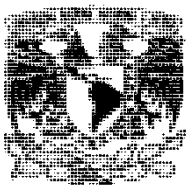


UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

WINDOWS COMMUNICATION FOUNDATION, UNA ALTERNATIVA PARA LA INTEROPERABILIDAD ENTRE PLATAFORMAS Y LENGUAJES DE PROGRAMACIÓN

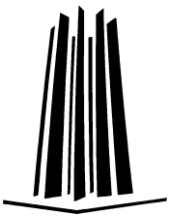
T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

P R E S E N T A:

RICARDO ALFREDO UGARTE REYES

ASESOR:
M. en C. JESÚS HERNÁNDEZ CABRERA



MÉXICO 2009

DEDICATORIA Y AGRADECIMIENTO

Para ti Fanny.
Esta tesis es para ti que hiciste
posible la terminación de esto.

Por ti Fanny.
Porque en los momentos difíciles,
eres mi motivación para seguir
siempre adelante.

Gracias Fanny.
Porque si no te hubiera conocido en aquel
ya lejano septiembre de 2005 nada
de esto hubiera sido posible.

Septiembre 2009.

ÍNDICE

CAPÍTULO 3: CARACTERÍSTICAS AVANZADAS DE WINDOWS COMMUNICATION FOUNDATION

CAPÍTULO 4. DESARROLLO DE UN SERVICIO WCF QUE PUEDE SER ACCEDIDO POR DIFERENTES CLIENTES WCF CONSTRUIDOS SOBRE DISTINTAS PLATAFORMAS Y LENGUAJES DE PROGRAMACIÓN

CAPÍTULO 5. DESARROLLO DE DIVERSOS CLIENTES WCF QUE HAGAN USO DE LAS OPERACIONES EXPUESTAS POR UN SERVICIO WCF.

INTRODUCCIÓN

La Ingeniería de Software es un conjunto de técnicas, procedimientos, reglas, normas o protocolos que se utilizan en la producción y mantenimiento de software de calidad. Sobre esta premisa se debería construir todo el software en México y en el mundo, sin embargo, esto no siempre es así.

Actualmente las empresas construyen sus sistemas y aplicaciones de software en una gran gama de lenguajes de programación, el producto final se ejecuta sobre una variedad de sistemas operativos. Suele pasar que, por diversas tomas de decisiones, los programas de una empresa requieren comunicarse con algunas aplicaciones de software de otra empresa para cumplir objetivos específicos. Si las aplicaciones de la segunda empresa están escritas en un lenguaje de programación distinto a las de la primera y además se ejecutan en un sistema operativo diferente, generalmente esto causa un gran problema de interoperabilidad. La solución primaria a este problema es el uso de servicios web básicos (conjunto de protocolos y estándares que permiten intercambiar información entre distintas aplicaciones desarrolladas en lenguajes de programación diferentes y que se ejecutan sobre cualquier plataforma) que se encuentran en algunos lenguajes de programación, sin embargo, la principal desventaja de estos servicios es que no están basados totalmente en las especificaciones *WS-**(modo abreviado para el creciente conjunto de especificaciones de servicios web (*WS*) tales como *WS-Security*, *WS-Trust*, *WS-Federation*, *WS-Policy*, *WSDL*, *WS-Management*, *WS-Coordination*, etc.), por ende surgen desventajas subyacentes al hacer uso de ellos; por ejemplo, no se garantiza la integridad de la información que se transfiere, inflexibilidad en el modo de transporte, etc., por lo tanto aunque los servicios web básicos resuelven el problema de interoperabilidad mencionado, surgen otros problemas más a resolver. El uso de las características extendidas de los servicios web como seguridad, transacciones, confiabilidad, etc. contenidas en las especificaciones *WS-** soluciona estos nuevos problemas. Aunque no es imposible construir servicios web que habiliten este tipo de características (por ejemplo en *.NET Framework* se debería usar *Enterprise Services* para administrar periodos de duración de objetos y definir transacciones distribuidas; *.NET Framework remoting* para la comunicación de *.NET Framework* a *.NET Framework*; *Web Services Enhancements* para implementar los acuerdo más recientes de los servicios web definidos, conocidos como *WS-** con la condición que todas las aplicaciones implicadas sean compatibles con las versiones de estas nuevas especificaciones o *Microsoft Message Queuing* para garantizar la entrega de datos), se expone a los desarrolladores de una aplicación de este tipo a una complejidad inmanejable tanto en su construcción como en su mantenimiento, provocando un bajo rendimiento del sistema.

Al momento de desarrollar esta tesis existen dos alternativas predominantes que implementan la mayoría de las especificaciones *WS-** y que dan solución total a los problemas anteriores, éstas son: *Windows Communication Foundation (WCF)* de *.NET Framework* y *Web Services Interoperability Technologies (WSIT)* de *Java*. Esta tesis se enfocará principalmente en *Windows Communication Foundation* por dos razones: por ser la tecnología que primariamente implementó la gran mayoría de las especificaciones *WS-** y para demostrar la potencialidad de *.NET Framework* como plataforma de programación de software. Esto no significa que se hará a un lado *Web Services Interoperability Technologies*, también se mostrará su funcionamiento interoperando con *Windows Communication Foundation*.

Windows Communication Foundation permite la fácil implementación de servicios web bajo el modelo de programación orientada a servicios en lenguajes de programación como *C#*, *C++*, etc.

para *.NET Framework*. Incluye funciones de serialización, control de versiones, integración e interoperabilidad con sistemas distribuidos construidos en el propio *.NET Framework* o en diferentes plataformas de software como *Java* y otras funciones más. *Windows Communication Foundation* implementa los servicios web interoperables que se complementan con seguridad multiplataforma, confiabilidad, transacciones y otros servicios. Las comunicaciones utilizan los protocolos de servicios web estándar entre otros.

Windows Communication Foundation se implementa esencialmente como un conjunto de clases encima del *Common Laanguage Runtime (CLR)* de *.NET Framework 2.0*. Por lo tanto, dado que extiende el entorno natural de programación de *.NET Framework*, *Windows Communication Foundation* permite crear aplicaciones orientadas a servicios y aplicaciones orientadas a objetos de una manera muy similar.

Aunque *Windows Communication Foundation* implanta un nuevo entorno de desarrollo para aplicaciones distribuidas comparado con los servicios web (*ASMX*) de *ASP.NET*, también está diseñado para interoperar perfectamente con aplicaciones que puedan o no estar construidas en *.NET Framework*. *Windows Communication Foundation* se diseñó para un escenario como el descrito al principio de esta introducción el cual es diverso pero realista y es la tecnología de *.NET Framework* para las aplicaciones que exponen y tienen acceso a servicios web.

Los diversos requerimientos de comunicación con una aplicación de servicio web construida con *Windows Communication Foundation* no son simples. Por ejemplo, en la interacción con una aplicación cliente construida con *.NET Framework*, el rendimiento se vuelve lo más importante, mientras que la interoperabilidad es trivial, porque ambos están generados en *.NET Framework*. Sin embargo, para la comunicación con una aplicación construida por ejemplo en *J2SE* usando *Web Services Interoperability Technologies*, la interoperabilidad se vuelve el objetivo principal. Los requerimientos de seguridad también son muy diferentes.

Si la aplicación de servicio web se construyera únicamente con *.NET Framework 2.0*, la aplicación debería utilizar más de una tecnología de comunicación para cumplir con requerimientos establecidos (seguridad, confiabilidad, etc.). Aunque esto es técnicamente posible, la aplicación resultante sería demasiado compleja de implementar y su posterior mantenimiento sería todo un reto. Con *Windows Communication Foundation*, la solución es mucho más fácil de implementar. *Windows Communication Foundation* se puede utilizar para todas las situaciones que implicarían el uso de varias tecnologías. Algunas de las características importantes de *Windows Communication Foundation* son las siguientes:

- ✓ Puesto que *Windows Communication Foundation* puede comunicarse utilizando servicios web estándar, la interoperabilidad es sencilla con otras plataformas que también admiten *Simple Object Access Protocol (SOAP)*. También se puede configurar y extender a *Windows Communication Foundation* para comunicarse con los servicios web utilizando mensajes no basados en *Simple Object Access Protocol*.
- ✓ Dado que el rendimiento de las aplicaciones es una preocupación primordial para la mayoría de los negocios. *Windows Communication Foundation* se desarrolló con el objetivo de ser una de las plataformas de aplicaciones distribuidas más rápidas.
- ✓ Cuando ambas partes de una comunicación se generan en *Windows Communication Foundation*, para permitir un rendimiento óptimo, la codificación de la conexión utilizada es una versión binaria optimizada de un conjunto de información *XML*. Los mensajes todavía cumplen con la estructura de datos de un mensaje *Simple Object Access Protocol*,

excepto su codificación pues utiliza una representación binaria de esa estructura de datos en lugar del formato de corchetes angulares y texto estándar de la codificación de texto de *XML 1.0*. El utilizar esta opción, tiene sentido para comunicarse con una aplicación cliente construida con *.NET Framework* porque el rendimiento es una inquietud importante. Cuando la aplicación cliente está construida por ejemplo en *Web Services Interoperability Technologies,* se utilizan como codificación corchetes angulares y texto estándar de la codificación de texto de *XML 1.0* pues la inquietud aquí es la interoperabilidad.

- ✓ La gestión de los periodos de duración de objetos, la definición de las transacciones distribuidas y otros aspectos son proporcionados por *Windows Communication Foundation*. Están disponibles para cualquier aplicación basada en *Windows Communication Foundation*, lo que significa que la aplicación puede utilizarlos con cualquiera de las otras aplicaciones con las que se comunica.
- ✓ Dado que admite un conjunto grande de las especificaciones de *WS-**, *Windows Communication Foundation* ayuda a proporcionar confiabilidad, seguridad y transacciones al comunicarse con cualquier plataforma (por ejemplo *Web Services Interoperability Technologies*) que también admite estas especificaciones.

El resultado de la unificación de tecnologías de *.NET Framework 2.0* y la implementación de las especificaciones *WS-** en *Windows Communication Foundation* de *.NET Framework 3.0* y *3.5* es una mayor funcionalidad y una complejidad significativamente reducida. Esta tesis ***Windows Communication Foundation, una alternativa para la interoperabilidad entre plataformas y lenguajes de programación*** proporciona una introducción a *Windows Communication Foundation*, mientras examina sus características y muestra su forma de uso. A continuación se dará una breve descripción de la estructura de este trabajo de tesis:

Capítulo 1. Generalidades de .NET Framework. El objetivo de este capítulo es dar una referencia breve pero concisa de la potencialidad y funcionamiento de *.NET Framework* así como del lenguaje de programación *C#*.

Capítulo 2. Características básicas de Windows Communication Foundation. El objetivo de este capítulo es proporcionar información sobre los elementos que permiten generar servicios web de *Windows Communication Foundation* básicos.

Capítulo 3. Características avanzadas de Windows Communication Foundation. El objetivo de este capítulo es mostrar características avanzadas que se pueden utilizar para crear servicios web de *Windows Communication Foundation* más complejos tomando como base las características básicas de los servicios web de *Windows Communication Foundation*.

Capítulo 4. Desarrollo de un servicio WCF que puede ser accedido por diferentes clientes WCF/WSIT construidos sobre distintas plataformas y lenguajes de programación. El objetivo de este capítulo es desarrollar (siguiendo las etapas del desarrollo de software) un servicio *Windows Communication Foundation* que haga uso de diversas características de los servicios web (por ejemplo: la implementación de seguridad vía certificados *X.509* para garantizar la integridad de la información, etc.), este servicio posteriormente será consumido por diferentes clientes para demostrar la interoperabilidad que proporciona *Windows Communication Foundation*. La construcción de este servicio se hará utilizando el lenguaje de programación *C# 3.0,* también se basará en las más recientes tecnologías de *.NET Framework* como *LINQ To SQL,* ésta es una herramienta que permite la fácil y eficiente manipulación de bases de datos relacionales permitiendo al usuario realizar consultas de

forma nativa en el lenguaje de programación que desee (*C#* en este caso), estas consultas son convertidas a *SQL* de forma trasparente al usuario cuando se requiere su ejecución. Una de sus principales ventajas es la de disminuir notablemente los errores que se producen al usar otras herramientas de acceso a bases de datos como *ADO.NET* de *.NET Framework* o *JDBC* de *Java* e incrementa el rendimiento de las aplicaciones.

Capítulo 5. Desarrollo de diversos clientes WCF/WSIT que hagan uso de las operaciones expuestas por un servicio WCF. El objetivo de este capítulo es la de demostrar que sin importar que tipo de aplicación cliente acceda al servicio *Windows Communication Foundation,* ésta será completamente interoperable. Siguiendo las etapas de desarrollo de software se construirán los siguientes clientes para consumir el servicio:

1) Un cliente de aplicación de escritorio escrito en *C# 3.0* para *.NET Framework 3.5 SP1* usando *Windows Presentation Foundation* para la *GUI*. Este cliente demostrará la interoperabilidad entre aplicaciones escritas en el mismo lenguaje de programación (*C#*) para la misma plataforma de programación de software (*.NET Framework*)
2) Un cliente de aplicación de escritorio escrito en *Java 6.0* para *J2SE 6.0 Update 4* usando *Swing* para la *GUI*. Este cliente demostrará la interoperabilidad entre aplicaciones escritas en diferentes lenguajes de programación (*C# – Java*) para diferentes plataformas de programación de software (*.NET Framework – J2SE*).
3) Un cliente de aplicación de escritorio escrito en *Visual Basic 9.0* para *.NET Framework 3.5 SP1* usando *Windows Forms* para la *GUI*. Este cliente demostrará la interoperabilidad entre aplicaciones escritas en diferentes lenguajes de programación (*C# – VB*) para la misma plataforma de programación de software (*.NET Framework*)
4) Un cliente de aplicación web escrito en *C# 3.0* usando *ASP.NET 3.5* para *.NET Framework 3.5 SP1*. Este cliente demostrará que la interoperabilidad presentada en los tres clientes anteriores no es exclusiva de aplicaciones de escritorio, sino que también se puede utilizar para aplicaciones web.

CAPÍTULO 1.
GENERALIDADES DE .NET FRAMEWORK.

1.1 .NET FRAMEWORK

.NET Framework es una plataforma de programación de software creada originalmente para su uso exclusivo en el sistema operativo *Windows.* Actualmente debido a la estandarización de la *Common Language Infraestructure (CLI)* y del lenguaje de programación *C#* a través de la *Organización Internacional de Estándares* (*ISO-23271 (CLI), ISO-23270 (C#)*) y de *Ecma International* (*ECMA-335 (CLI), ECMA-334 (C#)*) se pueden construir máquinas virtuales para distintos sistemas operativos. Un ejemplo de máquina virtual diferente a la de *Microsoft* basada en el estándar de *Common Language Infraestructure* es la *máquina virtual Mono* de El Proyecto Mono (*The Mono Project*). La creación de máquinas virtuales para diferentes plataformas provoca que al igual y como sucede con *Java* no se tenga la necesidad de recompilar o volver a escribir el código fuente original para poder ejecutar una aplicación de software en una plataforma específica.

Los objetivos para los que *.NET Framework* fue planeado son los siguientes:

- ✓ Dar al programador de aplicaciones de software un entorno de programación orientada a objetos. Los programas generados en este entorno se puede guardar y ejecutar en cualquiera de los siguientes tres modos: de modo local, de modo remoto o de modo mixto.
- ✓ Reducir de la mayor forma posible la implementación de software y problemas entre diferentes versiones (muy comunes en las *Dynamic Link Library* o *DLL*); ejecutar de forma segura el mismo, y eliminar los problemas de rendimiento que ofrecen otras plataformas de software similares.
- ✓ Unificar la experiencia de desarrollo de software entre aplicaciones muy diferentes, como pueden ser aplicaciones de escritorio, aplicaciones web o aplicaciones de servicios.
- ✓ Asegurar que el código de *.NET Framework* se puede integrar con diversos tipos de código estableciendo todo el funcionamiento y comunicación en estándares internacionales.

.NET Framework está compuesto por dos elementos principales: el *Common Language Runtime (CLR)* y la *biblioteca de clases base (BCL).* Estos elementos se explicarán más adelante en esta tesis.

1.1.1 .NET FRAMEWORK 1.0 y 1.1

Estas son las primeras versiones de *.NET Framework* aparecidas en el año 2002 y 2003 respectivamente, se mencionan en esta tesis para tener un marco histórico completo sobre las versiones de la plataforma, sin embargo actualmente ambas están en desuso. Estas versiones fueron sustituidas completamente por *.NET Framework 2.0* en el año 2005. La versión *3.0* en el año 2006 y la versión *3.5* en el año 2007 se integran sobre la versión *2.0*. La versión *4.0* (actualmente en la versión *Beta 1* al escribir estas líneas) mejorará la versión *2.0* al integrar por ejemplo computación paralela al modo de programación ya existente.

1.1.1.1 .NET FRAMEWORK 1.0

La primera versión de *.NET Framework* fue publicada en el año 2002. *.NET Framework 1.0* es compatible con *Windows 98*, *NT 4.0*, *2000* y *XP*. La versión *1.0* de *.NET Framework* forma parte de la primera versión del entorno de desarrollo integrado (*Integrated Development Environment, IDE*) *Microsoft Visual Studio .NET* (*Visual Studio 2002, versión 7.0*)

1.1.1.2 .NET FRAMEWORK 1.1

Esta es la primera actualización mayor de *.NET Framework 1.0* publicada en el año 2003. Esta versión forma parte de la segunda versión del entorno de desarrollo integrado *Microsoft Visual Studio .NET* (*Visual Studio 2003, versión 7.1*).

Cambios en la versión 1.1 en comparación con la versión 1.0

- ✓ Se permite que los ensamblados de *Windows Forms* se ejecutan de una manera semiconfiable desde Internet, se habilita el código de acceso de seguridad en aplicaciones web creadas en *ASP.NET.*
- ✓ Soporte nativo para *Open Database Connectivity* (*ODBC*) y para el Sistema Manejador de Bases de datos de *Oracle*. Anteriormente estaban disponible como agregados de *.NET Framework 1.0*.
- ✓ Aparece *.NET Compact Framework*, una versión de *.NET Framework* se utiliza para programar dispositivos móviles como *Asistentes Digitales Personales* (*PDA*), *teléfonos celulares*, etc.
- ✓ Soporte para el *Protocolo de Internet* (*Internet Protocol, IP*) versión *6* (*IPv6*).
- ✓ Numerosos cambios a las diversas clases que conformaban la versión anterior de *.NET Framework*.

1.1.2 COMPATIBILIDAD DE VERSIONES

En *.NET Framework*, la compatibilidad con versiones anteriores significa que un programa generado en una versión anterior de *.NET Framework* se debe ejecutar también en una versión posterior. La compatibilidad con versiones posteriores quiere decir que un programa generado usando una versión posterior de *.NET Framework* se deberá ejecutar también en una versión anterior.

.NET Framework es compatible con versiones anteriores. Por ejemplo, programas creados con la versión *2.0* se ejecutarán en la versión *3.0*, y programas que utilizan la versión *3.0* se ejecutarán en la versión 3.5. Sin embargo, como pasa en muchas otras plataformas de software, para que la compatibilidad con versiones posteriores sea óptima, es necesario modificar el programa para que se ejecute de forma correcta. Los programas que hayan sido creados con la versión *2.0* no se ejecutarán en versiones anteriores de *.NET Framework* (*1.0 y 1.1*) debido a que el *Common Language Runtime (CLR)* es distinto. La compatibilidad con versiones anteriores y con versiones posteriores, puede verse afectada si se realiza un cambio en *.NET Framework* que ayude a mejorar la seguridad o corrija una funcionalidad. La relación existente entre las versiones *2.0*, *3.0*, *3.5* y *4.0* de *.NET Framework* es distinto a la relación a las versiones *1.0*, *1.1* (ambas en desuso) y *2.0* de *.NET Framework*, éstas son completamente independientes unas de otras. Cuando las versiones *1.0*, *1.1* y *2.0* están en una misma computadora, cada una tiene su propio *Common Language Runtime*, su propia *biblioteca de clases base (BCL)*, su propio compilador, etc.

1.1.3 ARQUITECTURA

La versión *3.5 SP1* de *.NET Framework* (la más reciente y estable al escribir estas líneas) se basa en las versiones *2.0* y *3.0* y sus *Service Pack*[1] correspondientes. Esta versión actualiza los ensamblados de la versión *3.5* e incluye nuevos *Service Pack* para las versiones *2.0* y *3.0*.

[1] Los *Service Pack* (o en la sigla en inglés *SP*) consisten en un grupo de parches que actualizan, corrigen y mejoran aplicaciones y sistemas operativos. Esta denominación fue popularizada por *Microsoft* cuando comenzó a empaquetar grupos de parches que actualizaban su sistema operativo *Windows*. Fuente: Service Pack. http://es.wikipedia.org/wiki/Service_Pack

1.1.3.1 RELACIÓN ENTRE LAS VERSIONES 2.0, 3.0 Y 3.5 DE .NET FRAMEWORK

Los elementos que se listan enseguida forman parte de *.NET Framework 3.5 SP1*:

- ✓ *.NET Framework 2.0* y sus *Service Pack 1 y 2,* estos últimos mantienen actualizados los ensamblados de esta versión.
- ✓ *.NET Framework 3.0*, esta versión hace uso de los ensamblados de *.NET Framework 2.0* y sus *Service Pack* e incluye los ensamblados para las tecnologías propias de la versión (por ejemplo *PresentationFramework.dll* y *PresentationCore.dll* para *Windows Presentation Foundation (WPF)*). Los *Service Pack 1 y 2* de *.NET Framework 3.0*, actualizan los ensamblados de la versión.
- ✓ *.NET Framework 3.5*, esta versión incluye ensamblados que dan una funcionalidad extra a *.NET Framework 2.0* y *3.0*, por ejemplo *System.Data.Linq.dll* y *System.Xml.Linq.dll* para hacer uso de *LINQ.* El *Service Pack 1*, actualiza los ensamblados que son parte de *.NET Framework 3.5*.

Una versión posterior de *.NET Framework* instalará automáticamente las versiones anteriores si éstas aún no están instaladas. Una aplicación de software que hace uso de *Windows Presentation Foundation* y tiene como destino de uso a *.NET Framework 3.0* utiliza la misma instancia del ensamblado *mscorlib.dll* que una aplicación que utiliza *Windows Forms* y que tiene como destino de uso *.NET Framework 2.0*. Si el usuario final de una aplicación de software instaló una versión posterior de *.NET Framework* o actualiza su ensamblado *mscorlib.dll*, las dos aplicaciones utilizan la nueva versión del ensamblado.

1.1.3.2 CARACTERÍSTICAS INCLUIDAS EN .NET FRAMEWORK 2.0 Y POSTERIOR

A continuación se da una recapitulación de las tecnologías incluidas en *.NET Framework*. Esta lista únicamente incluye las principales características. No se menciona la versión *4.0* porque aun está en fase de desarrollo (*Beta 1* al escribir estas líneas) y las características pueden variar en la versión final de ésta.

.NET Framework 2.0, .NET Framework 2.0 SP1 y .NET Framework 2.0 SP2

Las siguientes tecnologías forman parte de *.NET Framework 2.0*.

- ✓ Compiladores para los lenguajes de programación *C#*, *C++*, *J#*. y *Visual Basic*.
- ✓ Common Language Runtime (CLR) y bibliotecas de clases base (BCL).
- ✓ Windows Forms.
- ✓ ASP.NET 2.0 y servicios web ASMX.
- ✓ ADO.NET 2.0.
- ✓ Tipos genéricos.

Los *Service Pack 1* y *2* de *.NET Framework 2.0* actualizan varios ensamblados de *.NET Framework 2.0*. Si un programa está basado en características introducidas en *.NET Framework 2.0 SP1* o *SP2* se puede usar *.NET Framework 2.0* como destino del programa y pedir al usuario que descargue desde Internet *.NET Framework 2.0 SP1* o *SP2*. Sin embargo, si el programa se basa en una funcionalidad más reciente a estos *Service Pack*, se recomienda el uso de *.NET Framework 3.5* como destino de la aplicación.

.NET Framework 3.0, .NET Framework 3.0 SP1 y .NET Framework 3.0 SP2

Para que *.NET Framework 3.0* pueda instalarse se necesita que *.NET Framework 2.0* esté instalado previamente en la computadora donde se desea usar. Si no es el caso, se instalará automáticamente.

Las siguientes tecnologías forman parte de *.NET Framework 3.0*:

- ✓ Windows Communications Foundation (*WCF*).
- ✓ Windows Presentation Foundation (*WPF*).
- ✓ Windows Workflow Foundation (*WF*).
- ✓ Windows CardSpace.

El *Service Pack 1* y el *Service Pack 2* de *.NET Framework 3.0* actualizan ensamblados que se incluyen con *.NET Framework 3.0* y agregan nuevas funcionalidades. Al igual que ocurre con la versión *2.0* , si el programa se basa en una funcionalidad más reciente a estos *Service Pack*, se recomienda el uso de *.NET Framework 3.5* como destino de la aplicación.

.NET Framework 3.5 y .NET Framework 3.5 SP1

.NET Framework 3.5 agrega nuevas características sus versiones predecesoras e incorpora tecnologías nuevas por medio de nuevos ensamblados.

Las tecnologías siguientes forman parte de la versión *3.5*:

- ✓ Nuevos compiladores para *C#*, *Visual Basic* y *C++*.
- ✓ Language Integrated Query (*LINQ*).
- ✓ ASP.NET AJAX.

El *Service Pack 1* de *.NET Framework 3.5* actualiza los ensamblados de *.NET Framework 3.5*. Las siguientes tecnologías forman parte de *.NET Framework 3.5 SP1*:

- ✓ ADO.NET Entity Framework.
- ✓ Datos dinámicos de *ASP.NET*.
- ✓ Compatibilidad con *SQL Server 2008*.
- ✓ Compatibilidad con *.NET Framework Client Profile* (únicamente contiene los ensamblados que se utilizan en las aplicaciones cliente).

1.1.4 COMMON LANGUAGE RUNTIME (CLR)

Para que la máquina virtual o *Common Language Runtime (CLR)* dé servicios al código administrado[2], el compilador de cualquier lenguaje enfocado a *.NET Framework* debe emitir metadatos los cuales describen tipos y referencias de código. Los metadatos se guardan junto con el código, un archivo ejecutable portable (*PE*) contiene metadatos. El *Common Language Runtime* hace uso de los metadatos para: encontrar y cargar clases, colocar instancias en memoria, invocar métodos, generar código nativo, exigir seguridad y controlar recursos en tiempo de ejecución.

La máquina virtual de *.NET Framework* administra de forma automática la existencia de los objetos creados por un programa y controla las referencias a éstos, liberándolos cuando ya no son necesarios. La recolección de elementos no utilizados (similar al utilizado en *Java*) elimina pérdidas de memoria así como otros errores habituales de programación (muy comunes en el lenguaje de programación *C* o *C++*). Aunque se recomienda utilizar completamente código administrado se puede utilizar código administrado y código no administrado (que no es reclamado por el recolector) conjuntamente en un programa de *.NET Framework*.

El *Common Language Runtime* facilita el diseño de aplicaciones cuyos objetos interactúan entre lenguajes de programación distintos. Los objetos que son creados en lenguajes de programación diferentes pueden comunicarse entre sí, esto permite integrar sus características de modo correc-

[2] Se denomina código administrado al código generado como resultado de compilar un lenguaje de programación enfocado a *Common Language Runtime (CLR)*. El código administrado hace uso de: la interoperabilidad, seguridad, control de excepciones, etc.

to. Para mayor información consultar el apartado: *1.1.6 Interoperabilidad entre lenguajes de programación* de esta tesis.

La máquina virtual utiliza metadatos para asegurar que la aplicación de software contiene todo lo necesario para su ejecución, esto garantiza que existen menos posibilidades de que la ejecución del código se interrumpa debido a una dependencia incorrecta (como ocurre con las *DLL*).

Los pasos siguientes muestran cómo lograr una ejecución administrada por parte del *Common Language Runtime*:

1. Elegir un compilador que tenga como destino un lenguaje de programación enfocado a *.NET Framework.* Se pueden utilizar uno o varios compiladores de lenguajes de programación como: *C#*, *C++*, *Visual Basic*, *J#*, *Delphi.NET*, I*ronPython*, *IronRuby*, *Fortran.NET*, *A#*, *F#*, *L#*, *P#*, *S#*, *Eiffel*, *Perl*, *Cobol*, etcétera.
2. Compilar el código a *Common Intermediate Language (CIL):* El compilador convierte el código fuente de un lenguaje enfocado a *.NET Framework* a *Common Intermediate Language* y crea los metadatos necesarios para su ejecución.
3. Compilar el *Common Intermediate Language* a código nativo: En tiempo de ejecución, un compilador *Just-In-Time (JIT)* convierte el *Common Intermediate Language* en código nativo (depende de la plataforma donde se ejecute el programa resultante).
4. Ejecutar código: El *Common Language Runtime* proporciona los recursos que permiten la ejecución de la aplicación de software.

1.1.5 BASE CLASS LIBRARY (BCL)

La biblioteca de clases base (*Base Class Library, BCL*) de *.NET Framework* es un conjunto de clases reutilizables que se integran con *Common Language Runtime (CLR)*. Para garantizar la interoperabilidad entre diferentes lenguajes de programación, la mayoría de los tipos de estas clases cumplen la *especificación de lenguaje común (CLS)* por lo cual, se pueden usar en cualquier lenguaje de programación siempre y cuando su compilador cumpla con la *CLS*.

Las clases de *.NET Framework* son la base sobre la que se crean programas en esta plataforma de programación de software, contiene clases que pueden cumplir con las siguientes funciones:

- ✓ Acceder a datos e interfaces gráficas de usuario (*GUI*).
- ✓ Ejecutar operaciones de Entrada/Salida.
- ✓ Encapsulamiento de estructuras de datos.
- ✓ Obtener información sobre objetos cargados en memoria.
- ✓ Realizar comprobaciones de seguridad.
- ✓ Representar clases base y excepciones.

Un programador de *.NET Framework* puede hacer uso de un repertorio completo de interfaces, clases abstractas y concretas. Es posible utilizar las clases sin ninguna modificación o derivarlas para extender su funcionalidad. Para hacer uso de una interfaz se debe crear una clase que la implemente o hacer una derivación de una clase que implemente la interfaz.

Las clases de *.NET Framework* utilizan una nomenclatura con sintaxis de punto como en *Java* u otros lenguajes de programación, esto indica una jerarquía. Esta sintaxis agrupa clases relacionadas en espacios de nombres (paquetes en *Java*) para hacer más fácil la referencia a ellos. La primera parte del nombre completo, hasta el punto situado más a la derecha es el nombre del espacio

de nombres (*namespace*), la última parte representa el nombre de la clase. Por ejemplo, *System.Collections.Generic.List<T>* representa a la clase *List<T>* (lista genérica) que pertenece al espacio de nombres *System.Collections.Generic* Los tipos del espacio de nombres *System.Collections.Generic* se usan para manipular colecciones genéricas de objetos.

System es el espacio de nombres raíz de las clases principales de *.NET Framework*. Este espacio de nombres tiene clases y estructuras que se utilizan en la mayoría de las aplicaciones de software como son: *Object* (raíz de la jerarquía de herencia), *Double*, *Int32, String, Char*, *Array*, etc. Muchas de estas clases son análogas a los tipos de datos primitivos que utilizan lenguajes de programación como *C++*. A parte de las clases base, el espacio de nombres *System* contiene otras clases, que cubren desde las clases que controlan excepciones hasta las clases que tratan aspectos a considerar en tiempo de ejecución como el recolector de elementos no utilizados (*Garbage Collector, GC*). *System* contiene espacios de nombres de segundo nivel o tercer nivel como *System.Data*, *System.Runtime*, *System.Imaging, System.Windows.Forms*, etc. En la tabla siguiente se muestra una pequeña lista de clases base que proporciona *.NET Framework*, se da una descripción breve de cada clase y se indica el tipo correspondiente en los lenguajes *C#*, *C++* y *Visual Basic*.

Categoría	Nombre de la clase	Descripción	Tipo en C#	Tipo en C++	Tipo en Visual Basic
Números enteros	Byte	Entero de 8 bits sin signo	byte	char	Byte
	SByte	Entero de 8 bits con signo. *	sbyte	signed char	SByte
	Int16	Entero de 16 bits con signo	short	short	Short
	Int32	Entero de 32 bits con signo	int	int ó long	Integer
	Int64	Entero de 64 bits con signo	long	__int64	Long
	UInt16	Entero de 16 bits sin signo. *	ushort	unsigned short	UShort
	UInt32	Entero de 32 bits sin signo. *	short	unsigned int ó unsigned long	UInteger
	UInt64	Entero de 64 bits sin signo. *	short	unsigned __int64	ULong
Punto flotante	Single	Número Flotante de 32 bits de precisión simple	float	float	Single
	Double	Número Flotante de 32 bits de precisión doble	double	double	Double
Lógicos	Boolean	Valor booleano (verdadero o falso)	bool	bool	Boolean
Otros	Char	Caracter Unicode de 16 bits	char	wchar_t	Char
	Decimal	Valor decimal de 128 bits	decimal	decimal	Decimal
	Object	Clase base en la jerarquía de objetos de .NET Framework	object	Object^	Object
	String	Cadena inmutable de longitud fija de caracteres Unicode.	string	String^	String

** No cumple la CLS (Porque no todos los lenguajes de programación aceptan números enteros sin signo)*

Tabla 1.1 Algunos tipos base proporcionados por .NET Framework

NOTA: En la tabla anterior se señaló el tipo propio de cada lenguaje, sin embargo esto no restringe el uso de la clase de la *BCL* directamente. Por ejemplo, en *C#* es lo mismo declarar un tipo booleano de la forma *bool nombre = true;* que *Boolean nombre = true;*

1.1.6 INTEROPERABILIDAD ENTRE LENGUAJES DE PROGRAMACIÓN

La interoperabilidad entre lenguajes de programación se le llama a la posibilidad de que el código escrito en un lenguaje de programación enfocado a *.NET Framework* pueda interactuar con código escrito en un lenguaje de programación diferente pero también enfocado a *.NET Framework*. La interoperabilidad permite optimizar la reutilización de código fuente, esto genera una mayor eficacia en el desarrollo de aplicaciones de software. Todos los lenguajes de programación enfocados a *.NET Framework* hacen uso del *Common Language Runtime (CLR).*

El *Common Language Runtime* ofrece una base para garantizar la interoperabilidad entre lenguajes de programación al imponer un sistema de tipos común (*Common Type System, CTS*), y al suministrar metadatos. Dado que todos los lenguajes de programación que usan el *Common Language Runtime* siguen las especificaciones del *sistema de tipos común* para definir y utilizar tipos, la utilización de tipos es eficiente y eficaz entre todos los lenguajes de programación. Los metadatos habilitan la interoperabilidad entre lenguajes de programación mediante un mecanismo que almacena y recupera la información de tipos. Un compilador almacena la información de tipos como *metadatos*, el *Common Language Runtime* los usa para proporcionar servicios mientras un programa se esté ejecutando. El *Common Language Runtime* administra la ejecución de programas construidos en varios lenguajes de programación debido a que la información de tipos es almacenada y recuperada de la misma manera, sin importar el lenguaje de programación en que se haya escrito el programa. El código administrado generado por un compilador de cualquier lenguaje de programación de *.NET Framework* debe cumplir con las siguientes características:

- ✓ El control de excepciones: Dado que el código fuente escrito en un lenguaje de programación específico puede lanzar una excepción en tiempo de ejecución, esa excepción debe ser interceptada y reconocida por cualquier objeto escrito en cualquier otro lenguaje de programación.
- ✓ Los lenguajes de programación enfocados a *.NET Framework* deben comprender el *CIL* (*Common intermediate language* o *Lenguaje intermedio común*) y los metadatos de *Common Language Runtime* para ser interoperables.
- ✓ Las clases pueden heredar la implementación de otras clases, con el fin de hacer uso de sus miembros sin importar en que lenguaje de programación se hayan escrito.

A pesar de que el *Common Language Runtime* permite que aplicaciones de software se ejecuten en un entorno de varios lenguajes de programación, no hay garantía de que funcionalidades específicas puedan ser utilizadas de forma total por lenguajes de programación usados por otros programadores. La razón principal es que el compilador de un lenguaje de programación enfocado a *.NET Framework* puede usar un sistema de tipos y metadatos específicos del lenguaje de programación. Si se desconoce el lenguaje de programación que va a hacer uso de los tipos, no se puede saber si las características expuestas van a estar accesibles. Si el lenguaje de programación permite usar enteros sin signo, se podría generar un método que acepte un parámetro de tipo *UInt64* (entero de 64 bits sin signo); sin embargo, ese método no sería útil para un lenguaje de programación que no tenga soporte para enteros sin signo. Entonces, para garantizar que el código administrado sea accesible desde cualquier lenguaje de programación, *.NET Framework* proporciona la *Common Language Specification (CLS)*.

La *Common Language Specification* es un conjunto de características básicas utilizadas por la mayoría de los lenguajes de programación. Las reglas de la *Common Language Specification* son un subconjunto del *Common Type System*, es decir, las reglas que se aplican al *Common Type System*

se aplican a *Common Language Specification*, salvo que reglas más estrictas en *Common Language Specification* digan lo contrario. La *Common Language Specification* mejora y garantiza la interoperabilidad entre lenguajes mediante la definición de características que están disponibles en una gran variedad de lenguajes de programación. La *Common Language Specification* se creó de forma que fuera lo suficientemente grande para incluir opciones que normalmente necesitan los programadores de software y lo suficientemente pequeño para que todos los lenguajes puedan aceptarlo. Además, se ha excluido de la *Common Language Specification* cualquier característica en la que no se permita verificar rápidamente la seguridad de tipos, de esta forma, todos los lenguajes compatibles con la *Common Language Specification* generan código que se puede comprobar.

1.1.7 COMMON LANGUAGE INFRASTRUCTURE (CLI)

La *Common Language Infrastructure (CLI)* es un estándar (*ISO 23271* y *ECMA 335*) que describe un entorno virtual para la ejecución de software, su principal característica es permitir que aplicaciones de software escritas en diferentes lenguajes de programación enfocados a *.NET Framework* se ejecuten en múltiples plataformas de hardware y software sin tener que reescribir o recompilar su código fuente. La arquitectura de *Common Language Infrastructure* debe:

- ✓ Realizar tareas de bajo nivel a través de *Just In Time (JIT).* Se cargan tipos en memoria, y se enlazan librerías y compilan a código nativo únicamente cuando es necesario.
- ✓ Otorgar la posibilidad de escribir aplicaciones de software interoperables sin importar en que plataforma se ejecute y en que lenguaje de programación se haya escrito.
- ✓ Cargar tipos de forma aislada en tiempo de ejecución, a su vez deben ser capaces de compartir recursos unos con otros.
- ✓ Empaquetar tipos en mediante metadatos y estos deben ser portables.
- ✓ Soportar que toda la arquitectura pueda acomodarse con poco impacto a nuevas incorporaciones y cambios.
- ✓ Resolver dependencias entre tipos en tiempo de ejecución haciendo uso de la descripción contenida en el ensamblado.
- ✓ Proveer todas las entidades de programación a través del sistema unificado de tipos (*CTS*, o *Common Type System*).
- ✓ Dar funcionalidades comunes mediante un conjunto de bibliotecas de clases que se puedan utilizar para generar aplicaciones de software.
- ✓ Ejecutar programas bajo un entorno que controle y cumpla con reglas específicas durante el tiempo de ejecución.

Algunos de los lenguajes de programación enfocados a *.NET Framework* y que cumplen por ende la *Common Language Infrastructure* son: *C#*, *C++*, *Visual Basic*, *J#*, *F#*, *Cobra, Delphi.NET*, *IronPython*, *P#*, *Chrome*, *Phalanger*, *IronRuby*, *Fortran.NET*, *A#*, *Boo*, *Nemerle*, *Mondrian*, *L#*, *Delta Forth.NET*, *S#*, *Eiffel*, *Perl*, *Cobol*, entre otros más.

1.1.8 C#

C# es un lenguaje de programación orientado a objetos que permite generar programas que se ejecutan en *.NET Framework*. Se usa para generar aplicaciones de escritorio, aplicaciones web, servicios web *XML*, aplicaciones distribuidas, aplicaciones de base de datos, y otras tareas más. *C#* es el lenguaje de programación más extendido de las plataformas de programación *.NET Framework* y *The Mono Project.*

La sintaxis del lenguaje de programación *C#* basada en llaves es muy similar a la sintaxis utilizada por los lenguajes de programación *C*, *C++* y *Java*. Los programadores que conocen cualquiera de estos lenguajes deberían crear aplicaciones de forma eficaz en *C#* de forma rápida. La sintaxis del lenguaje de programación *C#* reduce muchas de las complejidades de *C++* y otorga características como: delegados, tipos por valor que admiten valores *NULL*, enumeraciones, expresiones lambda y acceso directo a memoria a través de punteros, esta última característica y otras más no se encuentran en *Java*.

Al ser un lenguaje de programación orientado a objetos, *C#* cumple con los conceptos de herencia, encapsulamiento, y polimorfismo. Como en *Java,* todos los miembros (incluido el método *Main* que es el punto de entrada de la aplicación) se encuentran dentro de la definición de una clase. Una clase derivada puede heredar una y sólo una clase base, pero puede implementar un número *n* de interfaces tal y como lo hace *Java*. Los métodos que redefinen métodos virtuales de una clase base en una clase derivada requieren la palabra clave *override*, esto con el fin de evitar redefiniciones accidentales de métodos como suele suceder en *Java*. En *C#*, a diferencia de *Java*, existen las estructuras. Una estructura es como una clase, sin embargo, ésta puede implementar interfaces pero no permite la herencia.

A parte de los principios básicos de la programación orientada a objetos, *C#* facilita el desarrollo de de software a través características de lenguaje modernas como:

- ✓ Atributos que proporcionan metadatos sobre tipos que se usan en tiempo de ejecución.
- ✓ Propiedades que acceden a campos privados encapsulados.
- ✓ Documentación usando *XML*.
- ✓ *Delegados* que habilitan la notificación de eventos con seguridad de tipos.
- ✓ Language-Integrated Query (*LINQ*) que proporciona funciones de consulta integradas de forma nativa sobre una gran variedad de objetos que representan colecciones de datos.

C# permite la interacción con una aplicación de software nativo del sistema operativo haciendo uso de un proceso llamado *invocación de plataforma*. La invocación de plataforma (*P/Invoke*) permite a programas escritos en *C#* realizar las mismas tareas que ejecuta una aplicación escrita en *C++* de forma nativa. *C#* admite el uso de punteros (*pointers*) haciendo uso del concepto de *código no seguro* en los casos en donde el acceso directo a la memoria es totalmente indispensable. En comparación con los lenguajes de programación *C* y *C++*, no hay archivos de encabezado independientes, ni se requiere que los métodos y los tipos se declaren en un orden determinado. A diferencia de *Java*, un solo archivo de código fuente de *C#* puede definir cualquier número de clases, estructuras, interfaces y eventos.

1.1.9 COMMON INTERMEDIATE LANGUAGE (CIL)

El código fuente escrito en *C#* (o en cualquier otro lenguaje de programación enfocado a *.NET Framework*, aunque aquí se hace referencia a *C#*) se compila en un *lenguaje intermedio común* (*CIL*) de acuerdo al estándar de la *Common Language Infrastructure (CLI)*. El código de *lenguaje intermedio común* se almacena en un archivo ejecutable llamado ensamblado. Un ensamblado (**.exe* o **.dll*) contiene metadatos que proporcionan información sobre tipos y otros recursos. Cuando un programa escrito en *C#* se ejecuta, éste se carga en el *Common Language Runtime*, con lo que se realizan diversas acciones en función de la información de los metadatos. Si se cumplen los requisitos de seguridad, el *Common Language Runtime* realiza una compilación *Just In Time (JIT)* para convertir el código de *lenguaje intermedio común* en lenguaje de máquina nativo de

acuerdo a la plataforma. El código ejecutado por *Common Language Runtime* se denomina código administrado. En el diagrama siguiente se muestran las relaciones en tiempo de compilación y ejecución de archivos de código fuente de *C#*, el compilador de *C#*, los metadatos, el *Common Language Runtime,* la *Base Class Library* y el sistema operativo.

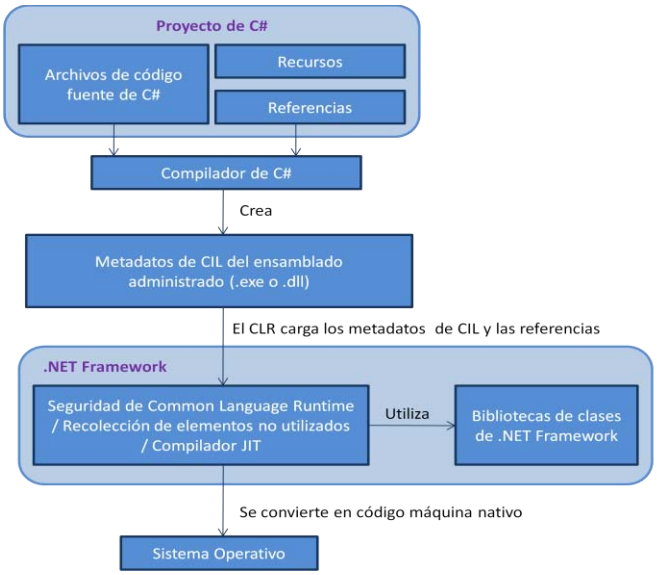


Figura 1.1 Relaciones en tiempo de compilación y tiempo de ejecución de los archivos de código fuente de *C#*, las bibliotecas de clases de .NET Framework, los ensamblados y el *CLR*.[3]

El *Common Intermediate Language* generado por el compilador de *C#* cumple con la *especificación de tipos común (CTS)*, este *Common Intermediate Language* puede interactuar con el *Common Intermediate Language* generado por cualquier lenguaje de programación que cumplan con la *CTS* de *.NET Framework*.

1.1.10 ADO.NET

Con el lanzamiento de *.NET Framework 1.0*, se presentó el modelo de acceso a datos: *ADO.NET 1.0* (*ActiveX Data Object .NET 1.0*). *ADO.NET* es la evolución de un modelo anterior (usado comúnmente en el lenguaje de programación Visual Basic 6.0) conocido como *ADO* (*ActiveX Data Object*). Aunque *ADO.NET* no es *ActiveX*[4], se mantuvo el nombre *ADO* como referencia a este modelo de acceso a bases de datos. *ADO.NET 1.0* proporciona la opción de trabajar con proveedores de datos específicos llamados Proveedores Administrados. Estos proveedores administrados ofrecen un mejor rendimiento ya que se comunican directamente con el origen de datos. También soporta comunicación con orígenes de datos vía *ODBC* u *OLEDB*, pero también.

ADO.NET presenta un modelo de clases como el siguiente:

- ✓ Connection: Representa una conexión con una base de datos de un sistema manejador de bases de datos (*SMBD*) relacional como *SQL Server*, *Oracle*, *MySQL, PosgreSQL*, etcétera.
- ✓ Command: Representa una instrucción de *SQL* que se ejecuta en una base de datos de un *Sistema Manejador de Bases de Datos*.

[3] Obtenido de: Introducción al lenguaje C# y .NET Framework. http://msdn.microsoft.com/es-es/library/z1zx9t92.aspx

[4] Las aplicaciones *ActiveX* están conceptualmente divididas en servidores, objetos que hacen que sus métodos y propiedades estén disponibles para los demás, y clientes, aplicaciones que usan objetos de servidor expuestos, métodos y propiedades. Algunos tipos de servidores, por ejemplo controles *ActiveX*, pueden disparar eventos que pueden ser después respondidos por el código de un cliente.

- ✓ DataReader: Proporciona una forma de leer una secuencia de registros sólo hacia delante en una base de datos de un *Sistema Manejador de Bases de Datos*.
- ✓ DataSet: Representa una caché de memoria interna de datos. Por ejemplo, se puede pensar como en un repositorio en memoria para almacenar datos que provienen de cualquier base de datos.
- ✓ DataAdapter: Representa un conjunto de comandos *SQL* y una conexión de base de datos que se utilizan para llenar el objeto *DataSet* y actualizar el origen de datos.

Los nombres de todas las clases mencionadas arriba son diferentes de acuerdo al *Sistema Manejador de Bases de Datos (SMBD)*. Al nombre de estas clases se le coloca un prefijo acorde al nombre del *SMBD*, por ejemplo para las 3 primeras clases, los nombres son como los que se muestran en la siguiente tabla:

Sistema Manejador de Bases de Datos	Nombre para Connection	Nombre para Command	Nombre para DataReader
SQL Server (Microsoft Corporation)	SqlConnection	SqlCommand	SqlDataReader
MySQL (Sun Microsystems)	MySqlConnection	MySqlCommand	MySqlDataReader
Oracle (Oracle Corporation)	OracleConnection	OracleCommand	OracleDataReader
PostgreSQL (PostgreSQL Global Development Group)	NpgsqlConnection	NpgsqlCommand	NpgsqlDataReader

Tabla 1.2 Nombres de las clases de ADO.NET de acuerdo al proveedor del SMBD

Las clases de la tabla anterior se encuentran en los siguientes espacios de nombres: *System.Data.SqlClient*, *MySql.Data.MySqlClient*, *System.Data.OracleClient* y *Npgsql* para *SQL Server*, *MySQL*, *Oracle* y *PostgreSQL* respectivamente. Cabe señalar que los controladores para *SQL Server* y *Oracle* se incluyen junto con *.NET Framework* mientras que el de *MySQL* y *PostgreSQL* se deben descargar desde la página web del *Sistema Manejador de Bases de Datos*.

Un tema importante que introdujo *ADO.NET 1.0*, fue el manejo de agrupamiento de conexiones, con esta característica se acabó con el costo excesivo que traía consigo el mantenimiento de una conexión abierta o la creación y destrucción de la misma. Con *ADO.NET* cuando una conexión es destruida en el código, el *Common Language Runtime (CLR)* de *.NET Framework* la mantiene disponible en un agrupamiento, de manera que la próxima vez que se necesite establecer nuevamente una conexión (con las mismas credenciales), la conexión es tomada del agrupamiento, sin que se necesite crear una nueva.

Un programador que utilice *Java* y que esté familiarizado con *Java Database Connectivity (JDBC)* notará que *ADO.NET* es similar a *JDBC,* en el procedimiento de acceso a bases de datos.

1.1.11 ASP.NET

ASP.NET permite el desarrollo de aplicaciones web con una cantidad mínima de código. *ASP.NET* es parte de *.NET Framework* y por ende puede acceder a su *biblioteca de clases base (BCL*). Las aplicaciones pueden escribirse en cualquier lenguaje de programación de *.NET Framework*.

ASP.NET es un *framework* de programación que se ejecuta en un servidor web para administrar de forma dinámica páginas web, este *framework* presenta un modelo unificado que responde a los

eventos generados por los clientes a través del código que se ejecuta en el servidor. Guarda de forma automática el estado de todos los controles de una página durante el ciclo vital de su procesamiento, permite encapsular la funcionalidad de la interfaz de usuario en controles cuyo funcionamiento es similar a los de *Windows Forms*. Los controles se pueden utilizar en diferentes páginas de *ASP.NET* visualizadas durante su ejecución. También contiene funciones para mantener una apariencia y comportamiento de los sitios web por medio de temas y máscaras.

Las páginas web creadas por *ASP.NET* pueden ser solicitadas por cualquier explorador de Internet del lado del cliente, entonces *ASP.NET* devuelve la página como *HTML* al explorador que realizó la solicitud. Se puede utilizar la misma página para varios exploradores, sin embargo, *ASP.NET* provee la capacidad de diseñar una página web que se ejecute en un explorador de Internet determinado, y de esta manera sacar provecho de las características propias del explorador. Las páginas web creadas con *ASP.NET* al hacer uso de *.NET Framework,* están totalmente orientadas a objetos. En las páginas web de *ASP.NET* se pueden utilizar elementos *HTML* que respondan a eventos y puedan trabajar con propiedades y métodos.

El compilador de *ASP.NET* compila todo el código que se haya escrito, esto permite obtener la seguridad de tipos y la mejora en el rendimiento entre otras ventajas. Una vez que se compiló el código, el *Common Language Runtime* compila una vez más el código de *ASP.NET* en código nativo, dando como resultado un mayor rendimiento. El resultado de compilar todos los componentes de una aplicación *ASP.NET* es un ensamblado que un entorno de hospedaje de *ASP.NET* (como puede ser *Internet Information Sservices, IIS*) puede utilizar para responder a las solicitudes de uno o varios clientes.

Servicios de seguridad

A parte de las características de seguridad propias de *.NET Framework*, *ASP.NET* proporciona clases de seguridad avanzada que permiten autenticar y autorizar el acceso a los usuarios entre otras tareas más. Es posible autenticar usuarios haciendo uso de una base de datos propia utilizando la autenticación mediante formularios *ASP.NET* (Esto se verá en el *capítulo 5* de esta tesis al crear el cliente de *ASP.NET* que accede al servicio *Windows Communication Foundation*). Es posible administrar por medio de roles las capacidades e información a las cuales puede acceder un usuario en una aplicación web mediante una base de datos que contenga la información de roles.

Funciones de administración de estado

ASP.NET suministra capacidades de administración de estado para almacenar información entre diferentes solicitudes de páginas web. Ejemplos de esta información son: la información de clientes o el contenido de un carro de la compra en comercio electrónico. Se puede manipular información definida por el programador, propia de la aplicación, propia de la sesión, propia del usuario y propia de la página.

Framework de servicios web XML

ASP.NET es compatible con el estándar de los servicios web *XML*. Un servicio web *XML* es una aplicación que puede implementar la lógica del negocio de una empresa y permite intercambiar información entre las aplicaciones utilizando servicios como el protocolo *HTTP* y la codificación *XML*. Dado que los servicios web *XML* no se encuentran relacionados con una convención en concreto, se puede tener acceso a los servicios web *XML* a través de programas escritos en cualquier lenguaje de programación, sin importar sobre qué sistema operativo se estén ejecutando.

Entorno de administración del ciclo de vida de las aplicaciones
ASP.NET incluye un entorno que controla el ciclo de vida de una aplicación desde que un usuario accede a un recurso en la aplicación hasta el momento en que se cierra la aplicación.

1.1.12 WINDOWS FORMS

Windows Forms es el nombre dado a la Interfaz de Programación de Aplicaciones (*API*) incluida como parte de *.NET Framework* desde la versión *1.0*. Proporciona acceso a elementos nativos de la interfaz gráfica de usuario del sistema operativo envolviéndola en código administrado. Si bien en el sistema operativo *Windows* se puede ver como un reemplazo de la anterior y compleja biblioteca *Microsoft Foundation Class* (*MFC*) creada en *C++*, no ofrece el *Modelo-Vista-Controlador* (*MVC*). Para hacer uso del *MVC* se debe utilizar *Windows Presentation Foundation (WPF)* de *.NET Framework 3.0*.

Al igual que *Abstract Window Toolkit* o *AWT* (el *API* de *Java* equivalente a *Windows Forms), Windows Forms* es una manera rápida y fácil para proporcionar componentes *GUI* desde *.NET Framework*. Por ejemplo, en el sistema operativo *Windows*, *Windows Forms* se basa en la *API* de la interfaz gráfica de usuario del sistema operativo. *Windows Presentation Foundation (WPF)* proporciona una forma mucho más independiente de la plataforma de diseño de interfaces gráficas de usuario contenida en el sistema operativo. Una aplicación de *Windows Forms* es una aplicación controlada por eventos soportados en *.NET Framework*. En el año 2008 el *API* de *System.Windows.Forms 2.0* de El *Proyecto Mono* (proyecto liderado por *Novell* para crear un conjunto de herramientas compatibles con *.NET Framework* basado en estándar de *ECMA* e *ISO*) se completó conteniendo el 100% de las clases de *System.Windows.Forms 2.0* de *.NET Framework*.

Un formulario de *Windows Forms* es una superficie visual que muestra información al usuario del programa. Generalmente las aplicaciones de *Windows Forms* se crean agregando controles (cuadros de texto, botones, listas, barras de herramientas, etc.) a los formularios y programando respuestas a eventos generados por el usuario, como un clic o doble clic del ratón o pulsación de teclas. Un control es un elemento que visualiza datos o permite leer la entrada de datos por parte del usuario del programa. La diferencia principal entre una aplicación de *Windows Forms* y una de *Consola* radica en que la primera reacciona a eventos que se ejecutan en tiempos aleatorios mientras que la segunda se ejecuta por lotes (una acción detrás de otra).

1.2 .NET FRAMEWORK 2.0

La versión *2.0* de *.NET Framework* es una mejora total a su predecesora. A continuación se muestra una imagen con los componentes principales de *.NET Framework 2.0* y una lista resumida de las nuevas características que presenta.

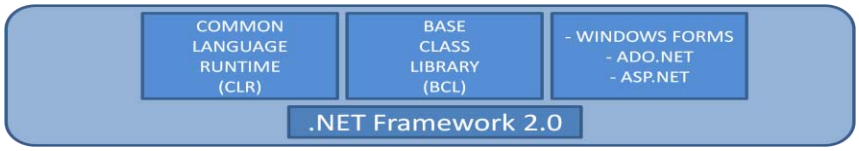


Figura 1.2. Componentes principales de .NET Framework 2.0

- ✓ Compatibilidad con plataformas de 64 bits: Permite crear aplicaciones de código administrado o utilizar aplicaciones de código no administrado en computadoras de *64 bits*.

- ✓ ADO.NET 2.0: Permite ejecutar operaciones asincrónicas a bases de datos, tipos de datos XML, atributos que permiten a las aplicaciones admitir varios conjuntos de resultados activos (*MARS*) con los sistemas manejadores de bases de datos, entre otros.
- ✓ Ejecutar solicitudes *TCP* desde aplicaciones: La clase *System.Net.Sockets.TcpListener*, permite crear sencillos servidores web que responden a las peticiones de un cliente. Este servidor web, permanece activo durante el período de vida del objeto System.Net.Sockets.*TcpListener*. La clase *System.Net.NetworkInformaction.Ping* permite determinar si una computadora remota se encuentra disponible desde una red. Este tipo es compatible con llamadas sincrónicas y asincrónicas.
- ✓ Redes: El espacio de nombres *System.Net* permite manejar aspectos relacionados con cualquier tipo de red.
- ✓ ASP.NET 2.0: Se realizaron mejoras importantes en todas las áreas de *ASP.NET*. Se agregaron controles nuevos que hacen más fácil la creación de páginas web dinámicas.
- ✓ Excepciones de seguridad: Se mejoró la clase *System.Security.SecurityException* para obtener de forma más sencilla las causas de excepciones de seguridad producidas en una aplicación.
- ✓ Obtener información de redes: Permite acceder a información referente a redes a través de las clases *IP, IPv4, IPv6, TCP y UDP* del espacio de nombres *System.Net.NetworkInformation.*
- ✓ Mejoras en servicios de interoperabilidad *COM*: Se mejoró el cálculo de referencias facilitando la interoperabilidad con código nativo del sistema operativo. Se aumentó el rendimiento de las llamadas entre aplicaciones en diferentes dominios de aplicación.
- ✓ Tipos y colecciones genéricas: Los tipos genéricos permiten declarar y definir clases, estructuras, interfaces, métodos y delegados con parámetros de tipo genérico. Los tipos genéricos generan un mayor rendimiento en las aplicaciones de software.
- ✓ Mejoras en funciones de Entrada/Salida: Se aumentó la funcionalidad de varias clases que implican procedimientos de *Entrada/Salida*. Se hace más fácil la manipulación de archivos de texto, la obtención de información sobre una unidad de disco, etc.
- ✓ Administración de certificados: Se agregan nuevas características para tratar todo lo relacionado a certificados *X.509*.
- ✓ Compatibilidad con *FTP*: Se permite acceder a recursos del Protocolo de transferencia de archivos (*FTP*) mediante las clases *WebRequest*, *WebResponse* y *WebClient* de *System.Net.*
- ✓ *Simple Mail Transfer Protocol (SMTP)*: Tanto el espacio de nombres *System.Net.Mail* como *System.Net.Mime* contienen clases que permiten a las aplicaciones de software enviar correo electrónico usando cualquiera de las opciones contenidas en el protocolo *SMTP*.
- ✓ Puerto serie: La clase *System.IO.Ports.SerialPort* permite al programador acceder a los puertos serie de la computadora cuando un programa lo requiera.

1.2.1 C# 2.0

A la par de la liberación de *.NET Framework 2.0*, aparece el lenguaje de programación *C# 2.0*, esta versión contiene las siguientes nuevas características:

- ✓ Clases estáticas: Estas son clases que solamente contienen miembros estáticos (no se pueden crear instancias de este tipo de clases). Se deben usar las palabras reservadas *static class* para definirlas. Antes de esta versión, se debía obligatoriamente definir el constructor de la clase con un ámbito privado (*private*) para evitar la creación de una instancia.
- ✓ Tipos genéricos: Los tipos genéricos se introdujeron al lenguaje para asegurar su compatibilidad con *.NET Framework 2.0*. Estos tipos permiten lograr un gran nivel de reutilización

de código e incrementan el rendimiento de las clases de colección. Los genéricos únicamente se diferencian por el tipo de dato específico que almacena. Con esta innovación es posible forzar que los parámetros de tipos sean tipos específicos.

- ✓ Métodos anónimos: Otorga al programador la capacidad de pasar como parámetro un bloque de código. Siempre que se espere un delegado, se puede utilizar un bloque de código en su lugar.
- ✓ Clases parciales: La definición de una clase parcial permite dividirla en varios archivos. Por ejemplo, el *diseñador de Windows* y el *diseñador relacional de Objetos* para *LINQ* de *Visual Studio* hacen uso esta particularidad para separar el código autogenerado del código propio de usuario. Para utilizar esta característica se deben utilizar las palabras reservadas *partial class*.
- ✓ Iteradores: El bucle *foreach* permite recorrer de forma secuencial todos los elementos de cualquier colección de datos de *.NET Framework*. La colección debe implementar la interfaz *IEnumerable<T>*.
- ✓ Tipos por valor que aceptan valores *NULL*: Habilita al programador definir tipos por valor que contengan un valor indefinido así como ocurre con los tipos por referencia.
- ✓ Ensamblados de confianza: Se puede acceder a tipos no públicos que se encuentran ubicados en otros ensamblados.
- ✓ Búferes de tamaño fijo: Utilidad que se usa para trabajar en un ambiente de código no seguro, se pueden declarar estructuras de tamaño fijo o arreglos usando la palabra reservada *stackalloc.*
- ✓ Control de alertas en el código: La directiva de pre-procesamiento de advertencia *#pragma* permite habilitar y deshabilitar ciertas advertencias lanzadas por el compilador de *C#*.
- ✓ Manipulación de delegados: Se proporciona una sintaxis simplificada para declarar delegados y utilizarlos en la aplicación.
- ✓ Accesibilidad en propiedades: Permite definir niveles de acceso diferentes para los descriptores de acceso *get* y *set* en propiedades.

1.2.2 ADO.NET 2.0

El lanzamiento de *.NET Framework 2.0*, trajo consigo una nueva versión de *ADO.NET*. En *ADO.NET 2.0* el diseño básico del modelo de objetos es el mismo que en la primera versión, sólo se agregaron varias características nuevas para hacer que las tareas comunes sean más fáciles de realizar. Una de las cosas nuevas que trajo *ADO.NET 2.0* fue un mejor diseño y funcionamiento del agrupamiento de conexiones[5] (*Connection Pooling*) y el manejo de la serialización *XML*.

Multiple Active Result Set o *MARS* (Varios conjuntos de resultados activos) es una característica que habilita la ejecución de varios lotes en una sola conexión hacia una base de datos. En la versión *1.0* y *1.1* de *ADO.NET*, únicamente era posible ejecutar un lote a la vez en una sola conexión. Una ejecución de múltiples lotes con *MARS* no significa que la ejecución de operaciones se realice de forma simultánea. Es otras palabras, cada lote es completamente independiente, esto es como si se usaran conexiones separadas consiguiendo un ahorro de recursos del sistema manejador de base de datos.

[5] La agrupación de conexiones reduce el número de veces que es necesario abrir nuevas conexiones. El concentrador mantiene la propiedad de la conexión física. Para administrar las conexiones, mantiene un conjunto de conexiones activas para cada configuración de conexión dada. Cada vez que un usuario llama a Open en una conexión, el agrupador comprueba si hay una conexión disponible en el grupo. Si hay disponible una conexión agrupada, la devuelve a la persona que llama en lugar de abrir una nueva. Cuando la aplicación llama a Close en la conexión, el agrupador la devuelve al conjunto agrupado de conexiones activas en lugar de cerrarla. Una vez que la conexión vuelve al grupo, ya está preparada para volverse a utilizar en la siguiente llamada a *Open*.

Se incluye una nueva *API* para examinar el esquema de una base de datos. También se tiene la opción de escribir código de acceso a datos independiente del proveedor de datos, así se tiene la posibilidad de ofrecer aplicaciones que se pueden conectar a cualquier origen de datos.

1.2.3 ASP.NET 2.0

.NET Framework 2.0 incorpora mejoras en muchas áreas de versiones previas de *ASP.NET*. Estas mejoras en lo referente a sitios web consisten en crear sitios con gran calidad de un modo más sencillo y rápido. Un sitio web a partir de esta versión, puede precompilarse para que genere código ejecutable en base a archivos de código fuente (estos pueden ser códigos fuente de *C#*, *C++*, etc. como de código *XML* usado en páginas **.aspx*). Una vez que se creó un sitio web, éste puede implementarse en un servidor de producción obteniendo un gran resultado en rendimiento. *ASP.NET 2.0* incluye diferentes clases y herramientas que ayudan a los programadores, administradores de servidor y hosts a administrar el sitio web. *ASP.NET* es compatible con una gran variedad de exploradores de Internet y dispositivos móviles. De forma predeterminada los controles propios de *ASP.NET 2.0* son completamente compatibles con los estándares de *XHTML versión 1.1*. Así, es posible especificar diferentes valores para las distintas propiedades en un mismo control para ejecutarse en diversos exploradores de Internet.

Para la creación de páginas web, la incorporación de muchos nuevos controles que permiten el agregado de funcionalidades para tareas comunes en páginas web dinámicas. Los controles que sirven para visualizar datos (como el *GridView*) permiten mostrar y modificar datos en páginas web de *ASP.NET 2.0* sin requerir que el programador escriba demasiado código fuente. La optimización del modelo de código de *ASP.NET 2.0* simplifica el desarrollo de páginas usando *ASP.NET 2.0*. El almacenamiento en memoria caché proporciona distintas maneras de guardar las páginas. A diferencia de otros *frameworks* para creación de páginas web dinámicas (como los proporcionados por *Java*), en *ASP.NET 2.0* es fácil personalizar sitios y páginas web de formas muy diversas. *ASP.NET 2.0* permite realizar seguimientos automáticos a usuarios que acceden al sitio.

1.2.4 TIPOS GENÉRICOS

Los tipos genéricos son clases, estructuras, interfaces y métodos que tienen parámetros de tipo (*T*) para uno o varios de los tipos que almacenan o utilizan. Una clase que define a una colección genérica puede usar un parámetro de tipo de acuerdo al tipo de objeto que almacena; los parámetros de tipo que almacenan aparecen como los tipos para las propiedades (*properties*) y para los tipos de parámetros de sus métodos. Un método genérico puede hacer uso de su parámetro de tipo para procesarlo o devolver ese mismo tipo al llamador. Los códigos siguientes muestran una definición de clase genérica simple.

Lenguaje de programación	Código fuente

C# 3.0	`public class Generico<T> {`
	`  public T Propiedad {get; set;}`
	`}`

Visual Basic 10	`Public Class Generico(Of T)`
	`  Public Property Propiedad() As T`
	`End Class`

Al crear una instancia de una clase genérica, se especifican los tipos reales que deben sustituirse para los parámetros de tipo. De esta forma, se establece una clase genérica, la cual es denominada

clase genérica construida, los tipos elegidos son sustituidos en todas las apariciones de los parámetros de tipo. El resultado de este proceso es una clase que garantiza una seguridad de tipos, como se ilustra en el código siguiente:

Lenguaje de programación	Código fuente
C# 3.0	`Generico<String> claseGenerica = new Generico<String>();` `claseGenerica.Propiedad = "Fanny";`
Visual Basic 10	`Dim claseGenerica As New Generico(Of String)` `claseGenerica.Propiedad = "Fanny"`

Entonces, el tipo (*T*) de la propiedad de la clase genérica es una clase del tipo *String.*

Las definiciones que se muestran a continuación se usan para entender los genéricos usados en *.NET Framework* (se ejemplifica usando *C#*):

- ✓ La declaración de una interfaz, clase o estructura es una definición de tipo genérico que utiliza parámetros de tipo los cuales puede almacenar o utilizar. Por ejemplo, la clase *Dictionary<TKey, TValue>* de *System.Collections.Generic* definida en *C#* puede contener dos tipos: claves (para el parámetro de tipo *TKey*) y valores (para el parámetro de tipo *TValue*). Dado que es una clase se pueden crear instancias de la misma.
- ✓ El término de tipo genérico abarca tanto los tipos construidos (*objetos*) como las definiciones de tipos genéricos (*clases*). Un objeto de tipo genérico, es el resultado de especificar los tipos de los parámetros de tipo genérico de una clase genérica. Un argumento de tipo genérico puede ser cualquier tipo soportado por *.NET Framework*.
- ✓ Los parámetros de tipo genérico generalmente son asignados en tiempo de compilación.
- ✓ Un método genérico es un método que alberga dos listas de parámetros: una de éstas contiene parámetros de tipo genérico y la segunda contiene parámetros formales. Los primeros pueden aparecer en el tipo de valor devuelto o incluso en los tipos para los parámetros formales.
- ✓ Existen restricciones, las cuales son límites que se colocan en parámetros de tipo genérico. Por ejemplo, se podría limitar un parámetro de tipo a tipos que implementen la interfaz genérica *IComparer<T>* en *C#* para garantizar el ordenamiento de esas instancias. Es posible también restringir los parámetros de tipo a tipos que tienen una clase base específica, que tengan un constructor predeterminado o que sean tipos por referencia o tipos por valor. Los usuarios de los tipos genéricos no podrán reemplazar los parámetros de tipo que no respeten estas restricciones.

1.2.4.1 COLECCIONES GENÉRICAS EN .NET FRAMEWORK.

La biblioteca de clases base (*BCL*) de *.NET Framework* proporciona varias clases de colección genéricas en los espacios de nombres *System.Collections.Generic* y *System.Collections.ObjectModel*.

Muchos de los tipos genéricos de colecciones son análogos directos de tipos no genéricos. Por ejemplo, la clase de colección *System.Collections.Generic.List<T>* es la versión genérica de *System.Collections.ArrayList*.
Para observar las diferencias entre ente una colección no genérica y genérica veamos un ejemplo. Supongamos que queremos agregar los datos de una persona (edad –Int32–, nombre –String–, fecha de nacimiento –DateTime– y lugar de trabajo –String–) usando una colección no genérica (*ArrayList*) y una genérica (*List<T>*) en *C# 3.0*.

Usando la clase *ArrayList* de *System.Collections* el procedimiento sería de la siguiente manera:

TIPO	CÓDIGO C#

Tipo no genérico (ArrayList)	`ArrayList arrayList = new ArrayList();`
	`arrayList.Add(new Object[] {`
	`  30,`
	`  "Fanny",`
	`  Convert.ToDateTime("05-04-1979"),`
	`  "La Isla Azul"`
	`});`

Ahora usando la clase *List<T>* de *System.Collections.Generic* el resultado sería:

TIPO	CÓDIGO C# 3.0

Tipo genérico (List<T>)	`List<Persona> list = new List<Persona>();`
	`list.Add(new Persona {`
	`  Edad = 30,`
	`  Nombre = "Fanny",`
	`  FechaNacimiento = Convert.ToDateTime("05-04-1979"),`
	`  Trabajo = "La Isla Azul"`
	`});`

Donde previamente para el segundo caso debimos haber declarado una clase llamada *Persona* de la siguiente forma:

CLASE PERSONA EN C# 3.0

`public class Persona {`
`  public int Edad { get; set; }`
`  public String Nombre { get; set; }`
`  public DateTime FechaNacimiento { get; set; }`
`  public String Trabajo { get; set; }`
`}`

Las diferencias entre la colección no genérica y genérica es la siguiente:
Puesto que en la primera clase de colección (*ArrayList*) insertamos los datos de la persona de forma directa (sin seguridad de tipos), esto no nos asegura que al momento que se desee utilizar los datos el resultado sea el esperado, es decir, al insertar la edad se colocó correctamente el número entero *30* pero si se hubiera introducido "*treinta*" no hubiera existido ningún problema en compilación pero sí en ejecución si se necesita trabajar con números enteros forzosamente. En cambio en la segunda clase de colección (*List<T>*) al crear previamente una clase "*Persona*" con la propiedad autoimplementada "*Edad*" del tipo entero (*Int32*) se logra la seguridad de tipos en compilación y en ejecución porque se obliga al programador a insertar un número entero.

1.2.4.2 VENTAJAS DE LOS TIPOS GENÉRICOS.

La principal ventaja al utilizar tipos genéricos es que no hay ninguna necesidad de escribir código fuente adicional para comprobar que el tipo que se introdujo es correcto, porque esto se hace en tiempo de compilación. Además de lo anterior se reduce la necesidad de convertir los tipos (como se hace en la clase *ArrayList*) y la posibilidad de que se produzcan excepciones dentro del programa en tiempo de ejecución. Por ejemplo, se puede crear una lista genérica del tipo *String* de la siguiente manera:

```
List<String> lista = new List<String>();
```

Con lo anterior se asegura que todos los datos que se introduzcan a través de métodos como *Add* sean del tipo *String,* cualquier otro tipo arrojaría un error en tiempo de compilación.

1.2.4.3 VENTAJAS DE LOS TIPOS GENÉRICOS DE .NET FRAMEWORK SOBRE JAVA.

Tanto *.NET Framework* como *Java* ofrecen tipos genéricos, el trato que ambas plataformas da internamente es diferente. En *.NET Framework* el compilador del *C#* valida la seguridad de tipos y la máquina virtual (*CLR*) trata estos tipos tal y como se especificaron en compilación. En *Java* el compilador del lenguaje al igual que en *C#* también valida la seguridad de tipos pero la máquina virtual (*JVM*) no los trata como el tipo especificado sino como tipos *Object*. Esto implica un rendimiento menor en el trato de tipos genéricos por parte de *Java* debido al *boxing* y *unboxing* que se debe hacer en tiempo de ejecución para los tipos. En la siguiente tabla se muestra el comportamiento de los tipos genéricos en ambas plataformas de programación de software tanto en tiempo de compilación como en tiempo de ejecución:

P. Software	Código fuente	Comportamiento en compilación	Comportamiento en ejecución
≥ .NET Framework 2.0 (≥ C# 2.0)	`List<String> lista = new List<String>();`	`List<String>`	`List<String>`
≥ Java 5.0	`ArrayList<String> lista = new ArrayList<String>();`	`ArrayList<String>`	`ArrayList<Object>`

Tabla 1.3 Comparación entre los tipos genéricos de .NET Framework y Java

Es importante señalar que la ejecución de los tipos genéricos en *Java* se realiza así (*String* en compilación y *Object* en ejecución) porque la *Java Virtual Machine* no se modificó para que aceptara los tipos genéricos como tal, mientras que el *Common Language Runtime* de *.NET Framework* sí se modificó para optimizar el rendimiento de las aplicaciones.

1.3 .NET FRAMEWORK 3.0

Esta versión de *.NET Framework* se incorpora a la versión *2.0* de *.NET Framework* y combina este marco de trabajo con nuevas tecnologías para crear:

- ✓ Experiencias de usuario visualmente atractivas.
- ✓ Comunicación interoperable confiable.
- ✓ Modelos de procesos de negocio.
- ✓ Administración de identidades de usuario.

En la siguiente imagen se pueden ver los componentes de *.NET Framework 3.0* y como se incorporan sobre *.NET Framework 2.0*:

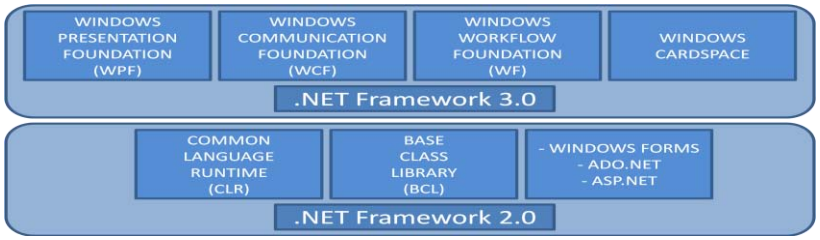


Figura1.3 La pila de .NET Framework (2.0 y 3.0)

Enseguida se explica de forma breve cada uno de los 4 componentes principales de *.NET Framework 3.0*:

1.3.1 WINDOWS PRESENTATION FOUNDATION

Windows Presentation Foundation (*WPF*) es un sistema que permite crear aplicaciones que proporcionan una gran apariencia en la capa visual. El núcleo de *Windows Presentation Foundation* aprovecha al máximo el hardware de gráficos existente actual. *Windows Presentation Foundation* contiene un completo conjunto de características de programación, entre las que se encuentran el *Lenguaje de marcado de aplicaciones extensible (XAML*), controles, gráficos en *2D* y *3D*, animaciones, estilos, plantillas, multimedia, texto, entre otras. *Windows Presentation Foundation* es parte de *.NET Framework*, por lo tanto permite crear aplicaciones que hacen uso de otras clases de la *biblioteca de clases base (BCL)*.

1.3.2 WINDOWS COMMUNICATION FOUNDATION

Windows Communication Foundation (*WCF*) ofrece un ambiente que engloba la construcción de aplicaciones basadas en computación distribuida, interoperables (incluso con tecnologías diferentes a *.NET Framework* como *Web Services Interoperability Technologies* de *Java*) a través de la creación de servicios web. *Windows Communication Foundation* reduce el tiempo y complejidad que conlleva el desarrollo de aplicaciones conectadas a través de un modelo de programación orientada a servicios. *Windows Communication Foundation* admite una gran cantidad de protocolos para realizar un intercambio de datos de forma segura y fiable, así como un abanico de opciones de codificación (como *XML*) y transporte (como *WsHttpBinding*).

1.3.3 WINDOWS WORKFLOW FOUNDATION

Windows Workflow Foundation (*WF*) es un modelo de programación y herramientas que generan de forma eficaz programas que funcionan bajo un flujo de trabajo. *Windows Workflow Foundation* permite a los programadores crear flujos de trabajo sistemáticos o flujos de trabajo humanos en sus programas. Se puede utilizar para resolver escenarios simples o escenarios complejos como los que se producen en grandes empresas, tales como el control de inventarios. *Windows Workflow Foundation* proporciona a los programadores una experiencia unificada de desarrollo de aplicaciones con otras tecnologías de *.NET Framework 3.0*, como *Windows Communication Foundation (WCF)* y *Windows Presentation Foundation (WPF)*.

1.3.4 WINDOWS CARDSPACE

Actualmente los usuarios de Internet utilizan un gran número de cuentas (correo electrónico, compras en línea, foros, etc.) y contraseñas. Muchas veces esto desencadena prácticas poco seguras, como la utilización del mismo nombre de usuario y contraseña en muchos sitios de Internet. Esto provoca robo de identidad, fraude y problemas de confidencialidad. *Windows CardSpace* es un sistema de identidad que permite a un usuario elegir una identidad de entre un conjunto de identidades que le pertenecen y utilizarla en ambientes donde son aceptadas.

1.4 .NET FRAMEWORK 3.5

.NET Framework 3.5 es la versión definitiva más actual de *.NET Framework* al momento de escribir estas líneas. Esta versión agrega múltiples mejoras a la máquina virtual *(CLR)* e incorpora diversas funcionalidades que de manera conjunta hacen posible trabajar con *Language Integrated Query (LINQ)*. A continuación se muestra una imagen que ejemplifica que tal y como ocurrió *con .NET*

Framework 3.0, esta versión también se integra con *.NET Framework 2.0*. Igualmente se muestra una lista de sus principales características:

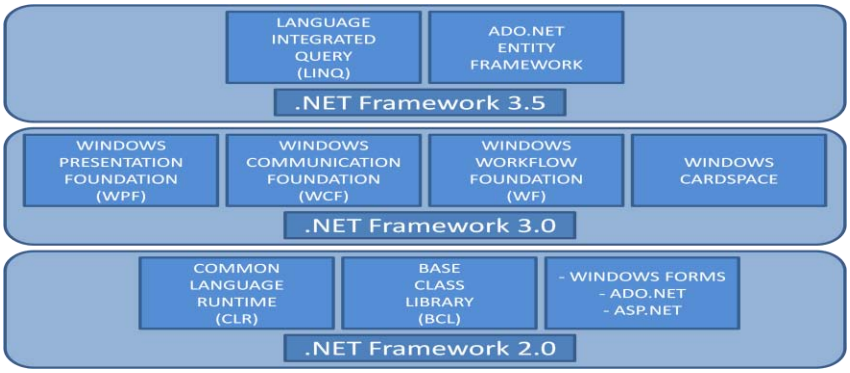


Figura 1.4 La pila de .NET Framework (2.0, 3.0 y 3.5)

- ✓ Conexión de red punto a punto: Esta es una tecnología de red sin servidor centralizado que habilita a varios dispositivos conectados a una red compartir recursos y establecer comunicación directa entre sí. El espacio de nombres *System.Net.PeerToPeer* suministra un repertorio de clases compatibles con el *Protocolo de resolución de nombres de mismo nivel (PNRP)* con el que es factible detectar nodos de un mismo nivel a través de objetos *PeerName* registrados en una red punto a punto. *PNRP* es capaz de resolver nombres de un mismo nivel en direcciones *IP* de tipo *IPv6* o *IPv4*.
- ✓ Recolección de basura: La clase *GCSettings* posee una nueva propiedad llamada *Latency-Mode* cuya funcionalidad es la de determinar el momento adecuado en el que el recolector de basura entra en la aplicación.
- ✓ Rendimiento del socket: La clase *System.Net.Sockets.Socket* se optimizó para que aplicaciones de software puedan utilizar operaciones de *Entrada/Salida* de forma asincrónica con el objetivo de obtener el mayor rendimiento posible. También se agregaron un conjunto de clases nuevas que forman parte de una serie de mejoras para el espacio de nombres *System.Net.Sockets*. Estas clases brindan un procesamiento asincrónico alterno que puede ser utilizado en aplicaciones de alto rendimiento. Estas optimizaciones se crearon principalmente para aquellas aplicaciones de servidores de red que requieren un alto rendimiento.
- ✓ Canalizaciones: Proporcionan una comunicación entre los procesos que se ejecutan en la misma computadora o en cualquier otra computadora de una red.
- ✓ Integración de *WCF* y *ASP.NET AJAX*: La integración de estas dos tecnologías proporciona un modelo de programación para la creación de aplicaciones web que pueden utilizar servicios de *Windows Communication Foundation*. En las aplicaciones *ricas de Internet (RIA)* que hacen uso de *AJAX*, el cliente (un explorador de Internet) realiza un Bloqueo de lectura y escritura mejorado: La clase System.Threading.*ReaderWriterLockSlim* proporciona un rendimiento mayor que *System.Threading .ReaderWriterLock* y es similar a la instrucción *lock* (útil para realizar exclusión mutua) de *C#*.
- ✓ Árboles de expresión: Proporcionan un mecanismo que permite representar a un código de nivel de lenguaje como si fueran datos. El espacio de nombres *System.Linq.Expressions* contiene las clases que permiten la creación de árboles de expresiones. Estas clases se pueden utilizar para representar diferentes operaciones a expresiones de código, por ejemplo una expresión para una comparación de igualdad.

- ✓ Mejoramiento de las zonas horarias: Las estructuras *DateTimeOffset* y *TimeZoneInfo*, optiman la compatibilidad ente zonas horarias y facilitan la creación de aplicaciones de software que trabajan con diferentes fechas, horas y zonas horarias.
- ✓ Solicitudes asincrónicas. La integración de *AJAX* en *ASP.NET 3.5* proporciona un modo sencillo para obtener acceso a pequeñas solicitudes a un servidor mediante el uso de *JavaScript* y *XML* en el explorador de Internet del lado del cliente.
- ✓ Windows Presentation Foundation: Contiene modificaciones y mejoras en varias áreas de la interfaz gráfica de usuario como controles, elementos de la interfaz de usuario *3D,* enlaces de datos, modelos de la aplicación, etc.

1.4.1 C# 3.0

La versión *3.0* de *C#* presenta nuevas características que mejoran y extienden a la versión *2.0* del lenguaje de programación. Estas características son funcionales por separado en diferentes maneras y en conjunto permiten el uso de *Language-Integrated Query (LINQ)*.

En seguida se muestran algunas de las características más importantes de *C# 3.0*:

- ✓ Inicializadores de objeto: Esta característica permite inicializar miembros de una clase en el momento de construir un objeto. La sintaxis es como la siguiente: *NombreClase clase = new NombreClase { Propiedad1 = valor };*
- ✓ Inicializadores de colección: Permite inicializar colecciones con una lista de inicialización al construir un objeto del tipo colección, esto puede sustituir llamadas concretas al método *Add* o algún otro.
- ✓ Tipos anónimos: La palabra reservada *var* indica al compilador de *C#* que debe inferir el tipo del elemento situado a la derecha de la palabra.
- ✓ Expresiones lambda: Permite insertar expresiones con parámetros de entrada que pueden enlazarse a delegados o árboles de expresión.
- ✓ Palabras reservadas de consulta: Son palabras que especifican cláusulas en una expresión de consulta: *from*, *where* (opcional), *orderby* (opcional), *join* (opcional), *select* o *group* e *into* (opcional)
- ✓ Métodos de extensión: Permite extender clases existentes con métodos estáticos que puedan ser invocados a través de la sintaxis de métodos de instancia.
- ✓ Métodos parciales: Permite declarar un método como parcial, el funcionamiento es análogo a las clases parciales o a un método abstracto principalmente.
- ✓ Propiedades autoimplementadas: Permite al programador declarar propiedades utilizando una sintaxis reducida. La sintaxis de estas propiedades es como sigue: *public T Nombre { get; [private] set; }*

1.4.2 LANGUAGE INTEGRATED QUERY (LINQ)

Las Consultas Integradas en el Lenguaje (*Language-Integrated Query, LINQ*) es una característica aparecida en *.NET Framework 3.5*. *Language-Integrated Query* añade y permite realizar consultas usando la sintaxis propia de lenguajes de programación enfocados a *.NET Framework.* Esta tecnología es compatible con cualquier colección de datos. *.NET Framework 3.5* contiene los ensamblados que habilitan el uso de *Language-Integrated Query* sobre colecciones de *.NET Framework*, bases de datos (*SQL Server* de forma nativa), *ADO.NET* y documentos con formato *XML*.

Los ensamblados de *LINQ* que pertenecen a *.NET Framework 3.5* son los siguientes:

- ✓ El ensamblado *System.Core.dll*: este ensamblado contiene el espacio de nombres *System.Linq*; aquí se encuentran un conjunto de métodos de extensión los cuales sirven para ejecutar procedimientos comúnes y consultas estándar sobre cualquier colección de datos que implementen la interfaz *System.Collections.IEnumerable* o *System.Collections.Generic.IEnumerable<T>*.
- ✓ El ensamblado *System.Xml.Linq.dll*: este ensamblado contiene el espacio de nombres *System.Xml.Linq*; a través de sus miembros se puede manipular la estructura de cualquier documento creado con *XML*. A esta herramienta se le denomina *LINQ para XML.*
- ✓ El ensamblado *System.Data.Linq.dll*: en este ensamblado se encuentran los espacios de nombres *System.Data.Linq.Mapping* y *System.Data.Linq;* el primero contiene clases para poder hacer un mapeo de una base de datos relacional (El *Diseñador Relacional de Objetos* de *Visual Studio 2008* hace uso de los miembros de este espacio de nombres para generar automáticamente el mapeo de una base de datos) y el segundo contiene clases para manipular los datos de una base de datos partiendo de su mapeo. A esta herramienta de acceso a bases de datos de le denomina *LINQ para SQL*.

1.4.2.1 LINQ PARA SQL (LINQ TO SQL)

LINQ para SQL permite manipular los datos contenidos en bases de datos relacionales a través de objetos de *Common Language Runtime*. La utilización eficaz y eficiente de *LINQ para SQL* depende en gran medida de que el programador además de conocer un lenguaje de programación enfocado a *.NET Framework*, también posea un conocimiento sólido en el manejo de bases de datos relacionales y en el uso de *Structured Query Language (SQL)*.

Al hacer uso de *LINQ para SQL*, la estructura de una base de datos es análoga a un modelo de objetos dependiente del lenguaje de programación que se esté utilizando. Cuando se tiene un programa que accede a una base de datos contenida en *SQL Server*, y se ejecuta una consulta integrada en el lenguaje, el *Common Language Runtime* de *.NET Framework* la convierte a *Transact-SQL* y la envía al servidor de bases de datos. Cuando el servidor procesa las instrucciones y devuelve los resultados al programa, el *Common Language Runtime* convierte el resultado en objetos que puedan ser entendidos por *.NET Framework*.

Ejemplo del uso de LINQ para SQL.

En el siguiente ejemplo que mostrará el uso de *LINQ para SQL* (*LINQ to SQL*), se observarán las características más básicas de esta tecnología. La razón de este ejemplo es dar una pequeña noción de esta tecnología para entender de una mejor manera los capítulos 4 y 5 de esta tesis.

Se tiene en *SQL Server* una base de datos llamada *LinqEjemplo* con 2 tablas (*Empleado* y *Departamento*) las cuales contienen los siguientes atributos:

EMPLEADO		
NOMBRE	**TIPO**	**¿NULO?**
IdEmpleado (PK)	int identity(1,1)	NOT NULL
IdDepartamento (FK)	int	NOT NULL
Nombre	nvarchar(50)	NOT NULL
ApellidoPaterno	nvarchar(25)	NOT NULL
ApellidoMaterno	nvarchar(25)	NULL
FechaNacimiento	datetime	NOT NULL

DEPARTAMENTO		
NOMBRE	**TIPO**	**¿NULO?**
IdDepartamento (PK)	int identity(1,1)	NOT NULL
Nombre	nvarchar(50)	NOT NULL

Tabla 1.4 Atributos de las tablas Empleado y Departamento

A su vez estas tablas tienen una relación uno a muchos a través del campo *IdDepartamento*. El contenido de las tablas en la base de datos es como se muestra a continuación:

Contenido de la tabla *Empleado*

IdEmpleado	IdDepartamento	Nombre	ApellidoPaterno	ApellidoMaterno	FechaNacimiento
1	1	Ruben	Plaza	Higuaín	31/12/1974 12:...
2	3	Raúl	González	Bautista	25/08/1977 12:...
3	1	Maite	Pérez	Valdéz	09/03/1983 12:...
4	3	Fernando	Blanco	Hernández	13/11/1980 12:...
5	4	Alejandro	Zárate	Álvarez	30/01/1968 12:...
6	2	Hugo	Reyes	López	02/09/1966 12:...
7	3	Emilie	de Ravin	*NULL*	16/05/1976 12:...
8	4	Gonzalo	Negredo	Güiza	21/10/1980 12:...
9	2	Sergio	Salgado	Ramos	24/06/1965 12:...
10	3	Antonio	Gómez	Guerrero	29/03/1979 12:...

Contenido de la tabla *Departamento*

IdDepartamento	Nombre
1	Gerencia
2	Finanzas
3	Informática
4	Recursos humanos

Tabla 1.5 Contenido de las tablas Empleado y Departamento

Usando *Visual Studio 2008* creamos una nueva solución de aplicación de consola y agregamos un nuevo "*LINQ to SQL classes mapped to relational objects*." a la solución con el nombre "*LinqEjemplo.dbml*". Esto agregará automáticamente referencias a los ensamblados *System.Data.Linq* y *System.Xml.Linq*.

Haciendo uso del *Server Explorer* de *Visual Studio 2008*, conectamos con la instancia de *SQL Server* y seleccionamos la base de datos "*LinqEjemplo*". Una vez conectado, el *Server Explorer* nos mostrará todo el contenido de la base de datos. En el apartado de tablas seleccionamos las tablas *Departamento* y *Empleado* y las arrastramos al "*Object Relational Designer*"

Una vez que el diseñador terminó de mapear las tablas de la base de datos (creando clases equivalentes a las tablas de la base de datos) y se muestra una imagen como la siguiente, ya podremos usar LINQ para manipular los datos de las tablas de la base de datos. La clase generada por el diseñador relacional de objetos es una clase parcial llamada *LinqEjemploDataContext* que hereda de la clase *DataContext*[6] del espacio de nombres *System.Data.Linq*, a partir de la cual podremos acceder a los elementos que se escojan de la base de datos.

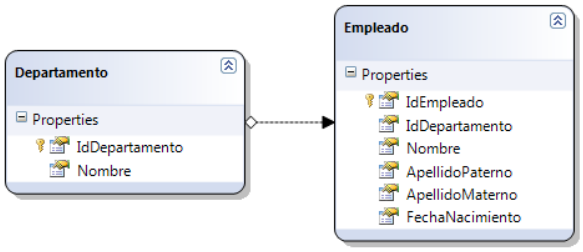


Figura 1.5 Diagrama de clases con el mapeo de las tablas de la base de datos

A continuación se muestra el equivalente en *LINQ* usando *C#* de las operaciones más comunes en *SQL* para *SQL Server* usando el mapeo de la base de datos creado anteriormente.

[6] La clase *DataContext* es el origen de todas las entidades asignadas en una conexión de base de datos. Realiza un seguimiento de los cambios realizados en todas las entidades recuperadas y mantiene una "memoria caché de identidad" que garantiza que las entidades que se recuperan más de una vez se representan utilizando la misma instancia de objeto. En general, una instancia de *DataContext* está diseñada para que dure una "unidad de trabajo", sea cual sea la definición de ese término en la aplicación. *DataContext* es un objeto ligero cuya creación no es costosa. Una aplicación *LINQ to SQL* típica crea instancias de *DataContext* en el ámbito de métodos o como miembro de clases de duración corta que representan un conjunto lógico de operaciones de base de datos relacionadas.

1.4.2.1.1 SELECCIÓN (SELECT)

SELECCIONAR TODOS LOS CAMPOS DE LA TABLA EMPLEADO.

SQL

```
SELECT * FROM Empleado;
```

ó

```
SELECT [t0].[IdEmpleado], [t0].[IdDepartamento], [t0].[Nombre],
[t0].[ApellidoPaterno], [t0].[ApellidoMaterno], [t0].[FechaNacimiento]
FROM [dbo].[Empleado] AS [t0];
```

LINQ to SQL

```
LinqEjemploDataContext db = new LinqEjemploDataContext();
var match = from empleado in db.Empleados
            select empleado;
```

SELECCIONAR LOS NOMBRES DE EMPLEADOS CUYA PRIMERA LETRA ESTÁ COMPRENDIDA ENTRE A Y M. ORDENARLOS ALFABÉTICAMENTE EN ORDEN ASCENDENTE.

SQL

```
SELECT Nombre FROM Empleado WHERE Nombre BETWEEN 'A%' AND 'N%'
ORDER BY Nombre ASC;
```

LINQ to SQL

```
LinqEjemploDataContext db = new LinqEjemploDataContext();
var match = from empleado in db.Empleados
            where empleado.Nombre.CompareTo("A") >= 0 &&
                  empleado.Nombre.CompareTo("N") < 0
            orderby empleado.Nombre ascending
            select empleado;
```

SELECCIONAR EL ID, NOMBRE COMPLETO, FECHA DE NACIMIENTO Y DEPARTAMENTO DE LOS EMPLEADOS USANDO INNER JOIN.

SQL

```
SELECT [t0].[IdEmpleado], [t1].[Nombre] AS [NombreDepartamento], [t0].[Nombre] AS
 [NombreEmpleado], [t0].[ApellidoPaterno], [t0].[ApellidoMaterno],
 [t0].[FechaNacimiento]
FROM [dbo].[Empleado] AS [t0]
 INNER JOIN [dbo].[Departamento] AS [t1]
 ON [t0].[IdDepartamento] = ([t1].[IdDepartamento]);
```

LINQ to SQL

```
LinqEjemploDataContext db = new LinqEjemploDataContext();
var match = from empleado in db.Empleados
            join departamento in db.Departamentos
            on empleado.IdDepartamento equals departamento.IdDepartamento
            select new {
                IdEmpleado = empleado.IdEmpleado,
                NombreDepartamento = departamento.Nombre,
                NombreEmpleado = empleado.Nombre,
                ApellidoPaterno = empleado.ApellidoPaterno,
                ApellidoMaterno = empleado.ApellidoMaterno,
                FechaNacimiento = empleado.FechaNacimiento,
            };
```

1.4.2.1.2 INSERTAR (INSERT)

INSERTAR UN NUEVO EMPLEADO A LA BASE DE DATOS.

SQL

```
INSERT INTO Empleado(IdDepartamento, Nombre, ApellidoPaterno, ApellidoMaterno,
FechaNacimiento)
VALUES (1, 'Fanny', 'Torres', null, '05-04-1979');
```

LINQ to SQL

```
LinqEjemploDataContext db = new LinqEjemploDataContext();
db.Empleados.InsertOnSubmit(new Empleado
{
    IdDepartamento = 1,
    Nombre = "Fanny",
    ApellidoPaterno = "Torres",
    ApellidoMaterno = null,
    FechaNacimiento = Convert.ToDateTime("05-04-1979")
});
db.SubmitChanges();
```

1.4.2.1.3 ACTUALIZAR (UPDATE)

ACTUALIZAR LA FECHA DE NACIMIENTO DE UN EMPLEADO EXISTENTE EN LA BASE DE DATOS.

SQL

```
UPDATE Empleado SET FechaNacimiento = '27-12-1977'
WHERE IdEmpleado = 7;
```

LINQ to SQL

```
LinqEjemploDataContext db = new LinqEjemploDataContext();
var match = from empleado in db.Empleados
            where empleado.IdEmpleado == 7
            select empleado;
foreach (Empleado empleado in match)
{
    empleado.FechaNacimiento = Convert.ToDateTime("27-12-1977");
}
db.SubmitChanges();
```

1.4.2.1.4 ELIMINAR (DELETE)

ELIMINAR UN EMPLEADO DE LA BASE DE DATOS DE ACUERDO A SU NÚMERO DE IDENTIFICACIÓN.

SQL

```
DELETE FROM Empleado WHERE IdEmpleado = 10;
```

LINQ to SQL

```
LinqEjemploDataContext db = new LinqEjemploDataContext();
var match = from empleado in db.Empleados
            where empleado.IdEmpleado == 10
            select empleado;


foreach (Empleado empleado in match)
{
    db.Empleados.DeleteOnSubmit(empleado);
}
db.SubmitChanges();
```

1.4.2.1.5 PROCEDIMIENTO ALMACENADO (STORED PROCEDURE)

OBTENER TODOS LOS EMPLEADOS Y EL NÚMERO DE EMPLEADOS.

SQL

CREACIÓN:

```
CREATE PROCEDURE MiProcedimientoAlmacenado
AS
BEGIN
       SELECT * FROM Empleado;
       SELECT COUNT(IdEmpleado) FROM Empleado;
END
```

EJECUCIÓN:

```
EXEC MiProcedimientoAlmacenado;
```

LINQ to SQL

Dado que el diseñador relacional de objetos solo mapea de forma adecuada procedimientos almacenados sencillos, se debe mapear el procedimiento almacenado de forma manual. Dado que todos los elementos de la base de datos están en la clase parcial *LinqEjemploDataContext* usaremos esta misma clase parcial para mapear el procedimiento almacenado de la siguiente manera:

MAPEO DEL PROCEDIMIENTO ALMACENADO:

```
public partial class LinqEjemploDataContext : DataContext
{
    [Function(Name = "dbo.MiProcedimientoAlmacenado")]
    [ResultType(typeof(Empleado))]
    [ResultType(typeof(Int32))]
    public IMultipleResults GetProcedimientoAlmacenado()
    {
        IExecuteResult result = this.ExecuteMethodCall(this,
            (MethodInfo)MethodInfo.GetCurrentMethod());
        return (IMultipleResults)result.ReturnValue;
    }
}
```

EJECUCIÓN DEL PROCEDIMIENTO ALMACENADO:

```
using (LinqEjemploDataContext db = new LinqEjemploDataContext())
{
    IMultipleResults multipleResults = db.GetProcedimiento();
    var queryEmpleados = multipleResults.GetResult<Empleado>();
    foreach (Empleado empleado in queryEmpleados)
    {
        Console.Write("Id Empleado: " + empleado.IdEmpleado + "\t");
        Console.WriteLine("Nombre de empleado: " + empleado.Nombre);
    }
    int queryCount = multipleResults.GetResult<Int32>().Single();
    Console.WriteLine("Cantidad de empleados: " + queryCount.ToString());
}
```

Algo importante de señalar es que, aunque los ensamblados nativos de *.NET Framework 3.5* únicamente proveen *LINQ to SQL* para el sistema manejador de bases de datos *SQL Server*, existen otros proveedores que ofrecen la capacidad de utilizar *LINQ to SQL* en otros manejadores de bases de datos. Un ejemplo de proveedor de *LINQ to SQL* es *DbLinq (http://groups.google.com/group/dblinq)*, un software *Open Source* que permite trabajar de forma eficiente con gran variedad de sistemas manejadores de bases de datos como: *Oracle*, *MySql*, *PostgreSql*, *SqlLite*, *Ingres* e incluso el propio *SQL Server.*

1.4.3 ASP.NET 3.5

La versión *3.5* de *.NET Framework* agrega características nuevas a *ASP.NET 2.0*. Una de las más importantes mejoras se encuentra enfocada en la compatibilidad que se obtiene en el desarrollo de sitios web que utilizan *Asynchronous JavaScript and XML (AJAX)*. *ASP.NET 3.5* agrega compatibilidad con el desarrollo de *AJAX* a través de un conjunto de nuevos controles y nuevas Interfaces de Programación de Aplicaciones (*API*). Se puede habilitar una página creada en *ASP.NET 2.0* para que ejecute *AJAX* agregando un control *ScriptManager* y un control *UpdatePanel*, de modo que la página web pueda actualizarse sin que sea necesario realizar una actualización completa de la página.

ASP.NET 3.5 agrega una biblioteca de cliente denominada *Microsoft AJAX Library* para el desarrollo de aplicaciones ricas de Internet (se pueden actualizar sin necesidad de realizar viajes de ida y vuelta al servidor). *Microsoft AJAX Library* es una biblioteca centrada en el cliente e independiente del explorador de Internet que utilice el usuario.

Esta versión de *ASP.NET* permite la generación de servicios web basados en *ASMX* o *WCF.* Por otra parte, servicios de aplicación que se ejecutan en el servidor (como la autenticación de formularios, la administración de roles), se exponen en *ASP.NET 3.5* también como servicios web que pueden ser utilizados en aplicaciones compatibles con *Windows Communication Foundation*. *ASP.NET 3.5* permite que las aplicaciones de software que se encuentran basadas en web compartan estos servicios de aplicación.

1.5 .NET FRAMEWORK 4.0

Aunque al momento de escribir esta tesis, la versión *4.0* de *.NET Framework* se encuentra en la *Beta 1*, algunas de las novedades que presentará esta versión son:

En lenguajes de programación:

- ✓ Soporte para lenguajes de programación como *C# 4.0*, *C++ 10.0*, *F# 1.9.6.16* y *Visual Basic 10* de forma nativa y completa sobre *Visual Studio 2010*.

En el *Common Language Runtime:*

- ✓ Soporte para escribir código asíncrono y multihilo para usarse en múltiples procesadores.
- ✓ Computación paralela. Conformada por la *Task Parallel Library (TPL).*
- ✓ Contratos de datos para expresar código en forma de precondiciones, postcondiciones y objetos invariantes. Que un programa compile no tiene que significar necesariamente que esté correcto. Si el programa está mal diseñado se diagnosticará en tiempo de ejecución.
- ✓ *Parallel Language Integrated Query* (*PLINQ*) o *LINQ* paralelo.
- ✓ Inicialización lazy: Permite que un objeto no se inicialice hasta que su uso sea necesario.
- ✓ Soporte para lenguajes dinámicos. Las comprobaciones se hacen en tiempo de ejecución (*dynamic language runtime, DLR*) y no en tiempo de compilación.
- ✓ El recolector de elementos no utilizados se optimizó para disponer de los recursos distribuidos en varios procesadores cuando se finalice el proceso.

En la librería de clases base:

- ✓ La clase *Tuples:* La clase *System.Tuple* es una clase genérica que mantiene un conjunto ordenado de datos híbridos, Teóricamente una *Tuple* puede almacenar *n* valores para *n* tipos genéricos declarados en la clase.
- ✓ La estructura *BigInteger*: La estructura *System.Numerics.BigInteger* soporta el manejo de número enteros más haya de las estructuras ordinarias de *Int16*, *Int32* e *Int64* de *.NET Framework 2.0*.
- ✓ Adiciones a muchas clases existentes para ofrecer servicios de programación paralela.

En red:

- ✓ Nuevas funciones para clases que permiten para trabajar en redes, la mayoría son cambios producidos por las nuevas características de seguridad de *Windows 7*.
- ✓ En *ASP.NET 4.0*: Nuevos controles, meta tags, datos dinámicos, etcétera.

En Windows Communication Foundation:

- ✓ Soporte para la especificación *WS-Discovery*.
- ✓ Extremos estándar como *MEX*, contrato *IMetadataExchange*.

En Windows Presentation Foundation:

- ✓ Nuevos controles como *DataGrid, Calendar, DataPicker* y actualizaciones sobre los existentes.

CAPÍTULO 2.
CARACTERÍSTICAS BÁSICAS DE WINDOWS COMMUNICATION FOUNDATION

2.1 WINDOWS COMMUNICATION FOUNDATION

Windows Communication Foundation (WCF) ofrece la capacidad de crear aplicaciones de software distribuidas interoperables. *Windows Communication Foundation* reduce el tiempo de creación de programas conectados entre sí por medio del modelo de programación orientada a servicios. La programación orientada a servicios está diseñada para facilitar el desarrollo de aplicaciones distribuidas, anteriormente estas aplicaciones eran construidas haciendo uso de diferentes tecnologías de *.NET Framework* como: los servicios web de *ASP.NET*, *Enterprise Services y* comunicación remota de *.NET Framework*. La programación orientada a servicios contiene conceptos de servicios web así como conceptos de programación orientada a objetos de *.NET Framework*. Dado que *Windows Communication Foundation* hace uso de *.NET Framework* la implementación de servicios de *Windows Communication Foundation* se puede hacer en lenguajes de programación enfocados a esta plataforma de software como *C#. Windows Communication Foundation* incluye diferentes tipos de funcionalidades entre las que se encuentran: serialización, control de versiones, interoperabilidad con tecnologías predecesoras a *Windows Communication Foundation* como: *Web Service Enhancements (WSE), Message Queue Server (MSMQ)*, servicios web *ASP.NET*, y otras funciones más.

Windows Communication Foundation acepta una gran cantidad de protocolos y estándares diseñados para el intercambio de información entre aplicaciones distribuidas de forma segura y fiable. Además de lo anterior, también acepta una amplia variedad de opciones de codificación y transporte. Debido a la aceptación global de los servicios web, la forma en que se desarrollan los servicios ha cambiado. Actualmente, las funciones que proporcionan los servicios web incluyen seguridad, coordinación de transacciones distribuidas y una comunicación de fiar. Las ventajas que se han presentado en los servicios web debido a las recientes exigencias de la industria se deberían reflejar en las tecnologías que los programadores usan hoy en día. *Windows Communication Foundation* hace uso de las ventajas ofrecidas por los actuales servicios web, siendo una de las dos tecnologías más poderosas (la otra es *Web Services Interoperability Technologies* de *Java*) para crear aplicaciones distribuidas confiables.

2.1.1 ARQUITECTURA DE WCF

La siguiente ilustración permite visualizar las capas principales que constituyen la arquitectura de *Windows Communication Foundation*:

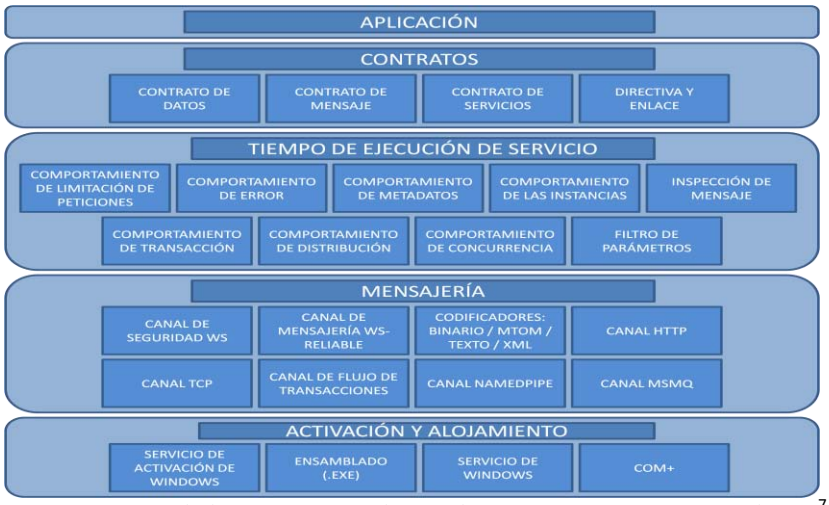


Figura 2.1 Capas de la arquitectura de Windows Communication Foundation[7]

A continuación se muestra información de cada una de las capas de la arquitectura de *Windows Communication Foundation*:

Contratos

En esta capa se encuentran varios contratos que definen diversos aspectos a usar en el sistema de mensajes. Los *contratos de datos* son clases que se definen para poder transferir información obtenida de un servicio. Estos contratos de datos se transfieren en formato *XML*, esto permite a cualquier sistema que entienda *XML* procesarlos. Los *contratos de mensaje* definen partes específicas del mensaje utilizando el protocolo *SOAP,* permitiendo un control más detallado sobre el mensaje cuando se requiere. Los *contratos de servicios* se especifican por medio de interfaces a través de alguno de los lenguajes de programación enfocados a *.NET Framework*. Las *directivas y enlaces* contienen información para comunicarse con un servicio de *Windows Communication Foundation*. Un enlace especifica como mínimo el transporte que se va a utilizar por ejemplo, *HTTP* o *TCP* y una codificación como *XML*. Una directiva incluye requisitos de seguridad y otras condiciones que se deben cumplir para establecer una comunicación.

Tiempo de ejecución de servicio

En esta capa se encuentran los comportamientos que se producen mientras dure la operación del servicio. El *comportamiento de la limitación de peticiones* controla cuántos mensajes se pueden procesar, este límite puede ser variable. El *comportamiento de error* especifica qué debe suceder cuando un error interno se presenta en el servicio, algo que puede hacer es comunicar al cliente que error ocurrió. El *comportamiento de metadatos* administra si los metadatos se deben exponer a diferentes clientes y cómo hacerlo. El *comportamiento de la instancia* indica el número de instancias del servicio de *Windows Communication Foundation* que se pueden ejecutar. El *comportamiento de transacción* permite la recuperación de transacciones si se produce un error en la ejecución. El *comportamiento de distribución* permite trabajar a un servicio de *Windows Communication Foundation* de forma distribuida. El *comportamiento de concurrencia* indica cuántos subprocesos se pueden ejecutar por instancia de servicio creada. El *filtro de parámetros* consiste en realizar acciones localizadas en encabezados de mensajes.

[7] Obtenido de: Arquitectura de Windows Communication Foundation. http://msdn.microsoft.com/es-mx/library/ms733128.aspx

Mensajería

En esta capa se encuentran los canales. Un canal es un componente que procesa un mensaje de una manera específica, un procesamiento puede ser la autenticación de un mensaje. Los canales funcionan sobre los mensajes y los encabezados del mensaje. Esta capa se diferencia de la capa de *tiempo de ejecución del servicio*, en que la segunda se encarga de procesar principalmente el contenido del cuerpo del mensaje.

Existen dos tipos de canales, estos se mencionan a continuación:

- ✓ Los canales de transporte: estos canales leen y escriben mensajes provenientes de una red. Algunos transportes hacen uso de codificadores para convertir los mensajes (información *XML*) hacia y desde la secuencia de bytes utilizada por la red. *HTTP*, *TCP* y *MSMQ* son ejemplos de transportes. *XML* y *binario optimizado* son ejemplos de codificaciones.
- ✓ Los canales de protocolo: estos canales leen y escriben encabezados extra en el mensaje, implementan protocolos de procesamiento de mensajes como *WS-Security* y *WS-Reliability. WS-Security* es una implementación de la especificación *WS-Security* contenida en *WS-** que habilita la seguridad en la capa del mensaje. *WS-Reliable* garantiza la entrega del mensaje. Los codificadores disponibles ofrecen una variedad de codificaciones que satisfacen las necesidades del mensaje. El *canal HTTP* indica que el *HyperText Transfer Protocol* se usa para entregar el mensaje. El *canal TCP* especifica que se usará el *Transmission Control Protocol* para enviar el mensaje. El *canal de flujo de transacciones* administra los modelos de mensajes de transacción. El *canal de MSMQ* permite la interoperabilidad con aplicaciones construidas con *MSMQ*.

Alojamiento y activación

Un servicio de *Windows Communication Foundation* es un ensamblado de *.NET Framework*. Como otros ensamblados, éste se debe poder ejecutar. A esto se le conoce como un servicio autohospedado. Sin embargo, ésta no es la única forma en la que se pueden ejecutar servicios de *Windows Communication Foundation*. Un servicio también se puede ejecutar por cualquiera de las siguientes formas: en un ensamblado externo construido con *.NET Framework* o dentro del sistema operativo *Windows* usando *Internet Information Server* (*IIS), o* el *Servicio de Activación de Windows* (*WAS*), o servicios de *Windows* o a través de un componente *COM+.*

2.1.2 SERVICIO (SERVICE)

Para que distintas aplicaciones de software creadas en lenguajes de programación diferentes y ejecutadas sobre diversas plataformas puedan comunicarse entre ellas necesitan hacer uso de un servicio. Un servicio incluye un conjunto de estándares y protocolos que contienen los mecanismos necesarios para intercambiar información entre aplicaciones de software por medio de redes de computadoras. La interoperabilidad que otorgan los servicios se consigue mediante la adopción de estándares. *W3C* y *OASIS* son las organizaciones responsables de mantener las especificaciones de los servicios web[8]. Para garantizar la interoperabilidad entre distintas implementaciones de servicios web la *Web Services Interoperability Organization* promueve mediante la especificación *Web Services Interoperability (WS-I*) la Interoperabilidad de los servicios web. Un servicio de *Windows Communication Foundation (WCF Service)* expone uno o más extremos (*endpoint*). Cada extremo expone una o más contratos de operaciones.

[8] Un servicio web es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

2.1.3 EXTREMO (ENDPOINT)

Los extremos de un servicio permiten que una aplicación cliente pueda comunicarse con un servicio de *Windows Communication Foundation*. Un extremo proporciona acceso a la funcionalidad que brinda un servicio a un cliente. La especificación de un extremo se puede hacer de forma imperativa mediante un lenguaje de programación como *C#* o de forma declarativa a través de un archivo de configuración *app.config*.

Windows Communication Foundation ofrece dos tipos de extremos:

- ✓ El extremo de aplicación: este extremo es un contrato de servicio implementado por un servicio de *Windows Communication Foundation* y expuesto por él mismo.
- ✓ El extremo de infraestructura: este extremo expone la infraestructura que otorga la funcionalidad necesaria y que no se relaciona con un contrato de servicio. Un ejemplo de este extremo podría ser un extremo que proporciona información sobre los metadatos del servicio.

El extremo (*Endpoint*) de un servicio de *Windows Communication Foundation* está compuesto por una dirección (*Address*), un enlace (*Binding*), un contrato (*Contract*) (el *ABC* de *Windows Communication Foundation*) y un comportamiento (*Behavior*). A continuación se explican cada uno de estos componentes:

A. La dirección (Address) de un extremo identifica al extremo e indica a los clientes del servicio de *Windows Communication Foundation dónde* se ubica éste. La dirección de extremo se manipula por medio de la clase *EndpointAddress*. Los miembros de esta clase son:
   - ✓ La propiedad *Uri:* representa la dirección del servicio de *Windows Communication Foundation*.
   - ✓ La propiedad *Identity:* representa la identidad de seguridad del servicio y una colección de encabezados de mensaje opcionales. Los encabezados se pueden usar para proporcionar información adicional de direccionamiento y más detallada para hacer uso del extremo.

B. El enlace (Binding) de un extremo indica *cómo* se debe comunicar éste con las aplicaciones cliente que realicen una solicitud, los aspectos a considerar son:
   - ✓ El protocolo de transporte que se debe usar en la comunicación, éste podrían ser: *Transmission Control Protocol (TCP)*, *Hypertext Transfer Protocol* (*HTTP)* o cualquier otro.
   - ✓ La codificación (*XML* o *binario optimizado*) que se usa para transferir mensajes.
   - ✓ Requisitos de seguridad como *Secure Sockets Layer (SSL)* o seguridad de mensaje *SOAP*.
   - ✓ El enlace que puede ser cualquiera de los enlaces proporcionados por *.NET Framework* y que derivan de la clase base abstracta *Binding*.

C. Un contrato (Contract) indica *qué* funcionalidad expone el extremo del servicio a un cliente. Un contrato especifica lo siguiente:
   - ✓ La forma en la que se va a transferir el mensaje.
   - ✓ Los contratos de operación que puede mandar a llamar un cliente.
   - ✓ Los parámetros de entrada requeridos para poder llamar a un contrato de operación.
   - ✓ El tipo de respuesta que puede esperar el cliente que realizó la llamada.

Comportamientos: Los comportamientos de extremo se usan para personalizar el comportamiento de forma local del extremo de un servicio. La propiedad *ListenUri* es un ejemplo de un comportamiento de extremo que se puede personalizar, ésta permite indicar una dirección de escucha diferente a la dirección *SOAP* o a la dirección del *Lenguaje de descripción de servicios web*[9] (*WSDL*).

2.1.4 DIRECCIÓN DE EXTREMO (ENDPOINTADDRESS)

Un extremo está asociado con una dirección, ésta se usa para hallar e identificar al extremo. La dirección se compone de un Identificador uniforme de recursos (*URI*), que indica la ubicación del extremo. La dirección del extremo se representa en mediante la clase *EndpointAddress*, esta clase contiene la propiedad *Identity* opcional, esta propiedad permite que otros extremos intercambian mensajes con él a través de la autenticación del extremo. Además de la propiedad anterior, contiene un conjunto de propiedades *Headers* también opcionales, que definen otros encabezados *SOAP* requeridos para usar el servicio. Estos encabezados opcionales dan información adicional de direccionamiento y más detallada para identificar o interactuar con el extremo. La dirección de un extremo se representa en la conexión con la especificación *WS-Addressing*.

Estructura URI
El identificador de recursos uniforme (*Uniform Resource Identifier, URI*) en la mayoría de transportes está conformado de cuatro partes. Las cuatro partes en las que se divide un identificador de recursos uniforme como este *http://www.ynnaf.com:8080/ynnafService.svc/ynnafEndpoint* son:

- ✓ Protocolo: *HTTP*
- ✓ Equipo (IP o DNS): *www.ynaaf.com*
- ✓ Puerto: *8080*
- ✓ Ruta de acceso (recurso): */ynnafService.svc/ynnafEndpoint*

Definición de una dirección para un Servicio
Como todo en *Windows Communication Foundation,* la dirección de extremo de un servicio se puede definir a través de código o de configuración. Es recomendable definir extremos de servicio a través de archivos de configuración. La ventaja de esto es que se pueden cambiar las direcciones de extremos sin tener que volver a compilar la aplicación. La definición de un extremo por medio de configuración, se hace con el elemento *<endpoint>.* Para definir la dirección de un extremo mediante código se hace con la clase *EndpointAddress*.

Extremos en WSDL
La dirección de un extremo se puede representar en el *Lenguaje de descripción de servicios web* (*WSDL)* mediante un elemento *EndpointReference* (*EPR)* de acuerdo a la especificación *WS-Addressing* en el elemento *wsdl:port* del extremo indicado. El *EndpointReference* contiene la dirección del extremo y todas las propiedades de la dirección.

2.1.5 ENLACE (BINDING)

Los enlaces en *Windows Communication Foundation* se usan para definir el transporte, la codificación y los detalles de protocolo que los clientes y servicios usan para comunicarse entre sí. Los

[9] WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligan después al protocolo concreto de red y al formato del mensaje.

enlaces funcionan para generar la representación de la conexión subyacente de los extremos, entonces, para lograr una correcta interoperabilidad la mayoría de los detalles de enlace deben ser acordados por las partes que se están comunicando. La manera más fácil de lograr la interoperabilidad es que todos los involucrados utilicen el mismo enlace.

Windows Communication Foundation separa cómo se escribe el software para una aplicación de cómo se comunica con otro software, este último es el enlace. Un enlace o *binding* está compuesto de un conjunto de elementos. Cada elemento tiene que ver con algún aspecto de comunicación con los clientes. Lo mínimo que un enlace debe incluir es un elemento de enlace de transporte, y un elemento referente a la codificación de mensajes (aunque que el enlace de transporte lo puede proporcionar de forma predeterminada). En tiempo de ejecución cada elemento de enlace puede contribuir en el código.

Los enlaces proporcionados por *Windows Communication Foundation* contienen elementos que se pueden utilizar en su configuración predeterminada o personalizada para cumplir las necesidades de una aplicación. Es posible trabajar fácilmente y en paralelo con diversas versiones de un enlace, la única que condición que se debe cumplir es dar un nombre diferente a las versiones del enlace.

Es posible crear un enlace personalizado siempre y cuando esté compuesto por los elementos requeridos para establecer una comunicación. Esto se hace si los enlaces proporcionados por *Windows Communication Foundation* no se adaptan a las necesidades específicas.

Existe una gran variedad de enlaces proporcionados por *Windows Communication Foundation*, todos ellos optimizados para escenarios concretos. Los enlaces proporcionados optimizan el desarrollo de servicios presentando únicamente las opciones que se aplican correctamente al escenario en cuestión. Por ejemplo, la funcionalidad del enlace *WSHttpBinding* proporciona interoperabilidad con servicios web que implementan las más recientes especificaciones *WS-**[10]. La siguiente tabla muestra una descripción de los enlaces proporcionados por *Windows Communication Foundation:*

ENLACE	DESCRIPCIÓN
BasicHttpBinding	Este enlace se usa para la comunicación con servicios web que cumplen con la especificación *WS-Basic Profile.* Los servicios web de *ASP.NET (ASMX)* y *Java* cumplen con ella. De manera predeterminada utiliza *HTTP* como medio de transporte y *XML* para codificar mensajes.
WSHttpBinding	Este es un enlace seguro e interoperable (compatible con *Web Services Interoperability Technologies*). Este enlace es adecuado para servicios que no son del tipo dúplex.
WSDualHttpBinding	Este es un enlace seguro e interoperable. Éste es adecuado para servicios del tipo dúplex y para comunicación por medio de intermediarios de *SOAP*.
WSFederationHttpBinding	Este es un enlace seguro e interoperable que usa la especificación *WS-Federation* para permitir a las organizaciones que se encuentran en una federación autenticar y autorizar eficazmente a los usuarios.
NetTcpBinding	Este es un enlace seguro y optimizado adecuado para la comunicación entre computadoras que usan aplicaciones de *Windows Communication Foundation*.
NetNamedPipeBinding	Este es un enlace seguro, confiable y optimizado adecuado para la comunicación entre computadoras que usan aplicaciones de *Windows Communication Foundation*.
NetMsmqBinding	Este es un enlace en cola adecuado para la comunicación entre computadoras que usan aplicaciones de *Windows Communication Foundation*.

[10] Las especificaciones para servicios Web *WS-** dan lugar a protocolos con capacidad de interactuar, manteniendo su interoperabilidad a nivel de Seguridad, intercambio fiable de mensajes y transacciones, en sistemas perfectamente integrados. Las especificaciones se establecen a partir de los estándares de base *XML* y *SOAP*.

NetPeerTcpBinding	Este es un enlace que permite la comunicación segura entre múltiples computadoras.
MsmqIntegrationBinding	Este es un enlace adecuado para la comunicación entre computadoras que usan una aplicación de *Windows Communication Foundation* y una aplicación de *Message Queue Server*.
BasicHttpContextBinding	Este es un enlace adecuado para la comunicación con los servicios web compatibles con el perfil *WS-Basic Profile* haciendo el uso de cookies *HTTP* para intercambiar el contexto.
NetTcpContextBinding	Este es un enlace seguro y optimizado adecuado para la comunicación entre aplicaciones de *Windows Communication Foundation* en distintas computadoras permitiendo usar encabezados *SOAP* para intercambiar el contexto.
WebHttpBinding	Este es un enlace utilizado para configurar los extremos de los servicios web de *Windows Communication Foundation* que se exponen a través de solicitudes *HTTP* en lugar de mensajes *SOAP*.
WSHttpContextBinding	Este es un enlace seguro e interoperable que permite utilizar encabezados SOAP para intercambiar el contexto. Este enlace es adecuado para servicios no dúplex.

Tabla 2.1 Enlaces proporcionados por WCF

2.1.6 CONTRATO (CONTRACT)

Un contrato define qué es lo que puede hacer un servicio. Los contratos que brinda *Windows Communication Foundation* son: de servicio, de operación, de mensaje, de error y de datos. Un *contrato de servicio* define ajustes de servicio como el espacio de nombres del servicio, el nombre del servicio y otros ajustes que conforman una unidad funcional única. Se recomienda que un contrato de servicio se defina creando un interfaz en un lenguaje de programación enfocado a *.NET Framework* aplicando el atributo *ServiceCon-tractAttribute.* El contrato de servicio real se obtiene de la implementación de las firmas de la interfaz. Un *contrato de operación* define los parámetros y el tipo del valor retornado por un método. Cuando se define un contrato de operación, éste debe contener el atributo *OperationContractAttribute*. Los métodos se pueden tomar un conjunto de tipos y devolver un tipo. En este caso, el sistema determinará el formato de los mensajes que han de intercambiarse. Un *contrato de mensaje* describe el formato de un mensaje. Este contrato puede declarar si los elementos del mensaje deben ir en encabezados, qué nivel de seguridad debe aplicarse a los elementos del mensaje, entre otros. Un *contrato de error* se asocia a un método de servicio (contrato de operación) para mostrar errores que se pueden retornar al llamador. Un método puede tener ninguno o muchos errores asociados a él. Estos errores son errores de *SOAP* que se modelan como excepciones en el modelo de programación. Para mayor información ver el siguiente capítulo de esta tesis. Los *contratos de datos* son tipos de que se pueden utilizar en cualquier parte del servicio. Estos contratos son clases que contienen el atributo *DataContractAttribute,* los miembros de estas clases son generalmente propiedades que contienen el atributo *DataMemberAttribute*. Si el servicio únicamente utiliza tipos simples y serializados de forma predeterminada, no hay necesidad de hacer uso de contratos de datos.

2.1.7 COMPORTAMIENTO (BEHAVIOR)

Un comportamiento controla diversos aspectos del servicio en tiempo de ejecución, estos pueden ser un extremo, un cliente o una operación. Los comportamientos se agrupan en función del ámbito, por ejemplo: los *comportamientos comunes* afectan de forma global a todos los extremos contenidos en el servicio; los *comportamientos de servicios* únicamente afectan temas relacionados con el servicio; los *comportamientos de extremos* solamente afectan a las propiedades de los extremos; y los *comportamientos de operaciones* solo afectan a operaciones determinadas. Entonces, un comportamiento de servicio indica cómo reacciona un servicio al producirse un exceso de

mensajes que amenaza sus funciones de control, un comportamiento de extremo, puede indicar cómo y dónde encontrar una credencial para seguridad.

2.1.8 HOSPEDAJE

Una vez que un servicio de *Windows Communication Foundation* está listo para funcionar se debe poder hospedar. Los servicios generados se pueden autohospedar o una aplicación diseñada para ese fin puede administrarlo. Un servicio autohospedado es aquel que ejecuta dentro de una aplicación que creó el programador específicamente para ese servicio, todas las actividades que se necesitan realizar para ejecutar un servicio son responsabilidad del programador, entre ellas están: abrir el servicio, escuchar por peticiones y cerrar el servicio. Aplicaciones que están diseñadas para alojar servicios de forma nativa son: *Internet Information Services (IIS)*, *Windows Activation Services (WAS)* y *Windows Services*. A diferencia de las primeras aplicaciones, éstas controlan toda la administración del servicio, por ejemplo, al desplegar un servicio de *Windows Communication Foundation* en *Internet Information Services* cuando éste recibe una petición inicia el servicio, responde las solicitudes y controla su tiempo de vida.

2.1.9 PROGRAMACIÓN DEL LADO DEL CLIENTE

Un cliente es una aplicación que despliega las operaciones del servicio como métodos. En el caso de *.NET Framework* toda aplicación puede alojar a un cliente de *Windows Communication Foundation*, incluso una aplicación que exponga un servicio. Por lo tanto, se puede crear un servicio que incluya clientes provenientes otros servicios. Una aplicación cliente se puede generar automáticamente utilizando las herramientas *ServiceModel Metadata Utility Tool (Svcutil.exe)* de *.NET Framework* o *WebService Import (wsimport.exe)* de *Java* siempre y cuando el servicio se esté ejecutando y publique metadatos. Dado que una aplicación cliente intercambia mensajes con extremos, se debe crear una instancia del cliente del servicio de *Windows Communication Foundation* y mandar a llamar a sus métodos a partir de ella.

2.1.10 INTERCAMBIO DE METADATOS

Los metadatos de un servicio web representan las características del servicio. *ServiceModel Metadata Utility Tool (Svcutil.exe)* de *.NET Framework* y *WebService Import (wsimport.exe)* de *Java* utilizan estos metadatos para poder generar un cliente del servicio que permita a una aplicación cliente interactuar con el servicio. Los metadatos de servicio incluyen documentos de esquema *XML* que definen contratos de datos del servicio, y documentos *WSDL* que describen los contratos de operación del servicio.
Windows Communication Foundation y *Web Services Interoperability Technologies* de *Java* generan de forma automática los metadatos del servicio a través de la inspección del servicio y de sus extremos. Se debe permitir explícitamente al comportamiento de los metadatos en un servicio para que puedan ser expuestos.

2.1.11 ADMINISTRACIÓN VÍA PROGRAMACIÓN VS. CONFIGURACIÓN

Un servicio de *Windows Communication Foundation* se puede controlar de tres formas: por código, por configuración o mixto. La ventaja que tiene la configuración sobre la primera es que permite que una persona que no sea el programador del servicio (como el administrador de la red donde se está ejecutando) indicar nuevos parámetros al servicio o al cliente o eliminarlos sin necesi-

dad de volver a compilar el código. La configuración otorga la capacidad de establecer direcciones de extremos y agregar extremos, enlaces y comportamientos. Aunque la codificación es menos flexible que la configuración, permite al programador del servicio obtener un control total de todos los componentes del servicio o del cliente. Si en la configuración se hace un ajuste que vaya en contra de la lógica del negocio de la empresa, éste se puede eliminar por medio de código.

2.1.12 CONFIABILIDAD

Los servicios de *Windows Communication Foundation* son confiables porque contienen las siguientes características de seguridad:

- ✓ Confidencialidad: encripta los mensajes para evitar que usuarios malintencionados puedan tener acceso de lectura; integridad: impide que usuarios malintencionados manipulen el mensaje.
- ✓ Autenticación: permite validar a los servicios y clientes.
- ✓ Autorización: permite tener un control del acceso a los recursos. Estas funciones se proporcionan mediante la utilización de mecanismos de seguridad como *HTTPS* o la implementación de una o más de las especificaciones de seguridad de *WS-**.

2.2 CONTRATOS DE SERVICIOS

Un contrato de servicio define ajustes de servicio, éste proporciona y establece información sobre:

- ✓ El conjunto de operaciones que puede realizar un servicio, su ubicación y su firma en términos de mensajes intercambiados.
- ✓ Los tipos de datos de estos mensajes.
- ✓ Los protocolos de transporte y los formatos de *serialización* que se usan para garantizar la correcta comunicación con el servicio de *Windows Communication Foundation*.

Un contrato de servicio para administrar una orden de compra podría contener un contrato de operación *CrearOrden* que acepte como parámetro de entrada el número de orden de compra y devuelva la información de la orden si se realizó el procesamiento correctamente o un error de lo contrario. También podría tener un contrato de operación llamado *ObtenerEstadoOrden* que acepte como parámetro de entrada el número de orden de compra y retorne la información sobre el estado de la orden (inicial, *atendiendo*, *enviando*, *enviado*, *etc*.). Entonces, un contrato de servicio como el anterior especificaría:

- ✓ El contrato de servicio consiste de dos contratos de operación: *CrearOrden* y *ObtenerEstadoOrden*.
- ✓ Los contratos de operación poseen mensajes de entrada (parámetro) y de salida (retorno).
- ✓ Los datos que llevan los mensajes de entrada y de salida.
- ✓ Instrucciones sobre la comunicación necesaria para procesar los mensajes correctamente.

La transferencia de contratos de servicio entre aplicaciones, se hace principalmente mediante formatos de *XML* estándar, como *Web Services Description Language (WSDL)* y *XML* Schema (*XSD*). Esta información de contrato público se usa para crear aplicaciones cliente que pueden comunicarse con el servicio, pues estos lenguajes están diseñados para interoperar con formatos y protocolos públicos compatibles con el servicio.

Web Services Description Language (WSDL) y *XML* Schema (XSD) son lenguajes adecuados para describir contratos de servicios, sin embargo son lenguajes difíciles de usar directamente. Esto

último es trivial porque únicamente son descripciones del servicio y no son usadas de manera directa por el programador al trabajar con el servicio. *Windows Communication Foundation* utiliza de forma familiar interfaces y clases para definir la estructura de un servicio e implementarlo. El contrato de servicio resultante se puede exportar como metadatos (*WSDL* y *XSD*) cuando un cliente necesite usarlo sobre todo para permitir la interoperabilidad entre otras plataformas. Los detalles de los mensajes *SOAP* como el transporte y seguridad, son tratados por *.NET Framework*, éste realiza de forma automática las conversiones necesarias del sistema de tipos del contrato de servicio al sistema a los tipos de *XML* y viceversa.

2.2.1 DISEÑO DE CONTRATOS DE SERVICIOS

Los contratos de servicios son un conjunto de contratos de operación. Los contratos de operación se definen creando un método y marcándolo con la clase de atributo *OperationContractAttribute*. Un contrato de servicio, entonces, contiene un grupo de contratos de operaciones en el interior de una interfaz marcada con la clase de atributo *ServiceContractAttribute*. Los métodos que no contienen el atributo *OperationContractAttribute* no son contratos de operación y por lo tanto no se exponen para ser utilizados por los clientes del servicio. Estos métodos únicamente pueden ser llamados desde el interior del servicio.

Las interfaces y las clases pueden representan una agrupación de contratos de operación, por consiguiente, las dos se pueden usar para definir un contrato de servicio. A pesar de lo anterior, se hace hincapié en usar interfaces porque organizan directamente los contratos de servicios. Si no se hace una implementación de la interfaz que contiene los contratos de operación, el contrato de servicios solo define operaciones que no tienen ninguna funcionalidad. Si se implementan los contratos de operación de la interfaz del contrato de servicio se habrá implementado un servicio *de Windows Communication Foundation*.

Las características de las interfaces de contratos de servicio son:

- ✓ Su uso es igual al de interfaces usadas en la programación orientada a objetos de *.NET Framework.*
- ✓ Si se modifica la implementación de la interfaz de un contrato de servicio, el contrato de servicio sigue siendo el mismo.
- ✓ Se puede controlar la versión de un servicio implementando una interfaz antigua y una interfaz nueva. Los clientes antiguos se conectan a la versión antigua y los clientes nuevos se conectan a la versión nueva.
- ✓ La interfaz del contrato de servicio puede implementar a su vez cualquier número de interfaces de contrato de servicio.
- ✓ Una clase puede implementar cualquier número interfaces de contratos de servicios.

El siguiente código en *C#* muestra cómo definir un contrato de servicio por medio de una interfaz:

```
// Defición del contrato ICalculadora.
[ServiceContract]
public interface ICalculadora {
    [OperationContract]
    double Suma(double a, double b);
    [OperationContract]
    double Resta(double a, double b);
    [OperationContract]
    double Multiplicacion(double a, double b);
```

```
    [OperationContract]
    double Division(double a, double b);
}
```

2.2.2 IMPLEMENTACIÓN DE CONTRATOS DE SERVICIOS

El siguiente código en *C#* muestra un servicio (*clase*) que implementa el contrato de servicio *ICalculadora* (*interfaz*) definido arriba:

```
// Implementación del contrato ICalculadora en la clase CalculadoraService.
public class CalculadoraService : ICalculadora {
    public double Suma(double a, double b) { return a + b; }
    public double Resta(double a, double b) { return a - b; }
    public double Multiplicacion(double a, double b) { return a * b; }
    public double Division(double a, double b) { return a / b; }
}
```

Análogamente al ejemplo anterior, un servicio (*clase*) puede exponer un contrato de servicio de forma directa. A continuación, se muestra el código en *C#* que define e implementa un contrato *CalculadoraService*.

```
// Defincion el contrato CalculadoraService directamente
// en la clase del servicio.
[ServiceContract]
class CalculadoraService {
    [OperationContract]
    public double Suma(double a, double b) { return a + b; }
    [OperationContract]
    public double Resta(double a, double b) { return a - b; }
    [OperationContract]
    public double Multiplicacion(double a, double b) { return a * b; }
    [OperationContract]
    public double Division(double a, double b) { return a / b; }
}
```

Se debe tener en cuenta que los servicios anteriores son diferentes porque sus nombres son diferentes. En el primer servicio, el contrato se llama *ICalculadora* mientras que, en el segundo, el contrato se llama *CalculadoraService*. El nombre lo define en dónde está colocado el atributo *ServiceContractAttribute*.

Aplicar al contrato de servicio directamente los atributos *ServiceContractAttribute* a la clase y *OperationContractAttribute* a sus métodos incrementa la velocidad y simplicidad de desarrollo de aplicaciones *Windows Communication Foundation.* Lamentablemente esta forma de implementación no permite la herencia múltiple (reglas de *.NET Framework*) y como resultado únicamente se puede implementar un contrato de servicio a la vez. Además, cualquier modificación a las firmas de los contratos operación de la clase modifica el contrato público del servicio, esto trae como consecuencia que clientes no modificados no usen el servicio.

2.3 CONTRATOS DE DATOS

Un contrato de datos es una clase que describe los datos que se van a intercambiar entre un servicio y un cliente. Para comunicarse un cliente y un servicio no tienen que tener los mismos tipos, sino los mismos contratos de datos. Un contrato de datos define para cada parámetro de tipo o tipo devuelto, qué se debe serializar (convertir en *XML*) para lograr un intercambio de información.

Windows Communication Foundation utiliza un motor de serialización[11] para serializar y deserializar los datos (convertir a *XML* y desde *XML*). En *.NET Framework* los tipos como números enteros, números de punto flotante, cadenas, *DateTime, XmlElement, Bitmap, Tuple (.NET Framework 4.0)* y otros tipos están serializados de forma predeterminada (la mayoría usando la interfaz *ISerializable*).

Los tipos que define un programador deben contener un contrato de datos definido para que sean serializables. La clase *DataContractSerializer*[12] puede deducir de forma predeterminada el contrato de datos y serializa todos los tipos públicos. Se serializan todos los campos y propiedades de lectura y escritura públicos de la clase. Se pueden ignorar miembros para la serialización haciendo uso del atributo *IgnoreDataMemberAttribute*. Se puede crear explícitamente (recomendado) un contrato de datos mediante el uso de los atributos *DataContractAttribute* y *DataMemberAttribute*. El atributo *DataContractAttribute* se aplica a clases, estructuras y enumeraciones. El atributo *DataMemberAttribute* se aplica a cada miembro público de la clase que funge como miembro de datos para indicar que se debe serializar.

2.3.1 CREAR UN CONTRATO DE DATOS PARA UNA CLASE O ESTRUCTURA

La creación de un contrato de datos se debe fundamentar en lo siguiente:

1. Declarar la clase que servirá como contrato de datos aplicando el atributo *DataContractAttribute*. Todas las clases públicas, incluidas aquellas sin atributos, son serializables. Si no se aplica el atributo *DataContractAttribute* explícitamente, la clase *DataContractSerializer* deduce un contrato de datos.
2. Definir los miembros (propiedades, campos o eventos) que deben serializarse aplicando el atributo *DataMemberAttribute* a cada uno de ellos. A estos miembros se les denomina miembros de datos. Todos los tipos públicos son serializables de forma predeterminada.

El siguiente código fuente en *C# 3.0* muestra cómo crear un contrato de datos para la clase *OrdenCompra* aplicando los atributos *DataContractAttribute* y *DataMemberAttribute* explícitamente.

```
[DataContract]
public class OrdenCompra {
    // Aplicar el DataMemberAttribute a las propiedades.
    [DataMember]
    public int IdOrdenCompra { get; set; }
    [DataMember]
    public DateTime FechaOrdenCompra { get; set; }
}
```

[11] La serialización consiste en un proceso de codificación de un Objeto (programación orientada a objetos) en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como *XML*. La serie de bytes o el formato pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno (por tanto, el nuevo objeto es un clon del original). La serialización es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.
Fuente: Wikipedia http://es.wikipedia.org/wiki/Serialización

[12] DataContractSerializer (Clase): Serializa y deserializa una instancia de un tipo en una secuencia o en un documento *XML* utilizando un contrato de datos proporcionado. Esta clase no puede heredarse. Fuente: Microsoft Developer Network.
http://msdn.microsoft.com/es-mx/library/system.runtime.serialization.datacontractserializer.aspx

A continuación se muestran como algunas de los tipos más utilizados de *.NET Framework* (por ejemplo: *Bitmap* y *Uri*) [13] están serializadas de forma predeterminada a través del atributo *SerializableAttribute*.

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public sealed class Bitmap : Image
```

```
[SerializableAttribute]
[TypeConverterAttribute(typeof(UriTypeConverter))]
public class Uri : ISerializable
```

El siguiente código fuente escrito en *C#* muestra un contrato de servicio (*interfaz*) en el cual se ejemplifica como los tipos primitivos no requieren un contrato de datos, mientras que un tipo complejo (clase *OrdenCompra* creada arriba) sí lo requirió:

```
[ServiceContract]
public interface ICompra {
    // No es necesario un contrato de datos ya que tanto el tipo
    // del parámetro como el tipo de retorno son tipos primitivos.
    [OperationContract]
    double ObtenerIVA(double precio);

    // No es necesario un contrato de datos ya que tanto el tipo
    // del parámetro como el tipo de retorno están marcados con
    // el atributo SerializableAttribute.
    [OperationContract]
    System.Drawing.Bitmap ObtenerImagen(System.Uri imagenUri);

    // El tipo OrdenCompra es de tipo complejo, entonces éste
    // requiere un contrato de datos. El tipo de retorno no lo
    // requiere porque es un tipo primitivo.
    [OperationContract]
    bool AprobarOrdenCompra(OrdenCompra ordenCompra);
}
```

Al crear un contrato de datos existen una serie de elementos a considerar:

- ✓ El atributo *IgnoreDataMemberAttribute* especifica que el miembro de datos no forma parte de un contrato de datos y no está serializado.
- ✓ El atributo *DataMemberAttribute* se aplica a campos, propiedades y eventos.
- ✓ Los niveles de accesibilidad *internal*, *private*, *protected* o *public* de miembros no afectan de manera alguna al contrato de datos.
- ✓ El atributo *DataMemberAttribute* se omite si se aplica a miembros estáticos.
- ✓ En el proceso de serialización se manda a llamar al código de obtención de propiedades para que los miembros de datos de propiedad obtengan el valor de las propiedades serializables.
- ✓ En el proceso de deserialización, se crea un objeto no inicializado, sin llamar a ningún constructor. Posterior a esto, se deserializan todos los miembros de datos.
- ✓ Un contrato de datos es válido si es posible serializar todos sus miembros de datos. Para los tipos genéricos no hay ningún requisito especial. Por ejemplo, veamos la siguiente clase genérica en *C#*:

```
[DataContract]
public class MiTipoGenerico1<T> {
    // Miembros de la clase
}
```

[13] Fuente:
Bitmap (Clase). http://msdn.microsoft.com/es-mx/library/system.drawing.bitmap.aspx
Uri (Clase). http://msdn.microsoft.com/es-mx/library/system.uri.aspx

Esta clase es serializable tanto si el tipo que se usa para el parámetro de tipo genérico (*T*) es serializable como si no lo es. Dado que debe ser posible serializar todos los miembros de datos, el tipo siguiente sólo es serializable si el parámetro de tipo genérico también es serializable, como se muestra en el siguiente código fuente en *C#*.

```
 [DataContract]
public class MiTipoGenerico2<T> {
    [DataMember]
    T elDato;
}
```

2.3.2 HERENCIA

Una clase que no contenga el atributo *DataContractAttribute* puede heredar de una clase que sí lo declare, el caso contrario no es posible. Esta especificación garantiza que se pueda tener compatibilidad con código escrito en versiones anteriores a *.NET Framework 3.0*.

2.3.3 NOMBRES DE CONTRATOS DE DATOS

Un servicio y un cliente pueden no compartir los mismos tipos, aun así pueden transferir datos entre sí siempre y cuando con contratos de datos sean equivalentes en ambos lados. La equivalencia se basa en el nombre del contrato de datos, sus miembros y los tipos manejados. Estos últimos se asignan mediante un mecanismo propio de la herramienta importadora del *WSDL*.

Las reglas básicas para los nombres de los contratos de datos de *Windows Communication Foundation* son:

- ✓ Análogamente al nombre completo de las clases en *.NET Framework*. El nombre completo de un contrato de datos se compone de su espacio de nombres y de su nombre.
- ✓ Los miembros de datos contenidos en los contratos de datos únicamente tienen nombres, sin espacio de nombres.
- ✓ El procesamiento de contratos de datos en *Windows Communication Foundation* es case sensitive (distingue entre mayúsculas y minúsculas) en espacios de nombres y en nombres de contratos de datos y miembros de datos.

Equivalencia entre contratos de datos.
Para que un servicio y un cliente puedan transferir información entre ellos de manera correcta, no es necesario que los contratos de datos existan exactamente en los dos puntos, basta con que sean equivalentes. La equivalencia de los contratos sea efectiva, el espacio de nombre y el nombre del contrato de datos debe ser igual, los tipos de los miembros de datos deben ser equivalentes y sus nombres iguales. Por ejemplo, si el tipo de un miembro de datos en un contrato de datos de *Windows Communication Foundation* es *System.String*, un cliente *Web Services Interoperability Technologies* que acceda al servicio de *Windows Communication Foundation* utilizará un tipo *java.lang.String*. Los nombres y espacios de nombres de contratos de datos y de los miembros de datos son case sensitiva, es decir, distinguen entre mayúsculas y minúsculas. Si dos tipos existen tanto en el remitente como en el receptor y no son equivalentes no se les dará el mismo espacio de nombres y nombre. Si se hiciera, generaría excepciones.

Los contratos de datos escritos en *C# 3.0* que se muestran a continuación son equivalentes:

```
[DataContract]
public class Cliente {
 [DataMember]
 public string NombreCompleto { get; set; }

 [DataMember]
 public string TelefonoCelular { get; set; }
}
```

```
[DataContract(Name = "Cliente")]
public class Persona {
 [DataMember(Name = "NombreCompleto")]
 private string NomPersona { get; set; }

 [DataMember(Name = "TelefonoCelular")]
 private string NumTelCel { get; set; }
}
```

2.4 CONTRATOS DE OPERACIONES

Los contratos de operaciones de un servicio de *Windows Communication Foundation* se utilizan para realizar un intercambio de mensajes de *SOAP* entre aplicaciones. Se incluyen tres tipos de contratos de operación en *Windows Communication Foundation,* estos son: el contrato de operación de *solicitud/respuesta*, *unidireccional* y *dúplex*.

2.4.1 OPERACIONES DE SOLICITUD/RESPUESTA (REQUEST/REPLY)

Un contrato de operación de *solicitud/respuesta* es aquel en el que una aplicación cliente recibe una respuesta relacionada con la petición. Éste es el Patrón de Intercambio de Mensaje (*MEP*[14]*)* predeterminado de los contratos de operaciones porque soporta tanto parámetros de entrada como de salida así como un tipo que se ha de retornar al llamador. En el ejemplo siguiente de código fuente en *C#*, se muestra un contrato de operación básico que toma recibe como parámetro un número entero de 32 bits y devuelve un valor booleano cuando se completó el procesamiento.

```
[OperationContract]
public bool Eliminar(int id);
```

A menos que especifique un tipo de contrato de operación diferente, incluso los contratos de operaciones que devuelven un tipo *void* se realiza una operación de *solicitud/respuesta*. A menos que una aplicación cliente invoque asíncronamente (a través de un *thread*) el llamado al contrato de operación, la aplicación detiene su procesamiento hasta ésta reciba el valor retornado, aunque el valor sea *void*. En el siguiente código fuente de *C#*, se muestra un contrato de operación que detiene la ejecución de una aplicación cliente hasta que ha recibido el valor *void* como respuesta desde el servidor.

```
[OperationContract]
public void Eliminar(int id);
```

Una de las desventajas del código anterior es puede disminuir el rendimiento de la aplicación cliente debido al tiempo que tarda en recibir la respuesta, La ventaja que se puede encontrar en los contratos de operación de *solicitud/respuesta* es que los errores de *SOAP* pueden retornarse en el mensaje de respuesta. Esta respuesta indica que se ha producido algún error durante la ejecución del servicio de *Windows Communication Foundation* ya sea en la comunicación o en el procesamiento. Los errores *SOAP* que se especifican en un contrato de operación se transfieren a la aplicación cliente a través de un objeto *FaultException*. Para obtener más información sobre errores de *SOAP*, consultar el siguiente capítulo de esta tesis.

[14] Un Patrón de Intercambio de Mensaje (*Message Exchange Pattern, MEP*) describe el patrón de mensajes requerido para que un protocolo de comunicaciones establezca o use una canal de comunicación.

2.4.2 OPERACIONES UNIDIRECCIONALES (ONE-WAY)

Cuando una aplicación cliente no debe esperar a que finalice la operación en el servicio de *Windows Communication Foundation* y no procesa errores *SOAP*, la operación puede especificar un contrato de operación unidireccional. Un contrato de operación unidireccional es aquel en el que una aplicación cliente invoca una operación y continúa su procesamiento después de que *Windows Communication Foundation* envía el mensaje en la red. Depende de la velocidad de la red, pero al menos que el mensaje que se está enviando sea extremadamente grande, la aplicación cliente sigue ejecutándose de manera casi inmediata a menos que se produzca una excepción local al enviar los datos. El intercambio de mensajes en el que se envía un mensaje y no se recibe ninguno (unidireccional) únicamente soporta un valor devuelto de *void*; si se declara un tipo diferente se arroja la excepción *InvalidOperationException*.
A diferencia de los contratos de operación de *solicitud/respuesta*, en los contratos de operación unidireccionales al no haber ningún mensaje de retorno, no se puede retornar ningún error de *SOAP* para indicar cualquier error en el procesamiento o la comunicación del servicio de *Windows Communication Foundation*. Para hacer uno de un contrato de operación unidireccional que debe devuelve un tipo *void*, se debe establecer la propiedad *IsOneWay* de la clase de atributo *OperationContractAttribute* en *true.* El siguiente código fuente en *C#* ejemplifica lo anterior:

```
[OperationContract(IsOneWay=true)]
public void Eliminar(int id);
```

Este contrato de operación es casi igual al ejemplo de *solicitud/respuesta*, la diferencia radica en la asignación de la propiedad *IsOneWay* en *true*. Esto quiere decir que, una vez que el contrato de operación haya terminado de procesarse no se enviará un mensaje de retorno al cliente, por lo tanto, una aplicación cliente continúa ejecutándose desde el momento en que el mensaje se entrega al canal de comunicación.

2.4.3 OPERACIONES DÚPLEX

Los contratos de operación dúplex se caracterizan porque tanto el servicio como del cliente pueden mandarse mensajes entre sí sin importar si se está usando un tipo de mensajería *unidireccional* o de *solicitud/respuesta*. Este tipo de comunicación dúplex o bidireccional se usa en los servicios que tienen que comunicarse directamente con el cliente, o para dar una condición asincrónica a cada uno de los involucrados en la transferencia del mensaje.
Los contratos de operación *dúplex* son más complejos que los contratos de operación *solicitud/respuesta* y *unidireccionales,* esto se debe al mecanismo extra que necesita para comunicarse con el cliente.
Además de crear un contrato de operación dúplex, también se debe crear un contrato de devolución de llamada (*callback*) y entonces asignar el tipo de ese contrato de devolución de llamada a la propiedad *CallbackContract* del atributo *ServiceContractAttribute* que marca al contrato de servicio (*interfaz*). Para implementar un modelo de comunicación dúplex, se crea una segunda interfaz que contenga las declaraciones de método a las que se llaman en el cliente.

Cuando un servicio de *Windows Communication Foundation* recibe un mensaje de comunicación dúplex, éste examina el elemento *ReplyTo* en ese mensaje entrante para determinar dónde se debe enviar la respuesta. Si el canal de comunicación que se utiliza para transferir el mensaje no se protege, un usuario que no es de confianza podría enviar un mensaje malintencionado a una computadora a través del elemento *ReplyTo*, provocando un ataque de denegación de servicio

(*Denial of Service, DoS*). Un *DoS* es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a usuarios legítimos, generalmente también provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales de su sistema.

2.4.4 PARÁMETROS DE SALIDA OUT Y REF

Cuando se programa haciendo uso de *Windows Communication Foundation* al igual que con otras aplicaciones de software de *.NET Framework*, se puede utilizar parámetros de entrada cuyo paso sea por valor o por referencia (palabras reservadas *ref* o *out* en *C#)*. Si se declara que el parámetro de un contrato de operación es *out* o *ref,* se indica que la modificación de los valores de los parámetros será devuelta por el contrato de operación. Un contrato de servicio como el siguiente contiene la declaración de un contrato de operación que aunque devuelve un tipo *void*, se requiere una operación de solicitud/respuesta debido el paso de parámetro por referencia.

```
[ServiceContractAttribute]
public interface IMiContrato {
    [OperationContract]
    public void LlenarDatos(ref TipoDatosPersonalizados datos);
}
```

Existen excepciones para el uso de estos parámetros, por ejemplo, únicamente se puede utilizar el enlace *NetMsmqBinding* para comunicarse con clientes si el contrato de operación devuelve un tipo *void*; no puede haber ningún valor de retorno, por lo tanto no se pueden usar parámetros de paso por referencia.

2.4.5 ESPECIFICAR EL NIVEL DE PROTECCIÓN DEL MENSAJE EN EL CONTRATO

Cuando se diseña un contrato se debe decidir el nivel de protección de los mensajes. Esto solo es necesario si el enlace que se va a usar para transferir la información tiene la seguridad desactivada (es decir, si se establece *System.ServiceModel.SecurityMode* en el valor *System.ServiceModel.SecurityMode.None*). Si no se tiene desactivada la seguridad, no se tiene que decidir sobre el nivel de protección de los mensajes del contrato. Salvo en el enlace *BasicHttpBinding*, los enlaces proporcionados por *.NET Framework*, otorgan un nivel de protección que hace innecesario el nivel de protección para cada operación o mensaje del servicio.
El nivel de protección especifica si los mensajes o partes de ellos están firmados, firmados y encriptados, o si se envían sin firmar o cifrar. El nivel de protección se puede establecer en varios lugares: para el contrato de servicio, para un contrato de operación, para un mensaje dentro de ese contrato de operación, o para una parte del mensaje. El valor que se haya establecido en un lugar se convierte en el valor predeterminado para los lugares menores a menos que se modifique explícitamente. Si un enlace (*binding*) no es capaz de ofrecer el nivel de protección mínimo necesario para el contrato, se arroja una excepción. Cuando no se establece explícitamente ningún valor de nivel de protección en el contrato, si así lo permite, el enlace controla de acuerdo a su configuración el nivel de protección para todos los mensajes. Este es el comportamiento predeterminado para asignar el nivel de protección.

Modificar el nivel de protección completo de *System.Net.Security.ProtectionLevel.EncryptAndSign* generalmente intercambia cierto grado de seguridad por un aumento del rendimiento. La decisión de modificar o no estos niveles depende de la importancia del valor de los datos que se intercambian y a la importancia de los contratos de operación involucrados.

En el siguiente código fuente en *C#* no se establece la propiedad *System.Servicemodel.Service-ContractAttribute.ProtectionLevel* o la propiedad equivalente *System.Servicemodel.Operation-ContractAttribute.Pro-tectionLevel* en el contrato.

```
[ServiceContract]
public interface ISinProteccionService {
    [OperationContract]
    public String ObtenerCadena();

    [OperationContract]
    public int ObtenerEntero();
}
```

Cuando se trabaja con una implementación como la anterior en un enlace como *WSHttpBinding* el modo de seguridad *(System.ServiceModel.SecurityMode)* predeterminado es el de mensaje (*Message*), todos los mensajes se encriptan y firman, dado que este enlace tiene un el nivel de protección *System.Net.Security.ProtectionLevel.EncryptAndSign* habilitado de forma predeterminada. Sin embargo, cuando el servicio se usa con un enlace como *BasicHttpBinding* el modo de seguridad (*SecurityMode*) predeterminado es ninguno (*None*), así que todos los mensajes se transfieren como texto. Ya que este enlace no provee seguridad de forma predeterminada, se omite el nivel de protección, es decir, los mensajes ni se encriptan ni se firman. Si se asigna el modo de seguridad (*SecurityMode*) a mensaje (*Message*) estos mensajes se encriptarían y se firmarían dado que ése sería el nivel de protección predeterminado del enlace (*System.Net.Security.Protection-Level.EncryptAndSign*). Cuando se desee explícitamente ajustar los niveles de protección de un contrato, se debe establecer la propiedad *ProtectionLevel* en el nivel que se requiera. Entonces, declarando un valor explícito de nivel de seguridad se exige que el enlace (si es que soporta ese valor de seguridad) provea la seguridad especificada. En el siguiente código fuente en *C#* se especifica explícitamente un valor *ProtectionLevel*, para el contrato de operación *ObtenerGuid*.

```
[ServiceContract]
public interface IConProteccionExplicitaService {
    [OperationContract]
    public string ObtenerCadena();
    [OperationContract(ProtectionLevel = ProtectionLevel.None)]
    public int ObtenerEntero();
    [OperationContract(ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    public int ObtenerGuid();// Globally Unique Identifier (GUID)
}
```

Cuando se trabaja con una implementación como la anterior en un enlace como *WSHttpBinding* el modo de seguridad *(System.ServiceModel.SecurityMode)* predeterminado es el de mensaje (*Message*), entonces, el comportamiento relativo a seguridad es el siguiente:

- ✓ Los mensajes del contrato de operación *ObtenerCadena* se encriptan y se firman.
- ✓ Los mensajes del contrato de operación *ObtenerEntero* se envían como texto sin encriptar ni firmar, es decir, se envían como texto sin formato.
- ✓ Los mensajes del contrato de operación *ObtenerGuid*, se encriptan y se firman como en el primer contrato de operación.

2.5 SESIONES, MANEJO DE INSTANCIAS Y CONCURRENCIA

Una sesión es la relación que involucra todos los mensajes transferidos entre un servicio y un cliente. El manejo de instancias se refiere al control del periodo de vida de los objetos del servicio definidos por el usuario y sus objetos *InstanceContext*. La concurrencia se refiere al control sobre

el número de subprocesos que se ejecutan simultáneamente en un objeto *InstanceContext*. En los siguientes apartados se explican cada uno de ellos.

2.5.1 SESIONES

Si la propiedad *System.ServiceModel.ServiceContractAttribute.SessionMode* de un servicio de *Windows Communication Foundation* se establece en *System.ServiceModel.SessionMode.Required*, quiere decir que todas las llamadas que se realicen a él deben formar parte de la misma conversación. Si se una sesión ha finalizado su tiempo de vida y una aplicación cliente transfiere un mensaje a través del canal de comunicación usando la sesión de la cual se dispusieron sus recursos se arrojará una excepción.

En *Windows Communication Foundation* las sesiones tienen las siguientes características principales:

- ✓ Las sesiones relacionan un conjunto de mensajes en una conversación establecida entre un servicio y un cliente. No existe un almacén de datos general asociado a una sesión *en Windows Communication Foundation*.
- ✓ Los mensajes transferidos durante una sesión se procesan como una estructura de datos que usa el método *FIFO* (se atienden en el orden en el que se recibieron)
- ✓ La aplicación cliente que hace la llamada inicia y finaliza explícitamente las sesiones.

Las diferencias entre sesiones de *ASP.NET* (la clase *System.Web.SessionState.HttpSessionState*) y *Windows Communication Foundation* que podemos encontrar son:

- ✓ Las sesiones *ASP.NET* siempre son iniciadas por servidor, en *Windows Communication Foundation* son iniciadas por el cliente
- ✓ Las sesiones *ASP.NET* están implícitamente desordenadas, en *Windows Communication Foundation* están ordenadas
- ✓ Las sesiones *ASP.NET* contienen un mecanismo de almacenamiento de datos general para todas las peticiones, en *Windows Communication Foundation* no existe un almacén de sesiones.

Las aplicaciones cliente inician sesiones, realizan una petición, reciben y procesan los mensajes enviados dentro de la sesión. Las aplicaciones de servicio pueden usar sesiones para agregar comportamiento adicional.

La propiedad *ServiceContractAttribute.SessionMode* se debe asignar a la enumeración *SessionMode* para exigir, permitir o prohibir que enlaces utilicen las sesiones entre los involucrados en la comunicación. Una sesión es una manera de poner en relación un conjunto de mensajes intercambiados entre dos o más extremos, es decir, entre un cliente y un servicio. La siguiente tabla muestra una descripción del los modos de sesión existentes en *Windows Communication Foundation:*

MODO DE SESIÓN	DESCRIPCIÓN
Allowed	Este modo de sesión indica que el contrato de servicio admite sesiones siempre y cuando el enlace también lo haga.
Required	Este modo de sesión indica que el contrato de servicio requiere un enlace con sesión. Si el enlace no acepta sesiones o no se configuró correctamente se produce una excepción.

NotAllowed	Este modo de sesión indica que el contrato de servicio no admite enlaces que están capacitados para iniciar sesiones.

Tabla 2.2 Miembros de SessionMode

2.5.2 CREACIÓN DE INSTANCIAS

La creación de instancias (propiedad *System.ServiceModel.ServiceBehaviorAttribute.InstanceContextMode*) maneja como un objeto *InstanceContext* se crea en respuesta a peticiones recibidas. La enumeración *InstanceContextMode* define los modos en que se pueden crear de instancias.

La siguiente tabla muestra los modos disponibles para la creación de instancias:

MODO DE CONTEXTO DE INSTANCIA	DESCRIPCIÓN
PerCall	Este modo de contexto de instancia indica que un nuevo objeto *InstanceContext* (y un objeto de servicio) se crea para cada solicitud de un cliente.
PerSession	Este modo de contexto de instancia indica que un nuevo objeto *InstanceContext* (y un objeto de servicio) se crea para cada nueva sesión del cliente y se mantiene durante el tiempo que dure esa sesión, este modo requiere de un enlace que use sesiones.
Single	Este modo de contexto de instancia indica que un objeto *InstanceContext* único (y un objeto de servicio) administra todas las solicitudes de clientes durante la vida de la aplicación.

Tabla 2.3 Miembros de InstanceContext

El siguiente código fuente en *C#* ejemplifica el valor predeterminado *PerSession* de *InstanceContextMode*, se establece de manera explícita en una clase de que implementa un contrato de servicio.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public class MiServicio : IServicio {
    // Contratos de operaciones
}
```

Si se utiliza el comportamiento de creación de instancias de un servicio de *Windows Communication Foundation* predeterminado (*PerSession* de *InstanceContextMode*) y un contrato de servicio admite el uso de sesiones, se pueden asignar uno o más contratos de operaciones del contrato de servicio estableciendo la propiedad *IsInitiating* en *true* para crear una instancia.

Los contratos de operación de inicio (*IsInitiating*) se deben llamar en primer lugar antes cualquier otro contrato de operación en una nueva sesión. Los contratos de operación que no son de inicio se pueden llamar después de que se haya hecho una llamada, por lo menos, a un contrato de operación de inicio. Se puede por tanto crear un constructor de sesión para un servicio de *Windows Communication Foundation* mediante contratos de operación de inicio diseñadas para dejar que un cliente (si presenta los datos adecuados) inicie una instancia de servicio. Esto último no es del todo cierto, pues el estado se asocia a la sesión y no el objeto de servicio pero para fines prácticos es adecuado. Los contratos de operación de inicio (en el comportamiento predeterminado de creación de instancias) se pueden llamar un número *n* de veces después de que se llame al primero y no se crean sesiones adicionales. Una vez que la sesión se establece, ésta se asocia a una instancia representada en tiempo de ejecución mediante un objeto *System.ServiceModel.InstanceContext*. Finalmente, el estado se asocia a la sesión y no al objeto de servicio.

2.5.3 INTERACCIÓN DE SESIONES CON LA CONFIGURACIÓN DE INSTANCECONTEXT

Las sesiones y la creación de instancias interactúan dependiendo de la combinación de los valores asignados a las propiedades que las controlan. La siguiente tabla muestra el resultado de un canal entrante (petición) de acuerdo a la combinación de los valores de las propiedades *System.ServiceModel.ServiceContractAttribute.SessionMode* y *System.ServiceModel.ServiceBehaviorAttribute.InstanceContextMode* del servicio.

SESSION MODE / INSTANCE CONTEXTMODE	Required	Allowed	NotAllowed
PerCall	✓ Comportamiento con canal con sesión: una sesión e *InstanceContext* para cada llamada. ✓ Comportamiento con canal sin sesión: se produce una excepción.	✓ Comportamiento con canal con sesión: una sesión e *InstanceContext* para cada llamada. ✓ Comportamiento con canal sin sesión: un *InstanceContext* para cada llamada.	✓ Comportamiento con canal con sesión: se produce una excepción. ✓ Comportamiento con canal sin sesión: un *InstanceContext* para cada llamada.
PerSession	✓ Comportamiento con canal con sesión: una sesión e *InstanceContext* para cada canal. ✓ Comportamiento con canal sin sesión: se produce una excepción.	✓ Comportamiento con canal con sesión: una sesión e *InstanceContext* para cada canal. ✓ Comportamiento con canal sin sesión: un *InstanceContext* para cada llamada.	✓ Comportamiento con canal con sesión: se produce una excepción. ✓ Comportamiento con canal sin sesión: un *InstanceContext* para cada llamada.
Single (Singleton)	✓ Comportamiento con canal con sesión: una sesión y un *InstanceContext* para todas las llamadas. ✓ Comportamiento con canal sin sesión: se produce una excepción.	✓ Comportamiento con canal con sesión: una sesión e *InstanceContext* para el singleton creado o especificado por el usuario. ✓ Comportamiento con canal sin sesión: un *InstanceContext* para el singleton creado o especificado por el usuario.	✓ Comportamiento con canal con sesión: se produce una excepción. ✓ Comportamiento con canal sin sesión: un *InstanceContext* para cada singleton creado o para el especificado por el usuario.

Tabla 2.4 Resultados de combinación de InstanceContextMode y SessionMode[15]

2.5.4 DESTRUCCIÓN DE INSTANCIAS

Si se usa la creación de instancias predeterminado (*PerSession* de *InstanceContextMode*) de *Windows Communication Foundation,* la misma instancia que se usó para crear el objeto cliente administra todas las llamadas a los contratos de operación del servicio. El modo de funcionamiento es similar al de una aplicación de *.NET Framework* que no sea enfocada a *Windows Communication Foundation,* por ejemplo, al hacer una instancia del objeto que refiere al servicio de *Windows Communication Foundation* en la aplicación cliente:

1. Se invoca al constructor del cliente de *Windows Communication Foundation*.
2. La instancia generada procesa todas las llamadas a los contratos de operación realizadas a la referencia de objeto de cliente de *Windows Communication Foundation*.
3. Se invoca a un destructor para destruir la referencia del objeto que se utilizó.

[15] Obtenido de: Sesiones, creación de instancias y concurrencia. http://msdn.microsoft.com/es-mx/library/ms731193.aspx

Si se utiliza el comportamiento de creación de instancias de un servicio de *Windows Communication Foundation* predeterminado (*PerSession* de *InstanceContextMode*) y un contrato de servicio admite el uso de sesiones, se pueden asignar uno o más contratos de operaciones del contrato de servicio estableciendo la propiedad *IsTerminating* en *true* para destruir la instancia.

Los contratos de operación de finalización (*IsTerminating*) a diferencia de los contratos de operación de inicio, se deben llamar como el último mensaje de una sesión existente. De forma predeterminada, *Windows Communication Foundation* recicla el objeto de servicio y su contexto después de que la aplicación cliente cierre la sesión a la que se asoció el servicio. Entonces, se puede crear un destructor declarando un contrato de operación de finalización que finalice de una manera adecuada la instancia de servicio.

Si bien el comportamiento de los contratos de operación de inicio y finalización es similar a los constructores y destructores de aplicaciones no enfocadas a *Windows Communication Foundation*, sólo es eso, una similitud. Cualquier contrato de operación de un servicio de *Windows Communication Foundation* puede ser una operación de inicio o finalización, incluso ambas al mismo tiempo.

El siguiente código fuente en C#, define al contrato de servicio *ICalculadora*. Se requiere que la instancia del cliente de *Windows Communication Foundation* llame primero a la operación *Iniciar* antes que a cualquier otra operación y que la sesión con este objeto de cliente de *Windows Communication Foundation* finalice cuando se llame al contrato de operación *Igual*.

```
[ServiceContract(Namespace = "http://www.ynnaf.com",
    SessionMode = SessionMode.Required)]
public interface ICalculadora {
    [OperationContract(IsOneWay = true, IsInitiating = true,
        IsTerminating = false)]
    void Iniciar();
    [OperationContract(IsOneWay = true, IsInitiating = false,
        IsTerminating = false)]
    void SumarAlResultado(double n);
    [OperationContract(IsOneWay = true, IsInitiating = false,
        IsTerminating = false)]
    void RestarAlResultado(double n);
    [OperationContract(IsOneWay = true, IsInitiating = false,
        IsTerminating = false)]
    void MultiplicarAlResultado(double n);
    [OperationContract(IsOneWay = true, IsInitiating = false,
        IsTerminating = false)]
    void DividirAlResultadoEntre(double n);
    [OperationContract(IsInitiating = false, IsTerminating = true)]
    double Igual();
}
```

Los servicios de *Windows Communication Foundation* no inician sesiones con las aplicaciones cliente. En las aplicaciones cliente de *Windows Communication Foundation*, hay una relación entre la duración del canal basado en sesión y la duración de la propia sesión. Entonces, los clientes crean nuevas sesiones creando nuevos canales basados en sesión y finalizan las sesiones existentes cerrando correctamente los canales basados en sesión. Un cliente generado con *.NET Framework* inicia una sesión con un extremo de servicio de *Windows Communication Foundation* llamando por medio de uno de los siguientes métodos:

- ✓ El método *System.ServiceModel.ClientBase.Open* en el objeto de cliente de *Windows Communication Foundation* para clientes generado en *.NET Framework* por la herramienta *Svcutil.exe*

- ✓ El método *System.ServiceModel.ICommunicationObject.Open* del canal retornado por una llamada al método *System.ServiceModel.ChannelFactory.CreateChannel*.
- ✓ Un contrato de operación de inicio (*IsInitiating*), de forma predeterminada, todos los contratos de operación son de inicio, al menos que se especifique explícitamente lo contrario. Cuando se llama al primer contrato de operación de inicio, el objeto del cliente de *Windows Communication Foundation* abre automáticamente el canal e inicia una sesión.

Una aplicación cliente construida con *.NET Framework* finaliza o destruye una sesión con el servicio llamando a alguno de los siguientes métodos:

- ✓ El método *System.ServiceModel.ClientBase.Close* en el objeto del cliente de *Windows Communication Foundation* generalmente generado con la herramienta *Svcutil.exe*.
- ✓ El método *System.ServiceModel.ICommunicationObject.Close* del canal retornado por una llamada al método *System.ServiceModel.ChannelFactory.CreateChannel*.
- ✓ Un contrato de operación de finalización, ninguna operación de forma predeterminada es de finalización; se debe especificar explícitamente en el contrato una operación a través de la propiedad *IsTerminating* asignándole el valor *true*.

2.5.5 CONCURRENCIA

Al control del número de subprocesos activos en *InstanceContext* durante cualquier momento se le llama concurrencia. La concurrencia puede controlarse en un servicio de *Windows Communication Foundation* mediante la propiedad *System.ServiceModel.ServiceBehaviorAttribute.ConcurrencyMode* por medio de la enumeración *ConcurrencyMode.*

Windows Communication Foundation suministra tres formas de concurrencia, éstas son:

- ✓ Single: En un contexto de instancia se puede tener un número determinado de subprocesos procesando mensajes. Los subprocesos que quieran hacer uso del mismo contexto de instancia se deben bloquear hasta que un subproceso salga del contexto de la instancia.
- ✓ Multiple: Cada instancia del servicio puede tener varios subprocesos procesando mensajes al mismo tiempo. El servicio debe ser seguro para los subprocesos al utilizar este modo de concurrencia.
- ✓ Reentrant: Cada instancia del servicio procesa un mensaje a la vez, sin embargo, acepta llamadas de operación reentrantes. El servicio sólo acepta estas llamadas cuando está llamando al exterior a través de un objeto de cliente *Windows Communication Foundation*.

La concurrencia tiene que ver con el modo en que se crean instancias. Si la creación de instancias está asignada con *PerCall*, la concurrencia es trivial dado que un nuevo *InstanceContext* se encarga de procesar cada mensaje, por lo tanto, únicamente hay un subproceso activo en cada *InstanceContext*.

El siguiente código fuente escrito en *C#* muestra cómo establecer la propiedad *ConcurrencyMode* en *Multiple* con un modo de contexto de instancia (*InstanceContextMode*) en *Single*.

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Multiple,
    InstanceContextMode = InstanceContextMode.Single)]
public class MiServicio : IMiServicio {
    // Firmas de contratos de operaciones
}
```

CAPÍTULO 3.
CARACTERÍSTICAS AVANZADAS DE WINDOWS COMMUNICATION FOUNDATION

3.1 EXCEPCIONES Y ERRORES

En las aplicaciones de software, las excepciones se usan para informar sobre errores que ocurren de forma local. Al igual que en las aplicaciones de escritorio o aplicaciones web creadas para *.NET Framework*, un servicio o cliente *Windows Communication Foundation* puede manejar excepciones de manera eficaz. Los errores en los servicios *Windows Communication Foundation* se utilizan para informar errores desde el servicio al cliente o del cliente al servicio. A parte de los errores que se transfieren vía mensajes, los canales de transporte usan frecuentemente mecanismos específicos para informar los errores producidos en el nivel de transporte. En un ambiente externo a *Windows Communication Foundation,* como el *HyperText Transfer Protocol (HTTP)* se utilizan códigos de respuesta (*400~417*) para informar un error producido en el cliente, por ejemplo: el error *403* indica que el usuario tiene prohibido el acceso al sitio web o el error *404* indica que la dirección introducida por el usuario no se encontró.

Cuando un servicio *Windows Communication Foundation* necesita lanzar una excepción al cliente, el programador debe considerar dos puntos importantes: lo primero es crear una excepción que implemente un tipo que pueda ser tratado de una manera adecuada por el cliente, lo segundo es ofrecer al cliente la información que indique: qué provocó el problema, qué consecuencia tuvo en la aplicación y cómo solucionarlo.

A diferencia de la mayoría de las aplicaciones (de escritorio y web) que se pueden crear en *.NET Framework*, y que están representados mediante la clase *System.Exception,* los servicios de *Windows Communication Foundation* informan sobre errores por medio de mensajes de error de *SOAP*. Los mensajes de error de *SOAP* son tipos que se incluyen en los metadatos de un ensamblado de un servicio, por lo tanto se crea un contrato de error que los clientes pueden usar para que la operación sea óptima. Dado que los errores de *SOAP* se expresan en formato *XML*, se genera una interoperabilidad entre distintos clientes que se ejecuten en cualquier plataforma, aumentando así el alcance del servicio *Windows Communication Foundation*.

A pesar que los servicios *Windows Communication Foundation* son capaces de manejar de forma interna excepciones de *.NET Framework*, cualquier información obtenida por una excepción debe enviarse al cliente convertida en un error *SOAP* en el servicio, una vez que el cliente recibe el error *SOAP,* éste debe convertirse a una excepción que sea manejada por él. Es posible usar excepciones predeterminadas o personalizadas para asignarlas a los mensajes de error *SOAP*. Entonces, es posible enviar dos tipos de errores a un cliente que hace uso del servicio: los declarados y los no declarados. Los *errores SOAP declarados* contienen el atributo *System.ServiceModel.FaultContractAttribute* en el contrato de operación, se especifica un tipo de error *SOAP* personalizado. Los errores *SOAP no declarados*, no se especifican explícitamente en el contrato de operación de un servicio *Windows Communication Foundation*.

Existen dos recomendaciones para el manejo de errores *SOAP* en un servicio de *Windows Communication Foundation*, la primera es definir el atributo *System.ServiceModel.FaultContractAttribute* en los contratos de operación que lo requieran para que los clientes del servicio sepan de forma correcta que respuesta pueden esperar y la segunda es devolver únicamente una excepción por

contrato de operación para evitar un ataque malicioso debido a la información extra sobre diferentes fallos.

3.1.1 ASIGNACIÓN DE EXCEPCIONES A ERRORES SOAP

La creación de un contrato de operación que arroja errores *SOAP* debe seguir varios pasos entre las que se incluyen: decidir en qué condiciones un servicio *Windows Communication Foundation* debería informar sobre errores, una vez que se realizó lo anterior se puede construir un error *SOAP* personalizado colocando el atributo *System.ServiceModel.FaultContractAttribute* en el contrato de operación que puede arrojar la excepción. Realizar los pasos anteriores garantiza la correcta administración de errores en las aplicaciones de *Windows Communication Foundation*. Una vez que se diseñó correctamente el mensaje de error, el siguiente paso es implementar el envío y la recepción de estos errores *SOAP*.

3.1.2 TIPOS DE EXCEPCIÓN

Las excepciones lanzadas por canales de comunicación son: *System.TimeoutException*, *System.ServiceMo-del.CommunicationException* u otra excepción derivada de la clase *CommunicationException*. Se pueden generar excepciones del tipo *ObjectDisposedException*, sin embargo, únicamente indican que el código de llamada utilizó de forma incorrecta el canal de comunicaciones. Si un canal es usado de forma correcta, sólo deben lanzar las excepciones mostradas en la tabla anterior. *Windows Communication Foundation* proporciona siete excepciones derivadas de la clase *CommunicationException* las cuales son utilizadas por el canal de comunicación. Estas excepciones se muestran en la siguiente tabla:

TIPO DE EXCEPCIÓN	SIGNIFICADO	CONTENIDO DE EXCEPCIÓN INTERNO	ESTRATEGIA DE LA RECUPERACIÓN
AddressAlreadyInUseException	La dirección del extremo especificada para realizar escuchas ya se está utilizando.	Si está presente, proporciona más detalles sobre el error de transporte que produjo esta excepción. Por ejemplo. *PipeException*, *SocketException ó HttpListenerException*.	Probar una dirección diferente.
AddressAccessDeniedException	El proceso no tiene acceso a la dirección del extremo especificada para realizar escuchas.	Si está presente, proporciona más detalles sobre el error de transporte que produjo esta excepción. Por ejemplo, *HttpListenerException* o *PipeException*.	Intentar con credenciales diferentes.
CommunicationObjectFaultedException	Normalmente se produce esta excepción porque una aplicación pasa por alto alguna excepción e intenta utilizar un objeto que ya ha fallado, posiblemente en un subproceso diferente al que detectó la excepción original.	Si está presente, proporciona detalles sobre la excepción interna.	Crear un nuevo objeto.
CommunicationObjectAbortedException	De forma similar a *CommunicationObjectFaultedException*, esta excepción indica que la aplicación ha llamado a *Abort* en el objeto, posiblemente desde otro subproceso, y el objeto ya no se puede utilizar por esa razón.	Si está presente, proporciona detalles sobre la excepción interna.	Crear un nuevo objeto.

Endpoint-NotFound-Exception	El extremo remoto de destino no está realizando escuchas. Esto puede ser el resultado de que cualquier parte de la dirección de extremo sea incorrecta, irresoluble, o que el extremo esté caído. Los ejemplos incluyen error de *DNS*, Administrador de la cola no disponible y servicio no ejecutándose.	La excepción interna proporciona detalles, normalmente del transporte subyacente.	Probar una dirección diferente. Alternativamente, el remitente puede esperar un poco e intentarlo de nuevo en caso de que el servicio estuviera caído
Protocol-Exception	Los protocolos de comunicaciones no coinciden entre los extremos. Por ejemplo, el tipo de contenido de marco no coincide o se ha excedido el tamaño máximo del mensaje.	Si está presente, proporciona más información sobre el error de protocolo concreto.	Asegurarse de que la configuración de protocolo del remitente y el destinatario coincidan.
ServerTooBusy-Exception	El extremo remoto está realizando escuchas pero no está preparado para procesar los mensajes.	Si está presente, la excepción interna proporciona el error de *SOAP* o detalles del error en el nivel de transporte.	Esperar y reintentar después la operación.
Timeout-Exception	La operación no se completó dentro del período de tiempo de espera.	Puede proporcionar los detalles sobre el tiempo de espera.	Esperar y reintentar después la operación.

Tabla 3.1 Tipos de excepciones predeterminadas de WCF[16]

Si es necesario tener una estrategia personalizada de recuperación de tipos de excepciones se deberá definir un nuevo tipo de excepción, la cual se debe derivar de la clase *CommunicationException* o de alguna de sus clases derivadas.

3.1.3 MENSAJES DE EXCEPCIÓN

Los mensajes de excepción van enfocados a los usuarios, estos deben proporcionar información capaz de ayudarlo a comprender y solucionar el problema que se haya presentado en un servicio de *Windows Communication Foundation*. Existen tres partes importantes en un mensaje de excepción, éstas son:

1. *Qué sucedió*. Se debe indicar de forma clara el problema. Un mensaje de excepción incorrecto es: "El valor de la etiqueta *XML* no es válido" ya que no se precisa el lugar específico donde se provocó el error. Un mensaje mejorado pero no óptimo es: "El valor de la etiqueta <*security*> no es válido". Un mensaje de excepción óptimo es: "El valor de la propiedad *mode* de la etiqueta <*security*> no es válido". Este es un mensaje más eficaz que los anteriores porque da información más precisa del error al usuario y permite que éste sea solucionado más rápidamente. Mensajes como el anterior debería ser usado como mínimo en mensajes de excepción usados en servicios de *Windows Communication Foundation.*
2. *La importancia del error*. Para mejorar el mensaje de excepción generado en el punto anterior se debería especificar que tan importante es el error, entonces, un mensaje que indique la importancia del error sería: "El valor de la propiedad *mode* de la etiqueta <*security*> no es válido, si no introduce un valor adecuado no se protegerá su mensaje".
3. *Cómo el usuario debería corregir el problema*. El mensaje de excepción debe contener información que ayude al usuario a solucionar el problema. El mensaje de excepción final sería el siguiente: "El valor de la propiedad *mode* de la etiqueta <*security*> no es válido, si no introduce un valor adecuado no se protegerá su mensaje. Para solucionar el problema

[16] Obtenido de: Administración de excepciones y errores. http://msdn.microsoft.com/es-es/library/ms789039.aspx

introduzca un valor correcto y vuelva a intentarlo". Un mensaje de excepción como el anterior debería ser usado mayoritariamente en los servicios de *Windows Communication Foundation.*

3.1.4 FAULTEXCEPTION

Todo contrato de operación que sea del tipo *petición/respuesta* retorna al llamador dos posibles mensajes: un mensaje con respuesta normal (si se ejecutó correctamente la operación), o un mensaje de error (si se produjo una excepción). Para poder enviar un mensaje de error (un *SOAP Fault* en términos de servicios web) se lanza una excepción *FaultException* cuando se implementa el servicio de *Windows Communication Foundation*. La clase *FaultException* genera un error sin tipo que es retornado al llamador con fines de depuración. Puesto que la clase *FaultException* hereda de la clase *CommunicationException*, se debe detectar (*catch*) una excepción *FaultException* antes de detectar una excepción *CommunicationException*.

Es recomendable utilizar el atributo *FaultContractAttribute* en los contratos de operación de los servicios de *Windows Communication Foundation* para retornar errores de *SOAP* en los casos en que el cliente necesita información sobre el error para mantener su aplicación. Se deben lanzar los objetos *FaultException* cuando se provocó un error y entonces el cliente debe recuperar la información proporcionada en la excepción.

FaultContractAttribute es un atributo que define uno o varios errores de *SOAP* que son retornados al llamador cuando se generan errores en ejecución en un contrato de operación de un servicio de *Windows Communication Foundation*. Se debe declarar en un contrato de operación el atributo *FaultContractAttribute* para emitir una o varias excepciones específicas que se incluirán en la descripción del Lenguaje de descripción de servicios web (*WSDL*) correspondiente al contrato de operación del servicio de *Windows Communication Foundation* como mensajes de error de *SOAP* que pueden ser retornados por la operación. En aplicaciones basadas en *SOAP* como los servicios de *Windows Communication Foundation (WCF)*, los contratos de operación del servicio de *Windows Communication Foundation* ofrecen la información del error de procesamiento mediante mensajes de error de *SOAP*.

Como se mencionó anteriormente, es recomendable que se declare explícitamente la clase de atributo *FaultContractAttribute* para especificar adecuadamente los errores de *SOAP* que un cliente que hace uso del servicio de *Windows Communication Foundation* puede obtener en el curso normal de un contrato de operación. Los miembros de una clase de excepción *FaultException* son:

- ✓ La propiedad *Action:* permite controlar la acción que realizará el mensaje de error.
- ✓ La propiedad *DetailType:* esta propiedad obtiene el tipo del objeto serializado que detalla el mensaje de error.
- ✓ Las propiedades *Name* y *Namespace:* Estas propiedades permiten controlar el nombre y el espacio de nombres, respectivamente del mensaje de error.
- ✓ La propiedad *HasProtectionLevel:* Permite conocer si un mensaje de error contiene un nivel de protección especificado, si este es el caso, la propiedad *ProtectionLevel* lo controlará. Si se envía un mensaje de error que contiene información confidencial o puede desencadenar problemas de seguridad, se debe asignar la propiedad *ProtectionLevel*.
- ➢ Si se establece explícitamente la propiedad *ProtectionLevel.Sign.ProtectionLevel* o *ProtectionLevel.Encrypt-AndSign.ProtectionLevel*, se deberá hacer uso de un enlace con seguri-

dad habilitado por medio de la propiedad *System.ServiceModel.SecurityMode* o se generará una excepción.

- Si se hace uso de un enlace que hace uso de la seguridad de forma predeterminada, aunque no se establezca la propiedad *ProtectionLevel* en algún nivel de protección, se codificarán y firmarán obligatoriamente todos los datos de la aplicación.
- Si se usa un enlace que no tiene la seguridad habilitada de forma predeterminada por ejemplo el enlace: *System.ServiceModel.BasicHttpBinding* y no se establece la propiedad *ProtectionLevel* explícitamente, no se protegerá ninguno de los datos que se transfieren.

Asignar la propiedad *ProtectionLevel* en *EncryptAndSign* para los mensajes del error en muchos de los casos es suficiente para obtener seguridad en la transferencia de la información. Si se desea retornar un error específico desde una operación marcada con *FaultContractAttribute*, se debe lanzar una excepción de tipo *FaultException<TDetail>* (donde el parámetro de tipo es la información del error que debe ser serializable) cuando excepción se genera durante la ejecución de un contrato de operación. Los programas cliente que hacen uso de los servicios de *Windows Communication Foundation* deben tratar el error de *SOAP* como el mismo tipo declarado en el cliente; es decir, como *FaultException<TDetail>*. *FaultContractAttribute* sólo se usa para especificar los errores de *SOAP* para contratos de operación de *petición/respuesta* (bidireccionales), los contratos de operación unidireccionales no manejan errores de *SOAP*, como consecuencia no se les puede definir un atributo del tipo *FaultContractAttribute*.

Para transportar la información del error usando el atributo *FaultContractAttribute* se debe utilizar cualquier parámetro de tipo que sea serializable a través del atributo *System.Runtime.Serialization.DataContract-Serializer* o por cualquier otro que permita serializar objetos. Por ejemplo, si se desea especificar que los clientes de un servicio de *Windows Communication Foundation* pueden esperar un error de *SOAP* que contiene una estructura *Int32 (número entero de 32 bits)*, se debe colocar ese parámetro de tipo en la clase de atributo *FaultContractAttribute* en el contrato de operación del servicio. El código siguiente muestra en *C#* como definir que un contrato de operación puede arrojar una excepción hacia un cliente:

```
[ServiceContract]
public interface IMiServicio {
    [OperationContract]
    [FaultContract(typeof(int))]
    int MiMetodo(int n1, int n2);
}
```

En la implementación del contrato de operación del servicio de *Windows Communication Foundation*, se puede arrojar una excepción *FaultException<TDetail>* donde el parámetro de tipo es el tipo que contiene la información del error (en este caso anterior el tipo es un *Int32*). Por ejemplo:

```
throw new FaultException<int>(4);
```

El ejemplo anterior muestra como un código de error puede ser enviado en una estructura *Int32*, sin embargo este tipo de error no es muy eficiente ya que no muestra información suficiente sobre el error que se produjo en el servicio de *Windows Communication Foundation*. Como se ha estado mencionando en esta tesis, los servicios de *Windows Communication Foundation* especifican detalladamente la información de los errores de *SOAP* al cliente. El siguiente código fuente escrito en *C#* ejemplifica el uso de la clase de atributo *FaultContractAttribute* para especificar que el contrato de operación *ValidarLongitudNombre(String nombre)* puede devolver un error de *SOAP*

del tipo *ValidarNombreFault* si la longitud del nombre introducido como parámetro es mayor a 7 caracteres, los pasos a para definir lo anterior son:

1. Definir la clase que fungirá como excepción personalizada:

```
[DataContract]
public class ValidarNombreFault {
    public ValidarNombreFault(String mensaje) {
        this.Mensaje = mensaje;
    }

    [DataMember]
    public string Mensaje { get; set; } // C# 3.0
    }
}
```

2. Definir la interfaz que fungirá como contrato de servicio:

```
 [ServiceContract(Namespace = "http://www.ynnaf.com")]
public interface IValidarNombre {
    [OperationContract]
    [FaultContract(
      typeof(ValidarNombreFault),
      Action = "http://www.ynnaf.com/ValidarNombreFault",
      ProtectionLevel = ProtectionLevel.EncryptAndSign)]
    String ValidarLongitudNombre(String nombre);
}
```

3. Implementar el contrato de servicio y arrojar la *FaultException* cuando la condición no se cumpla:

```
public class ValidarNombreService : IValidarNombre {
    public String ValidarLongitudNombre(String nombre) {
        Console.WriteLine("El cliente introdujo: " + nombre);
        // Si el nombre es mayor a 7 caracteres arrojar la excepción
        if (nombre.Length <= 7)
            return "El servicio responde, la longitud de: " + nombre +
                " es correcta";
        else
            throw new FaultException<ValidarNombreFault>(
                new ValidarNombreFault("Un error ocurrió. El nombre: " +
                    nombre + "es mayor a 7 caracteres"));
    }
}
```

El siguiente programa escrito en *C#* ejemplifica como una aplicación de consola de *.NET Framework* (que sirve como aplicación cliente) trata las errores de *SOAP* que se arrojan desde el servicio de *Windows Communication Foundation.* Las excepciones se tratan desde la más específica hasta la más general. Una excepción del tipo *FaultException<ValidarNombreFault>* se produce si la longitud del nombre pasado al parámetro es mayor a 7 caracteres:

```
public class ClienteWCF {
    public static void Main(String[] args) {
        ValidarNombreServiceClient wcfCliente = new ValidarNombreServiceClient();
        try {
            // Llamar al método en el servicio.
            Console.WriteLine("Introduce el nombre a validar: ");
            String nombre = Console.ReadLine();
            Console.WriteLine("Respuesta: " +
                wcfClient.ValidarLongitudNombre(nombre));
            Console.WriteLine("Presiona ENTER para salir...");
            Console.ReadKey();
            // Hecho.
```

```
            wcfCliente.Close();
            Console.WriteLine("Finalizado.");
        } catch (TimeoutException problemaTiempo) {
            Console.WriteLine("El tiempo de operación del servicio expiró: " +
                problemaTiempo.Message);
            Console. ReadKey();
            wcfCliente.Abort();
        } catch (FaultException<ValidarNombreFault> validarNombreFault) {
            Console.WriteLine(validarNombreFault.Detail.Mensaje);
            Console. ReadKey();
            wcfCliente.Abort();
        } catch (FaultException errorDesconocido) {
            Console.WriteLine("Se recibió un error desconocido. " +
                errorDesconocido.Message);
            Console. ReadKey();
            wcfCliente.Abort();
        } catch (CommunicationException problemaComunicacion) {
            Console.WriteLine("Ocurrió un problema de comunicación. " +
                problemaComunicacion.Message + problemaComunicacion.StackTrace);
            Console. ReadKey();
            wcfCliente.Abort();
        }
    }
}
```

3.2 SEGURIDAD

La seguridad de *Windows Communication Foundation (WCF)* se basa en estructuras de seguridad propias, estructuras de seguridad existentes y en las especificaciones establecidas para los mensajes SOAP. Puesto que *Windows Communication Foundation* es un conjunto de Interfaces de Programación de Aplicaciones que sirven para construir aplicaciones distribuidas, la seguridad en la capa de comunicación entre servicios y clientes es muy importante. *Windows Communication Foundation* proporciona diferentes mecanismos para construir aplicaciones distribuidas seguras, entre estos se encuentran: *Secure Hypertext Transfer Protocol (HTTPS)*, autenticación vía nombre de usuario (*username*) y contraseña (*password*), autenticación vía certificados *X.509*, entre otros. Una de las ventajas al usar *SOAP* como protocolo de comunicación es que todos los mecanismos de seguridad se hacen interoperables. Los mecanismos mencionados anteriormente se transfieren vía *SOAP* y con interoperables ya que haciendo uso de *XML*. Así, de esta forma los mensajes se transfieren se vía este protocolo toman ventaja de las especificaciones de *XML* como firmas digitales y codificación. Los mensajes que maneja *Windows Communication Foundation* contribuyen a confiar en la información que se transfiere en aspectos comerciales.

3.2.1 VENTAJAS DE LA SEGURIDAD DE WINDOWS COMMUNICATION FOUNDATION

.NET Framework permite crear dos tipos de aplicaciones que funcionan con *Windows Communication Foundation:* aplicaciones que funcionan como servicio o como cliente. Los mensajes que se transfieren entre aplicaciones de servicio y cliente lo hacen a través Internet, pasando por numerosos firewalls y una gran cantidad de intermediarios. La transferencia de mensajes a través de los medios anteriores introduce una gran cantidad de amenazas en la seguridad. A continuación se listan una serie amenazas que *Windows Communication Foundation* ayuda a combatir:

- ✓ *Observación del tráfico de red para obtener información confidencial*. Si un cliente de un banco pide transferir fondos de una cuenta a otra, y el mensaje que se transfiere de un punto a otros no está protegido, un usuario malintencionado puede interceptar el mensaje y obtener la información confidencial del cliente. Entonces como éste tiene el número de cuenta y la contraseña del cliente, puede realizar una transferencia de fondos desde la cuenta que se expuso al peligro.

- ✓ *Entidades deshonestas que actúan como servicios sin conocimiento del cliente*. A este tipo de ataque se le conoce como ataque de suplantación de identidad o *phishing*, la forma de en la que funciona este ataque es el siguiente: un usuario malintencionado actúa como un servicio legal que obtiene información confidencial de clientes que confían en él de forma equívoca. El usuario deshonesto usa la información obtenida de los clientes engañados para transferir fondos desde las cuentas que se expusieron al peligro.
- ✓ *Modificación de los mensajes transferidos para lograr un resultado diferente al que generó el autor de la llamada*. El funcionamiento de este ataque es el de modificar el número de cuenta que da un cliente para realizar un depósito monetario, esto provoca el destino final del depósito sea la cuenta del usuario que ideó el ataque al cliente.
- ✓ *Reproducción de la misma orden de compra*. Este ataque consiste en que un usuario malintencionado reproduzca una orden de compra *n* veces y hace que la página de ventas en línea envíe todas esas órdenes de compra al cliente que no los ha pedido más que una sola vez.
- ✓ *Incapacidad de un servicio para autenticar un cliente*. El problema que se presenta en este casi es que el servicio no puede garantizar que el cliente verdadero haya realizado la transacción.

Windows Communication Foundation reduce el riesgo que cualquiera de los cinco puntos anteriores ocurra debido principalmente a las opciones de encriptación de mensajes que posee y a que solamente los dos puntos involucrados en la transferencia pueden encriptar y desencriptar los mensajes.

La seguridad de transferencia que brinda *Windows Communication Foundation* ofrece las siguientes soluciones para los problemas expuestos arriba:

- ✓ Confidencialidad de la información del mensaje. Esta solución es aplicable para el problema 1.
- ✓ Autenticación del punto final de servicio (el que genera la respuesta a la petición de un cliente). Esta solución es aplicable para el problema 2.
- ✓ Integridad de la información contenida en el mensaje. Esta solución es aplicable para el problema 3.
- ✓ Detección de la reproducción de mensajes. Esta solución es aplicable para el problema 4.
- ✓ Autenticación del cliente principal (el que genera la petición al servicio). Esta solución es aplicable para el problema 5

Los servicios web pueden exponerse a diferentes tipos de clientes como: clientes internos, clientes externos o clientes desde Internet. Para garantizar la seguridad, *Windows Communication Foundation* proporciona una integración con modelos de autenticación de seguridad existentes. La seguridad que otorga *Windows Communication Foundation* trabaja sobre una gran variedad credenciales y modelos de autenticación, entre estos se encuentran:

- ✓ Credencial de cliente de certificado. Por ejemplo, certificados *X.509.*
- ✓ Credencial de cliente de nombre de usuario (*username*) y contraseña (*password*).
- ✓ Credenciales de Windows (usando el protocolo *Kerberos* y *NT LanMan (NTLM)*).
- ✓ Autor de llamada anónimo.

3.2.2 NORMAS E INTEROPERABILIDAD

Actualmente existen grandes sistemas construidos en diferentes lenguajes de programación ejecutándose sobre distintas plataformas, muchas veces incompatibles entre ellos. Para que las aplicaciones distribuidas sean funcionales, deben ser interoperables entre diferentes proveedores y la seguridad también lo debe ser. *Windows Communication Foundation* admite una gran variedad de normas de seguridad interoperable, entre ellas se encuentran: *WS-Security*, *WS-Trust*, *WS-SecureConversation*, y *WS-SecurityPolicy*.
Para el perfil de seguridad básico (*Basic Security Profile*), la clase *System.ServiceModel.BasicHttpBinding* está destinada poder establecer enlaces para este perfil; la clase *System.ServiceModel.WSHttpBinding* (que se utilizará en el caso práctico del capítulo 4 de esta tesis) se usa para establecer enlaces con las normas de seguridad más recientes, todas ellas contenidas en las especificaciones *WS-** (en específico *WS-Security 1.1* y *WS-SecureConversation*). Basándose en estas normas, la seguridad proporcionada por *Windows Communication Foundation* puede ser interoperable con servicios web que se ejecutan en sistemas operativos distintos a *Microsoft Windows* como *Linux, MacOS, Solaris,* etc.

3.2.3 SEGURIDAD DISTRIBUIDA DE APLICACIONES

Windows Communication Foundation (WCF) divide en tres áreas funcionales la seguridad, estas son: la *seguridad de la transferencia*, el *control de acceso* y la *auditoría*. En los siguientes apartados se describe de forma breve cada una de estas áreas.

3.2.3.1 SEGURIDAD DE LA TRANSFERENCIA

La seguridad de la transferencia abarca tres funciones de seguridad principales: integridad, confidencialidad y autenticación. Estas funciones garantizan sean enviados y recibidos de una forma segura desde el origen hasta el destino La siguiente tabla describe las tres funciones que constituyen la seguridad de transferencia:

FUNCIÓN	DESCRIPCIÓN
Integridad	La integridad es la garantía de detectar si un mensaje se transfirió de forma completa y precisa, esto debe realizarse después de haberse transferido de un punto a otro. La integridad se usa para evitar la alteración del mensaje ya que el camino de un punto a otro, éste debió ser leído por muchos actores. Normalmente se logra mediante realizando una firma digital sobre un mensaje.
Confidencialidad	La confidencialidad es la garantía mantener un mensaje ilegible para nadie que no sea el destinatario. Este mecanismo se realiza a menudo codificando datos utilizando una clave pública y una clave privada. Si un cliente debe enviar su número de tarjeta de crédito a un sitio de Internet, este debe enviarse de forma confidencial de Internet.
Autenticación	La autenticación es la capacidad de comprobar la identidad de un usuario. Dado que el propietario de una cuenta bancaria debería ser el único que pueda retirar fondos. Un mecanismo que permite otorgar una identidad única a un usuario único es asignarle un nombre de usuario y contraseña. Otro mecanismo consiste en hacer uso de un *certificado X.509* proporcionado por un tercero.

Tabla 3.2 Funciones de la seguridad de transferencia

Escenarios de seguridad de transferencia
Algunos de los escenarios en los que se utiliza la seguridad de transferencia de *Windows Communication Foundation* son los siguientes:

- ✓ Transferencia haciendo uso de un nombre de usuario y contraseña y *Secure Hypertext Transfer Protocol (HTTPS)*. Las aplicaciones que usan *Windows Communication Foundation* (ya sea como servicio o cliente) se desarrollan para trabajar en Internet. Las credenciales que presentan los clientes se comparan con los datos contenidos en una base de datos que contienen nombres de usuario y contraseñas. El servicio de de servicio y cliente se ejecuta en una dirección *Secure Hypertext Transfer Protocol* haciendo uso de un certificado *Secure Sockets Layer (SSL)* de confianza.
- ✓ Transferencia haciendo uso de certificados. Como se dijo en el punto anterior Las aplicaciones que usan *Windows Communication Foundation* (ya sea como servicio o cliente) se desarrollan para trabajar en Internet. Tanto el cliente como el servicio poseen certificados *X.509*, estos se pueden utilizar para proteger los mensajes (encriptándolos). Una de las ventajas de usar estos certificados es realizar transacciones importantes que requieren integridad y confidencialidad del mensaje además de permitir la autenticación mutua.

3.2.3.1.1 MODOS DE SEGURIDAD DE TRANSPORTE Y DE MENSAJE

En *Windows Communication Foundation* se usan dos mecanismos para hacer uso de la seguridad de transferencia, estos son: el modo de seguridad de *transporte* y el modo de seguridad de *mensaje*.

- ✓ *Modo de seguridad de transporte:* Este modo usa un protocolo de nivel de transporte, como *Secure Hypertext Transfer Protocol (HTTPS)*, para obtener la seguridad de transferencia. Las ventajas del modo de transporte son las de estar ampliamente adoptado, estar disponible en una gran cantidad de plataformas y ser menos complejo desde el punto de vista computacional. La desventaja que posee es que sólo protege mensajes de punto a punto.
- ✓ *Modo de seguridad de mensaje*. Este modo hace uso de la especificación *WS-Security* y otras especificaciones para obtener la seguridad de transferencia. A diferencia del modo de seguridad de transporte, el modo de seguridad de mensaje se hace directamente en los mensajes *SOAP,* está incluida en envolturas *SOAP* junto con los datos de la aplicación. Este modo de seguridad posee las ventajas de ser independiente del protocolo de transporte, más extensible y de garantizar la seguridad global a diferencia del modo de seguridad de transporte que solo hace frente al protocolo punto a punto. La desventaja que se puede encontrar es que el rendimiento es menor que el modo de seguridad de *transporte* debido a que se debe hacer uso de *XML* en los mensajes *SOAP*.

Modos de seguridad

En la siguiente tabla se describen los diferentes modos de seguridad de transferencia que posee *Windows Communication Foundation*:

MODO	DESCRIPCIÓN
Ninguno	Este modo no proporciona seguridad ni en el nivel de transporte ni en el nivel del mensaje. Dentro de los enlaces que maneja *Windows Communication Foundation* únicamente el *BasicHttpBinding* tiene este modo habilitado de forma predeterminada.
Transporte	Este modo utiliza un protocolo de transporte seguro como *Secure Hypertext Transfer Protocol (HTTPS)* para transferir la información de un punto a otro dentro de Internet.
Mensaje	Este modo implementa la seguridad dentro del mensaje *SOAP* (basado en los estándares de *WS-Security*) para transferir mensajes de un punto a otro dentro de Internet.
Modo mixto	En este modo de seguridad se utiliza la seguridad de transporte en el lado del servidor y la seguridad de mensajes para el lado del cliente.

Ambos	Este modo utiliza el modo de seguridad de transporte y de mensaje tanto del lado del cliente como del servidor. Únicamente está disponible en el enlace *NetMsmqBinding.*

Tabla 3.3 Modos de seguridad de transferencia

Seguridad de transferencia y credenciales

Un conjunto de datos que se presentan para identificar la identidad de un cliente es una credencial. Mostrar una credencial implica demostrar la posesión de los datos. *Windows Communication Foundation* permite trabajar con una gran diversidad de credenciales en los modos de seguridad de transporte y seguridad de mensaje. Si se va especificar una credencial, esto se debe hace en un enlace de *Windows Communication Foundation*. En México, un permiso para conducir es un ejemplo de credencial. Un permiso contiene datos que representan la identidad y capacidades de un ciudadano. Entre las características que contiene están: prueba de posesión en forma de la fotografía del ciudadano; una entidad emisora de confianza, es decir, un departamento gubernamental de licencias; y un sello en la licencia, puede contener un holograma, que confirma que la licencia no se ha falsificado. Como se ha mencionado anteriormente *Windows Communication Foundation* acepta dos tipos de credenciales: la primera el nombre de usuario y contraseña y la segunda *certificados X.509*. Comparándolo con el ejemplo de la licencia para conducir:

- ✓ La *credencial de nombre de usuario y contraseña:* el nombre de usuario representa la identidad que se reclama y la contraseña es la prueba de posesión. La autoridad de confianza es el sistema que se encargue de validar el nombre de usuario y la contraseña generalmente contenidos en una base de datos propia.
- ✓ La *credencial del certificado:* el nombre del sujeto dentro del certificado se usa para representar la identidad que se reclama. La prueba de posesión de la credencial se haciendo uso de la clave privada que se usó para generar una firma.

Credenciales de un cliente en seguridad de transporte

A continuación se describen los valores que pueden usarse al crear una aplicación que utiliza la seguridad de transporte:

CONFIGURACIÓN	DESCRIPCIÓN
None	Indica que el cliente no necesita hacer uso de ninguna credencial para autenticarse. Con esto se hace una conversión a un cliente anónimo.
Básica	Indica que se usará una autenticación básica.
Implícita	Indica que se usará una autenticación implícita.
Ntlm	Indica que se hará uso de la autenticación de *Windows* mediante negociación *SSPI* en un dominio de *Windows*. La negociación *SSPI* resulta en el uso del protocolo *Kerberos* o *NT LanMan (NTLM)*.
Windows	Indica que se usará la autenticación de Windows usando *SSPI* en un dominio de *Windows*. *SSPI* escoge el protocolo *Kerberos* o *NTLM* como servicio de autenticación. La negociación *SSPI* prueba primero el protocolo *Kerberos*; si eso falla, utiliza *NTLM*.
Certificado	Indica que se hará uso de la autenticación de cliente utilizando un certificado *X.509*.

Tabla 3.4 Tipos de credencial en transporte

Credenciales de un cliente en seguridad de mensaje

A continuación se describen los valores que pueden usarse al crear una aplicación que utiliza la seguridad de mensaje:

CONFIGURACIÓN	DESCRIPCIÓN
None	Indica que se podrá interactuar con clientes anónimos.
Windows	Indica que los intercambios de mensajes *SOAP* ocurren bajo la autenticación de una credencial de *Windows*. Se usa el mecanismo de negociación *SSPI* para optar entre el protocolo *Kerberos* o *NTLM* como servicio de autenticación.
Nombre de usuario	Indica que el servicio requiere que el cliente se autentique a través de un nombre de usuario y contraseña. Dado que *Windows Communication Foundation* no permite operaciones criptográficas con el nombre de usuario y contraseña, *Windows Communication Foundation* garantiza que el transporte se asegure al usar credenciales de nombres de usuario y contraseña.
Certificate	Indica que la autenticación del cliente deberá hacerse utilizando un certificado *X.509*.
CardSpace	Indica que la autenticación del cliente se hará mediante *Windows CardSpace*.

Tipo 3.5 Tipos de credencial en mensaje

Programación de credenciales

El modelo de programación utilizado en *Windows Communication Foundation* habilita la posibilidad de especificar las credenciales a usar por medio de comportamientos de servicio y de canal. La seguridad de *Windows Communication Foundation* tiene dos tipos comportamientos sobre los cuales de pueden usar de credenciales: comportamientos de credenciales de servicio y comportamientos de credenciales de canal. Estos comportamientos de credenciales de *Windows Communication Foundation* permiten especificar las credenciales que se van a usar para cumplir con la seguridad expresada a través de los enlaces proporcionados. En *Windows Communication Foundation* todos los clientes heredan de la clase *ClientBase,* la propiedad *ClientCredentials* permite especificar los valores de credencial del cliente. En la clase *ServiceCredentials* se especifican los certificados validar al cliente y las credenciales del servicio de *Windows Communication Foundation* para varios tipos de credenciales de cliente.

Negociación en la seguridad de mensajes

La negociación en la seguridad de mensajes permite que la credencial otorgada por el servicio de *Windows Communication Foundation* se pueda configurar en el cliente. Si por ejemplo se desea guardar el certificado proporcionado por el servicio en el *almacén de certificados del sistema operativo Windows*, se deberá utilizar una herramienta como *Microsoft Management Console (MMC, mmc.exe)* para lograrlo.

El modo de seguridad de mensajes permite que el servicio intercambie con el cliente (como parte de una negociación inicial) sus credenciales para establecer (una vez autenticados) transferencias de mensajes sin problemas.

3.2.3.2 CONTROL DE ACCESO

También conocido como autorización, el control de acceso permite a distintos usuarios obtener privilegios diferentes para realizar consultas sobre datos. Si se tienen los archivos de finanzas de una empresa, estos contienen información confidencial, por lo tanto sólo los administradores pueden consultar esta información. Entonces, la autorización está basada tanto en el rol "*administrador*" como en la identidad específica del administrador, esto último es para evitar que un administrador tenga acceso a información exclusiva de otro administrador. En *Windows Communication Foundation*, la autorización se proporciona mediante el atributo *PrincipalPermissionAttribute*.

3.2.3.3 AUDITORÍA

La auditoría suministra un mecanismo para que un administrador del sistema se percate de un ataque que se haya producido o esté ejecutándose en el momento. Lo anterior es posible gracias a

que las aplicaciones construidas usando *Windows Communication Foundation* registran eventos de seguridad como éxito o error. A parte de lo anterior, la auditoría permite a un programador encontrar problemas de seguridad. S se produce un error por ejemplo en la autorización de usuario, un programador puede solucionar el problema examinando el registro de eventos generado por la auditoría.

3.2.4 ESCENARIOS DE SEGURIDAD COMUNES

A continuación se describen algunos de los casos más comunes referentes a la seguridad en *Windows Communication.* El último escenario (*Seguridad con certificados mutuos, en servidor y cliente*) es el que aplicará en el caso práctico de los capítulos 4 y 5 de esta tesis.

3.2.4.1 CLIENTE Y SERVICIO WCF NO PROTEGIDO

La ilustración siguiente muestra un ejemplo de un cliente y servicio no seguros usando *Windows Communication Foundation (WCF)*.

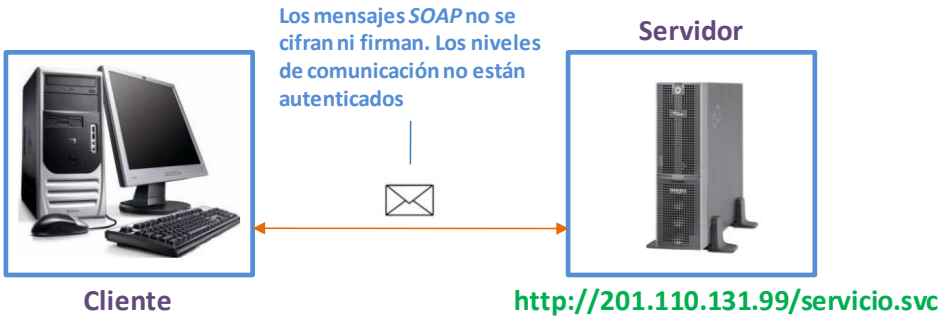


Figura 3.1 Enlace no seguro en Windows Communication Foundation[17]

CARACTERÍSTICA	DESCRIPCIÓN
Modo de seguridad	Ninguno
Transporte	HTTP
Enlace	*BasicHttpBinding*.
Interoperabilidad	Con clientes de servicios web existentes y servicios
Autenticación	Ninguno
Integridad	Ninguno
Confidencialidad	Ninguno

SERVICIO WINDOWS COMMUNICATION FOUNDATION NO PROTEGIDO

Creación del extremo del servicio en archivo de código fuente de C# (*.cs)

El siguiente código fuente en *C#* muestra cómo crear extremo sin seguridad. Como se ha mencionado, de forma predeterminada, *BasicHttpBinding* tiene el modo de seguridad establecido en *None*.

```
public class HostServicio {
    public static void Main(String[] args) {
        Uri httpUri = new Uri("http://201.110.131.99/Servicio");
        // Crear el ServiceHost.
        ServiceHost miServiceHost = new ServiceHost(typeof(Servicio), httpUri);
```

[17] Obtenido de: Cliente y servicio de Internet no protegidos. http://msdn.microsoft.com/es-mx/library/ms733091.aspx

```
        // Crear un enlace que usa HTTP. De forma predeterminada,
        // este enlace no es seguro.
        BasicHttpBinding miEnlace = new BasicHttpBinding();
        // Agregar un extremo al servicio.
        miServiceHost.AddServiceEndpoint(typeof(IServicio), miEnlace, "");
        // Abrir el servicio y esperar por llamadas.
        miServiceHost.Open();
        Console.Write("Escuchando....");
        Console.ReadLine();
        // Cerrar el servicio cuando se presione una tecla.
        miServiceHost.Close();
    }
}
```

Creación del extremo del servicio con archivo de configuración App.config

El código siguiente configura el extremo anterior usando *XML*:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors />
    <services>
      <service behaviorConfiguration="" name="ServicioBehavior">
        <endpoint address="http://201.110.131.99/Servicio"
                  binding="basicHttpBinding"
                  bindingConfiguration="Basic_NoSeguro"
                  name="BasicHttp IServicio"
                  contract="Ynnaf.IServicio" />
      </service>
    </services>
    <bindings>
      <basicHttpBinding>
        <binding name="Basic_NoSeguro" />
      </basicHttpBinding>
    </bindings>
    <client />
  </system.serviceModel>
</configuration>
```

CLIENTE WINDOWS COMMUNICATION FOUNDATION NO PROTEGIDO

Creación del cliente en archivo de código fuente de C# (*.cs)

El código siguiente en *C#* muestra un cliente *WCF* básico que tiene acceso a un extremo no seguro:

```
public class Cliente {
    public static void Main(String[] args) {
        // Crear una instancia de BasicHttpBinding.
        // De forma predeterminada, no es seguro.
        BasicHttpBinding miEnlace = new BasicHttpBinding();

        // Crear la cadena de la dirección, u obtenerlo
        // desde un archivo de configuración.
        string httpUri = "http://201.110.131.99/Servicio";

        // Crear una dirección de extremo con la dirección.
        EndpointAddress miExtremo = new EndpointAddress(httpUri);

        // Crear una instancia del cliente WCF. El código del cliente
        // se generó usando la herramienta Svcutil.exe.
        ServicioClient wcfCliente = new ServicioClient(miEnlace, miExtremo);
        try {
            wcfCliente.Open();
            // Empezar a usar el servicio.
            Console.WriteLine(wcfClient.Metodo());
```

```
            // Cerrar el cliente.
            wcfCliente.Close();
        } catch (Exception ex) {
            Console.WriteLine(ex.Message);
            wcfCliente.Abort();
        } finally {
            Console.WriteLine("Cliente cerrado");
            Console.ReadLine();
        }
    }
}
```

Creación del cliente con archivo de configuración App.config

El siguiente código configura el cliente en un archivo de configuración *XML*:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IServicio" >
          <security mode="None">
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://201.110.131.99/Servicio/NoSeguro"
                binding="basicHttpBinding"
                bindingConfiguration="BasicHttpBinding_IServicio"
                contract="IServicio"
                name="BasicHttpBinding_IServicio" />
    </client>
  </system.serviceModel>
</configuration>
```

3.2.4.2 SEGURIDAD DE TRANSPORTE CON AUTENTICACIÓN BÁSICA

La imagen siguiente muestra un servicio y un cliente de *Windows Communication Foundation (WCF)*. El servidor necesita un *certificado X.509* válido que se puede utilizar para *Capa de sockets seguros (SSL)* y los clientes deben confiar en el certificado del servidor.

Figura 3.2 Enlace seguro con autenticación básica[18]

CARACTERÍSTICA	DESCRIPCIÓN
Modo de seguridad	Transporte
Transporte	HTTP
Enlace	*WSHttpBinding*.

[18] Obtenido de: Seguridad de transporte con autenticación básica. http://msdn.microsoft.com/es-mx/library/ms733775.aspx

Interoperabilidad	Con clientes de servicios web existentes y servicios
Autenticación (servidor) **Autenticación (cliente)**	Sí (utilizando *HTTPS*) Sí (a través del nombre de usuario/Contraseña)
Integridad	Sí
Confidencialidad	Sí

SERVICIO WCF CON SEGURIDAD DE TRANSPORTE Y AUTENTICACIÓN BÁSICA

Creación del extremo del servicio en archivo de código fuente de C# (*.cs)

El código siguiente muestra cómo crear un extremo de servicio que utiliza un nombre de usuario y contraseña para la seguridad de la transferencia. Se debe considerar que el servicio exige un *certificado X.509* que autentique al cliente.

```
public class HostServicio {
    public static void Main(String[] args) {
        // Crear el enlace.
        WSHttpBinding miEnlace = new WSHttpBinding();
        miEnlace.Security.Mode = SecurityMode.Transport;
        miEnlace.Security.Transport.ClientCredentialType =
            HttpClientCredentialType.Basic;

        // Crear el URI para el extremo.
        Uri httpUri = new Uri("https://201.110.131.99/Servicio");
        // Crear el Service Host y agregar un extremo.
        ServiceHost miServiceHost = new ServiceHost(
            typeof(Servicio), httpUri);
        miServiceHost.AddServiceEndpoint(
            typeof(IServicio), miEnlace, "");

        // Abrir el servicio.
        miServiceHost.Open();
        Console.WriteLine("Escuchando...");
        Console.WriteLine("Presiona ENTER para salir.");
        Console.ReadLine();
        // Cerrar el servicio.
        miServiceHost.Close();
    }
}
```

Creación del extremo del servicio con archivo de configuración App.config

Lo siguiente configura un servicio para utilizar la autenticación básica con seguridad de nivel de transporte:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="NombreUsuarioConTransporte">
          <security mode="Transport">
            <transport clientCredentialType="Basic" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <services>
      <service name="Ynnaf.Servicio">
        <endpoint address="https://201.110.131.99/Servicio"
                  binding="wsHttpBinding"
                  bindingConfiguration="NombreUsuarioConTransporte"
                  name="BasicEndpoint"
                  contract="Ynnaf.IServicio" />
```

```
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

CLIENTE WCF PARA SERVICIO WCF CON SEGURIDAD DE TRANSPORTE Y AUTENTICACIÓN BÁSICA

Creación cliente en archivo de código fuente de C# (*.cs)

En el siguiente código el usuario debe proporcionar un nombre de usuario y contraseña válidos. Los métodos para asignar el nombre de usuario y la contraseña no se muestra aquí. Se debe usar una interfaz de usuario para pedir la información al usuario. Un punto muy importante de recordar es que el nombre de usuario y contraseña sólo se pueden establecer utilizando el código:

```
public class Cliente {
    public static void Main(String[] args) {
        // Crear el enlace.
        WSHttpBinding miEnlace = new WSHttpBinding();
        miEnlace.Security.Mode = SecurityMode.Transport;
        miEnlace.Security.Transport.ClientCredentialType =
            HttpClientCredentialType.Basic;

        // Create the endpoint address. Note que el nombre de la computadora
        // debe coincidir con el asunto o campo DNS del certificado X.509
        // usado para autenticar el service.
        EndpointAddress endPointAddress = new
            EndpointAddress("https://201.110.131.99/Servicio");

        // Crear el cliente. El  código para el cliente Servicio no se
        // muestra aquí.
        ServicioClient wcfCliente = new ServicioClient(miEnlace, endPointAddress);
        // El cliente debe proveer un nombre de usuario y contraseña. El código
        // para retornar el nombre de usuario y contraseña no se muestran aquí.
        // Use una base de datos para almacenar nombres de usuario y contraseñas.
        wcfCliente.ClientCredentials.UserName.UserName = NombreUsuario();
        wcfCliente.ClientCredentials.UserName.Password = PasswordUsuario();
        try {
            // Comenzar a usar el cliente.
            wcfCliente.Open();
            Console.WriteLine(wcfClient.Metodo());
            Console.ReadLine();

            // Cerrar el cliente.
            wcfCliente.Close();
        } catch (Exception ex) {
            Console.WriteLine(ex.Message);
            wcfCliente.Abort();
        } finally {
            Console.WriteLine("Cliente cerrado");
            Console.ReadLine();
        }
    }
}
```

Creación del extremo del servicio con archivo de configuración App.config

Como se mencionó anteriormente únicamente se puede hacer vía código. El código de configuración siguiente muestra como configurarlo:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="WSHttpBinding IServicio" >
          <security mode="Transport">
            <transport clientCredentialType="Basic" />
```

```
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client>
      <endpoint address="https://201.110.131.99/Servicio"
                binding="wsHttpBinding"
                bindingConfiguration="WSHttpBinding_IServicio"
                contract="IServicio"
                name="WSHttpBinding_IServicio" />
    </client>
  </system.serviceModel>
</configuration>
```

3.2.4.3 SEGURIDAD DE MENSAJES CON CERTIFICADOS MUTUOS

El escenario siguiente muestra un servicio y cliente *Windows Communication Foundation* protegidos utilizando el modo de seguridad de mensajes vía certificados mutuos. Tanto el cliente como el servicio se autentican con certificados x.509:

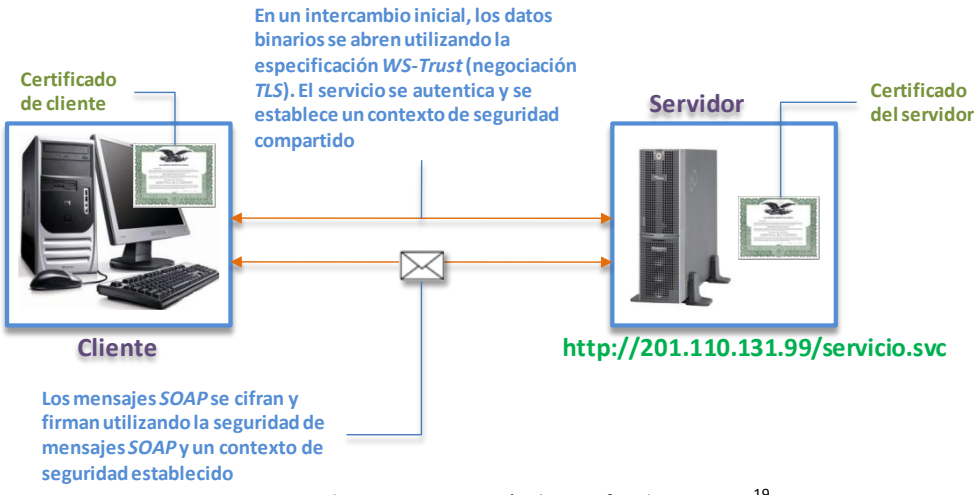


Figura 3.5 Enlace seguro a través de certificados mutuos[19]

CARACTERÍSTICA	DESCRIPCIÓN
Modo de seguridad	Mensaje
Transporte	HTTP
Enlace	*WSHttpBinding*.
Interoperabilidad	WCF – WCF y WCF – WSIT
Autenticación (servidor)	Si se utiliza inicialmente la negociación de Seguridad de la capa de transporte (TLS), se requiere autenticación de servidor utilizando el certificado del servicio
Autenticación (cliente)	Utilizar certificado de cliente y negociación TLS
Integridad	Sí, utilizando el protocolo TLS
Confidencialidad	Sí, utilizando el protocolo TLS

SERVICIO WCF CON SEGURIDAD DE MENSAJES UTILIZANDO CERTIFICADOS MUTUOS

Creación del extremo del servicio en archivo de código fuente de C# (*.cs)

El código siguiente en *C#* muestra cómo crear un extremo de servicio que utilice la seguridad del mensaje para establecer un contexto seguro:

[19] Obtenido de: Seguridad de mensajes con certificados mutuos. http://msdn.microsoft.com/es-mx/library/ms733102.aspx

```
public class HostServicio {
    public static void Main(String[] args) {
        // Crear el enlace.
        WSHttpBinding miEnlace = new WSHttpBinding();
        binding.Security.Mode = SecurityMode.Message;
        binding.Security.Message.ClientCredentialType =
             MessageCredentialType.Certificate;

        // Crear la URI para el extremo.
        Uri httpUri = new Uri("http://201.110.131.99/Servicio");
        // Crear el service host.
        ServiceHost miServiceHost =
            new ServiceHost(typeof(Servicio), httpUri);
        miServiceHost.AddServiceEndpoint(typeof(IServicio), miEnlace, "");

        // Especificar un certificado para autenticar el servicio.
        miServiceHost.Credentials.ServiceCertificate.
            SetCertificate(StoreLocation.LocalMachine,
            StoreName.My, X509FindType.FindBySubjectName,
            "CertificadoServicio");

        // Abrir el servicio.
        miServiceHost.Open();
        Console.WriteLine("Escuchando...");
        Console.ReadLine();
        // Cerrar el servicio.
        miServiceHost.Close();
    }
```

Creación del extremo del servicio con archivo de configuración App.config

El archivo de configuración del servicio es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="ServiceCredentialesBehavior">
          <serviceCredentials>
            <serviceCertificate findValue="CertificadoServicio"
                                x509FindType="FindBySubjectName" />
          </serviceCredentials>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service behaviorConfiguration="ServiceCredentialesBehavior"
               name="Ynnaf.Servicio">
        <endpoint address="http://201.110.131.99/Servicio"
                  binding="wsHttpBinding"
                  bindingConfiguration="WSHttpBinding_IServicio"
                  name="CertificadoSeguroParaCliente"
                  contract="Ynnaf.IServicio" />
      </service>
    </services>
    <bindings>
      <wsHttpBinding>
        <binding name="WSHttpBinding_IServicio">
          <security mode="Message">
            <message clientCredentialType="Certificate" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client />
  </system.serviceModel>
</configuration>
```

CLIENTE WCF PARA SERVICIO WCF CON SEGURIDAD DE MENSAJES UTILIZANDO CERTIFICADOS MUTUOS

Creación del cliente en archivo de código fuente de C# (.cs)*

El siguiente código crea un cliente del servicio. El enlace es óptimo para la seguridad del modo de mensaje y el tipo de credencial de cliente está establecido en *Certificate*:

```
public class Cliente {
    public static void Main(String[] args) {
        // Crear el enlace.
        WSHttpBinding miEnlace = new WSHttpBinding();
        miBinding.Security.Mode = SecurityMode.Message;
        miBinding.Security.Message.ClientCredentialType =
            MessageCredentialType.Certificate;

        // Crear la dirección de extremo.
        EndpointAddress endPointAddress = new
            EndpointAddress("http://201.110.131.99/Servicio");

        // Crear el cliente.
        ServicioClient wcfCliente =
            new ServicioClient(miEnlace, endPointAddress);
        // Especificar un certificado para usar para autenticar el cliente.
        wcfClient.ClientCredentials.ClientCertificate.SetCertificate(
            StoreLocation.CurrentUser,
            StoreName.My, X509FindType.FindBySubjectName,
            "CertificadoCliente");

        // Comenzar a usar el cliente.
        try {
            wcfClient.Open();
            Console.WriteLine(wcfClient.Metodo());
            Console.ReadLine();
            // Cerrar el cliente.
            wcfClient.Close();
        } catch (Exception ex) {
            Console.WriteLine(ex.Message);
            wcfClient.Abort();
        } finally {
            Console.WriteLine("Cliente cerrado");
            Console.ReadLine();
        }
    }
}
```

Creación del cliente con archivo de configuración App.config

La configuración siguiente especifica el certificado de cliente mediante un comportamiento del extremo. El código también utiliza un elemento *<identity>* para especificar un *Domain Name System (DNS)* de la identidad del servidor esperada:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="endpointCredentialesBehavior">
          <clientCredentials>
            <clientCertificate findValue="CertificadoCliente"
               storeLocation="LocalMachine"
               x509FindType="FindBySubjectName" />
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
```

```
      <wsHttpBinding>
        <binding name="WSHttpBinding_IServicio" >
          <security mode="Message">
            <message clientCredentialType="Certificate" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://201.110.131.99/Servicio"
                behaviorConfiguration="endpointCredentialesBehavior"
                binding="wsHttpBinding"
                bindingConfiguration="WSHttpBindingIServicio"
                contract="IServicio"
                name="WSHttpBinding_IServicio">
        <identity>
          <dns value="CertificadoServicio" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

CAPÍTULO 4.
DESARROLLO DE UN SERVICIO WCF QUE PUEDE SER ACCEDIDO POR DIFERENTES CLIENTES WCF/WSIT CONSTRUIDOS SOBRE DISTINTAS PLATAFORMAS Y LENGUAJES DE PROGRAMACIÓN

INTRODUCCIÓN

Algunos aspectos importantes a tener en cuenta al leer el código fuente que se muestran en este cuarto capítulo son:

- ✓ Al inicio o final de cada fragmento de código fuente que se muestra, se da un resumen de los procesos que se realizan en el código.
- ✓ En ocasiones, dentro del mismo código fuente se insertan comentarios describiendo lo que se ejecutará en las líneas que los prosiguen. Se debe tomar en cuenta que los comentarios en el lenguaje de programación *C#* inician con los caracteres // y finalizan hasta un salto de línea (los comentarios de la forma */* MiComentario */* no se utilizan en este trabajo) y en el lenguaje de programación *XML* inician con los caracteres <!-- y finalizan con los caracteres -->.
- ✓ El código mostrado no es todo el que se necesita para hacer funcionar la aplicación correctamente, únicamente se muestran fragmentos de código para procesos específicos.
- ✓ Se da por hecho que el lector tiene un conocimiento sólido de la plataforma de programación de software *.NET Framework* así como de los lenguajes de programación *C#* y *XML*. Esto se debe a que esta tesis no se centra en ofrecer al lector un conocimiento desde cero de todos los componentes de *.NET Framework*.

4.1 SERVICIO WINDOWS COMMUNICATION FOUNDATION

En este capítulo mostraré como crear un servicio *Windows Communication Foundation (WCF)* por medio de *.NET Framework 3.0*. Este servicio permitirá la interoperabilidad entre distintas plataformas y lenguajes de programación. El desarrollo de esta aplicación se basará en las siguientes etapas de desarrollo de software: *planteamiento del problema*, *requerimientos*, *análisis*, *diseño*, *implementación* y *despliegue*.

4.2 PLANTEAMIENTO DEL PROBLEMA

La empresa *YnnafSoftware* (antes *YnniSoftware*), decide implementar un nuevo sistema que le permita llevar el control sobre los distintos productos de software y hardware que vende. La decisión de realizar este nuevo sistema es debido al crecimiento que tuvo la empresa recientemente (actualmente cuenta con sucursales, socios y sitio de Internet los cuales atienden a clientes de forma descentralizada). El sistema con el que cuenta la empresa es como el que muestra la siguiente figura:

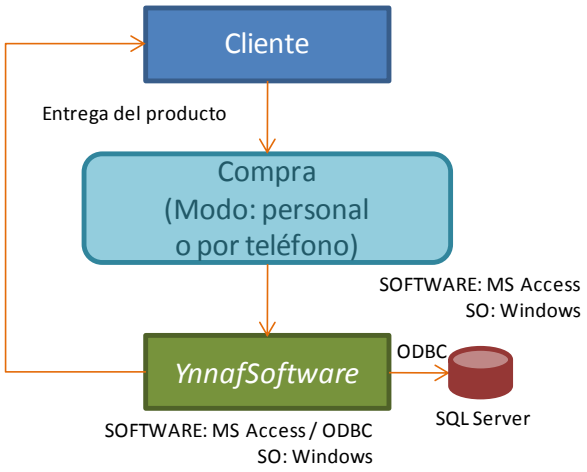


Figura 4.1 Actual sistema de la empresa YnnafSoftware (obsoleto)

La lógica del negocio del sistema (ahora obsoleto) de la empresa funciona de la siguiente manera:

1. Un cliente ordena uno o varios de los productos que vende la empresa. Esta orden de compra se puede hacer de cualquiera de las siguientes formas: *personal* o *vía telefónica*.
2. La empresa toma la orden de compra, valida que el producto esté en existencia y el pago se haya efectuado correctamente. Si todo lo anterior se cumple, se entrega o envía el producto al cliente.

La eficiencia de este sistema centralizado no es suficiente para las nuevas características de la empresa. Ahora que se tiene un mayor volumen de ventas se debe crear un nuevo sistema que sea capaz de soportarlo. Se necesita crear una aplicación distribuida para cumplir eficientemente con las ventas.

4.3 REQUERIMIENTOS

El sistema para satisfacer las nuevas necesidades de la empresa *YnnafSoftware* deberá contener las siguientes modificaciones a la lógica del negocio expuesta en el planteamiento del problema:

1. Se agregará una nueva forma de venta a las ya existentes: ventas vía *Internet*.
2. Capacidad para acceder a la lógica del negocio por otro software que se ejecute tanto dentro como fuera de la empresa. En el sistema anterior solamente se puede acceder con software desde el interior de la empresa. El nuevo sistema deberá tener la capacidad de interoperar con diferentes plataformas y lenguajes de programación. Este punto se debe a que la empresa *YnnafSoftware* (antes *YnniSoftware*) realizó una reciente fusión con la empresa *AfiSoftware*, y que realizó varios acuerdos comerciales con distintas empresas. Entonces, la lógica del negocio de la empresa deberá ser accedida por tres tipos diferentes de software, los cuales se listan a continuación:
   a) Aplicaciones cliente que se ejecutarán desde computadoras con sistema operativo *Windows* que son utilizados por empleados de la empresa *YnnafSoftware* (tanto en la matriz como en sucursales propias). Creadas específicamente para el nuevo sistema, estos programas deben ser aplicaciones de escritorio.
   b) Aplicaciones de escritorio existentes construidas en *Java 2 Standard Edition (J2SE) versión 6.0* que se pueden ejecutar o no en un sistema operativo *Windows*. Debido a la reciente fusión con otra empresa (*AfiSoftware*), esta aplicación existente debe poder

tener acceso a la lógica de negocio de la nueva aplicación para proporcionar una experiencia unificada a los clientes de las empresas fusionadas.

c) Las aplicaciones de los socios de la empresa que se pueden ejecutar en una *gran variedad de plataformas*, cada una de ellas situada dentro de una empresa que tiene una organización empresarial con la compañía.

3. La nueva aplicación distribuida deberá ser capaz de dar servicio a las siguientes aplicaciones: la aplicación que controle toda la lógica del negocio de la empresa, la aplicación que se utilizará en las sucursales de la empresa, la aplicación web para las ventas vía Internet, la aplicación existente construida en *J2SE 6.0* y a las aplicaciones de los asociados. Estas últimas están en su mayoría construidas sobre las plataformas de software *.NET Framework* y *Java*.
4. La nueva lógica del negocio se comportará de la siguiente manera:
   a) Hay cuatro tipos de clientes: *1)* Los que son atendidos directamente por la empresa matriz *YnnafSoftware* (que no son parte de su estructura empresarial), *2)* La empresa fusionada (ahora sucursales), *3)* Sucursales propias y *4)* Socios comerciales.

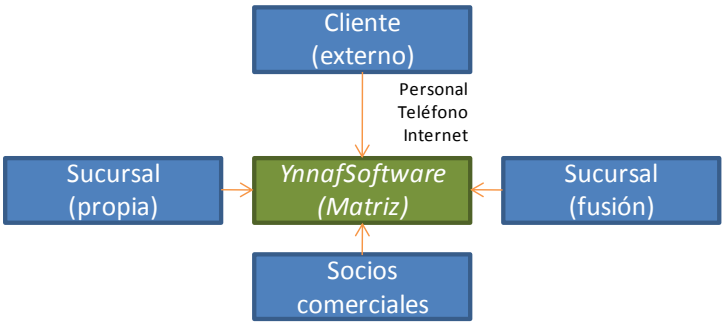


Figura 4.2 Clientes que atiente la empresa YnnafSoftware

Sin embargo, la forma en que se atienden a estos clientes no es de la misma. Para la sucursal propia, sucursal fusión y socios comerciales, la entrega de pedidos se hace de acuerdo a las necesidades de cada uno, es decir, cada determinado tiempo este tipo de clientes pide una determinada cantidad de productos a la matriz para mantenerlos en su propio almacén, cuando estas reservas se están acabando, piden una nueva dotación a la empresa. La entrega de los productos de los clientes (que no pertenecen a la organización empresarial anterior) que acuden directamente a la matriz por cualquiera de los 3 medios de compra existentes será totalmente responsabilidad de ésta (a diferencia de los otros tipos de cliente). Una vez que la empresa toma la orden de compra de cualquiera de los cuatro tipos de clientes, se valida que el producto se encuentre en existencia, el pago se haya hecho correctamente (para los clientes que acuden directamente a la matriz, acceden vía sitio web o los asociados) y entonces se entrega o envía el producto al cliente. Todos los tipos de clientes deberán conectarse a la aplicación vía *Internet* de forma segura.

b) El acceso a la base de datos contenida en *SQL Server*, se deberá hacer utilizando un lenguaje de programación diferente a *SQL.* Este lenguaje debe ofrecer una mayor seguridad, mejoras en el rendimiento y facilidad de mantenimiento y actualización. No se deberá conectar por lo tanto usando *ODBC* como se hacía en el sistema ahora obsoleto.

c) La autentificación con la aplicación que expone la lógica del negocio se deberá hacer a través de certificados *X.509* por parte del servicio y del cliente (autenticación mutua).

La autorización de acceso a la lógica del negocio se deberá realizar por medio de nombre de usuario y contraseña, esta última deberá estar encriptada en el algoritmo de seguridad Hash *SHA-512*.

d) Se deben implementar las funcionalidades que van a ser accedidas por todas las aplicaciones cliente en una sola aplicación (servicio). Estas funcionalidades deberán estar restringidas de acuerdo a los tipos de clientes. Los clientes únicamente pueden consultar información propia no la de los demás.

En resumen, el funcionamiento de la nueva lógica del negocio que será expuesta por la aplicación, deberá comportarse como se muestra en la siguiente figura:

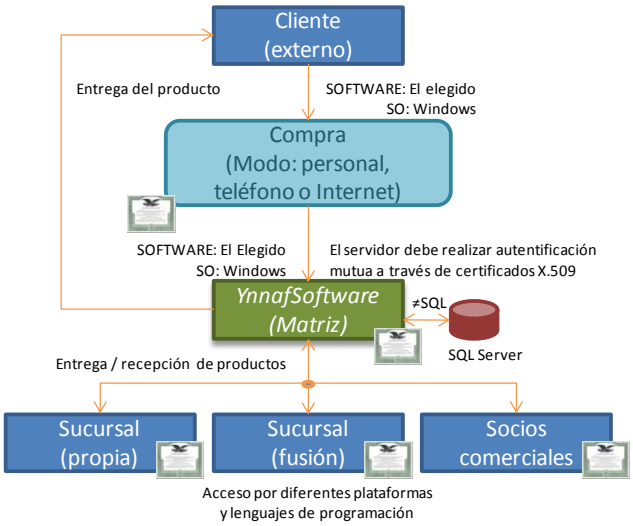


Figura 4.3 Lógica del negocio requerida para el nuevo sistema

4.4 ANÁLISIS

Dado que algunas de las herramientas con las que cuenta el actual sistema de la empresa *YnnafSoftware* son reutilizables (el sistema operativo *Windows* y el sistema manejador de bases de datos *SQL Server*), lo único que se debe elegir es sobre que lenguaje de programación se programará el nuevo sistema.

Se sabe que las dos plataformas de software más poderosas y utilizadas de la actualidad son *.NET Framework* y *Java*. Una vez que se analizaron estas alternativas, se opta por utilizar *.NET Framework* de *Microsoft* por las razones que se listan a continuación:

- ✓ La curva de aprendizaje es más corta para los programadores con los que cuenta la empresa *YnnafSoftware*, pues al utilizar previamente *Visual Basic For Applications (VBA)* para *Microsoft Access* están más acostumbrados a lenguajes de programación de *Microsoft*.
- ✓ Todos los lenguajes de programación (*C#, VB.NET, etc.*) orientados a *.NET Framework* ofrecen el paradigma de la programación orientada a objetos, muy útil para la reutilización de código fuente.
- ✓ Para implementar esta nueva lógica del negocio y habilitar la comunicación con otro software, el nuevo sistema deberá utilizar la herramienta *Windows Communication Foundation (WCF)* de *.NET Framework 3.0* para la creación de aplicaciones distribuidas.

- ✓ La capacidad de *.NET Framework* para *interoperar* entre distintos lenguajes de programación enfocados a .NET Framework como *C#*, *VB.NET*, etc. Permite construir el sistema en múltiples lenguajes.
- ✓ Los programas que utilicen las sucursales propias, el sitio de Internet y el programa maestro que controla todas las ventas, pueden construirse sobre esta plataforma sin dificultad, esto es debido a que brinda al programador una experiencia afín entre tipos de aplicaciones muy diferentes, como aplicaciones de escritorio o aplicaciones de web.
- ✓ Dado que se requiere utilizar un lenguaje diferente a *SQL* para acceder a la base de datos de *SQL Server* y obtener los máximos beneficios. La plataforma *.NET Framework 3.5* provee las Consultas Integradas en el Lenguaje para *SQL* (*LINQ to SQL*) para este propósito.
- ✓ La interoperabilidad de las aplicaciones construidas en *.NET Framework* con otros sistemas operativos como *Linux* (en caso que se necesite) a través de máquinas virtuales como *Mono*.

Entonces, las herramientas que se utilizarán para construir el nuevo sistema en *.NET Framework 3.0* serán:

- ✓ Sistema operativo: *Microsoft Windows Server 2008*
- ✓ Plataforma de programación: *.NET Framework 2.0, 3.0 y 3.5*
- ✓ Lenguaje de programación: *C#3.0*
- ✓ Servidor HTTP: *Internet Information Service (IIS) 7.0*
- ✓ Sistema manejador de bases de datos: *SQL Server 2005*
- ✓ Entorno de desarrollo integrado: *Visual Studio 2008*

Una vez que se eligió la plataforma de software con la que se construirá el sistema, la lógica del negocio funcionará como se muestra en la siguiente imagen:

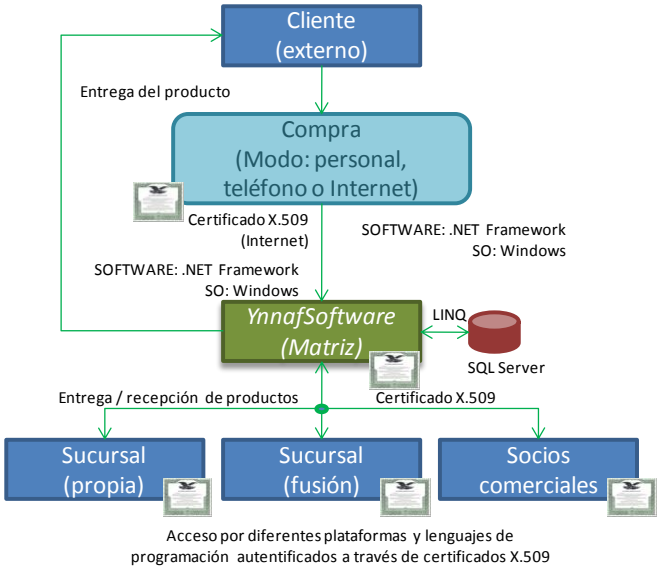


Figura 4.4 Lógica del negocio del nuevo sistema

4.5 DISEÑO

A continuación se muestran características a considerar en la implementación del servicio *WCF* que expondrá la lógica del negocio del nuevo sistema de la empresa *YnnafSoftware*.

4.5.1 CASOS DE USO DETALLADOS

Caso de uso:	**Autenticación de cliente y servicio.**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* autentica a los clientes que se conectan con él.
Pre-condiciones:	El cliente debe tener un certificado *X.509* otorgado por la empresa.
Post-condiciones:	El cliente es autenticado ante el servicio *WCF*
Versión:	0.0.1
Fecha de creación	10-02-2009
Fecha de actualización:	

FLUJOS				
Flujo básico:	Autenticación con certificados *X.509* correctos			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide realizar una transacción con el servicio *WCF* expuesto en la dirección: *http://localhost/servicioWCF/YnnafSoftwareService.svc*	2	El servicio recibe información del certificado del cliente, si el certificado coincide con el propio, se autentifica al cliente ante el servicio *WCF*. Se crea una instancia y sesión del servicio que será utilizada durante todo el tiempo que el cliente esté conectado.	
3	El usuario una vez autenticado, puede ahora iniciar sesión para obtener un nivel de acceso a la lógica del negocio.			
Flujo alternativo:	Autenticación con certificados *X.509* incorrectos			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide realizar una transacción con el servicio *WCF* expuesto en la dirección: *http://localhost/servicioWCF/YnnafSoftwareService.svc*	2	El servicio recibe información del certificado del cliente, si el certificado no coincide con el propio no se autentica al cliente y se envía automáticamente una excepción al cliente indicando que las credenciales que presentó para la autenticación son incorrectas.	

Caso de uso:	**Asignación de nivel de acceso a la lógica del negocio.**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* autoriza el acceso a la implementación del contrato de servicio a los diversos clientes.
Pre-condiciones:	El cliente debe estar registrado en la base de datos.
Post-condiciones:	El cliente obtiene un nivel de acceso a la implementación del contrato de servicio.
Versión:	0.0.1
Fecha de creación	10-02-2009
Fecha de actualización:	

FLUJOS				
Flujo básico:	Inicio de sesión con nombre de usuario y/o contraseña válidos			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	Dado que el cliente fue autenticado debido a su certificado *X.509*, y tiene una instancia y sesión en el servicio *WCF;* decide hacer uso del servicio *WCF*, iniciando sesión.	2	El servicio recibe la solicitud de inicio de sesión del cliente a través de su nombre de usuario y contraseña. Si los datos proporcionados son correctos, se asigna un nivel de acceso de acuerdo al valor contenido en la base de datos y se retorna al cliente un	E1, E2

			valor indicándole que se autorizó su acceso.	
3	El usuario puede utilizar el servicio de acuerdo a su nivel de acceso.			
Flujo alternativo:	Autorización de uso con nombre de usuario y/o contraseña inválidos			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	Dado que el cliente fue autenticado debido a su certificado *X.509*, y tiene una instancia y sesión en el servicio *WCF;* decide hacer uso del servicio *WCF*, iniciando sesión.	2	El servicio recibe la solicitud de inicio de sesión del cliente a través de su nombre de usuario y contraseña. Si el usuario no existe en la base de datos o los datos son incorrectos, se le retorna una excepción indicándole que no se encontró su nombre de usuario o contraseña.	E1, E2

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**
E1	Excepción de SQL	Se envía una excepción con datos filtrados provenientes de la excepción *SQL* producida en el *SMBD*. Estas excepciones pueden ser de: conexión, nombre de columna no válido, autorización de acceso a la tabla, inserción de valor nulo, violación de llave primaria, foránea o única entre otras. Se debe contactar al administrador del sistema.
E2	Excepción diferente a una excepción SQL	Se envía una excepción mencionando que se produjo un error desconocido en el servicio. Se debe contactar al administrador del sistema.

Caso de uso:	**Selección de datos.**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* se comporta al seleccionar datos desde cualquier tabla de la base de datos del sistema de acuerdo a una petición del cliente.
Pre-condiciones:	El cliente debe tener un nivel de acceso a la lógica del negocio que le permita seleccionar datos en la base de datos.
Post-condiciones:	El cliente obtiene los datos que solicitó.
Versión:	0.0.1
Fecha de creación	12-02-2009
Fecha de actualización:	

FLUJOS				
Flujo básico:	Selección de datos con nivel de acceso correcto			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente solicita datos de cualquiera de las tablas que contienen los datos de la empresa.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para seleccionar datos. Se conecta a la base de datos, se seleccionan todos los datos pedidos y se retornan al cliente.	E1, E2
4	El cliente puede utilizar esos datos de forma local.			
Flujo alternativo:	Selección de datos con nivel de acceso no válido			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente solicita datos de cualquiera de las tablas que contienen los datos de la empresa.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para seleccionar datos, si no es así, el servicio retorna una excepción al cliente indicándole que no tiene permiso de acceso a ese contrato de operación.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**
E1	Excepción de SQL	Se envía una excepción con datos filtrados provenientes de la excepción *SQL* producida en el *SMBD*. Estas excepciones pueden ser de:

		conexión, nombre de columna no válido, autorización de acceso a la tabla, inserción de valor nulo, violación de llave primaria, foránea o única entre otras. Se debe contactar al administrador del sistema.
E2	Excepción diferente a una excepción SQL	Se envía una excepción mencionando que se produjo un error desconocido en el servicio. Se debe contactar al administrador del sistema

Caso de uso:	**Búsqueda de datos.**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* se comporta al buscar datos desde cualquier tabla de la bases de datos del sistema de acuerdo a una petición del cliente. Las búsquedas siempre se hacen por patrón y de acuerdo al campo más importante de la tabla.
Pre-condiciones:	El cliente debe tener un nivel de acceso a la lógica del negocio que le permita buscar datos en la base de datos.
Post-condiciones:	El cliente obtiene los datos que solicitó.
Versión:	0.0.1
Fecha de creación	12-02-2009
Fecha de actualización:	

FLUJOS

Flujo básico: Búsqueda de datos con nivel de acceso correcto

ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente solicita datos específicos en cualquiera de las tablas que contienen datos de la empresa.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para buscar datos. Se conecta a la base de datos, se buscan los datos pedidos y se retornan al cliente.	E1, E2
4	El cliente puede utilizar esos datos de forma local.			

Flujo alternativo: Búsqueda de datos con nivel de acceso no válido

Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente solicita datos específicos en cualquiera de las tablas que contienen datos de la empresa.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para buscar datos, si no es así, el servicio retorna una excepción al cliente indicándole que no tiene permiso de acceso a ese contrato de operación.	

EXCEPCIONES

Identificador	**Nombre**	**Respuesta del sistema**
E1	Excepción de SQL	Se envía una excepción con datos filtrados provenientes de la excepción *SQL* producida en el *SMBD*. Estas excepciones pueden ser de: conexión, nombre de columna no válido, autorización de acceso a la tabla, inserción de valor nulo, violación de llave primaria, foránea o única entre otras. Se debe contactar al administrador del sistema.
E2	Excepción diferente a una excepción SQL	Se envía una excepción mencionando que se produjo un error desconocido en el servicio. Se debe contactar al administrador del sistema

Caso de uso:	**Inserción de datos**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* se comporta al insertar datos en cualquier tabla de la base de datos del sistema de acuerdo a una petición del cliente.
Pre-condiciones:	El cliente debe tener un nivel de acceso a la lógica del negocio que le permita insertar datos en la base de datos.
Post-condiciones:	El cliente insertó los datos que necesitaba.
Versión:	0.0.2
Fecha de creación	14-02-2009

Fecha de actualización:				
FLUJOS				
Flujo básico:	Inserción de datos con nivel de acceso y lista de datos correctos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide a insertar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para insertar datos. Se verifica que la lista de elementos a insertar enviada por el cliente contenga elementos. Se conecta a la base de datos y se insertan los datos. Se retorna al cliente el número de elementos insertados.	E1, E2
5	El cliente recibe la información de cuántos elementos se insertaron como resultado de su petición.			
Flujo alternativo 1:	Inserción de datos con nivel de acceso no válido para la operación.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide a insertar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para insertar datos, si no es el adecuado se le envía una excepción indicándole que no tiene permiso de acceso a ese contrato de operación.	
Flujo alternativo 2:	Inserción de datos con lista vacía.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide a insertar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para insertar datos. Se verifica que la lista de elementos a insertar enviada por el cliente contenga elementos, si está vacía, se le envía una excepción indicándole que la lista de los datos a insertar no puede estar vacía.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**
E1	Excepción de SQL	Se envía una excepción con datos filtrados provenientes de la excepción *SQL* producida en el *SMBD*. Estas excepciones pueden ser de: conexión, nombre de columna no válido, autorización de acceso a la tabla, inserción de valor nulo, violación de llave primaria, foránea o única entre otras. Se debe contactar al administrador del sistema.
E2	Excepción diferente a una excepción SQL	Se envía una excepción mencionando que probablemente se enviaron datos incoherentes al servicio o se produjo un error desconocido. Si se cree que los datos con coherentes, entonces se debe contactar al administrador del sistema

Caso de uso:	**Actualización de datos**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* se comporta al actualizar datos en cualquier tabla de la base de datos del sistema de acuerdo a una petición del cliente.
Pre-condiciones:	El cliente debe tener un nivel de acceso a la lógica del negocio que le permita actualizar datos en la base de datos.
Post-condiciones:	El cliente actualizó los datos que necesitaba.
Versión:	0.0.2
Fecha de creación	14-02-2009
Fecha de actualización:	
FLUJOS	
Flujo básico:	Inserción de datos con nivel de acceso y lista de datos correctos.
ACTOR	**SISTEMA**

Paso	Acciones	Paso	Acciones	Excepción
1	El cliente decide actualizar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para actualizar datos. Se verifica que la lista de elementos a actualizar enviada por el cliente contenga elementos. Se conecta a la base de datos y se actualizan los datos. Se retorna al cliente el número de elementos actualizados.	E1, E2
3	El cliente recibe la información de cuántos elementos se actualizaron como resultado de su petición.			
Flujo alternativo 1:	Inserción de datos con nivel de acceso no válido para la operación.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide actualizar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para actualizar datos, si no es el adecuado se le envía una excepción indicándole que no tiene permiso de acceso a ese contrato de operación.	
Flujo alternativo 2:	Inserción de datos con lista vacía.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide actualizar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para actualizar datos. El servicio verifica que la lista de elementos a actualizar enviada por el cliente contenga elementos, si está vacía, se le envía una excepción indicándole que la lista de los datos a actualizar no puede estar vacía.	

EXCEPCIONES

Identificador	Nombre	Respuesta del sistema
E1	Excepción de SQL	Se envía una excepción con datos filtrados provenientes de la excepción *SQL* producida en el *SMBD*. Estas excepciones pueden ser de: conexión, nombre de columna no válido, autorización de acceso a la tabla, inserción de valor nulo, violación de llave primaria, foránea o única entre otras. Se debe contactar al administrador del sistema.
E2	Excepción diferente a una excepción SQL	Se envía una excepción mencionando que probablemente se enviaron datos incoherentes al servicio o se produjo un error desconocido. Si se cree que los datos con coherentes, entonces se debe contactar al administrador del sistema

Caso de uso:	**Eliminación de datos**
Tipo:	Básico
Actor principal:	Clientes del servicio *WCF*
Descripción:	Se describe como el servicio *WCF* se comporta al eliminar datos en cualquier tabla de la base de datos del sistema de acuerdo a una petición del cliente.
Pre-condiciones:	El cliente debe tener un nivel de acceso a la lógica del negocio que le permita eliminar datos en la base de datos.
Post-condiciones:	El cliente eliminó los datos que necesitaba.
Versión:	0.0.1
Fecha de creación	14-02-2009
Fecha de actualización:	
FLUJOS	
Flujo básico:	Eliminación de datos con nivel de acceso y lista de datos correctos.

ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide eliminar un conjunto de datos en cualquiera de las tablas de la	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para eliminar	E1, E2

	base de datos del sistema.		datos. El servicio verifica que la lista de elementos a eliminar enviada por el cliente contenga elementos. Se conecta a la base de datos y se eliminan los datos. Se retorna al cliente el número de elementos eliminados.	
3	El cliente recibe la información de cuántos elementos se eliminaron como resultado de su petición.			
Flujo alternativo 1:	Inserción de datos con nivel de acceso no válido para la operación.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide eliminar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para eliminar datos, si no es el adecuado se le envía una excepción indicándole que no tiene permiso de acceso a ese contrato de operación.	
Flujo alternativo 2:	Inserción de datos con lista vacía.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide eliminar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema.	2	El servicio verifica que el cliente tiene el nivel de acceso necesario para eliminar datos. El servicio verifica que la lista de elementos a eliminar enviada por el cliente contenga elementos, si está vacía, se le envía una excepción indicándole que la lista de los datos a eliminar no puede estar vacía.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**
E1	Excepción de SQL	Se envía una excepción con datos filtrados provenientes de la excepción SQL producida en el *SMBD*. Estas excepciones pueden ser de: conexión, nombre de columna no válido, autorización de acceso a la tabla, inserción de valor nulo, violación de llave primaria, foránea o única entre otras. Se debe contactar al administrador del sistema.
E2	Excepción diferente a una excepción SQL	Se envía una excepción mencionando que probablemente se enviaron datos incoherentes al servicio o se produjo un error desconocido. Si se cree que los datos con coherentes, entonces se debe contactar al administrador del sistema

4.5.2 MODELO ENTIDAD-RELACIÓN, ESTRUCTURA Y PERMISOS DE ACCESO A LA BASE DE DATOS

El modelo entidad relación de la base de datos del nuevo sistema construida en *SQL Server 2005* es el siguiente:

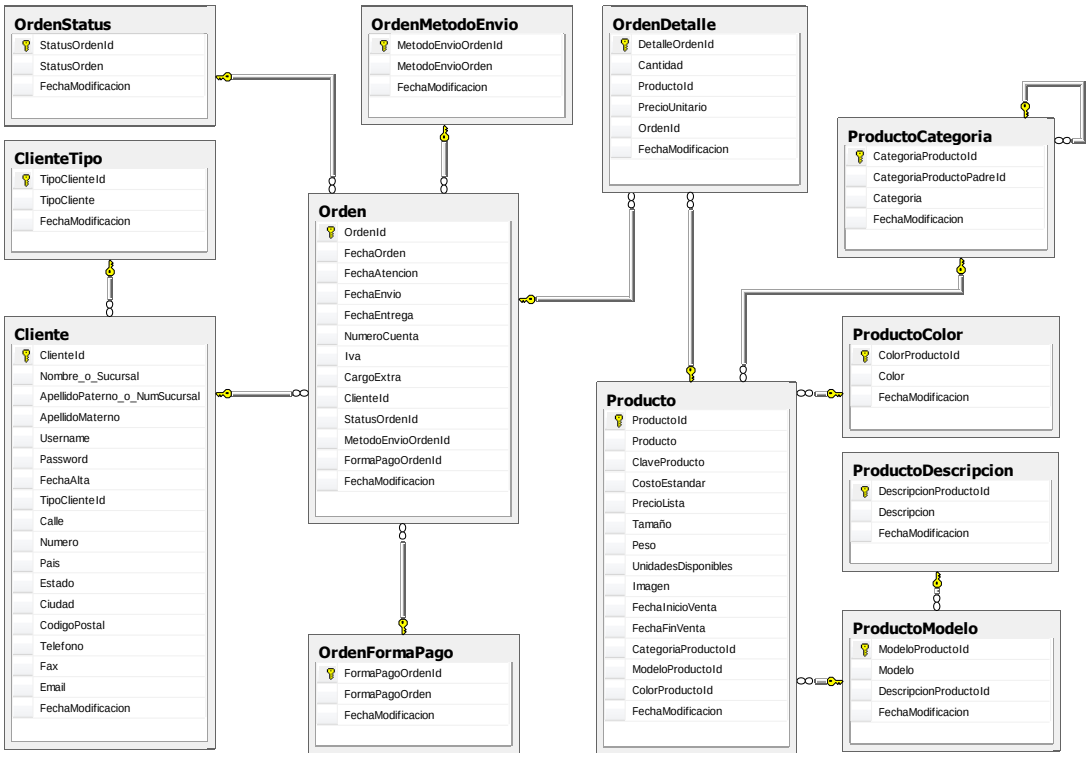


Figura 4.5 Modelo entidad-relación

La funcionalidad, estructura, llaves primarias (*PK*), únicas (*UQ*) y relaciones (*FK*) de las tablas que constituyen la base de datos (*YnnafSoftware.mdf*) del sistema son las siguientes:

Producto: Contiene información sobre los productos.

Campos (Nombre / Tipo / Nulo / Default)			
ProductoId	int identity (1,1)	NOT NULL	[Ninguno]
Producto	nvarchar(100)	NOT NULL	[Ninguno]
ClaveProducto	nvarchar(50)	NOT NULL	[Ninguno]
CostoEstandar	decimal(18, 2)	NULL	[Ninguno]
PrecioLista	decimal(18, 2)	NOT NULL	[Ninguno]
Tamaño	nvarchar(50)	NULL	[Ninguno]
Peso	nvarchar(50)	NULL	[Ninguno]
UnidadesDisponibles	int	NOT NULL	[Ninguno]
Imagen	varbinary(MAX)	NOT NULL	Imagen No Disponible - Array
FechaInicioVenta	datetime	NULL	[Ninguno]
FechaFinVenta	datetime	NULL	[Ninguno]
CategoriaProductoId	int	NOT NULL	[Ninguno]
ModeloProductoId	int	NOT NULL	[Ninguno]
ColorProductoId	int	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
ProductoId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
Producto/CategoriaProductoId		ProductoCategoria/CategoriaProductoPadreId	
Producto/ColorProductoId		ProductoColor/ColorProductoId	
Producto/ModeloProductoId		ProductoModelo/ModeloProductoId	
Producto/ProductoId		OrdenDetalle/ProductoId	

ProductoCategoria: Contiene información sobre las categorías de productos.

Campos (Nombre / Tipo / Nulo / Default)			
CategoriaProductoId	int identity (1,1)	NOT NULL	[Ninguno]
CategoriaProductoPadreId	int	NULL	[Ninguno]
Categoria	nvarchar(50)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
CategoriaProductoId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones			
ProductoCategoria/CategoriaProductoId		ProductoCategoria/CategoriaProductoPadreId	
ProductoCategoria/CategoriaProductoId		Producto/CategoriaProductoId	

ProductoColor: Contiene información sobre las colores que pueden tener los productos.

Campos (Nombre / Tipo / Nulo / Default)			
ColorProductoId	int identity (1,1)	NOT NULL	[Ninguno]
Color	nvarchar(30)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
ColorProductoId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
ProductoCategoria/ColorProductoId		Producto/ColorProductoId	

ProductoDescripcion: Contiene diferentes descripciones de productos.

Campos (Nombre / Tipo / Nulo / Default)			
DescripcionProductoId	int identity (1,1)	NOT NULL	[Ninguno]
Descripcion	nvarchar(500)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
DescripcionProductoId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
ProductoDescripcion/DescripcionProductoId		ProductoModelo/DescripcionProductoId	

ProductoModelo: Contiene información sobre diferentes modelos de productos.

Campos (Nombre / Tipo / Nulo / Default)			
ModeloProductoId	int identity (1,1)	NOT NULL	[Ninguno]
Modelo	nvarchar(50)	NOT NULL	[Ninguno]
DescripcionProductoId	int	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
ModeloProductoId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
ProductoModelo/ModeloProductoId		Producto/ModeloProductoId	
ProductoModelo/DescripcionProductoId		ProductoDescripcion/DescripcionProductoId	

Orden: Contiene información general sobre las órdenes de compra.

Campos (Nombre / Tipo / Nulo / Default)			
OrdenId	int identity (1,1)	NOT NULL	[Ninguno]
FechaOrden	datetime	NULL	[Ninguno]
FechaAtencion	datetime	NULL	[Ninguno]
FechaEnvio	datetime	NULL	[Ninguno]
FechaEntrega	datetime	NULL	[Ninguno]
NumeroCuenta	nvarchar(30)	NULL	[Ninguno]
Iva	decimal(18, 2)	NOT NULL	0.15

CargoExtra	decimal(18, 2)	NOT NULL	0.00
ClienteId	varbinary(MAX)	NOT NULL	[Ninguno]
StatusOrdenId	int	NOT NULL	[Ninguno]
MetodoEnvioOrdenId	int	NOT NULL	[Ninguno]
FormaPagoOrdenId	int	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
ModeloProductoId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
Orden/ClienteId		Cliente/ClienteId	
Orden/StatusOrdenId		OrdenStatus/StatusOrdenId	
Orden/MetodoEnvioOrdenId		OrdenMetodoEnvio/MetodoEnvioOrdenId	
Orden/FormaPagoOrdenId		OrdenFormaPago/FormaPagoOrdenId	
Orden/OrdenId		OrdenDetalle/OrdenId	

OrdenStatus: Contiene información sobre los posibles estados de una orden.

Campos (Nombre / Tipo / Nulo / Default)			
StatusOrdenId	int identity (1,1)	NOT NULL	[Ninguno]
StatusOrden	nvarchar(25)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
StatusOrdenId			
Llaves únicas (UQ)			
StatusOrden			
Relaciones (FK)			
OrdenStatus/StatusOrdenId		Orden/StatusOrdenId	

OrdenMetodoEnvio: Contiene información sobre los posibles métodos de envío de una orden.

Campos (Nombre / Tipo / Nulo / Default)			
MetodoEnvioOrdenId	int identity (1,1)	NOT NULL	[Ninguno]
MetodoEnvioOrden	nvarchar(50)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
MetodoEnvioOrdenId			
Llaves únicas (UQ)			
MetodoEnvioOrden			
Relaciones (FK)			
OrdenMetodoEnvio/MetodoEnvioOrdenId		Orden/MetodoEnvioOrdenId	

OrdenFormaPago: Contiene información sobre las posibles formas de pago de una orden.

Campos (Nombre / Tipo / Nulo / Default)			
FormaPagoOrdenId	int identity (1,1)	NOT NULL	[Ninguno]
FormaPagoOrden	nvarchar(50)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
FormaPagoOrdenId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
OrdenFormaPago/FormaPagoOrdenId		Orden/FormaPagoOrdenId	

OrdenDetalle: Contiene información detallada sobre las órdenes de compra.

Campos (Nombre / Tipo / Nulo / Default)			
DetalleOrdenId	int identity (1,1)	NOT NULL	[Ninguno]
Cantidad	int	NOT NULL	[Ninguno]
ProductoId	int	NOT NULL	[Ninguno]
PrecioUnitario	decimal(18, 2)	NOT NULL	[Ninguno]
OrdenId	int	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()

Llaves primarias (PK)	
ModeloProductoId	
Llaves únicas (UQ)	
[Ninguna]	
Relaciones (FK)	
OrdenDetalle/OrdenId	Orden/OrdenId
OrdenDetalle/ProductoId	Producto/ProductoId

Cliente: Contiene información sobre los clientes de la empresa.

Campos (Nombre / Tipo / Nulo / Default)			
ClienteId	int identity (1,1)	NOT NULL	[Ninguno]
Nombre_o_Sucursal	nvarchar(50)	NOT NULL	[Ninguno]
ApellidoPaterno_o_NumSucursal	nvarchar(50)	NOT NULL	[Ninguno]
ApellidoMaterno	nvarchar(50)	NULL	[Ninguno]
Username	nvarchar(50)	NOT NULL	[Ninguno]
Password	nvarchar(150)	NOT NULL	[Ninguno]
FechaAlta	datetime	NOT NULL	getdate()
TipoClienteId	int	NOT NULL	[Ninguno]
Calle	nvarchar(50)	NOT NULL	[Ninguno]
Numero	nvarchar(50)	NOT NULL	[Ninguno]
Pais	nvarchar(50)	NOT NULL	[Ninguno]
Estado	nvarchar(50)	NOT NULL	[Ninguno]
Ciudad	nvarchar(50)	NOT NULL	[Ninguno]
CodigoPostal	nvarchar(20)	NOT NULL	[Ninguno]
Telefono	nvarchar(20)	NOT NULL	[Ninguno]
Fax	nvarchar(20)	NOT NULL	[Ninguno]
Email	nvarchar(75)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
ClienteId			
Llaves únicas (UQ)			
[Ninguna]			
Relaciones (FK)			
Cliente/ClienteId		Orden/ClienteId	
Cliente/TipoClienteId		ClienteTipo/TipoClienteId	

ClienteTipo: Contiene información sobre los clientes de la empresa.

Campos (Nombre / Tipo / Nulo / Default)			
TipoClienteId	int identity (1,1)	NOT NULL	[Ninguno]
TipoCliente	nvarchar(50)	NOT NULL	[Ninguno]
FechaModificacion	datetime	NOT NULL	getdate()
Llaves primarias (PK)			
TipoClienteId			
Llaves únicas (UQ)			
TipoCliente			
Relaciones (FK)			
ClienteTipo/TipoClienteId		Cliente/TipoClienteId	

Información a considerar para las tablas anteriores: *1)* La tabla *OrdenDetalle* aunque no necesita de una llave primaria, ésta se colocó para evitar problemas de inserción en *LINQ to SQL*, *2)* El valor *getdate()* en *Default* es una función de agregado de *SQL Server (Transact-SQL)* para obtener la hora actual del sistema, *3)* El valor *Default* de la columna Imagen de la tabla *Producto*, es un arreglo de bytes que representa a una imagen que contiene el texto "Imagen no disponible" y *4)* La única relación que tiene eliminación en *Cascada* es la de las tablas *Orden/OrdenDetalle.*

Además de las tablas anteriores, se crearán los siguientes procedimientos almacenados en la base de datos de *SQL Server*:

DEFINICIÓN DEL PROCEDIMIENTO ALMACENADO	FUNCIÓN
sp_GetProductos	Obtiene todos los productos, categorías de productos, modelos de productos, descripciones de productos y colores de productos. Se deberá implementar con las sentencias *Join* necesarias.
sp_GetProductosByNombre(@nombreProducto nvarchar(50), @productosEncontrados int = 0 output)	Obtiene productos buscados de acuerdo al nombre que se le mande al parámetro de entrada. También obtiene todas las categorías de productos, modelos de productos, descripciones de productos y colores de productos. Se deberá implementar con las sentencias *Join* necesarias y la búsqueda se deberá de hacer de la siguiente manera: *LIKE '%' + @nombreProducto + '%'*. También se retorna el número de productos encontrados en el parámetro de salida del procedimiento.
sp_GetOrdenById(@numeroOrden int, @clienteId int)	Obtiene los datos generales y particulares de una orden de compra de acuerdo al Id de la orden y del cliente. Estos valores se insertan en los parámetros de entrada.
sp_GetOrdenByClienteId(@clienteId int)	Obtiene los datos generales y particulares de todas las órdenes hechas por un cliente. La búsqueda de las órdenes se hace de acuerdo al Id del cliente en el parámetro de entrada.
sp_CreateOrden(@listaProductos varchar(1000), @fechaOrden datetime, @fechaAtencion datetime, @fechaEnvio datetime, @fechaEntrega datetime, @numeroCuenta nvarchar(30), @iva decimal(18, 2), @cargoExtra decimal(18, 2), @statusOrdenId int, @clienteId int, @metodoenvioOrdenId int, @formaPagoOrdenId int, @ordenId int output, @descripcionResultado nvarchar(150) output)	Crea una orden de compra. Se le deben pasar como parámetros los campos de las tablas *Orden* y *OrdenDetalles*. El parámetro *listaProductos* debe ser una cadena que siga el patrón: *"Cantidad, ProductoId_1, Cantidad, ProductoId_2, …"*. El patrón será descompuesto en el procedimiento para hacer la correcta orden de compra de productos. El parámetro de salida *OrdenId* es el que se devolverá al cliente del servicio *WCF* indicando el número de orden que se acaba de crear. Si ocurre o no un error al procesar la orden, se le devolverá la descripción del resultado de la orden al cliente en el parámetro de salida *descripcionResultado*. Este procedimiento almacenado debe usar transacciones para asegurar la integridad de los datos.
sp_CreateClienteASPNET(@nombre_o_Sucursal nvarchar(50), @apellidoPaterno_o_NumeroSucursal nvarchar(50), @apellidoMaterno nvarchar(50), @username nvarchar(50), @password nvarchar(50), @tipoClienteId int, @calle nvarchar(50), @numero nvarchar(50), @ciudad nvarchar(50), @estado nvarchar(50), @pais nvarchar(50), @codigoPostal nvarchar(50), @telefono nvarchar(50), @fax nvarchar(50), @email nvarchar(50), @existeUsername int OUTPUT, @numeroError int OUTPUT)	Crea un cliente proveniente desde el sitio web creado en *ASP.NET*. Los parámetros de entrada de este procedimiento almacenado son iguales a los campos de la tabla *Cliente*. Los parámetros de salida y su funcionalidad son los siguientes: *existeUsername,* a través del cual se informará al cliente si el nombre de usuario ya existe en la base de datos (se retornará un valor entero de uno si existe); *numeroError,* es utilizado para retornar el número de error producido en el procedimiento almacenado (si es que se produce alguno se retorna un valor diferente de cero). Este procedimiento almacenado debe utilizar transacciones para asegurar la integridad de los datos de la tabla *Cliente* de la base de datos.

El Sistema Manejador de Bases de Datos (*SQL Server*) trabajará con los siguientes cuatro tipos de usuarios de acuerdo a los requerimientos del sistema:

- ✓ AutorizacionUser: Se encargará de autorizar la entrada de cualquier usuario a la lógica del negocio.

- ✓ YnnafSoftwareUser: Este tipo de usuario se encargará de administrar todo el sistema.
- ✓ SucursalUser: Será el usuario proveniente de una sucursal propia o de fusión.
- ✓ SitioWebUser: Será el usuario que acceda desde el sitio de Internet.

Los usuarios anteriores accederán a los miembros de la base de datos (*YnnafSoftware*) con los siguientes privilegios:

TIPO Y NOMBRE DEL ELEMENTO	*AutorizacionUser*					*YnnafSoftwareUser*					*SucursalUser*					*SitioWebUser*				
	S	I	U	D	E	S	I	U	D	E	S	I	U	D	E	S	I	U	D	E
Tabla Cliente	*S*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*N*	*N*	*N*	*N*	*X*	*N*	*S*	*S*	*N*	*X*
Tabla ClienteTipo	*S*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*N*	*N*	*N*	*N*	*X*	*N*	*N*	*N*	*N*	*X*
Tabla Dirección	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*N*	*X*
Tabla Orden	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*S*	*N*	*N*	*X*	*S*	*S*	*N*	*N*	*X*
Tabla OrdenDetalle	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*S*	*N*	*N*	*X*	*S*	*S*	*N*	*N*	*X*
Tabla OrdenFormaPago	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*N*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla OrdenMetodoEnvio	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*N*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla OrdenStatus	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*N*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla Producto	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla ProductoCategoria	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla ProductoColor	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla ProductoDescripcion	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
Tabla ProductoModelo	*N*	*N*	*N*	*N*	*X*	*S*	*S*	*S*	*S*	*X*	*S*	*N*	*N*	*N*	*X*	*S*	*N*	*N*	*N*	*X*
P.A. sp_GetProductos	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*
P.A. sp_GetProductosByNombre	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*
P.A. sp_GetOrdenById	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*
P.A. sp_GetOrdenByClienteId	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*S*
P.A. sp_CreateOrden	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*
P.A. sp_CreateClientASPNET	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*	*X*	*X*	*X*	*X*	*N*	*X*	*X*	*X*	*X*	*S*

Donde:

En negrita:	*En cursiva:*
S = Select, **I** = Insert, **U** = Update, **D** = Delete y **E** = Execute	*S* = Sí, *N* = No y *X* = No aplica

4.5.3 DEFINICIÓN DE MIEMBROS DEL CONTRATO DE SERVICIO

Información a considerar para todos los miembros del servicio WCF: El nombre de los *contratos de operación*, *contratos de datos*, *enumeraciones*, *estructuras* y *métodos* es tan descriptivo que no es necesario mencionar a detalle su funcionalidad. La funcionalidad de cada uno de ellos va acorde con su nombre. A continuación se muestran las definiciones que deben tener todos los miembros del servicio *WCF*.

4.5.3.1 DEFINICIONES DE CONTRATOS DE OPERACIÓN

La definición de los contratos de operación que se muestran a continuación es extraída de acuerdo a: los casos de uso detallados, la definición de los procedimientos almacenados de *SQL Server* o simplemente son métodos auxiliares:

El ***primer caso de uso*** no contiene métodos implementados explícitamente. El método que cumple con el ***segundo caso de uso*** es: bool IniciarSesion(String username, String password). Los métodos que cumplen con el ***tercer caso de uso*** son: List<ProductoColor> GetColoresProductos(); List<ProductoDescripcion> GetDescripcionesProductos(); List<ProductoModelo_Join> GetModelosProductos(); List<ProductoCategoria_Join> GetCategoriasProductos(); List<Producto_Join> GetProductos();List<OrdenFormaPago> GetFormasPagoOrden(); List<OrdenMetodoEnvio> GetMetodosEnvioOrden(); List<Cliente_Join> GetClientes();List<ClienteTipo> GetTiposCliente(). Los métodos que cumplen con el ***cuarto caso de uso*** son: List<Producto_Join> GetProductosByNombre(String nombreProducto); Cliente GetClienteById(int idCliente); Los métodos que cumplen con el ***quinto caso de uso*** son: int InsertCategorias(List<ProductoCategoria> listaCategorias); int InsertColores(List<ProductoColor> listaColores); int InsertDescripciones(List<ProductoDescripcion> listaDescripciones); int InsertModelos(List<ProductoModelo> listaModelos); int InsertProductos(List<Producto> listaProductos); int InsertDetalleOrden(List<OrdenDetalle> listaDetalleOrden); int InsertCliente(Cliente

cliente); int InsertClientes(List<Cliente> listaClientes). Los métodos que cumplen con el ***sexto caso de uso*** son: int UpdateCategorias(List<ProductoCategoria> listaCategorias); int UpdateColores(List<ProductoColor> listaColores); int UpdateDescripciones(List<ProductoDescripcion> listaDescripciones); int UpdateModelos(List<ProductoModelo> listaModelos); int UpdateProductos(List<Producto> listaProductos); int UpdateOrden(Orden orden); int UpdateDetalleOrden(List<OrdenDetalle> listaDetalleOrden); int UpdateCliente(Cliente cliente); int UpdateClientes(List<Cliente> listaClientes). Los métodos que cumplen con el ***séptimo caso de uso*** son: int DeleteCategorias(List<int> listaIdCategorias); int DeleteColores(List<int> listaIdColores); int DeleteDescripciones(List<int> listaIdDescripciones); int DeleteModelos(List<int> listaIdModelos); int DeleteProductos(List<int> listaIdProductos); int DeleteOrden(int idOrden); int DeleteDetalleOrden(List<int> listaIdDetalleOrden); int DeleteCliente(int idCliente); int DeleteClientes(List<int> listaIdClientes).

Las definiciones de los métodos que implementan los ***procedimientos almacenados*** del sistema son: Sp_Productos Sp_GetProductos();Sp_Productos Sp_GetProductosByNombre(String nombreProducto, ref int productosEncontrados); Sp_Orden_Id Sp_GetOrdenById(int numeroOrden, ref decimal subtotal, ref decimal subtotalCargoExtra, ref decimal subtotalIva, ref decimal total); Sp_Orden_ClienteId Sp_GetOrdenByClienteId(ref decimal subtotal, ref decimal subtotalCargoExtra, ref decimal subtotalIva, ref decimal total); void Sp_CreateClienteASPNET(Cliente cliente, ref int existeUsername, ref int numeroError); int Sp_CreateOrden(Orden orden, String listaProductos, ref int ordenId, ref String descripcionResultado).

La definición del método que ***destruye una instancia*** creada previamente es: bool FinalizarSesion(). La definición del método que comprueba si un ***usuario existe*** previamente en la base de datos es: bool ComprobarUsername(String usuario).

NOTA: Todos los contratos de operación deben estar serializados. Únicamente el contrato de operación *IniciarSesion* contiene la propiedad *IsInitiating* asignada a *true y la propiedad IsTerminating* asignada a *false*. Solamente el contrato de operación *FinalizarSesion* contiene la propiedad *IsInitiating* asignada a *true* y la propiedad *IsTerminating* asignada a *true*. Todos los demás contratos de operación tienen las propiedades *IsInitiating* asignada a *false y la propiedad IsTerminating* asignada a *false.*

4.5.3.2 DEFINICIONES DE CONTRATOS DE DATOS.

Contratos de datos dependientes de los contratos de operación:

Para ***Joins*** son: Cliente_Join, Orden_Join, OrdenDetalle_Join, Producto_Join, ProductoCategoria_Join y ProductoModelo_Join
Para ***FaultException*** son: AutorizacionFault, ClienteFault, OrdenFault, ProductoFault
Para ***procedimientos almacenados*** son: Sp_Orden_Id, Sp_Orden_ClienteId, Sp_Productos

NOTA: Todos los contratos de datos y miembros de datos deben estar serializados. Los miembros de datos de los contratos de datos para *Joins* dependen de las columnas de las tablas que se quieren obtener desde la base de datos de *SQL Server* en una consulta. Los contratos de datos para *FaultException* deben contener los miembros de datos *int NumeroError* y *String Mensaje* para obtener información del mensaje. Los miembros de datos de los contratos de datos para *procedimientos almacenados* dependen de los conjuntos de resultados que retorne el procedimiento almacenado desde *SQL Server*.

4.5.3.3 DEFINICIONES DE ENUMERACIONES

Enumeración (*Enum*) para almacenar los probables números de error: NumeroErrorEnum. No debe estar serializada.

4.5.3.4 DEFINICIONES DE ESTRUCTURAS

Estructura (*Struct*) para almacenar las posibles descripciones de errores: MensajeErrorStruct. No debe estar serializada.

4.5.3.5 DEFINICIONES DE MÉTODOS

Para ***tokenizar los id y cantidades de productos***: String AgruparCadenaTokenizada(String cadenaTokenizada). No es un contrato de operación. Usado para el segundo parámetro del procedimiento almacenado: int Sp_CreateOrden(Orden orden, String listaProductos, ref int ordenId, ref String descripcionResultado).

4.6 IMPLEMENTACIÓN

A continuación se muestra (en forma reducida por cuestiones de espacio), la implementación de esta aplicación de acuerdo a los pasos desarrollados anteriormente. Se darán ejemplos de código únicamente de algunos contratos de operación del servicio *WCF*. Por ejemplo, para contratos de operación que realizan *selección*, solo se mostrará el contrato de operación *GetProductos (selecciona todos los productos).* Esto no significa que los contratos de operación que no se muestren sean totalmente diferentes en su implementación, con esto quiero decir que el contrato de operación *GetClientes* se implementa de forma similar al contrato de operación *GetProductos,* solo que enfocado a su respectiva tabla en la base de datos.

4.6.1 MAPEO Y SERIALIZACIÓN DE LA BASE DE DATOS CONTENIDA EN SQL SERVER PARA LINQ TO SQL

Antes que cualquier otra cosa se debe realizar el mapeo de la base de datos de la empresa *YnnafSoftware* contenida en *SQL Server 2005* para que podamos acceder a ella a través de *LINQ to SQL*. Los pasos que debemos seguir en *Visual Studio 2008* para conseguir esto son los que se muestran a continuación:

Agregar un nuevo elemento *LINQ to SQL Classes* a la solución a través del menú *Project -> Add New Item…*. En la ventana que se abre, establecer el nombre *YnnafSoftware.dbml* y dar clic en el botón *Add.*

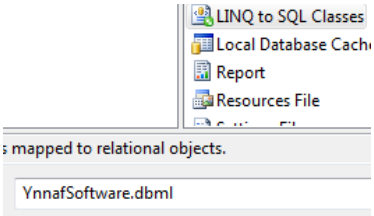


Figura 4.6 Agregado de un objeto *LINQ to SQL Classes*

A través del *Server Explorer* de *Visual Studio 2008*, conectar con la base de datos de *SQL Server 2005.* Debemos dar clic en el botón *Connect to Database*, en la ventana que aparece a continuación debemos llenar los campos necesarios: en *Data source* seleccionar *Microsoft SQL Server (SqlClient)*, en *Server Name* colocar la instancia de *SQL Server* donde se encuentra la base de datos, en *Log on to the server* seleccionar *Use SQL Server Authentication* e insertar el *nombre de usuario* y *contraseña* (se debe tener los permisos necesarios para leer la base de datos). Finalmente en *Connect to a database*, seleccionar *Select or enter a database name* e insertar el nombre de la base de datos del nuevo sistema, en este caso el nombre es *YnnafSoftware*.

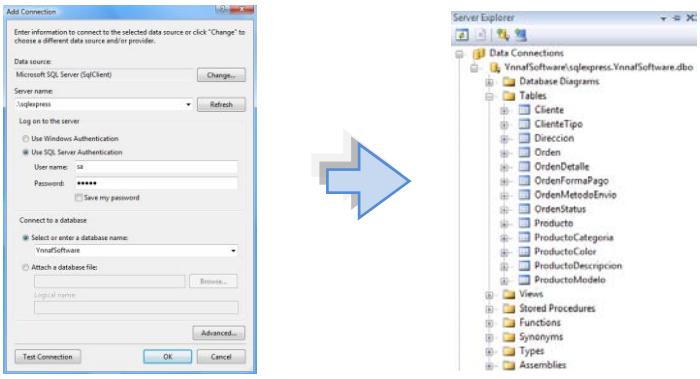


Figura 4.7 Conexión con la base de datos de SQL Server a través de Visual Studio 2008

Una vez que se conectó con la base de datos de *SQL Server* y se muestra todo su contenido (como se puede observar en la figura 4.7), seleccionar todas las tablas y arrastrarlas al diseñador relacional de objetos.

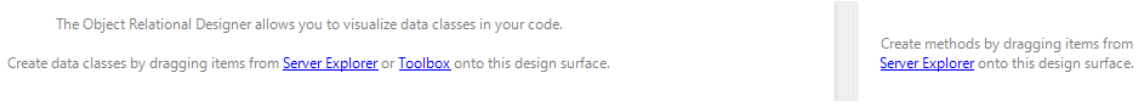


Figura 4.8 Diseñador Relacional de Objetos de LINQ to SQL

Nota: El arrastre de las tablas de la base de datos de *SQL Server* se debe hacer en la superficie de la izquierda (figura 4.8). La superficie de la derecha sirve para mapear procedimientos almacenados o funciones, sin embargo estos mapeos no son eficaces para procedimientos almacenaos que regresan conjuntos de resultados múltiples. Para ver como mapear los procedimientos almacenados con múltiples conjuntos de resultados (como son los que se usan en este sistema) de forma manual y correcta, referirse a *4.7.5 Creación en SQL Server, mapeo y ejecución en LINQ to SQL de procedimientos almacenados.*

Al finalizar el mapeo de las tablas de la base de datos *YnnafSoftware* del nuevo sistema se deberá observar un resultado como el siguiente:

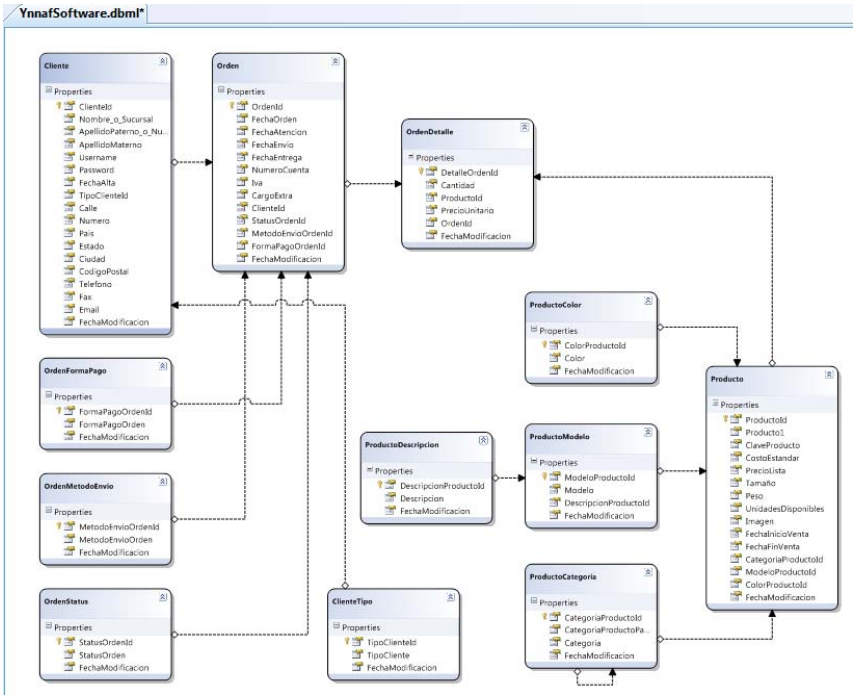


Figura 4.9 Resultado del mapeo de la base de datos por parte del Diseñador Relacional de Objetos

Una vez que se tiene el mapeo de la base de datos de *SQL Server*, debemos serializar todas las clases que representan las tablas de la base de datos para que sean funcionales con el servicio *WCF*. Esto lo haremos de la siguiente forma: Dar clic en el *Diseñador Relacional de Objetos*, ir a *propiedades* y cambiar el modo de serialización (*Serialization Mode*) de *None* a *Unidirectional* como se muestra en la siguiente imagen:

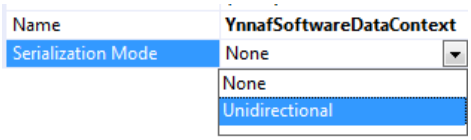


Figura 4.10 Serialización de las clases generadas por *el Diseñador Relacional de Objetos*

En los siguientes fragmentos de código fuente se puede ver la definición del código fuente antes y después de realizar la serialización necesaria para *Windows Communication Foundation*.

```
[Table(Name="dbo.Cliente")]
public partial class Cliente : INotifyPropertyChanging, INotifyPropertyChanged{
        [Column(Storage="_ClienteId", AutoSync=AutoSync.OnInsert,
             DbType="Int NOT NULL IDENTITY", IsPrimaryKey=true, IsDbGenerated=true)]
        public int ClienteId {
                get{ return this._ClienteId; }
                set{
                        if ((this._ClienteId != value)){
                                this.OnClienteIdChanging(value);
                                this.SendPropertyChanging();
                                this._ClienteId = value;
                                this.SendPropertyChanged("ClienteId");
                                this.OnClienteIdChanged();
                        }
                }
        }
        // Código con el resto del mapeo de la tabla Cliente
}
```

Código fuente sin serializar (no funcional con *WCF*) sin los atributos *DataContract* ni *DataMember*

```
[Table(Name="dbo.Cliente")]
[DataContract()]
public partial class Cliente : INotifyPropertyChanging, INotifyPropertyChanged{
        [Column(Storage="_ClienteId", AutoSync=AutoSync.OnInsert,
             DbType="Int NOT NULL IDENTITY", IsPrimaryKey=true, IsDbGenerated=true)]
        [DataMember(Order=1)]
        public int ClienteId {
                get{ return this._ClienteId; }
                set{
                        if ((this. ClienteId != value)){
                                this.OnClienteIdChanging(value);
                                this.SendPropertyChanging();
                                this._ClienteId = value;
                                this.SendPropertyChanged("ClienteId");
                                this.OnClienteIdChanged();
                        }
                }
        }
       // Código con el resto del mapeo de la tabla Cliente
}
```

Código fuente serializado (funcional con *WCF*) con los atributos *DataContract* y *DataMember*

4.6.2 CREACIÓN DEL CONTRATO DE SERVICIO

A continuación se muestra una versión reducida (para fines explicativos) de la creación del contrato de servicio que implementa la lógica del negocio del servicio *WCF*. Requiere usar *sesiones* como se vio en diseño.

```
using System;
using System.ServiceModel;
using YnnafSoftwareServiceWCF.DataContract.Faults;

namespace YnnafSoftwareServiceWCF {
    [ServiceContract(Namespace = "http://www.ynnafSoftware.com",
        Name = "YnnafSoftwareServiceWCF", SessionMode = SessionMode.Required)]
    public interface IYnnafSoftware_Principal : IYnnafSoftware_Productos,
        IYnnafSoftware_Ordenes, IYnnafSoftware_Clientes {
        [OperationContract(Name = "IniciarSesion", IsInitiating = true,
            IsTerminating = false)]
        [FaultContract(typeof(AutorizacionFault))]
        bool IniciarSesion(String username, String password);

        [OperationContract(Name = "FinalizarSesion", IsInitiating = false,
            IsTerminating = true)]
        [FaultContract(typeof(AutorizacionFault))]
        bool FinalizarSesion();
    }
}
```

En el código anterior, podemos ver que además de definir las firmas de los contratos de operación (*IniciarSesion* y *FinalizarSesion*) de la interfaz *IYnnafSoftware_Principal,* este contrato (*interfaz*) también implementa las interfaces *IYnnafSoftware_Productos*, *IYnnafSoftware_Ordenes*, *IYnnafSoftware_Clientes*; las firmas de los contratos de operación que contienen estas interfaces están relacionadas de acuerdo a los sufijos del nombre de la interfaz. La definición de estas interfaces es como se muestra a continuación:

```
[ServiceContract(Namespace = "http://www.ynnafSoftware.com",
    Name = "YnnafSoftwareServiceWCF", SessionMode = SessionMode.Required)]
public interface IYnnafSoftware_Productos {
        // Declaración de contratos de operación relacionados a Productos
}
```

```
[ServiceContract(Namespace = "http://www.ynnafSoftware.com",
    Name = "YnnafSoftwareServiceWCF", SessionMode = SessionMode.Required)]
public interface IYnnafSoftware_Ordenes {
        // Declaración de contratos de operación relacionados a Órdenes
}
```

```
[ServiceContract(Namespace = "http://www.ynnafSoftware.com",
    Name = "YnnafSoftwareServiceWCF", SessionMode = SessionMode.Required)]
public interface IYnnafSoftware_Clientes {
        // Declaración de contratos de operación relacionados a Clientes
}
```

Debe tomarse en cuenta que el correcto funcionamiento de estas interfaces depende de que, el atributo *ServiceContract* sea idéntico en las 4 interfaces.

4.6.3 IMPLEMENTACIÓN DE CONTRATOS DE OPERACIONES

La implementación del contrato de servicio (*interfaz*) que contiene la lógica del negocio de la empresa, se realiza como se pidió en el diseño del sistema. La implementación de los contratos de operación se hará haciendo uso de *LINQ to SQL* tal como se pidió en los requerimientos del sistema. Con el propósito de obtener un código fuente más fácil de mantener, esta implementación se realiza sobre clases parciales. En seguida se muestran las definiciones de las clases parciales que implementan el contrato de servicio:

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public partial class YnnafSoftwareService : IYnnafSoftware_Principal {
    // Implementación de contratos de operación de IYnnafSoftware_Principal
}
```

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public partial class YnnafSoftwareService : IYnnafSoftware_Productos {
    // Implementación de contratos de operación de IYnnafSoftware_Productos
}
```

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public partial class YnnafSoftwareService : IYnnafSoftware_Ordenes {
    // Implementación de contratos de operación de IYnnafSoftware_Ordenes
}
```

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
public partial class YnnafSoftwareService : IYnnafSoftware_Clientes {
    // Implementación de contratos de operación de IYnnafSoftware_Clientes
}
```

A continuación se muestran algunas implementaciones (con su correspondiente explicación) de contratos de operaciones que se usan a lo largo del servicio *WCF*. Se hace hincapié en dar una demostración de un contrato de operación que efectúa selección, inserción, actualización y eliminación usando *LINQ to SQL*. La demostración un contrato de operación para procedimientos almacenados se hará más adelante en esta tesis.

Implementación del contrato de operación que se debe llamar por los clientes del servicio *WCF* que requieren obtener una autorización (y nivel de acceso) a la lógica del negocio de la empresa.

```
private int nivelAcceso = 0;
private int idCliente = 0;

public bool IniciarSesion(String username, String password) {
    try {
```

```
        if (String.IsNullOrEmpty(username) || String.IsNullOrEmpty(password))
            throw new FaultException<AutorizacionFault>(new AutorizacionFault());
        ConnectionStringSettings settings;
        settings =
           ConfigurationManager.ConnectionStrings["AutorizacionUserConnectionString"];
        using (YnnafSoftwareDataContext db = new
           YnnafSoftwareDataContext(settings.ConnectionString)) {
            var match = from autorizacion in db.Clientes
                        join tipo in db.ClienteTipos on
                            autorizacion.TipoClienteId equals tipo.TipoClienteId
                        where autorizacion.Username.Equals(username) &&
                            autorizacion.Password.Equals(password)
                        select new {
                            NivelAcceso = autorizacion.TipoClienteId,
                            IdCliente = autorizacion.ClienteId
                        };
            if (match.Count() > 0) {
                foreach (var m in match) {
                    this.nivelAcceso = m.NivelAcceso;
                    this.idCliente = m.IdCliente;
                }
                return true;
            } else {
                throw new FaultException<AutorizacionFault>(new AutorizacionFault());
            }
        }
    } catch (SqlException ex) {
        FinalizarSesion();
        int numeroError = ex.Number;
        String mensajeError = ex.Message;
        String mensajeErrorFiltrado = new FiltroSqlServerError().
             GetError(ref numeroError, mensajeError);
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = mensajeErrorFiltrado,
            NumeroError = numeroError
        }, new FaultReason(mensajeErrorFiltrado));
    } catch (FaultException<AutorizacionFault>) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Login,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Login),
        }, new FaultReason(MensajeErrorStruct.Login));
    } catch (Exception ex) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Desconocido,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido)
        }, new FaultReason(ex.Message));
    }
}
```

En resumen, lo que hace este contrato de operación es lo siguiente: *1)* Verificar que el nombre de usuario y contraseña no estén vacíos, si lo están, regresar una *FaultException* al llamador, *2)* Obtener la cadena de conexión a la base de datos desde un archivo *app.config (ver 4.6.7)*, *3)* Conectar con la base de datos, *4)* Buscar en la base de datos de *SQL Server* usando *LINQ to SQL* el nombre de usuario y contraseña proporcionados, *5)* Si se encontró el nombre de usuario y contraseña asignar los campos *nivelAcceso* e *idCliente* con los valores obtenidos desde la base de datos y retornar un valor booleano verdadero (*true*); de lo contrario, retornar una *FaultException* al llamador del contrato de operación. Si ocurre un error en el proceso, los bloques *catch* retornaran al llamador de la operación una *FaultException* mostrando el error producido.

Implementación del contrato de operación que se debe llamar por los clientes del servicio *WCF* que requieren obtener una lista de productos desde el servicio *WCF*.

```
public List<Producto_Join> GetProductos() {
    ConnectionStringSettings settings;
```

```
    String cs = (nivelAcceso == 0) ? "AutorizacionUserConnectionString" :
        (nivelAcceso == 1) ? "YnnafSoftwareUserConnectionString" :
        (nivelAcceso == 2) ? "SucursalUserConnectionString" :
        (nivelAcceso == 3) ? "SitioWebUserConnectionString" : String.Empty;

    settings = ConfigurationManager.ConnectionStrings[cs];

    if (this.nivelAcceso == 0) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Login,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Login),
        }, new FaultReason(MensajeErrorStruct.Login));
    }

    using (YnnafSoftwareDataContext db = new
            YnnafSoftwareDataContext(settings.ConnectionString)) {
        try {
            var match = from producto in db.Productos
                        join productoCategoria in db.ProductoCategorias on
                             producto.CategoriaProductoId equals
                             productoCategoria.CategoriaProductoId
                        join productoModelo in db.ProductoModelos on
                             producto.ModeloProductoId equals
                             productoModelo.ModeloProductoId
                        join productoDescripcion in db.ProductoDescripcions on
                             productoModelo.DescripcionProductoId equals
                             productoDescripcion.DescripcionProductoId
                        join productoColor in db.ProductoColors on
                             producto.ColorProductoId equals
                             productoColorProductoId
                        select new Producto_Join
                        {
                            ProductoId = producto.ProductoId,
                            Producto = producto.Producto1,
                            ClaveProducto = producto.ClaveProducto,
                            Color = productoColor.Color,
                            ColorProductoId = producto.ColorProductoId,
                            CostoEstandar = producto.CostoEstandar,
                            PrecioLista = producto.PrecioLista,
                            Tamaño = producto.Tamaño,
                            Peso = producto.Peso,
                            Categoria = productoCategoria.Categoria,
                            CategoriaProductoId = producto.CategoriaProductoId,
                            Modelo = productoModelo.Modelo,
                            ModeloProductoId = producto.ModeloProductoId,
                            Descripcion = productoDescripcion.Descripcion,
                            UnidadesDisponibles = producto.UnidadesDisponibles,
                            Imagen = producto.Imagen,
                            FechaInicioVenta = producto.FechaInicioVenta,
                            FechaFinVenta = producto.FechaFinVenta,
                            FechaModificacion = producto.FechaModificacion
                        };
            return match.ToList();
        } catch (SqlException ex) {
            int numeroError = ex.Number;
            String mensajeError = ex.Message;
            String mensajeErrorFiltrado = new
                FiltroSqlServerError().GetError(ref numeroError, mensajeError);
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = mensajeErrorFiltrado,
                NumeroError = numeroError
            }, new FaultReason(mensajeErrorFiltrado));
        } catch (Exception) {
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = MensajeErrorStruct.Desconocido,
                NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido)
            }, new FaultReason(MensajeErrorStruct.Desconocido));
        }
    }
}
```

En resumen, lo que hace este contrato de operación es lo siguiente: *1)* Obtener con que cadena de conexión se va a conectar a la base de datos, esto depende del nivel de acceso obtenido en el contrato de operación *IniciarSesion*, *2)* Verificar el nivel de acceso, si éste es igual a cero, regresar al llamador del contrato de operación un *FaultException*, *3)* Conectar a la base de datos, *4)* A través de sentencias *Join* en *LINQ to SQL* obtener todos los productos contenidos en la base de datos de *SQL Server* y *5)* Retornar al llamador una lista genérica con los datos obtenidos en el anterior paso. Si ocurre un error en el proceso, los bloques *catch* retornaran al llamador de la operación una *FaultException* mostrando el error producido.

Implementación del contrato de operación que se debe llamar por los clientes del servicio *WCF* que requieren insertar nuevos productos a través del servicio *WCF*.

```
public int InsertProductos(List<Producto> listaProductos) {
    int registrosAfectados = 0;
    ConnectionStringSettings settings;

    String cs = (nivelAcceso == 0) ? "AutorizacionUserConnectionString" :
        (nivelAcceso == 1) ? "YnnafSoftwareUserConnectionString" :
        (nivelAcceso == 2) ? "SucursalUserConnectionString" :
        (nivelAcceso == 3) ? "SitioWebUserConnectionString" : String.Empty;

    settings = ConfigurationManager.ConnectionStrings[cs];

    if (this.nivelAcceso == 0 || this.nivelAcceso == 2 || this.nivelAcceso == 3) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Autorizacion,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Autorizacion),
        }, new FaultReason(MensajeErrorStruct.Autorizacion));
    }

    if (listaProductos.Count > 0) {
        try {
            using (YnnafSoftwareDataContext db = new
                    YnnafSoftwareDataContext(settings.ConnectionString)) {
                foreach (Producto producto in listaProductos) {
                    producto.FechaModificacion = DateTime.Now;
                    db.Productos.InsertOnSubmit(producto);
                    registrosAfectados++;
                }
                db.SubmitChanges(); // Guardar físicamente
                return registrosAfectados;
            }
        } catch (SqlException ex) {
            int numeroError = ex.Number;
            String mensajeError = ex.Message;
            String mensajeErrorFiltrado = new FiltroSqlServerError().GetError(
                ref numeroError, mensajeError);
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = mensajeErrorFiltrado,
                NumeroError = numeroError
            }, new FaultReason(mensajeErrorFiltrado));
        } catch (Exception ex) {
            throw new FaultException<ProductoFault>(new ProductoFault
            {
                Mensaje = MensajeErrorStruct.Desconocido,
                NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido)
            }, new FaultReason(MensajeErrorStruct.Desconocido));
        }
    } else {
        throw new FaultException<ProductoFault>(new ProductoFault {
            Mensaje = MensajeErrorStruct.ListaVacia,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.ListaVacia)
        }, new FaultReason(MensajeErrorStruct.ListaVacia));
    }
}
```

En resumen, lo que hace este contrato de operación es lo siguiente: *1)* Obtener la cadena de conexión con la que se va a conectar a la base de datos, esto depende del nivel de acceso obtenido en el contrato de operación *IniciarSesion*, *2)* Verificar el nivel de acceso, si éste es diferente de uno, regresar al llamador del contrato de operación un *FaultException*, *3)* Verificar que la lista de productos enviada al servicio WCF sea mayor a cero, de lo contrario regresar al llamador del contrato de operación un *FaultException 4)* Conectar a la base de datos, *5)* Iterar a través de todo el contenido de la lista genérica e insertar los nuevos productos en la base de datos de *SQL Server* usando *LINQ to SQL* y *6)* Retornar al llamador la cantidad de registros afectados. Si ocurre un error en el proceso, los bloques *catch* retornaran al llamador de la operación una *FaultException* mostrando el error producido.

Implementación del contrato de operación que se debe llamar por los clientes del servicio *WCF* que requieren actualizar productos existentes a través del servicio *WCF*.

```
public int UpdateProductos(List<Producto> listaProductos) {
    int registrosAfectados = 0;
    ConnectionStringSettings settings;

    String cs = (nivelAcceso == 0) ? "AutorizacionUserConnectionString" :
        (nivelAcceso == 1) ? "YnnafSoftwareUserConnectionString" :
        (nivelAcceso == 2) ? "SucursalUserConnectionString" :
        (nivelAcceso == 3) ? "SitioWebUserConnectionString" : String.Empty;

    settings = ConfigurationManager.ConnectionStrings[cs];

    if (this.nivelAcceso == 0 || this.nivelAcceso == 2 || this.nivelAcceso == 3) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Autorizacion,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Autorizacion),
        }, new FaultReason(MensajeErrorStruct.Autorizacion));
    }

    if (listaProductos.Count > 0) {
        List<Producto> match = new List<Producto>();
        try {
            using (YnnafSoftwareDataContext db = new
                    YnnafSoftwareDataContext(settings.ConnectionString)) {
                foreach (Producto idProducto in listaProductos) {
                    var query = from producto in db.Productos
                                where producto.ProductoId == idProducto.ProductoId
                                select producto;

                    foreach (Producto color in query) {
                        match.Add(color);
                    }
                }

                foreach (Producto anterior in match) {
                    Producto nuevo = listaProductos.Find(
                         p => p.ProductoId == anterior.ProductoId);
                    anterior.ProductoId = nuevo.ProductoId;
                    anterior.Producto1 = nuevo.Producto1;
                    anterior.ClaveProducto = nuevo.ClaveProducto;
                    anterior.CostoEstandar = nuevo.CostoEstandar;
                    anterior.PrecioLista = nuevo.PrecioLista;
                    anterior.Tamaño = nuevo.Tamaño;
                    anterior.Peso = nuevo.Peso;
                    anterior.UnidadesDisponibles = nuevo.UnidadesDisponibles;
                    anterior.Imagen = nuevo.Imagen;
                    anterior.FechaInicioVenta = nuevo.FechaInicioVenta;
                    anterior.FechaFinVenta = nuevo.FechaFinVenta;
                    anterior.CategoriaProductoId = nuevo.CategoriaProductoId;
                    anterior.ModeloProductoId = nuevo.ModeloProductoId;
```

```
                    anterior.ColorProductoId = nuevo.ColorProductoId;
                    anterior.FechaModificacion = DateTime.Now;
                    registrosAfectados++;
                }
                db.SubmitChanges(); // Guardar físicamente
                return registrosAfectados;
            }
        } catch (SqlException ex) {
            int numeroError = ex.Number;
            String mensajeError = ex.Message;
            String mensajeErrorFiltrado = new FiltroSqlServerError().
                GetError(ref numeroError, mensajeError);
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = mensajeErrorFiltrado,
                NumeroError = numeroError
            }, new FaultReason(mensajeErrorFiltrado));
        } catch (Exception ex) {
            System.Diagnostics.Debug.WriteLine(ex.Message);
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = MensajeErrorStruct.Desconocido,
                NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido)
            }, new FaultReason(MensajeErrorStruct.Desconocido));
        }
    } else {
        throw new FaultException<ProductoFault>(new ProductoFault {
            Mensaje = MensajeErrorStruct.ListaVacia,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.ListaVacia)
        }, new FaultReason(MensajeErrorStruct.ListaVacia));
    }
```

En resumen, lo que hace este contrato de operación es lo siguiente: *1)* Obtener la cadena de conexión con la que se va a conectar a la base de datos, esto depende del nivel de acceso obtenido en el contrato de operación *IniciarSesion*, *2)* Verificar el nivel de acceso, si éste es diferente de uno, regresar al llamador del contrato de operación un *FaultException*, *3)* Verificar que la lista de productos enviada al servicio WCF sea mayor a cero, de lo contrario regresar al llamador del contrato de operación un *FaultException 4)* Conectar a la base de datos, *5)* Iterar a través de todo el contenido de la lista genérica y actualizar los anteriores datos de los productos existentes con los nuevos en la base de datos de *SQL Server* usando *LINQ to SQL* y *6)* Retornar al llamador la cantidad de registros afectados. Si ocurre un error en el proceso, los bloques *catch* retornaran al llamador de la operación una *FaultException* mostrando el error producido.

Implementación del contrato de operación que se debe llamar por los clientes del servicio *WCF* que requieren eliminar productos existentes a través del servicio *WCF*.

```
public int DeleteProductos(List<int> listaIdProductos) {
    int registrosAfectados = 0;
    ConnectionStringSettings settings;

    String cs = (nivelAcceso == 0) ? "AutorizacionUserConnectionString" :
        (nivelAcceso == 1) ? "YnnafSoftwareUserConnectionString" :
        (nivelAcceso == 2) ? "SucursalUserConnectionString" :
        (nivelAcceso == 3) ? "SitioWebUserConnectionString" : String.Empty;

    settings = ConfigurationManager.ConnectionStrings[cs];

    if (this.nivelAcceso == 0 || this.nivelAcceso == 2 || this.nivelAcceso == 3) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Autorizacion,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Autorizacion),
        }, new FaultReason(MensajeErrorStruct.Autorizacion));
    }

    if (listaIdProductos.Count > 0) {
        try {
```

```
            using (YnnafSoftwareDataContext db = new YnnafSoftwareDataContext()) {
                try {
                    foreach (int id in listaIdProductos) {
                        Producto producto = db.Productos.Single(
                             p => p.ProductoId == id);
                        db.Productos.DeleteOnSubmit(producto);
                        registrosAfectados++;
                    }
                } catch (Exception) {
                    registrosAfectados = 0;
                }
                db.SubmitChanges(); // Guardar físicamente
                return registrosAfectados;
            }
        } catch (SqlException ex) {
            int numeroError = ex.Number;
            String mensajeError = ex.Message;
            String mensajeErrorFiltrado = new FiltroSqlServerError().
                GetError(ref numeroError, mensajeError);
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = mensajeErrorFiltrado,
                NumeroError = numeroError
            }, new FaultReason(mensajeErrorFiltrado));
        } catch (Exception ex) {
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = MensajeErrorStruct.Desconocido,
                NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido)
            }, new FaultReason(MensajeErrorStruct.Desconocido));
        }
    } else {
        throw new FaultException<ProductoFault>(new ProductoFault {
            Mensaje = MensajeErrorStruct.ListaVacia,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.ListaVacia)
        }, new FaultReason(MensajeErrorStruct.ListaVacia));
    }
}
```

En resumen, lo que hace este contrato de operación es lo siguiente: *1)* Obtener la cadena de conexión con la que se va a conectar a la base de datos, esto depende del nivel de acceso obtenido en el contrato de operación *IniciarSesion*, *2)* Verificar el nivel de acceso, si éste es diferente de uno, regresar al llamador del contrato de operación un *FaultException*, *3)* Verificar que la lista de productos enviada al servicio WCF sea mayor a cero, de lo contrario regresar al llamador del contrato de operación un *FaultException 4)* Conectar a la base de datos, *5)* Iterar a través de todo el contenido de la lista genérica y eliminar los productos existentes de acuerdo al Id de producto obtenido a través de la iteración de la base de datos de *SQL Server* usando *LINQ to SQL* y *6)* Retornar al llamador la cantidad de registros afectados. Si ocurre un error en el proceso, los bloques *catch* retornaran al llamador de la operación una *FaultException* mostrando el error producido.

4.6.4 CREACIÓN DE CONTRATOS DE DATOS

Los contratos de datos utilizados para el funcionamiento del servicio *WCF* son de tres tipos: A) *FaultException<T>*, B) *Joins* y C) *Procedimientos almacenados.*

Contratos de datos para sentencias Join

La utilidad de estos contratos de datos es la de mantener un código fuente más limpio al realizar una consulta en *LINQ to SQL*. A continuación se muestra la definición del contrato de operación *Producto_Join*, este contrato sirve para almacenar el resultado de una consulta que contiene sentencias *Join* que combinan todas las tablas relacionadas con productos, para ver el funcionamiento de esta contrato de datos ver: *Implementación del contrato de operación que se debe llamar por*

los clientes del servicio WCF que requieren actualizar productos existentes a través del servicio WCF en el subcapítulo 4.6.3 de esta tesis

```
[DataContract(
    Namespace = "http://www.ynnafSoftware.com/DataContract/Joins",
    Name = "Producto_Join")]
public class Producto_Join {
    [DataMember(Name = "ProductoId")]
    public int ProductoId { get; set; }
    [DataMember(Name = "Producto")]
    public String Producto { get; set; }
    [DataMember(Name = "PrecioLista")]
    public decimal PrecioLista { get; set; }
    [DataMember(Name = "CategoriaProductoId")]
    public int CategoriaProductoId { get; set; }
    [DataMember(Name = "Categoria")]
    public String Categoria { get; set; }
    [DataMember(Name = "ModeloProductoId")]
    public int ModeloProductoId { get; set; }
    [DataMember(Name = "Modelo")]
    public String Modelo { get; set; }
    [DataMember(Name = "Descripcion")]
    public String Descripcion { get; set; }
    [DataMember(Name = "ColorProductoId")]
    public int ColorProductoId { get; set; }
    [DataMember(Name = "Color")]
    public String Color { get; set; }
    // Otros DataMember() que cumplen con los valores deseados
}
```

Contratos de datos para FaultException<T>

La utilidad de estos contratos de datos es la de retornar excepciones *FaultException<T>* personalizadas al llamador de los contratos de operaciones cuando ocurre una excepción en el servicio *WCF*.

```
[DataContract(Namespace = "http://www.ynnafSoftware.com/DataContract/Faults",
    Name = "ClienteFault")]
public class ClienteFault {
    // Constructores
    public ClienteFault(int numeroError) { this.NumeroError = numeroError; }
    public ClienteFault(String mensaje) { this.Mensaje = mensaje; }
    public ClienteFault() {
        this.NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido);
        this.Mensaje = "Mensaje no especificado";
    }
    public ClienteFault(int numeroError, String mensaje) {
        this.NumeroError = numeroError;
        this.Mensaje = mensaje;
    }

    // Miembros de datos
    [DataMember(Name = "NumeroError")]
    public int NumeroError { get; set; }

    [DataMember(Name = "Mensaje")]
    public String Mensaje { get; set; }
}
```

En este contrato de operación podemos ver que contiene 2 miembros de datos (*DataMember)*, uno para regresar información sobre el número de error (*NumeroError*) y otro para regresar la descripción del error (*Mensaje*). Estos valores son obtenidos desde la enumeración *NumeroErrorEnum* y la estructura *MensajeErrorStruct* respectivamente.

Contratos de datos para Procedimientos almacenados

La utilidad de estos contratos de datos es la de retornar un conjunto de resultados al llamador de un contrato de operación que manipule un procedimiento almacenado. A continuación podemos observar un contrato de operación que sirve para almacenar todos los conjuntos de resultados (*Resultset*) retornados por el procedimiento almacenado *sp_GetProductos* almacenado en *SQL Server*. Cada conjunto de resultados corresponde con una lista genérica. Se debe notar que este contrato hace referencia a otros contratos de datos, como el contrato *Productos_Join* mencionado anteriormente.

```
[DataContract(
    Namespace = "http://www.ynnafSoftware.com/DataContract/StoredProcedures",
    Name = "Sp_Productos")]
public class Sp_Productos {
    [DataMember(Name = "ListaProductos")]
    public List<Producto_Join> ListaProductos { get; set; }
    [DataMember(Name = "ListaCategoriasProductos")]
    public List<ProductoCategoria_Join> ListaCategoriasProductos { get; set; }
    [DataMember(Name = "ListaModelosProductos")]
    public List<ProductoModelo_Join> ListaModelosProductos { get; set; }
    [DataMember(Name = "ListaDescripcionesProductos")]
    public List<ProductoDescripcion> ListaDescripcionesProductos { get; set; }
    [DataMember(Name = "ListaColoresProductos")]
    public List<ProductoColor> ListaColoresProductos { get; set; }
}
```

4.6.5 PROCEDIMIENTOS ALMACENADOS: CREACIÓN EN SQL SERVER, MAPEO Y EJECUCIÓN EN LINQ TO SQL

Aunque el servicio *WCF* utiliza *varios procedimientos almacenados*, en este apartado nos enfocaremos únicamente en la construcción del procedimiento almacenado que permite traer toda la información referente a los *productos* de la empresa buscados por nombre. El procedimiento almacenado el cual se llama *sp_GetProductosByNombre* obtiene los siguientes 5 conjuntos de resultados:

NOMBRE DE LA TABLA	CAMPOS PROPIOS A SELECCIONAR	CAMPOS (DE TABLAS EXTERNAS) QUE TAMBIÉN SE DEBEN SELECCIONAR
1. Producto	ProductoId, Producto, ClaveProducto, CostoEstandar, PrecioLista, Tamaño, Peso, UnidadesDisponibles, Imagen, FechaInicioVenta, FechaFinVenta, CategoriaProductoId, ModeloProductoId, ColorProductoId, Color, FechaModificacion	Categoria (ProductoCategoria), Modelo (ProductoModelo), Descripcion (ProductoDescripcion), Color (ProductoColor)
2. ProductoCategoria	CategoriaProductoId, CategoriaProductoPadreId, Categoria AS Categoria, FechaModificacion	Categoria AS CategoriaPadre (ProductoCategoria)
3. ProductoModelo	ModeloProductoId, Modelo, DescripcionProductoId, FechaModificacion	Descripcion (ProductoDescripcion)
4. ProductoDescripcion	TODOS LOS CAMPOS	NINGUNO
5. ProductosColor	TODOS LOS CAMPOS	NINGUNO

Además de los 5 conjuntos de resultados, el procedimiento almacenado también retorna un parámetro de salida indicando cuántos productos se encontraron de acuerdo al nombre del producto buscado. Por lo tanto, la creación de este procedimiento almacenado en *SQL Server* utilizando *TRANSACT-SQL* es como se muestra a continuación:

```
CREATE PROCEDURE sp_GetProductosByNombre
(
        @nombreProducto nvarchar(50),
        @productosEncontrados int = 0 output
```

```
)
AS
BEGIN

-- OBTENER PRODUCTOS
SELECT Producto.ProductoId, Producto.Producto, Producto.ClaveProducto,
 Producto.CostoEstandar, Producto.PrecioLista, Producto.Tamaño, Producto.Peso,
 Producto.UnidadesDisponibles, Producto.Imagen, Producto.FechaInicioVenta,
 Producto.FechaFinVenta, Producto.CategoriaProductoId, ProductoCategoria.Categoria,
 Producto.ModeloProductoId, ProductoModelo.Modelo, ProductoDescripcion.Descripcion,
 Producto.ColorProductoId, ProductoColor.Color, Producto.FechaModificacion
FROM Producto
       INNER JOIN ProductoCategoria
              ON Producto.CategoriaProductoId = ProductoCategoria.CategoriaProductoId
       INNER JOIN ProductoModelo
              ON Producto.ModeloProductoId = ProductoModelo.ModeloProductoId
       INNER JOIN ProductoDescripcion
              ON ProductoDescripcion.DescripcionProductoId =
                  ProductoModelo.DescripcionProductoId
       INNER JOIN ProductoColor
              ON Producto.ColorProductoId = ProductoColor.ColorProductoId
WHERE Producto.Producto LIKE '%' + @nombreProducto + '%'
ORDER BY Producto.Producto ASC;

-- OBTENER CATEGORIAS DE PRODUCTOS
SELECT ProductoCategoria1.CategoriaProductoId,
 ProductoCategoria1.CategoriaProductoPadreId,
 ProductoCategoria1.Categoria AS Categoria,
 ProductoCategoria2.Categoria AS CategoriaPadre,
 ProductoCategoria1.FechaModificacion
FROM ProductoCategoria AS ProductoCategoria1
       LEFT OUTER JOIN ProductoCategoria AS ProductoCategoria2
              ON ProductoCategoria1.CategoriaProductoPadreId =
                  ProductoCategoria2.CategoriaProductoId
ORDER BY Categoria ASC;

-- OBTENER MODELOS DE PRODUCTOS
SELECT ProductoModelo.ModeloProductoId, ProductoModelo.Modelo,
 ProductoModelo.DescripcionProductoId, ProductoDescripcion.Descripcion,
 ProductoModelo.FechaModificacion
FROM ProductoModelo INNER JOIN ProductoDescripcion
       ON ProductoModelo.DescripcionProductoId =
           ProductoDescripcion.DescripcionProductoId
ORDER BY ProductoModelo.Modelo ASC;

-- OBTENER DESCRIPCIONES DE PRODUCTOS
SELECT * FROM ProductoDescripcion
ORDER BY ProductoDescripcion.Descripcion ASC;

-- OBTENER COLORES DE PRODUCTOS
SELECT * FROM ProductoColor
ORDER BY ProductoColor.Color ASC;

-- OBTENER LA CANTIDAD DE PRODUCTOS ENCONTRADOS
SET @productosEncontrados =
       (SELECT COUNT(*) FROM Producto WHERE Producto.Producto LIKE '%' +
           @nombreProducto + '%');
END
```

Como se comentó en el apartado *1.4.2.1 Linq para Sql (Linq to Sql)* de esta tesis, el *diseñador relacional de objetos* de *Visual Studio 2008* no es capaz de mapear adecuadamente un *procedimiento almacenado* con más de un *conjunto de resultados*, por lo que este trabajo le toca hacerlo manualmente al programador del sistema.

Cuando se requiere obtener datos de tablas sin la combinación de algunas de ellas, basta con usar la clase generada por el *Diseñador Relacional de Objetos;* sin embargo, al utilizar la cláusula *JOIN* en las consultas del procedimiento almacenado y *Windows Communication Foundation* debemos

crear nuestras propias clases, capaces de almacenar los datos obtenidos como resultado de la consulta *SELECT-FROM-JOIN-ON*.

La declaración de estas clases se hace serializándolas con el atributo *DataContract*, a su vez también serializamos sus miembros con el atributo *DataMember*. A continuación se muestra la definición y serialización (necesaria para *WCF*) de la clase que servirá para almacenar los datos del tercer conjunto de datos del procedimiento almacenado declarado anteriormente:

```
[DataContract(
    Namespace = "http://www.ynnafSoftware.com/DataContract/Joins",
    Name = "ProductoModelo_Join")]
public class ProductoModelo_Join {
    [DataMember(Name= "ModeloProductoId")]
    public int ModeloProductoId { get; set; }
    [DataMember(Name = "Modelo")]
    public String Modelo { get; set; }
    [DataMember(Name = "DescripcionProductoId")]
    public int DescripcionProductoId { get; set; }
    [DataMember(Name = "Descripcion")]
    public String Descripcion { get; set; }
    [DataMember(Name = "FechaModificacion")]
    public DateTime FechaModificacion { get; set; }
}
```

Cosas importantes que podemos observar aquí son:

- ✓ El nombre de la clase es *ProductoModelo_Join* (podría haber sido cualquier otro). Servirá para almacenar los datos provenientes de la tercera consulta que se ejecuta en el procedimiento almacenado. Esta clase está serializada a través del atributo *DataContract*.
- ✓ El nombre de cada una de las propiedades debe ser idéntico al nombre del campo de la tabla en la base de datos (ver tabla: *Campos a seleccionar en el procedimiento almacenado*) para realizar un correcto acoplamiento entre los datos. Todas las propiedades están serializadas por medio del atributo *DataMember*.

Una vez que se declararon y serializaron los contratos de datos (*clases*) para contener los resultados provenientes de las consultas con sentencias *JOIN*, se procede a declarar el método que servirá para llamar al procedimiento almacenado en la base de datos. Utilizando la clase parcial *YnnafSoftwareDataContext* resultante del mapeado de la base de datos, creamos un nuevo archivo de *C#* e implementaremos en la clase parcial *YnnafSoftwareDataContext* un método al que llamaremos *Sp_GetProductosByNombre*, el cual deberá contener los siguientes atributos:

- ✓ Function: Con la propiedad *Name* para indicar el nombre del procedimiento almacenado, tal cual esta declarado en la base de datos de *SQL Server*.
- ✓ ResultType: A este atributo se le debe asignar el tipo (*clase*) que contendrá los resultados devueltos por las consultas ejecutadas en el procedimiento almacenado. El orden debe ser el mismo en el que se ejecutan en el procedimiento almacenado.

El tipo devuelto por el método, y la implementación del mismo es un patrón que sigue *LINQ to SQL* para ejecutar procedimientos almacenados que devuelven múltiples conjuntos de resultados.

Los parámetros del procedimiento almacenado deben contener el atributo *Parameter* con la propiedad *DbType* asignada al tipo de dato utilizado en *SQL Server* (como *String)*. El tipo del parámetro debe ser equivalente en *C#* y *Transact-SQL.* El nombre de los parámetros debe ser el mismo que en el procedimiento almacenado (si no se especifica en la propiedad *Name* del atributo *Para-*

meter el nombre del parámetro*)*. Si el parámetro del procedimiento almacenado es de salida, en *C#* se deberá pasar por referencia (*ref*).

La declaración del método capaz de ejecutar el procedimiento almacenado es la siguiente:

```
public partial class YnnafSoftwareDataContext : DataContext {
    [Function(Name = "dbo.sp_GetProductosByNombre")]
    [ResultType(typeof(Producto_Join))]
    [ResultType(typeof(ProductoCategoria_Join))]
    [ResultType(typeof(ProductoModelo_Join))]
    [ResultType(typeof(ProductoDescripcion))]
    [ResultType(typeof(ProductoColor))]
    public IMultipleResults Sp_GetProductosByName(
        [Parameter(DbType = "NVarChar(50)")] String nombreProducto,
        [Parameter(DbType = "Int")] ref int productosEncontrados) {
        IExecuteResult result = this.ExecuteMethodCall(this,
            (MethodInfo)MethodInfo.GetCurrentMethod(), new Object[] {
                // Pasar los parámetros al procedimiento.
                nombreProducto, productosEncontrados
            });
        // Obtener el valor del parámetro 1 (parámetros en base cero),
        // es decir, el parámetro de salida del procedimiento
        productosEncontrados = Convert.ToInt32(result.GetParameterValue(1));
        return (IMultipleResults)result.ReturnValue;
    }
}
```

Sin embargo, la interfaz *IMultipleResults* al no ser serializable no puede utilizarse directamente con el servicio *WCF* para transferir la información a los clientes del servicio. Para solucionar esto haremos uso del contrato de datos *Sp_Productos* (para ver la definición de esta clase, consultar: *4.6.4 Creación de contratos de datos, Contratos de datos para Procedimientos almacenados*) para retornar la información al llamador del contrato de operación. El siguiente código muestra como retornar el resultado generado por el procedimiento almacenado al llamador:

```
public Sp_Productos Sp_GetProductosByNombre(String nombreProducto,
    ref int productosEncontrados) {
    ConnectionStringSettings settings;

    String cs = (nivelAcceso == 0) ? "AutorizacionUserConnectionString" :
        (nivelAcceso == 1) ? "YnnafSoftwareUserConnectionString" :
        (nivelAcceso == 2) ? "SucursalUserConnectionString" :
        (nivelAcceso == 3) ? "SitioWebUserConnectionString" : String.Empty;

    settings = ConfigurationManager.ConnectionStrings[cs];

    if (this.nivelAcceso == 0) {
        throw new FaultException<AutorizacionFault>(new AutorizacionFault {
            Mensaje = MensajeErrorStruct.Login,
            NumeroError = Convert.ToInt32(NumeroErrorEnum.Login),
        }, new FaultReason(MensajeErrorStruct.Login));
    }

    using (YnnafSoftwareDataContext db = new
        YnnafSoftwareDataContext(settings.ConnectionString)) {
        try {
            IMultipleResults multipleResults =
                db.Sp_GetProductosByName(nombreProducto, ref productosEncontrados);
            Sp_Productos sp_productos = new Sp_Productos();

            var queryProductos = multipleResults.GetResult<Producto_Join>();
            var queryProductoCategoria =
                multipleResults.GetResult<ProductoCategoria_Join>();
            var queryProductoModelo = multipleResults.GetResult<ProductoModelo_Join>();
            var queryProductoDescripcion =
                multipleResults.GetResult<ProductoDescripcion>();
```

```
            var queryProductoColor = multipleResults.GetResult<ProductoColor>();

            sp_productos.ListaProductos = queryProductos.ToList();
            sp_productos.ListaCategoriasProductos = queryProductoCategoria.ToList();
            sp_productos.ListaModelosProductos = queryProductoModelo.ToList();
            sp_productos.ListaDescripcionesProductos = queryProductoDescripcion.ToList();
            sp_productos.ListaColoresProductos = queryProductoColor.ToList();

            return sp_productos;
        } catch (SqlException ex) {
            int numeroError = ex.Number;
            String mensajeError = ex.Message;
            String mensajeErrorFiltrado = new FiltroSqlServerError().
                GetError(ref numeroError, mensajeError);
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = mensajeErrorFiltrado,
                NumeroError = numeroError
            }, new FaultReason(mensajeErrorFiltrado));
        } catch (Exception) {
            throw new FaultException<ProductoFault>(new ProductoFault {
                Mensaje = MensajeErrorStruct.Desconocido,
                NumeroError = Convert.ToInt32(NumeroErrorEnum.Desconocido)
            }, new FaultReason(MensajeErrorStruct.Desconocido));
        }
    }
}
```

En resumen, lo que hace este contrato de operación es lo siguiente: *1)* Obtener la cadena de conexión con la que se va a conectar a la base de datos, esto depende del nivel de acceso obtenido en el contrato de operación *IniciarSesion*, *2)* Verificar el nivel de acceso, si éste es igual a cero, regresar al llamador del contrato de operación un *FaultException*, *3)* Conectar a la base de datos, *4)* Llamar al método (contrato de operación) que accede al procedimiento almacenado de *SQL Server*, pasándole los parámetros *nombreProducto* y *productosEncontrados* (este último por referencia) *5)* Construir una instancia del contrato de datos que almacenará los conjuntos de resultados devueltos por el procedimiento almacenado, *6)* Obtener cada uno de los conjuntos de resultados y almacenarlos en variables de inferencia, *7)* Asignar las propiedades del contrato de datos a resultados obtenidos (variables inferidas) y *8)* Retornar el contrato de datos al llamador. Si ocurre un error en el proceso, los bloques *catch* retornaran al llamador de la operación una *FaultException* mostrando el error producido.

4.6.6 CERTIFICADOS X.509: CREACIÓN E INSTALACIÓN DE LOS CERTIFICADOS YNNAFSOFTWARESERVICE.PFX E YNNAFSOFTWARECLIENT.PFX

Dado que en los requerimientos se necesita un *certificado X.509* (como medio de seguridad) que se utilizará como credencial, crearemos uno para usarse sobre el servicio *WCF*. El archivo *PFX (**P**ersonal In**f**ormation E**x**change)* que obtendremos (con la llave privada y pública) será el que le demos a los distintos clientes para autenticarse. A continuación se describen los pasos para crear este certificado usando *Visual Studio 2008 Command Prompt* Incluido junto con la instalación de *Visual Studio 2008*:

1. Abrir el programa *Visual Studio 2008 Enviroment Prompt,* generalmente ubicado en: *C:\Archivos de programa\Microsoft Visual Studio 9.0\VC.*
2. Cambiar el directorio a *C:*\.
3. Insertar la siguiente instrucción:
   makecert.exe -n "CN=YnnafSoftwareCA" -r -sv YnnafSoftwareCA.pvk YnnafSoftwareCA.cer -pe

donde:

-**n** *x509name*: Especifica el nombre de certificado del sujeto. Este nombre debe cumplir la norma X.500. El método más sencillo consiste en especificar el nombre entre comillas, precedido de *CN=*; por ejemplo, "*CN=MiNombreCA*".

-r: Especifica que el certificado se firmará automáticamente.

-sv *privateKeyFile:* Especifica el archivo que contiene el contenedor de clave privada.

-pe: Marca como exportable la clave privada generada. Esto permite que dicha clave se incluya en el certificado.

4. Insertar contraseña (2) de la clave privada. En este caso será *YnnafSoftwareCA* y dar clic en *Aceptar.*
5. Volver a escribir la contraseña: *YnnafSoftwareCA* y dar clic en el botón *Aceptar.*
6. Insertar la siguiente instrucción:
   makecert.exe -sr LocalMachine -ss My -a sha1 -n "CN=YnnafSoftwareService" -sky exchange -iv YnnafSoftwareCA.pvk -ic YnnafSoftwareCA.cer -pe
   donde:

   -**sr** *location*: Especifica la ubicación del almacén de certificados del sujeto. *Location* puede ser *currentuser* (valor predeterminado) o *localmachine*.

   -**ss** *store*: Especifica el nombre del almacén de certificados del sujeto que almacena el certificado de salida.

   -**a** *algorithm*: Especifica el algoritmo de firma. Debe ser *md5* (valor predeterminado) o *sha1*.

   -**n** *x509name*: Especifica el nombre de certificado del sujeto. Este nombre debe cumplir la norma *X.500*. El método más sencillo consiste en especificar el nombre entre comillas, precedido de *CN=*; por ejemplo, "*CN=MiNombre*".

   -**sky** *keytype*: Especifica el tipo de clave del sujeto, que debe ser *signature*, *exchange* o un número entero que represente un tipo de proveedor. De forma predeterminada, se puede pasar *1* para una clave de intercambio y *2* para una clave de firma.
7. Insertar la contraseña (*YnnafSoftwareCA*) que se insertó en el paso 4 (Firma del emisor) y dar clic en el botón *Aceptar*. Si el proceso se ejecutó correctamente aparecerá el mensaje: *Succeeded* en la consola*.*
8. Insertar la siguiente instrucción:
   certmgr.exe -add -r LocalMachine -s My -c -n YnnafSoftwareService -r LocalMachine -s My
   donde:

   -add: Agrega certificados, listas *CTL* y listas *CRL* a un almacén de certificados.

   -r *registryLocation*: Identifica la ubicación del Registro del almacén de sistema. Esta opción sólo se tiene en cuenta si se especifica la opción *-s*. El parámetro *registrylocation* ha de tomar alguno de los valores siguientes: *currentUser* indica que el almacén de certificados está bajo la clave *HKEY_CURRENT_USER*. Éste es el valor predeterminado o *localMachine* indica que el almacén de certificados está bajo la clave *HKEY_LOCAL_MACHINE*.

   -s: Indica que el almacén de certificados es un almacén de sistema. Si no se especifica esta opción, el almacén es del tipo *StoreFile*.

   -c: Agrega certificados cuando se utiliza con la opción *-add*. Elimina certificados cuando se utiliza con la opción *-delete*. Guarda certificados cuando se utiliza con la opción *-put*. Muestra certificados cuando se utiliza sin las opciones *-add*, *-delete* y *-put*.

-n *commonNameString*: Especifica el nombre común del certificado que se agrega, elimina o guarda. Esta opción sólo se puede utilizar con certificados; no se puede utilizar con listas *CTL* y listas *CRL*.

Si el proceso se ejecutó correctamente aparecerá el mensaje: *CertMgr Succeeded*

9. Abrir la *Microsoft Managament Console (mmc.exe).*
10. En *Certificados -> Equipo local*, dar clic derecho sobre el certificado recién agregado seleccionar *Todas las tareas -> Exportar*.
11. En la ventana *Asistente para exportación de certificados* dar clic en el botón *Siguiente*.
12. Seleccionar *Exportar la clave privada* y dar clic en el botón *Siguiente*.
13. De forma obligatoria está seleccionada la opción: *Intercambio de información personal: PKCS #12 (.PFK)*, también se debe seleccionar: *Si es posible, incluir todos los certificados en la ruta de acceso de certificación* y dar clic en el botón *Siguiente*.
14. Insertar la contraseña (2) para la protección de la llave privada. En este caso será *YnnafSoftwareService*. Dar clic en el botón *Siguiente*.
15. Especificar la ruta de exportación. Se hará en *C:\YnnafSoftwareService.pfx*, dar clic en *Siguiente*.
16. Dar clic en el botón *Finalizar* después de verificar que los datos son correctos.
17. Dar clic en el botón *Aceptar* cuando se avise que la operación de exportación se realizó con éxito.
18. Nuevamente en la *Microsoft Managament Console*, eliminar el certificado *YnnafSoftwareService* a través del cual exportamos el archivo *PFX*.
19. Dar clic derecho sobre *Certificados* en *Certificados (Equipo local) -> Personal*, seleccionar *Todas las tareas -> Importar*
20. En la ventana del asistente de importación de certificados dar clic en el botón *Siguiente*
21. Buscar el archivo *C:\YnnafSoftwareService.pfx* asignarlo en el campo del nombre del archivo y dar clic en el botón *Siguiente*.
22. Asignar la contraseña de la llave privada y dar clic en el botón *Siguiente*
23. Seleccionar *Colocar todos los certificados en el siguiente almacén: Personal* y dar clic en *Siguiente*.
24. Dar clic en el botón *Finalizar* después de verificar que los datos son correctos.
25. Dar clic en el botón *Aceptar* cuando se avise que la operación de importación se realizó con éxito.

Una vez finalizada la creación e instalación del *certificado X.509* en el servidor, podemos crear el certificado que se le distribuirá a os clientes. Los pasos a seguir son los siguientes: Repetir desde paso 6 al 25 del apartado *4.7.6 Creación e instalación del certificado X.509 del lado del servidor*. Cambiar todos los nombre *YnnafSoftwareService* por *YnnafSoftwareClient*, por ejemplo, la instrucción del paso 6 cambia a: *makecert.exe -sr LocalMachine -ss My -a sha1 -n "CN=YnnafSoftwareClient" -sky exchange -iv YnnafSoftwareCA.pvk -ic YnnafSoftwareCA.cer –pe* y la instrucción del paso 8 cambia a: *certmgr.exe -add -r LocalMachine -s My -c -n YnnafSoftwareClient -r LocalMachine -s My*. El certificado *X.509* que se distribuirá a los clientes *WCF* será el que se encuentre en el archivo *C:\YnnafSoftwareClient.pfx*. A los clientes *WSIT* también se les deberá dar el certificado *X.509* en el archivo *C:\YnnafSoftwareService.pfx.*

4.6.7 CONFIGURACIÓN DE LA EXPOSICIÓN DEL SERVICIO WCF

A continuación se muestra como se debe definir el archivo de configuración *app.config* para exponer el servicio *WCF (*de forma auto hospedada para fines de desarrollo*)*. Este archivo también con-

tiene las cadenas de conexión utilizadas para conectar con *SQL Server*, estas cadenas se encuentran en este archivo como medida de seguridad en caso de ofuscación del ensamblado del servicio *WCF*.

```
<!-- Archivo de configuración App.config del Servicio WCF -->
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <!-- Cadenas de conexión para los 4 diferentes usuarios del servicio WCF -->
  <connectionStrings>
    <add name="YnnafSoftwareUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=YnnafSoftwareUser;Password=YnnafSoftwarePass"
         providerName="System.Data.SqlClient" />
    <add name="SucursalUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=SucursalUser;Password=SucursalPass"
         providerName="System.Data.SqlClient" />
    <add name="SitioWebUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=SitioWebUser;Password=SitioWebPass"
         providerName="System.Data.SqlClient" />
    <add name="AutorizacionUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=AutorizacionUser;Password=AutorizacionPass"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <!-- Configuración del enlace WsHttpBinding para el servicio WCF -->
    <bindings>
      <wsHttpBinding>
        <binding name="EnlaceWsHttp" bypassProxyOnLocal="true" useDefaultWebProxy="false">
          <!-- Indicar la protección del mensaje a través de certificado X.509 -->
          <security mode="Message">
            <transport clientCredentialType="None" />
            <message clientCredentialType="Certificate" negotiateServiceCredential="false"
                     algorithmSuite="Basic128" establishSecurityContext="true" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <!-- Comportamiento del servicio WCF -->
    <behaviors>
      <serviceBehaviors>
        <behavior name="ComportamientoMetadatos">
          <!-- Exponer el WSDL, asignar a "false" cuando ya no se necesite exponer -->
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceCredentials>
            <!-- Declaración del certificado X.509 para autenticar al cliente -->
            <clientCertificate>
              <certificate findValue="YnnafSoftwareClient" storeLocation="LocalMachine"
                           storeName="My" x509FindType="FindBySubjectName" />
              <authentication certificateValidationMode="None" />
            </clientCertificate>
            <!-- Declaración del certificado X.509 para el servidor -->
            <serviceCertificate findValue="YnnafSoftwareService"
                                storeLocation="LocalMachine"
                                storeName="My" x509FindType="FindBySubjectName" />
          </serviceCredentials>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <!-- Declaración de las características del servicio WCF -->
    <services>
```

```
      <service behaviorConfiguration="ComportamientoMetadatos"
               name="YnnafSoftwareServiceWCF.YnnafSoftwareService_Principal">
        <!-- Declaración del enlace seguro WsHttpBinding -->
        <!-- La propiedad bindingNamespace es obligatoria para que el
             servicio funcione en Java en Windows Vista -->
        <endpoint address="wsYnnafSoftwareService"
                  contract="YnnafSoftwareServiceWCF.IYnnafSoftware_Principal"
                  name="WsHttpBinding_YnnafSoftwareService"
                  bindingNamespace="http://localhost:80/servicioWCF"
                  bindingName="YnnafSoftwareService"
                  bindingConfiguration="EnlaceWsHttp"
                  binding="wsHttpBinding">
          <identity>
            <!-- La identidad del usuario de acuerdo al certificado X.509 -->
            <certificateReference findValue="YnnafSoftwareService"
                                  storeLocation="LocalMachine"
                                  storeName="My" x509FindType="FindBySubjectName"/>
          </identity>
        </endpoint>
        <!-- Enlace que expone los metadatos -->
        <endpoint contract="IMetadataExchange" binding="mexHttpBinding" address="mex" />
        <host>
          <!-- Declaración de la dirección donde se aloja el servicio WCF -->
          <baseAddresses>
            <add baseAddress="http://localhost:8080/servicioWCF/YnnafSoftwareService"/>
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

En el archivo anterior se puede ver que está compuesto por los siguientes elementos principales:

- ✓ connectionStrings: Almacena las cadenas de conexión a través de las cuales los cuatro distintos clientes de la empresa acceden a la base de datos de *SQL Server*. Esta es una forma segura de mantener cadenas de conexión a bases de datos.
- ✓ system.serviceModel.bindings: Aquí se controla lo relativo a la configuración del enlace, es decir, en esta sección colocamos explícitamente que el servicio *WCF* utilizará seguridad sobre el *Mensaje* utilizando *certificados X.509*.
- ✓ system.serviceModel.behaviors: En esta sección se indica cuando se va a exponer o no el *WSDL* del servicio; además, se indica cuales son los certificados usados para la autenticación del cliente y cuál es el certificado que usará el servicio *WCF*.
- ✓ system.serviceModel.services: Se declara el extremo del servicio (y como se va a comportar el enlace), cómo identificar al cliente y cuál es la dirección base del servicio *WCF*. Esta dirección base es necesaria para fines de desarrollo (para auto hospedar el servicio *WCF*). En un ambiente de producción, esta dirección la eliminaremos como veremos en *4.8.1 Configuración de la aplicación web que publicará el servicio Windows Communication Foundation en Internet Information Services (IIS) 7.0*.

4.7 DESPLIEGUE

A continuación se muestra los pasos a seguir para configurar, levantar y ejecutar el servicio *Windows Communication Foundation* en *IIS 7.0.*

4.7.1 CONFIGURACIÓN DE LA APLICACIÓN WEB QUE PUBLICARÁ EL SERVICIO WINDOWS COMMUNICATION FOUNDATION EN INTERNET INFORMATION FOUNDATION (IIS) 7.0

Una vez que hemos finalizado de escribir el servicio *Windows Communication Foundation* (*WCF*), debemos construir el ensamblado del proyecto (presionando *F6* en *Visual Studio 2008*), el cual contendrá toda la funcionalidad de la lógica de negocio de la empresa. Ya que tenemos el ensamblado construido (archivo *YnnafSoftwareServiceWCF.dll*), debemos publicarlo a través del *Internet Information Services* (*IIS*) *7.0*. Los pasos que debemos seguir son los siguientes:

Crear un nuevo proyecto en *Visual Studio 2008*. Ir al menú *File -> New -> Project… -> Visual C# -> Web -> WCF Service Application* -> Presionar el botón *OK*

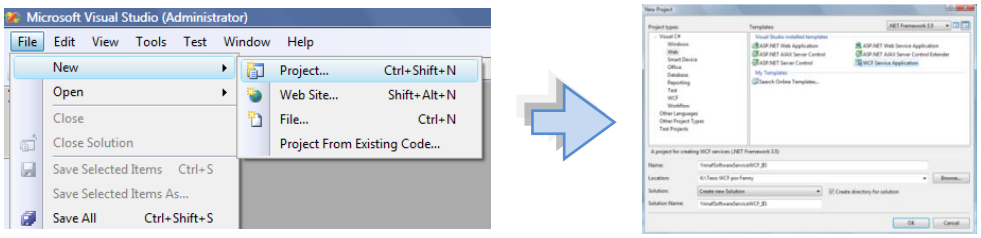


Figura 4.11 Creación de Aplicación Web para publicación del servicio WCF.

Esto generará un proyecto con los siguientes elementos: directorio *App_Data*, *interfaz IService1.cs*, clase *Service1.svc* y archivo de configuración *Web.config*. De entre todos estos elementos, debemos eliminar la interfaz *IService1.cs* y el directorio *App_Data.*

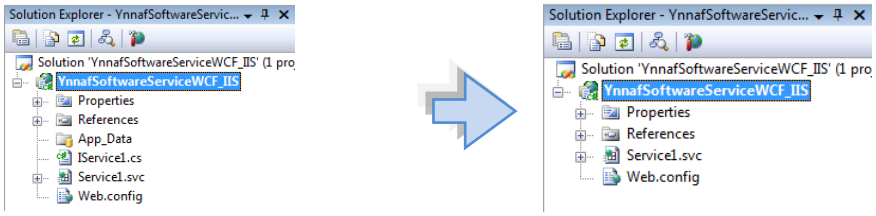


Figura 4.12 Archivos necesarios para la publicación del servicio WCF

Agregar la referencia al servicio *WCF* contenido en el ensamblado *YnnafSoftwareServiceWCF.dll*

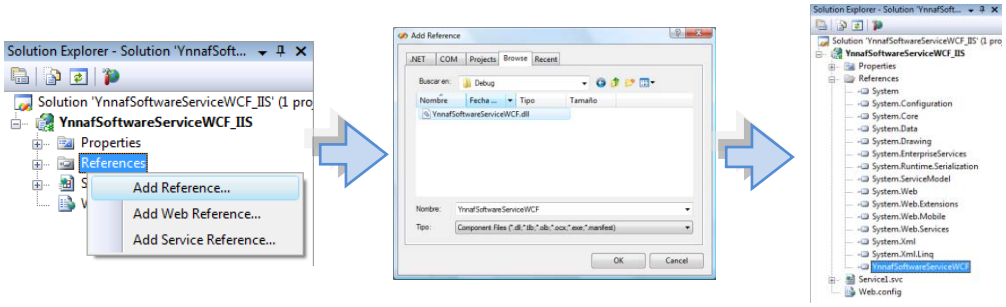


Figura 4.13 Referencia al servicio WCF (YnnafSoftwareServiceWCF.dll)

El archivo *Service1.svc* lo renombraremos (*Rename*) a *YnnafSoftwareService.svc* y editaremos su código *XML* dando clic derecho sobre él y seleccionando *View Markup*:

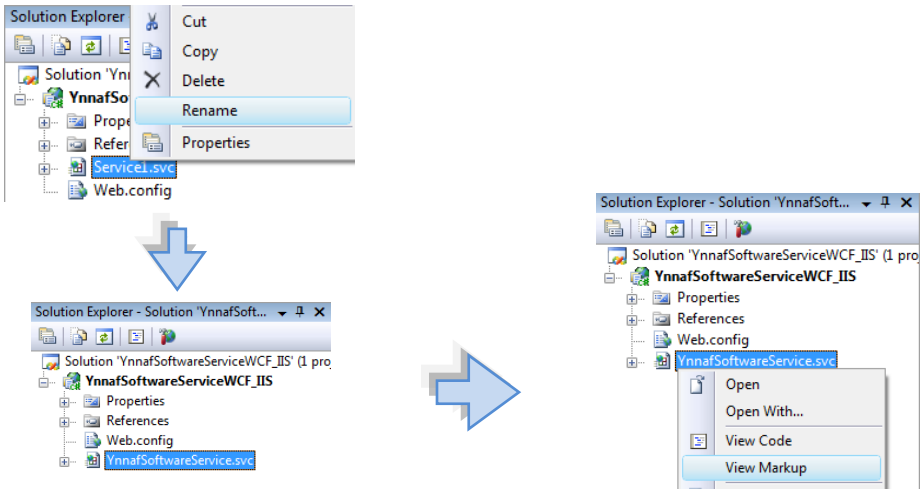


Figura 4.14 Renombramiento de archivo YnnafSoftwareService.svc y edición de su código XML

El código que veremos después de realizar los pasos anteriores es el siguiente:

```
<%@ ServiceHost Language="C#" Debug="true" Service="YnnafSoftwareServiceWCF_IIS.Service1"
CodeBehind="Service1.svc.cs" %>
```

Este código *XML* lo debemos modificar de la siguiente manera para que se adapte a las características de nuestro servicio *WCF*. En el elemento *Service* colocaremos el espacio de nombres (*namespace*) y la clase que implementa el contrato de servicio (*IYnnafSoftwareService*), los elementos *Language*, *Debug* y *CodeBehind* los eliminaremos porque no los necesitamos para la ejecución. El resultado de esta modificación será:

```
<%@ ServiceHost Service="YnnafSoftwareServiceWCF.YnnafSoftwareService" %>
```

Además de la modificación al código *XML*, editaremos el archivo *YnnafSoftwareService.svc.cs* para que también se adapte a las características del servicio *WCF.* Abrimos el archivo de *C#* y eliminamos la indicación de implementación de la interfaz *IService1* y los métodos que implementa, es decir, los métodos *GetData* y *GetDataUsingDataContract* A continuación se muestra el código que se debe editar:

```
using System;

namespace YnnafSoftwareServiceWCF_IIS {
    public class YnnafSoftwareService : IService1 {
        public String GetData(int value) {
            return string.Format("You entered: {0}", value);
        }

        public CompositeType GetDataUsingDataContract(CompositeType composite) {
            if (composite.BoolValue) {
                composite.StringValue += "Suffix";
            }
            return composite;
        }
    }
}
```

El resultado final de la edición del archivo *YnnafSoftwareService.svc.cs* (código anterior) deberá ser como se muestra a continuación:

```
using System;

namespace YnnafSoftwareServiceWCF_IIS {
    public class YnnafSoftwareService { }
}
```

En el archivo *Web.config* eliminaremos todo su contenido (el que se creó de forma predeterminada) y lo reemplazaremos por el casi todo el código del archivo *app.config* del servicio *WCF*. El siguiente código *XML* será el que coloca en el archivo:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <!-- Cadenas de conexión para los 4 diferentes usuarios del servicio WCF -->
  <connectionStrings>
    <add name="YnnafSoftwareUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=YnnafSoftwareUser;Password=YnnafSoftwarePass"
         providerName="System.Data.SqlClient" />
    <add name="SucursalUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=SucursalUser;Password=SucursalPass"
         providerName="System.Data.SqlClient" />
    <add name="SitioWebUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=SitioWebUser;Password=SitioWebPass"
         providerName="System.Data.SqlClient" />
    <add name="AutorizacionUserConnectionString"
         connectionString="Data Source=.\sqlexpress;Initial Catalog=YnnafSoftware;User
                           ID=AutorizacionUser;Password=AutorizacionPass"
         providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <!-- Configuración del enlace WsHttpBinding para el servicio WCF -->
    <bindings>
      <wsHttpBinding>
        <binding name="EnlaceWsHttp" bypassProxyOnLocal="true" useDefaultWebProxy="false">
          <!-- Indicar la protección del mensaje a través de certificado X.509 -->
          <security mode="Message">
            <transport clientCredentialType="None" />
            <message clientCredentialType="Certificate" negotiateServiceCredential="false"
                     algorithmSuite="Basic128" establishSecurityContext="true" />
          </security>
        </binding>
      </wsHttpBinding>
    </bindings>
    <!-- Comportamiento del servicio WCF -->
    <behaviors>
      <serviceBehaviors>
        <behavior name="ComportamientoMetadatos">
          <!-- Exponer el WSDL, asignar a "false" cuando ya no se necesite exponer -->
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceCredentials>
            <!-- Declaración del certificado X.509 para autenticar al cliente -->
            <clientCertificate>
              <certificate findValue="YnnafSoftwareClient" storeLocation="LocalMachine"
                           storeName="My" x509FindType="FindBySubjectName" />
              <authentication certificateValidationMode="None" />
            </clientCertificate>
            <!-- Declaración del certificado X.509 para el servidor -->
            <serviceCertificate findValue="YnnafSoftwareService"
                                storeLocation="LocalMachine"
                                storeName="My" x509FindType="FindBySubjectName" />
          </serviceCredentials>
```

```
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <!-- Declaración de las características del servicio WCF -->
    <services>
      <service behaviorConfiguration="ComportamientoMetadatos"
               name="YnnafSoftwareServiceWCF.YnnafSoftwareService_Principal">
        <!-- Declaración del enlace seguro WsHttpBinding -->
        <!-- La propiedad bindingNamespace es obligatoria para que el servicio
             funcione en Java -->
        <endpoint address="wsYnnafSoftwareService"
                  contract="YnnafSoftwareServiceWCF.IYnnafSoftware_Principal"
                  name="WsHttpBinding_YnnafSoftwareService"
                  bindingNamespace="http://localhost:80/servicioWCF"
                  bindingName="YnnafSoftwareService"
                  bindingConfiguration="EnlaceWsHttp"
                  binding="wsHttpBinding">
          <identity>
            <!-- La identidad del usuario de acuerdo al certificado X.509 -->
            <certificateReference findValue="YnnafSoftwareService"
                                  storeLocation="LocalMachine"
                                  storeName="My" x509FindType="FindBySubjectName"/>
          </identity>
        </endpoint>
        <!-- Enlace que expone los metadatos -->
        <endpoint contract="IMetadataExchange" binding="mexHttpBinding" address="mex" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Se debe notar que se omitieron las siguientes líneas con respecto a la versión original del servicio *WCF*:

```
<host>
    <baseAddresses>
        <add baseAddress="http://localhost:8080/servicioWCF/YnnafSoftwareService"/>
    </baseAddresses>
</host>
```

Esto se debe a que no es necesario colocar una dirección base para el servicio *WCF*, la dirección que se utilizará será la que utilice *IIS 7.0 + Alias + archivo .svc (http://localhost/servicioWCF/YnnafSoftwareService.svc)*

Una vez que hemos configurado todo lo anterior, construimos la aplicación web que publicará el servicio *WCF* presionando *F6* en *Visual Studio 2008*. Esto nos deberá crear el ensamblado *YnnafSoftwareServiceWCF_IIS.dll* en el directorio *Bin* del proyecto. En este directorio también se encuentra el ensamblado *YnnafSoftwareService.dll* (el servicio *WCF*).

Nombre	Fecha modificación	Tipo	Tamaño
YnnafSoftwareServiceWCF.dll	18/03/2009 06:40 p.m.	Extensión de la aplicación	161 KB
YnnafSoftwareServiceWCF	18/03/2009 06:40 p.m.	Program Debug Database	280 KB
YnnafSoftwareServiceWCF_IIS.dll	19/03/2009 12:55 p.m.	Extensión de la aplicación	4 KB
YnnafSoftwareServiceWCF_IIS	19/03/2009 12:55 p.m.	Program Debug Database	8 KB

Figura 4.15 Directorio Bin de la aplicación web del servicio WCF

4.7.2 PERMISO DE LECTURA SOBRE CERTIFICADO X.509 A INTERNET INFORMATION SERVICES (IIS) 7.0

Para que el servicio *Windows Communication Foundation* (*WCF*) se pueda ejecutar sobre *Internet Information Services (IIS) 7.0*, se deben otorgar derechos de acceso a la clave privada del certificado *X.509 YnnafSoftwareService* (provisto por la empresa) a *IIS 7.0*. Si no se otorgan estos permisos de acceso a *IIS 7.0* sobre el certificado *X.509* almacenado en *LocalMachine/My,* aparecerá un mensaje de error como el siguiente al tratar de ejecutar el servicio *WCF* en *IIS 7.0*:

Error de servidor en la aplicación '/servicioWCF'.

El conjunto de claves no existe
Descripción: Excepción no controlada al ejecutar la solicitud Web actual. Revise el seguimiento de la pila para obtener más información acerca del error y dónde se originó en el código.

Detalles de la excepción: System.Security.Cryptography.CryptographicException: El conjunto de claves no existe

Error de código fuente: Se ha generado una excepción no controlada durante la ejecución de la solicitud Web actual. La información sobre el origen y la ubicación de la excepción pueden identificarse utilizando la excepción del seguimiento de la pila siguiente.

Seguimiento de la pila:
[CryptographicException: El conjunto de claves no existe]
[ArgumentException: El certificado 'CN=YnnafSoftwareService' debe tener una clave privada capaz de realizar intercambio de claves. El proceso debe tener derechos de acceso a la clave privada.]
[ServiceActivationException: El servicio '/servicioWCF/YnnafSoftwareService.svc' no se puede activar debido a una excepción producida durante la compilación. El mensaje de excepción es: El certificado 'CN=YnnafSoftwareService' debe tener una clave privada capaz de realizar intercambio de claves. El proceso debe tener derechos de acceso a la clave privada.]

Para otorgar los permisos de acceso a *IIS* sobre los certificados *X.509*, utilizaremos la herramienta: *Microsoft Windows HTTP Services (WinHTTP) Certificate Configuration Tool (WinHttpCertCfg.exe)*. Esta herramienta se puede descargar desde el sitio: *http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f&displaylang=en.*

Una vez que descargamos el programa (*winhttpcertcfg.msi*), lo instalamos en la computadora. El programa generalmente se instala en el directorio *C:\Archivos de programa\Windows Resource Kits\Tools.*

Ya instalado, abrimos la consola de Windows (*cmd.exe*) con derechos de administrador. Nos movemos hasta el directorio *C:\Archivos de programa\Windows Resource Kits\Tools* e insertamos el siguiente comando: *WinHttpCertCfg.exe -g -c LOCAL_MACHINE\My -s YnnafSoftwareService -a IIS_IUSRS* para otorgar permisos de acceso al usuario *IIS_IUSRS*. Este usuario es el que utiliza el proceso que ejecuta *IIS 7.0* para trabajar. Si se ejecutó correctamente el otorgamiento de derechos de acceso a *IIS 7.0* sobre el certificado *X.509 YnnafSoftwareService* aparecerá algo como lo siguiente en la consola de Windows:

```
Microsoft (R) WinHTTP Certificate Configuration Tool
Copyright (C) Microsoft Corporation 2001.

Matching certificate:
CN=YnnafSoftwareService

Granting private key access for account:
    BUILTIN\IIS_IUSRS
```

Figura 4.16 Otorgamiento exitoso de derechos de acceso al usuario IIS_IUSRS

Para verificar cuáles usuarios tienen derecho de acceso al certificado *X.509 YnnafSoftwareService* ejecutaremos la siguiente línea de comando *WinHttpCertCfg.exe -l -c LOCAL_MACHINE\My -s YnnafSoftwareService*. Si los permisos se asignaron correctamente se verá una pantalla como la siguiente:

```
Microsoft (R) WinHTTP Certificate Configuration Tool
Copyright (C) Microsoft Corporation 2001.

Matching certificate:
CN=YnnafSoftwareService

Additional accounts and groups with access to the private key include:
   NT AUTHORITY\SYSTEM
   BUILTIN\Administradores
   BUILTIN\IIS_IUSRS
```

Figura 4.17 Usuarios con derecho de acceso al certificado X.509 YnnafSoftwareService

4.7.3 LEVANTAMIENTO Y EJECUCIÓN DEL SERVICIO WINDOWS COMMUNICATION FOUNDATION SOBRE INTERNET INFORMATION SERVICES (IIS) 7.0

Abrimos el *Administrador de Internet Information Services (IIS) 7.0 (inetmgr.exe)*

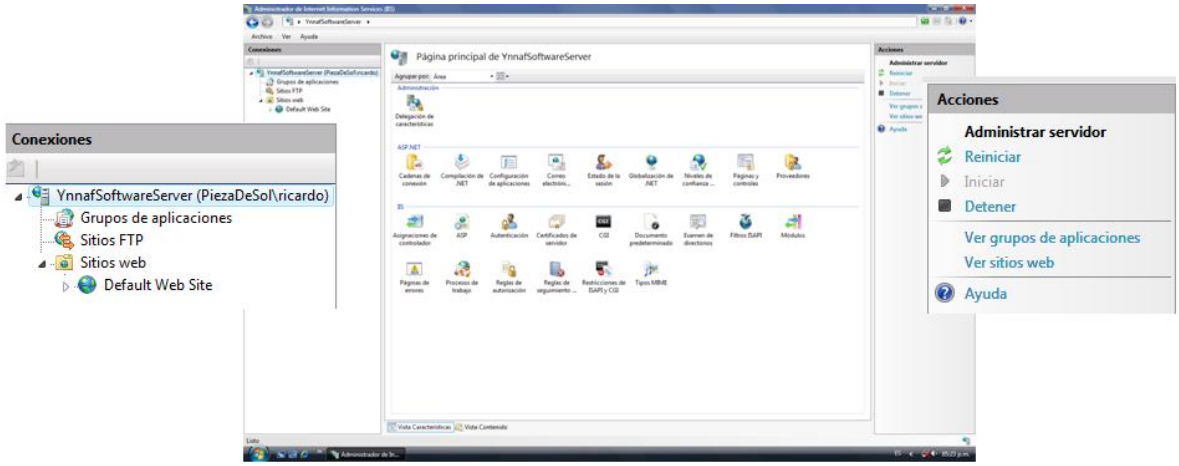


Figura 4.18 Administrador de Internet Information Services (IIS) 7.0

Los pasos que debemos seguir para levantar el servicio *WCF* son los siguientes: damos clic derecho sobre *Default Web Site* y seleccionamos *Agregar aplicación…*. En la ventana que aparecerá, insertamos un *Alias* (*servicioWCF*) y la ruta de acceso física al servicio. Para este último la ruta deberá ser la carpeta base del directorio *Bin* (donde se encuentran los ensamblados *YnnafSoftwareService_IIS.dll* e *YnnafSoftwareService.dll*). Por ejemplo, si los ensamblados anteriores se encuentran en *C:\YnnafSoftwareServiceWCF_IIS\-YnnafSoftwareServiceWCF_IIS\Bin*, la ruta de acceso física que se deberá colocar en la ventana será: *C:\YnnafSoftwareServiceWCF_IIS\YnnafSoftwareServiceWCF_IIS.* Damos clic en el botón *Aceptar.*

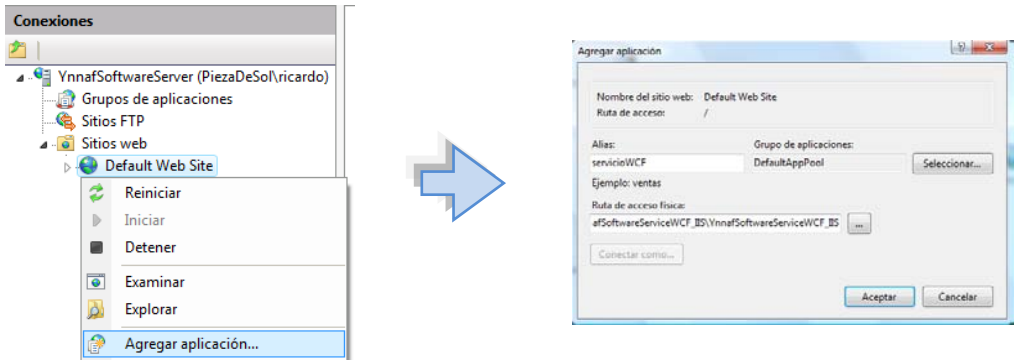


Figura 4.19 Levantamiento del servicio WCF en IIS 7.0

Finalmente, para comprobar que efectivamente nuestro servicio *WCF* esta ejecutándose correctamente en el servidor *IIS 7.0*, abrimos nuestro explorador de Internet (*Internet Explorer*, *Firefox*, etc.) e insertamos la dirección: *http://localhost/servicioWCF/YnnafSoftwareService.svc* donde:

- ✓ http://localhost: es la dirección donde se ejecuta *IIS 7.0.*
- ✓ servicioWCF: es el alias de la aplicación (servicio *WCF*) hospedada en *IIS 7.0.*
- ✓ YnnafSoftwareService.svc: es el archivo que accede al servicio *WCF* (creado en el apartado *4.7.1*).

Si *IIS 7.0* se está ejecutando y el servicio *WCF* se encuentra correctamente configurado, veremos en nuestro explorador de Internet algo como la siguiente imagen:

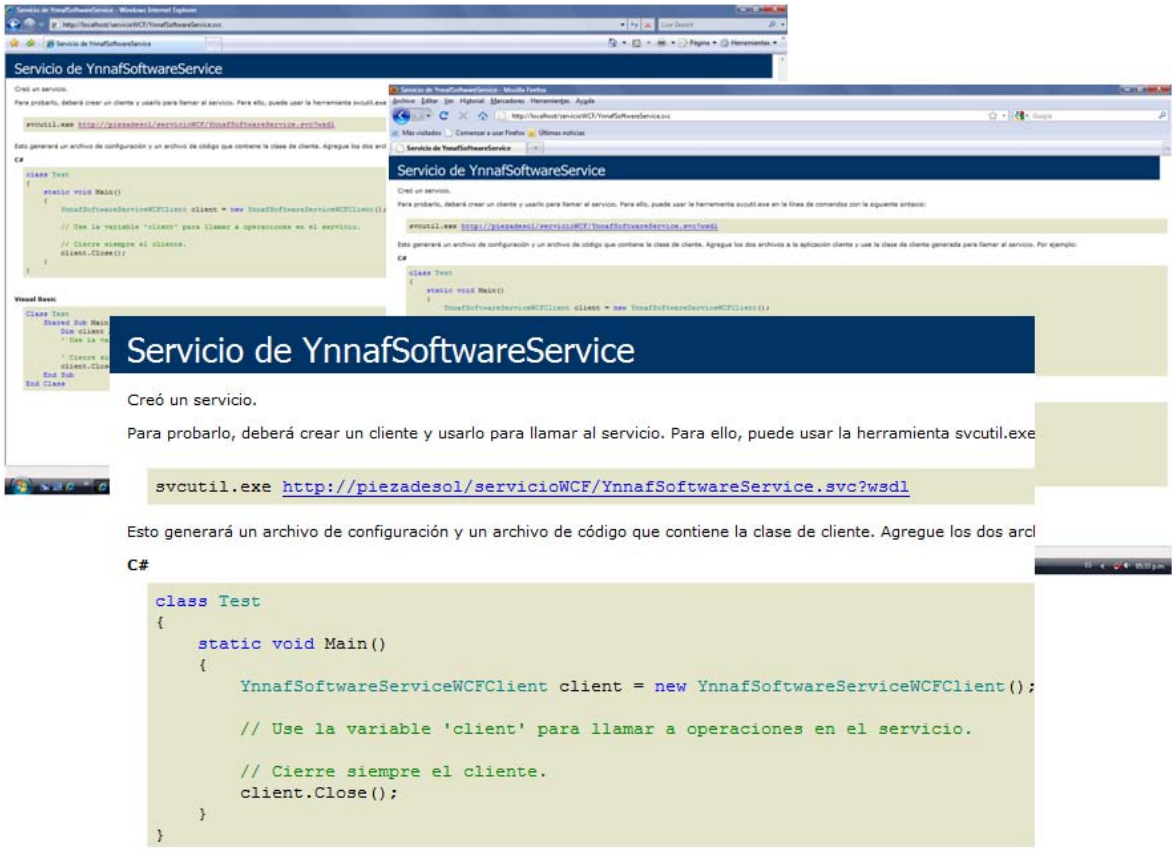


Figura 4.20 Servicio WCF '*YnnafSoftwareService'* ejecutándose correctamente en IIS 7.0
(A la izquierda Microsoft Internet Explorer 8.0, a la derecha Mozilla FireFox 3.5)

CAPÍTULO 5.
DESARROLLO DE DIVERSOS CLIENTES WCF/WSIT QUE HACEN USO DE LAS OPERACIONES EXPUESTAS POR EL SERVICIO WCF.

INTRODUCCIÓN

El objetivo primordial de este quinto capítulo es el de mostrar como las aplicaciones de escritorio construidas en *C#/WPF*, *Java/Swing* y *VB.NET/Windows Forms* y la aplicación web construida en *C#/ASP.NET* (cuatro aplicaciones construidas en diferentes plataformas de software, lenguajes de programación y enfocadas a distintos propósitos) pueden consumir el servicio de *Windows Communication Foundation* creado en el capítulo anterior.

Algunos aspectos importantes a tener en cuenta al leer el código fuente que se muestran en este quinto capítulo son:

- ✓ Al inicio o final de cada fragmento de código fuente mostrado, se da un resumen de los procesos que se realizan en el código.
- ✓ Dentro del mismo código fuente se insertan comentarios describiendo lo que se ejecutará en las líneas que los prosiguen. Se debe tomar en cuenta que los comentarios se pueden identificar de la siguiente manera: en los lenguajes de programación *C#, Java* y *JavaScript* inician con los caracteres // y finalizan hasta un salto de línea (los comentarios de la forma */* MiComentario */* no se utilizan en este trabajo); en el lenguaje de programación *VB.NET* inician con el caracter ' y finalizan hasta un salto de línea siempre y cuando no esté precedido del caracter _ y finalmente en los lenguajes de programación *XML* y *XAML* inician con los caracteres <!-- y finalizan con los caracteres -->.
- ✓ Además de los comentarios anteriores dentro del código, en *C#, VB.NET* y *Java* se pueden ver documentación (para fines de desarrollo en un *IDE* por ejemplo) dentro del código como la siguiente: ///<summary> ///*MiDoc* ///</summary> para *C#,* '''<summary> '''*MiDoc* '''</summary> para *VB.NET* y /** *MiDoc* */ para *Java* y *JavaScript*.
- ✓ El código mostrado no es todo el que se necesita para hacer funcionar las aplicaciones correctamente, únicamente se muestran fragmentos de código para procesos específicos.
- ✓ Se da por hecho que el lector tiene un conocimiento sólido de las plataformas de programación de software *.NET Framework* (*C#*, *VB.NET*, *Windows Forms*, *WPF*, *XML*, *XAML* y *ASP.NET*) y *Java* (lenguaje de programación *Java* y *Swing*). Esto se debe a que esta tesis no se centra en ofrecer al lector un conocimiento desde cero de todos los componentes de *.NET Framework, Java* u otras tecnologías implicadas en el desarrollo de software.

5.1 CREACIÓN DE UN CLIENTE WCF EN C#. PROGRAMA PRINCIPAL (APLICACIÓN DE ESCRITORIO)

A continuación se muestra como un servicio *Windows Communication Foundation* (*WCF*) obtiene una completa interoperabilidad con un cliente *Windows Communication Foundation.* Ambos construidos bajo una plataforma de software y lenguaje de programación iguales. Se observará como el servicio *Windows Communication Foundation* (construido en el capítulo anterior) escrito en *C# 3.0* interopera al 100% con un cliente *WCF* construido también en *C# 3.0*. Entonces, la plataforma de software para ambos es *.NET Framework*, el cliente *WCF* es una aplicación de escritorio.

5.1.1 REQUERIMIENTOS

Se requiere construir un cliente *Windows Communication Foundation (WCF)* en *C#* para consumir el servicio *WCF* que expone la lógica del negocio de la empresa. Esta nueva aplicación deberá cumplir con las siguientes condiciones:

1. Establecer comunicación con el servicio *WCF* utilizando el certificado *X.509* provisto por la empresa para acceder a la lógica del negocio.
2. Gráficamente debe ser una aplicación atractiva al usuario. De preferencia no se debería usar *Windows Forms* como herramienta de creación de interfaces gráficas de usuario.
3. Se deben poder administrar todos los datos de la base de datos del sistema (por medio del servicio *WCF*). Esto implica que todo lo relacionado a los productos, clientes y órdenes deben ser administrado desde este programa.

5.1.2 ANÁLISIS

Dado que el servicio *WCF* está construido en *C# versión 3.0* para la plataforma de software *.NET Framework,* en el cliente también se utilizará la versión *3.0* del lenguaje de programación para obtener los máximos beneficios. Ya que también se requiere que la aplicación gráficamente sea atractiva al usuario y que de preferencia no se emplee *Windows Forms* para la creación de la interfaz gráfica de usuario, se utilizará *Windows Presentation Foundation* (*WPF*) para este fin. Las ventanas del programa se crearán utilizando la interfaz *Ribbon*. Además se utilizará el *WPFToolkit* (ensamblado referenciado al proyecto) para obtener una cantidad mayor de controles *WPF*.

5.1.3 DISEÑO

A continuación se muestran los casos de uso detallados que se deben seguir para los diferentes procedimientos que se utilizarán a lo largo del programa.

5.1.3.1 CASOS DE USO DETALLADOS

Caso de uso:	**Inicio de sesión con el servicio WCF.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C#* utilizando *WPF*			
Descripción:	Se describe como se obtiene instancia con el servicio *WCF*.			
Pre-condiciones:	El cliente *WCF* debe poseer los certificados *X.509* adecuados para el servicio *WCF*			
Post-condiciones:	El cliente *WCF* obtiene instancia y sesión con el servicio *WCF*.			
Versión:	0.0.1			
Fecha de creación	13-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Inicio de sesión con nombre de usuario y contraseña válidos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide ejecutar cualquiera de las dos tareas para las que sirve el programa. Se debe iniciar una instancia y sesión con el servicio *WCF* llamando al contrato de operación *IniciarSesion (username, password)*	2	Se retorna un valor booleano verdadero si el nombre de usuario y contraseña se encuentran en la base de datos. Como consecuencia se crea una instancia y sesión en el servicio *WCF*. Se mantiene en la sesión el nivel de acceso del usuario.	
Flujo alternativo:	Inicio de sesión con nombre de usuario y contraseña no válidos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**

1	El usuario decide ejecutar cualquiera de las dos tareas para las que sirve el programa. Se debe iniciar una instancia y sesión con el servicio *WCF* llamando al contrato de operación *IniciarSesion (username, password)*	2	Se retorna una excepción al cliente indicando que no se encontró el nombre de usuario o contraseña. Esta excepción debe ser interceptada por el cliente *WCF* escrito en *C#*.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**

Caso de uso:	**Selección de datos.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C#* utilizando *WPF*			
Descripción:	Se describe como el cliente *WCF* se comporta al obtener cualquier dato desde el servicio WCF.			
Pre-condiciones:	Se debe tener un nivel de acceso que permita obtener datos desde la base de datos.			
Post-condiciones:	El cliente obtiene los datos que solicitó.			
Versión:	0.0.1			
Fecha de creación	13-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Selección de datos con nivel de acceso correcto			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario solicita datos de cualquiera de las tablas que contienen los datos de la empresa.	2	Si se tiene el nivel de acceso necesario para seleccionar datos, se retornan los datos al cliente *WCF*.	
Flujo alternativo:	Selección de datos con nivel de acceso no válido			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario solicita datos de cualquiera de las tablas que contienen los datos de la empresa.	2	Si no se tiene el nivel de acceso necesario para seleccionar datos, se debe interceptar la excepción retornada por el servicio *WCF* y mostrar la información del error al usuario.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Inserción de datos.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C#* utilizando *WPF*			
Descripción:	Se describe como el cliente *WCF* se comporta al insertar datos a través del servicio *WCF*			
Pre-condiciones:	Se debe tener un nivel de acceso que permita insertar datos a través del servicio *WCF*.			
Post-condiciones:	El cliente insertó los datos que necesitaba.			
Versión:	0.0.1			
Fecha de creación	13-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Inserción de datos con nivel de acceso y lista de datos correctos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide insertar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema a través del servicio *WCF*	2	Si se tiene el nivel de acceso necesario para insertar datos, se retornan la cantidad de productos insertados en la base de datos.	
Flujo alternativo:	Inserción de datos con nivel de acceso y lista de datos incorrectos.			

Paso	Acciones	Paso	Acciones	Excepción
1	El usuario decide insertar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema a través del servicio *WCF*	2	Si no se tiene el nivel de acceso necesario para insertar datos o los datos son incorrectos, se debe interceptar la excepción retornada por el servicio *WCF* y mostrar la información del error al usuario.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**

Caso de uso:	**Actualización de datos.**
Tipo:	Básico
Actor principal:	Cliente *WCF* escrito en *C#* utilizando *WPF*
Descripción:	Se describe como el cliente *WCF* se comporta al actualizar datos a través del servicio *WCF*
Pre-condiciones:	Se debe tener un nivel de acceso que permita actualizar datos a través del servicio *WCF*.
Post-condiciones:	El cliente actualizó los datos que necesitaba.
Versión:	0.0.1
Fecha de creación	13-02-2009
Fecha de actualización:	

FLUJOS

Flujo básico:	Actualización de datos con nivel de acceso y lista de datos correctos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide actualizar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema a través del servicio *WCF*	2	Si se tiene el nivel de acceso necesario para actualizar datos, se retornan la cantidad de productos actualizados en la base de datos.	
Flujo alternativo:	Actualización de datos con nivel de acceso y lista de datos incorrectos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide actualizar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema a través del servicio *WCF*	2	Si no se tiene el nivel de acceso necesario para actualizar datos o los datos son incorrectos, se debe interceptar la excepción retornada por el servicio *WCF* y mostrar la información del error al usuario.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**

Caso de uso:	**Eliminación de datos.**
Tipo:	Básico
Actor principal:	Cliente *WCF* escrito en *C#* utilizando *WPF*
Descripción:	Se describe como el cliente *WCF* se comporta al eliminar datos a través del servicio *WCF*
Pre-condiciones:	Se debe tener un nivel de acceso que permita eliminar datos a través del servicio *WCF*.
Post-condiciones:	El cliente eliminó los datos que necesitaba.
Versión:	0.0.1
Fecha de creación	13-02-2009
Fecha de actualización:	

FLUJOS

Flujo básico:	Eliminación de datos con nivel de acceso y lista de datos correctos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide eliminar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema a través del	2	Si se tiene el nivel de acceso necesario para eliminar datos, se retornan la cantidad de productos eliminados en la base de	

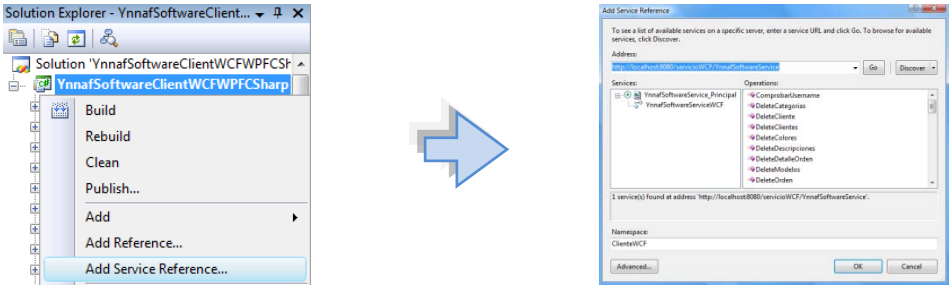
	servicio *WCF*		datos.	
Flujo alternativo:	Eliminación de datos con nivel de acceso y lista de datos incorrectos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El cliente decide eliminar un conjunto de datos en cualquiera de las tablas de la base de datos del sistema a través del servicio *WCF*	2	Si no se tiene el nivel de acceso necesario para eliminar datos o los datos son incorrectos, se debe interceptar la excepción retornada por el servicio *WCF* y mostrar la información del error al usuario.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

5.1.4 IMPLEMENTACIÓN

A continuación se muestra la implementación del programa en *C# 3.0/WPF*:

5.1.4.1 IMPORTACIÓN DEL WSDL DEL SERVICIO WCF

Para poder generar el código que nos permita acceder al servicio *WCF* debemos seguir los siguientes pasos: *1)* Crear una referencia de servicio usando la dirección provista por la empresa *YnnafSoftware* para el servicio *WCF*: *http://localhost/servicioWCF/YnnafSoftwareService.svc*. *2)* El espacio de nombres (*Namespace*) para acceder al servicio en el cliente será: *ClienteWCF*



Importación del servicio WCF en Visual Studio 2008

Una vez finalizada la generación del código fuente local del servicio *WCF*, podemos ver que los códigos tanto en el servicio como en el cliente son similares, el modo de uso depende del lenguaje de programación; aquí al ser el mismo lenguaje, el modo de uso es el mismo:

DEFINICIÓN DEL CONTRATO DE OPERACIÓN SP_CREATEORDEN EN EL SERVICIO WCF

```
[OperationContract(Name = "Sp_CreateOrden", IsInitiating = false, IsTerminating = false)]
[FaultContract(typeof(OrdenFault))]
int Sp_CreateOrden(Orden orden, String listaProductos, ref int ordenId,
ref String DescripcionResultado);
```

EL MISMO CONTRATO DE OPERACIÓN EN EL CLIENTE WCF CONSTRUIDO EN C# 3.0/WPF TRAS LA IMPORTACIÓN (ESTE CÓDIGO ES GENERADO AUTOMÁTICAMENTE)

```
[System.ServiceModel.OperationContractAttribute(IsInitiating=false,
Action="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrden",
ReplyAction="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenResponse")]
[System.ServiceModel.FaultContractAttribute(typeof(YnnafSoftwareClientWCFWPFCSharp.Cliente
WCF.OrdenFault),
Action="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenOrdenFaultFault
", Name="OrdenFault", Namespace="http://www.ynnafSoftware.com/DataContract/Faults")]
int Sp_CreateOrden(YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Orden orden, string
listaProductos, ref int ordenId, ref string descripcionResultado);
```

5.1.4.2 INSTANCIA SINGLETON PARA EL CLIENTE WCF

Puesto que el servicio *WCF* maneja instancias que guardan valores ayudándose de sesiones, es necesario mantener desde el cliente *WCF* una instancia única con el servicio *WCF* para obtener un correcto funcionamiento. Para lograr esto, utilizaremos el patrón *Singleton*[20]. A continuación se muestra el código fuente escrito en *C# 3.0* que lleva a cabo esta tarea:

```
/// <summary>
/// Permite manipular una única instancia del servicio WCF
/// </summary>
public sealed class InstanceClienteWCF {
    // Mantiene la instancia única
    private static volatile InstanceClienteWCF instance = null;

    // Para ser utilizado en la sentencia lock. Esta sentemcia se
    // utiliza para asegurar que un bloque de código se ejecuta
    // hasta ser completado sin interrupciones.
    private static readonly object padlock = new object();

    // Evita la construcción de un objeto de esta clase de forma externa
    private InstanceClienteWCF() { }

    /// <summary>
    /// Obtiene la instancia única del cliente WCF
    /// </summary>
    public static InstanceClienteWCF Instance {
        get {
            if (instance == null) {
                lock (padlock) {
                    if (instance == null)
                        instance = new InstanceClienteWCF();
                }
            }
            return instance;
        }
    }

    /// <summary>
    /// Obtiene o establece la instancia del servicio WCF
    /// </summary>
    public ClienteWCF.YnnafSoftwareServiceWCFClient ServicioWCF { get; set; }
}
```

Con este código fuente (contenido en el espacio de nombres *YnnafSoftwareClientWCFWPFCSharp*) aseguramos tener una y solo una instancia del cliente *WCF* incluso en entornos multihilo. La forma en que se obtendrá la instancia será usando la instrucción: *InstanceClienteWCF.Instance.*
Este tipo de implementación del patrón *Singleton* contiene el patrón *Double-Check Locking*. *Double-Check Locking* garantiza que únicamente se bloquea el acceso concurrente cuando la condición se satisface y una vez que se tiene el bloqueo se vuelve a evaluar la misma condición que ocasionó el bloqueo. Así, si dos hilos tratan de entrar a este segmento de código el primero va a cumplir con las 2 condiciones pero el segundo sólo cumple la primera condición dado que al momento de preguntar por segunda vez, el primer hilo ocasionó el bloqueo.

[20] El patrón Singleton garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ésta instancia. Fuente: Microsoft Developer Network. El Patrón Singleton. http://msdn.microsoft.com/es-es/library/bb972272.aspx

5.1.4.3 ENCRIPTACIÓN DE CONTRASEÑAS EN SHA-512.

Dado que el servicio *WCF* requiere que la contraseña necesaria para autorizar el acceso del usuario a la lógica del negocio esté encriptada en el algoritmo de seguridad *SHA-512* usando la codificación *UTF-8 (almacenada en Base 64)*, se debe hacer de la siguiente forma usando *C#*:

```
public static String EncriptarSha512(String cadenaSinEncriptar) {
    try {
        byte[] cadenaSinEncriptarArray = Encoding.UTF8.GetBytes(cadenaSinEncriptar);
        SHA512Managed sha512 = new SHA512Managed();
        byte[] cadenaEncriptadaArray = sha512.ComputeHash(
            cadenaSinEncriptarArray, 0, cadenaSinEncriptarArray.Length);
        return Convert.ToBase64String(
            cadenaEncriptadaArray, 0, cadenaEncriptadaArray.Length);
    } catch (Exception ex) {
        Console.WriteLine(ex.Message);
        return null;
    }
}
```

Hay que destacar lo siguiente: para que este método funcione adecuadamente se deben importar los espacios de nombres: *System.Security.Cryptography* y *System.Text*.

NOTA: A diferencia de *Java* (como veremos más adelante en la construcción del cliente *WSIT* para el servicio *WCF*), las clases para generar la encriptación de cadenas no son propietarias de *Microsoft* y nos garantiza que no habrá problemas de compatibilidad futuras debido a posibles desapariciones de estas clases en la Biblioteca de Clases Base.

5.1.4.4 WPF TOOLKIT Y RIBBON LIBRARY

Con el propósito de facilitar el desarrollo de la aplicación con *Windows Presentation Foundation (WPF),* vamos a hacer uso del ensamblado *WPFToolkit.dll* disponible desde la página: *http://wpf.codeplex.com/Release/-ProjectReleases.aspx?ReleaseId=25047.* El *WPF Toolkit* es una colección de componentes y características construidos en *WPF* y que no se encuentran de forma nativa en *.NET Framework 3.0* ó *3.5*. Para esta aplicación, vamos a hacer uso del control *DataGrid* (control de lista enlazada a datos que muestra los elementos del origen de datos en una tabla. El control *DataGrid* permite seleccionar, ordenar y editar estos elementos) entre otros. Para que nuestra aplicación *WPF* pueda hacer uso de este ensamblado, debemos registrarlo en cada una de las ventanas que lo requieran como se muestra a continuación:

```
xmlns:toolkit="http://schemas.microsoft.com/wpf/2008/toolkit"
```

Para poder construir las ventanas con la interfaz *Ribbon* nos apoyaremos del ensamblado *RibbonControlsLibrary.dll* disponible desde la página: *http://msdn.microsoft.com/es-mx/office/-aa973809(en-us).aspx -> License the Office UI*. Para que la aplicación *WPF* pueda hacer uso de los controles de este ensamblado, se debe registrar en cada una de las ventanas que lo requieran como se muestra a continuación:

```
xmlns:ribbon=
"clr-namespace:Microsoft.Windows.Controls.Ribbon;assembly=RibbonControlsLibrary"
```

NOTA: Estas 2 características de *WPF* (controles de *Ribbon* y *WPFToolkit*) probablemente estarán disponibles en la versión final de *.NET Framework 4.0,* es decir, estarán contenidas en ensambla-

dos nativos de *.NET Framework*. En la *Beta 1* de la versión *4.0* de *.NET Framework* al momento de escribir esta tesis, únicamente se encuentran los controles de *WPF Toolkit.*

5.1.4.5 LLAMADAS A CONTRATOS DE OPERACIONES DEL SERVICIO WCF

A continuación se muestra de forma sintetizada por cuestiones de espacio, como se llaman diversos contratos de operación a través de la aplicación construida con *WPF* y *C#*. Se darán ejemplos de código únicamente de algunos contratos de operación. Se debe notar que no se coloca el código para inicializar todas las interfaces gráficas de usuario.

5.1.4.5.1 INICIO DE SESIÓN

Lo primero que necesitamos hacer para utilizar el programa es autenticar el cliente *WCF* ante el servicio *WCF*. Para lograr esto, se crea una instancia de *YnnafSoftwareServiceWCFClient*, se establecen los certificados *X.509* y se asigna esta instancia a la instancia *Singleton*. Una vez hecho lo anterior, se debe obtener autorización de acceso a la lógica del negocio del sistema, se debe entonces llamar al contrato de operación *IniciarSesion(username, password)* del servicio *WCF*. Si el nombre de usuario y contraseña son correctos se asigna a la propiedad *DialogResult* el valor devuelto por el servicio (*true*) y se podrá usar el programa, de lo contrario se caerá en el bloque *catch* y no se podrá usar el programa. A partir de la instancia *Singleton* accederemos en toda la ejecución del programa al servicio *WCF*. En el código fuente siguiente, podemos ver cómo se realiza el procedimiento descrito anteriormente:

```
private void IngresarButton_Click(object sender, RoutedEventArgs e) {
    try {
        // Inicio de sesión
        ClienteWCF.YnnafSoftwareServiceWCFClient cliente = new
            YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.YnnafSoftwareServiceWCFClient();
        // Obtenemos el certificado desde el almacén MMC. StoreName.My -> Personal
        cliente.ClientCredentials.ClientCertificate.SetCertificate(
            StoreLocation.LocalMachine, StoreName.My, X509FindType.FindBySubjectName,
            "YnnafSoftwareClient");
        InstanceClienteWCF.Instance.ServicioWCF = cliente;
        this.DialogResult = InstanceClienteWCF.Instance.ServicioWCF.IniciarSesion(
            this.UsernameTextBox.Text,
            Encriptacion.EncriptarSha512(this.PasswordPasswordBox.Password));
    } catch (Exception ex) {
        MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
            MessageBoxImage.Error);
    }
}
```

5.1.4.5.2 OBTENCIÓN, INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN DE PRODUCTOS.

Se debe notar que el enlace de datos necesario para el *DataGrid* (vista en *MVC*) es único para los cuatro procedimientos (obtención, inserción, actualización y eliminación de productos)

5.1.4.5.2.1 ENLACE DE DATOS EN DATAGRID

La definición del control *DataGrid* de *WPFControlToolkit.dll* que mostrará los datos, se hará vía código *XAML*[21] y se controlará vía código *C#*, el primero nos va servir para tratar todo lo relacionado con la vista mientras que el segundo nos servirá para mantener un control sobre la vista.

[21] Lenguaje de marcado de aplicaciones extensible (*XAML*) es un lenguaje de marcado para la programación de aplicaciones declarativa. *Windows Presentation Foundation (WPF)* implementa un cargador *Lenguaje de marcado de aplicaciones extensible (XAML)* y proporciona compatibilidad el lenguaje *Lenguaje de marcado de aplicaciones extensible (XAML)* para los tipos de *Windows Presentation Foundation (WPF)*, de tal forma que se puede crear la mayor parte de la interfaz de usuario de la aplicación en marcado *Lenguaje de marcado de aplicaciones extensible (XAML)*.

5.1.4.5.2.1.1 CÓDIGO XAML

El siguiente código *XAML* permite definir todo lo relacionado con la vista:

```
<toolkit:DataGrid x:Name="ProductosDataGrid" ItemsSource="{Binding}"
             AlternationCount="2"
             AutoGenerateColumns="False"
             BorderBrush="Transparent"
             Background="Transparent"
             ColumnHeaderStyle="{DynamicResource dgHeaderStyleNegro}"
             RowStyle="{DynamicResource dgRowStyleNegro}"
             CellStyle="{DynamicResource dgCellStyle}"
             SelectionMode="Extended" SelectionUnit="FullRow"
             GridLinesVisibility="All"
             VerticalGridLinesBrush="DarkGray"
             RowDetailsVisibilityMode="VisibleWhenSelected"
             CanUserAddRows="False" CanUserDeleteRows="False"
             BeginningEdit="ProductosDataGrid_BeginningEdit"
             RowEditEnding="ProductosDataGrid RowEditEnding"
             PreviewKeyDown="ProductosDataGrid_PreviewKeyDown">

    <!-- Definición de columnas -->
    <toolkit:DataGrid.Columns>
        <!--Nombre-->
        <toolkit:DataGridTemplateColumn x:Name="NombreProductoDataGridTemplateColumn"
            Header="Nombre" MinWidth="150">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Producto}" MinHeight="100"
                        MinWidth="100" Padding="0,40,0,0" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
            <toolkit:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate x:Name="NombreProductoDataTemplate">
                    <TextBox Style="{StaticResource TextBoxInError}">
                        <Binding Path="Producto" UpdateSourceTrigger="PropertyChanged"
                            Mode="TwoWay">
                        </Binding>
                    </TextBox>
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellEditingTemplate>
        </toolkit:DataGridTemplateColumn>
        <!-- Imagen -->
        <toolkit:DataGridTemplateColumn Header="Imagen" MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <Image Source="{Binding Path=Imagen,
                        Converter={StaticResource ConversorImagenes}}"
                        MinHeight="100" MaxWidth="100" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
            <toolkit:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Vertical">
                        <Image Source="{Binding Path=Imagen,
                            Converter={StaticResource ConversorImagenes}}"
                            MinHeight="77" MaxWidth="77" />
                        <Button Content="Cargar imagen" Click="ImagenButton_Click" />
                    </StackPanel>
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellEditingTemplate>
        </toolkit:DataGridTemplateColumn>
        <!—Otras definiciones de columnas-->
        <!—...-->
    </toolkit:DataGrid.Columns>
</toolkit:DataGrid>
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *DataGrid* que servirá para mostrar información sobre productos obtenida desde el código de *C#*. *2)* Definición de las columnas, generalmente la propiedad del control que permita visualizar datos al usuario (*Text*, *TextBinding*, *SelectedItemBinding*, etc.), está enlazada (*Binding*) con la propiedad de la fuente de datos que se desea mostrar. Por ejemplo, para mostrar el nombre del producto, la propiedad *Text* del control *TextBlock* está enlazada con la propiedad *Producto* de la lista genérica que sirve como fuente de datos.

5.1.4.5.2.1.2 CÓDIGO C# PARA OBTENCIÓN DE PRODUCTOS

El siguiente código *C#* permite tratar todo lo relacionado al modelo:

```
private void VerProductoRibbonButton_Executed(object sender, ExecutedRoutedEventArgs e){
    try {
        // Usando el procedimiento almacenado obtenemos todo lo relacionado con productos
        ClienteWCF.Sp_Productos sp_productos =
            InstanceClienteWCF.Instance.ServicioWCF.Sp_GetProductos();

        // Llenamos las listas genéricas locales
        Colecciones.productos_join = sp_productos.ListaProductos.ToList();
        Colecciones.productoColor = sp_productos.ListaColoresProductos.ToList();
        Colecciones.productoCategoria_join =
            sp_productos.ListaCategoriasProductos.ToList();
        Colecciones.productoDescripcion =
            sp_productos.ListaDescripcionesProductos.ToList();
        Colecciones.productoModelo_join = sp_productos.ListaModelosProductos.ToList();

        // Obtenemos el DataGrid
        ProductosDataGridWPF dgWPF =
            (ProductosDataGridWPF)this.GridProductosDataGrid.Children[0];

        // Vinculamos las listas genéricas con los controles
        dgWPF.ColorDataGridComboBoxColumn.ItemsSource =
            Colecciones.productoColor.Select(p => p.Color);
        dgWPF.CategoriaDataGridComboBoxColumn.ItemsSource =
            Colecciones.productoCategoria_join.Select(p => p.Categoria);
        dgWPF.ModeloDataGridComboBoxColumn.ItemsSource =
            Colecciones.productoModelo_join.Select(p => p.Modelo);
        dgWPF.ProductosDataGrid.ItemsSource = Colecciones.productos_join;

        // Limpiar el estado previo de la barra de estado
        this.EstadoTextBlock.Text = String.Empty;
    } catch (NullReferenceException) {
        MessageBox.Show(this, "No está autentificado.", "Prohibido",
            MessageBoxButton.OK, MessageBoxImage.Exclamation);
    } catch (Exception ex) {
        MessageBox.Show(this, ex.Message, "Error",
            MessageBoxButton.OK, MessageBoxImage.Exclamation);
    }
}
```

En resumen lo que el código anterior efectúa es lo siguiente: *1)* Obtener desde el método *Sp_GetProductos* toda la información referente a los productos de la empresa, *2)* Asignar los valores obtenidos a listas genéricas locales y *3)* Asignar a la propiedad *ItemsSource* de los controles que muestran información al usuario la lista genérica local correspondiente.

5.1.4.5.2.1.3 CÓDIGO C# PARA INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN DE PRODUCTOS

Conociendo que el código *XAML* es el mismo que el utilizado en la obtención de productos, el código *C#* para insertar, actualizar y eliminar productos es el siguiente:

```
private void GuardarProductoRibbonButton_Executed(object sender,
    ExecutedRoutedEventArgs e) {
    if (this.ribbon.SelectedTab.Label == "Productos") {
```

```
        #region INSERT Productos
        if (this.AgregarProductoRibbonToggleButton.IsChecked == true &&
            this.EditarProductoRibbonToggleButton.IsChecked == false &&
            this.EliminarProductoRibbonToggleButton.IsChecked == false) {
            try {
                List<ClienteWCF.Producto> lista = new
                    List<YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Producto>();
                ProductosDataGridWPF dgWPF =
                    (ProductosDataGridWPF)this.GridProductosDataGrid.Children[0];

                dgWPF.ProductosDataGrid.CanUserAddRows = false;
                dgWPF.ProductosDataGrid.CanUserDeleteRows = false;

                foreach (ClienteWCF.Producto_Join p in dgWPF.ProductosDataGrid.Items) {
                    lista.Add(new YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Producto {
                        ProductoId = p.ProductoId,
                        Producto1 = p.Producto,
                        ClaveProducto = p.ClaveProducto,
                        CostoEstandar = p.CostoEstandar,
                        PrecioLista = p.PrecioLista,
                        Tamaño = p.Tamaño,
                        Peso = p.Peso,
                        UnidadesDisponibles = p.UnidadesDisponibles,
                        Imagen = p.Imagen,
                        FechaInicioVenta = p.FechaInicioVenta,
                        FechaFinVenta = p.FechaFinVenta,
                        CategoriaProductoId = p.CategoriaProductoId,
                        ModeloProductoId = p.ModeloProductoId,
                        ColorProductoId = p.ColorProductoId,
                        FechaModificacion = DateTime.Now
                    });
                }
                this.EstadoTextBlock.Text = "Se insertaron: " +
                    InstanceClienteWCF.Instance.ServicioWCF.InsertProductos(
                        lista.ToArray()) + " productos";
            } catch (NullReferenceException) {
                MessageBox.Show(this, "No está autentificado.", "Prohibido",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            } catch (Exception ex) {
                MessageBox.Show(this, ex.Message, "Error",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            }
        }
        #endregion
        #region UPDATE Productos
        else if (this.AgregarProductoRibbonToggleButton.IsChecked == false &&
            this.EditarProductoRibbonToggleButton.IsChecked == true &&
            this.EliminarProductoRibbonToggleButton.IsChecked == false) {
            try {
                List<ClienteWCF.Producto> lista = new
                    List<YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Producto>();
                ProductosDataGridWPF dgWPF =
                    (ProductosDataGridWPF)this.GridProductosDataGrid.Children[0];

                foreach (ClienteWCF.Producto_Join p in dgWPF.ProductosEditados) {
                    lista.Add(new YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Producto {
                        ProductoId = p.ProductoId,
                        Producto1 = p.Producto,
                        ClaveProducto = p.ClaveProducto,
                        CostoEstandar = p.CostoEstandar,
                        PrecioLista = p.PrecioLista,
                        Tamaño = p.Tamaño,
                        Peso = p.Peso,
                        UnidadesDisponibles = p.UnidadesDisponibles,
                        Imagen = p.Imagen,
                        FechaInicioVenta = p.FechaInicioVenta,
                        FechaFinVenta = p.FechaFinVenta,
                        CategoriaProductoId = p.CategoriaProductoId,
                        ModeloProductoId = p.ModeloProductoId,
                        ColorProductoId = p.ColorProductoId,
                        FechaModificacion = DateTime.Now
```

```
                    });
                }
                this.EstadoTextBlock.Text = "Se editaron: " +
                    InstanceClienteWCF.Instance.ServicioWCF.UpdateProductos(
                        lista.ToArray()) + " productos";
            } catch (NullReferenceException) {
                MessageBox.Show(this, "No está autentificado.", "Prohibido",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            } catch (Exception ex) {
                MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
                    MessageBoxImage.Exclamation);
            }
        }
        #endregion
        #region DELETE Productos
        else if (this.AgregarProductoRibbonToggleButton.IsChecked == false &&
            this.EditarProductoRibbonToggleButton.IsChecked == false &&
            this.EliminarProductoRibbonToggleButton.IsChecked == true) {
            try {
                List<int> lista = new List<int>();
                ProductosDataGridWPF dgWPF =
                    (ProductosDataGridWPF)this.GridProductosDataGrid.Children[0];

                foreach (int n in dgWPF.ProductosEliminados) {
                    lista.Add(n);
                }
                this.EstadoTextBlock.Text = "Se eliminaron: " +
                    InstanceClienteWCF.Instance.ServicioWCF.DeleteProductos(
                        lista.ToArray()) + " productos";
            } catch (NullReferenceException) {
                MessageBox.Show(this, "No está autentificado.", "Prohibido",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            } catch (Exception ex) {
                MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
                    MessageBoxImage.Exclamation);
            }
        }
        #endregion
        else {
            this.EstadoTextBlock.Text = "No ha seleccionado alguna operación a realizar.";
        }
    }
}
```

En resumen lo que el código anterior efectúa es lo siguiente, ***Para insertar:*** Se obtienen las clases *Producto_Join* que están enlazadas con el *DataGrid,* cada una de estas clases es descompuesta para generar la clase *Producto* y se agrega esta nueva clase a una lista genérica de este último tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *InsertProductos* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.* ***Para actualizar:*** Se obtienen las clases *Producto_Join* desde una propiedad que almacena los productos que fueron modificados por el usuario, cada una de estas clases es descompuesta para generar la clase *Producto* y se agrega esta nueva clase a una lista genérica de este último tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *UpdateProductos* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.* ***Para eliminar:*** Se obtienen las estructuras *Int32 (*contiene el número entero del Id del producto a eliminar*)* desde una propiedad que almacena los productos que fueron eliminados por el usuario, cada una de estas estructuras es agregada a una lista genérica de este tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *DeleteProductos* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.*

5.1.4.5.3 OBTENCIÓN, INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN DE ÓRDENES DE COMPRA.

Se debe notar que el enlace de datos necesario para el *DataGrid* (vista en *MVC*) es único para los cuatro procedimientos (obtención, inserción, actualización y eliminación de órdenes de compra). Para definir el funcionamiento del control *DataGrid* de *WPFControlToolkit.dll* que almacenará los datos, vamos a hacerlo vía código *XAML* y *C#*, el primero nos va servir para tratar todo lo relacionado con la vista mientras que el segundo nos servirá para mantener un control adecuado del comportamiento de la vista.

5.1.4.5.3.1 CÓDIGO XAML PARA ORDEN DE COMPRA GENERAL

El siguiente código *XAML* permite tratar todo lo relacionado con una orden de compra de forma general en relación con la vista:

```
<toolkit:DataGrid x:Name="ordenesDataGrid" ItemsSource="{Binding}"
             AlternationCount="2"
             AutoGenerateColumns="False"
             BorderBrush="Transparent"
             ColumnHeaderStyle="{DynamicResource dgHeaderStyleNegro}"
             RowStyle="{DynamicResource dgRowStyleNegro}"
             CellStyle="{DynamicResource dgCellStyle}"
             SelectionMode="Extended"
             SelectionUnit="FullRow"
             GridLinesVisibility="All"
             VerticalGridLinesBrush="DarkGray"
             RowDetailsVisibilityMode="VisibleWhenSelected"
             CanUserAddRows="False"
             CanUserDeleteRows="False"
             Margin="0,27,0,0" Height="76" VerticalAlignment="Top"
             BeginningEdit="ordenesDataGrid_BeginningEdit"
             RowEditEnding="ordenesDataGrid_RowEditEnding"
             PreviewKeyDown="ordenesDataGrid_PreviewKeyDown">

    <!-- Definición de columnas -->
    <toolkit:DataGrid.Columns>
        <!-- Detalle -->
        <toolkit:DataGridTemplateColumn Header="Detalle" MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <Button Name="DetalleButton" Content="Detalle"
                            Click="DetalleButton_Click" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
        </toolkit:DataGridTemplateColumn>
        <!--Id Orden-->
        <toolkit:DataGridTemplateColumn x:Name="OrdenIdDataGridTemplateColumn"
            Header="Id Orden" MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding OrdenId}" MinHeight="23"
                        MinWidth="100" Padding="0,5,0,0" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
        </toolkit:DataGridTemplateColumn>
        <!-- Fecha de orden -->
        <toolkit:DataGridTemplateColumn Header="Fecha orden" MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Path=FechaOrden, StringFormat=d,
                        ConverterCulture=es-MX}" MinHeight="23" MinWidth="100"
                        Padding="0,5,0,0" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
            <toolkit:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate>
                    <toolkit:DatePicker SelectedDate="{Binding Path=FechaOrden,
                        Mode=TwoWay}" SelectedDateFormat="Short" />
                </DataTemplate>
```

```
            </toolkit:DataGridTemplateColumn.CellEditingTemplate>
        </toolkit:DataGridTemplateColumn>
        <!—Otras definiciones de columnas-->
        <!—...-->
    </toolkit:DataGrid.Columns>
</toolkit:DataGrid>
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *DataGrid* que servirá para mostrar información general sobre las órdenes de compra obtenida desde el código de *C#* y *2)* Definición de las columnas, generalmente la propiedad del control que permita visualizar datos al usuario (*Text*, *TextBinding*, *SelectedDate*, etc.), está enlazada (*Binding*) con la propiedad de la fuente de datos que se desea mostrar. Por ejemplo, para mostrar el Id de la orden, la propiedad *Text* del control *TextBlock* está enlazada con la propiedad *OrdenId* de la lista genérica que sirve como fuente de datos.

5.1.4.5.3.2 CÓDIGO XAML PARA ORDEN DE COMPRA PARTICULAR

El siguiente código *XAML* permite tratar todo lo relacionado con una orden de compra de forma particular (detalles de la orden de compra de forma general) en relación con la vista:

```
<toolkit:DataGrid x:Name="ordenDetalleDataGrid" ItemsSource="{Binding}"
            AlternationCount="2"
            AutoGenerateColumns="False"
            BorderBrush="Transparent"
            ColumnHeaderStyle="{DynamicResource dgHeaderStyleNegro}"
            RowStyle="{DynamicResource dgRowStyleNegro}"
            CellStyle="{DynamicResource dgCellStyle}"
            SelectionMode="Extended"
            SelectionUnit="FullRow"
            GridLinesVisibility="All"
            VerticalGridLinesBrush="DarkGray"
            RowDetailsVisibilityMode="VisibleWhenSelected"
            CanUserAddRows="False"
            CanUserDeleteRows="False"
            Margin="0,107,0,25">
    <toolkit:DataGrid.Columns>
        <!--Producto-->
        <toolkit:DataGridComboBoxColumn x:Name="ProductosDataGridComboBoxColumn"
                  SelectedItemBinding="{Binding Path=Producto}"
                  SelectedValueBinding="{Binding Path=ProductoId}"
                  TextBinding="{Binding Path=Producto}" TextSearch.TextPath="Producto"
                  SelectedValuePath="ProductoId" DisplayMemberPath="Producto"
                  Header="Producto" Width="200" >
            <toolkit:DataGridComboBoxColumn.EditingElementStyle>
                <Style TargetType="ComboBox">
                    <Setter Property="IsEditable" Value="false" />
                    <Setter Property="Background" Value="LightGoldenrodYellow" />
                    <Setter Property="Foreground" Value="Green" />
                    <Setter Property="FontWeight" Value="Bold" />
                    <EventSetter Event="SelectionChanged"
                          Handler="ProductosDataGridComboBoxColumn SelectionChanged" />
                </Style>
            </toolkit:DataGridComboBoxColumn.EditingElementStyle>
        </toolkit:DataGridComboBoxColumn>
        <!--Modelo-->
        <toolkit:DataGridTemplateColumn Header="Modelo" MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Path=Modelo}" MinHeight="23"
                               MinWidth="100" Padding="0,5,0,0" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
        </toolkit:DataGridTemplateColumn>
        <!—Otras definiciones de columnas-->
        <!—...-->
    </toolkit:DataGrid.Columns>
</toolkit:DataGrid>
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *DataGrid* que servirá para mostrar información detallada sobre la orden de compra obtenida desde el código de *C#* y *2)* Definición de las columnas, generalmente la propiedad del control que permita visualizar datos al usuario (*Text*, *TextBinding*, *SelectedDate*, etc.), está enlazada (*Binding*) con la propiedad de la fuente de datos que se desea mostrar. Por ejemplo, para mostrar el nombre del producto, la propiedad *SelectedItemBinding* del control *ComboBox* está enlazada con la propiedad *Productos* y la propiedad *SelectedValueBinding* del control *ComboBox* está enlazada con la propiedad *ProductoId* de la lista genérica que sirve como fuente de datos. Este tipo de enlace (dos propiedades enlazadas en un *ComboBox*) sirve para conocer cuál es el producto que se seleccionó y poder así, actualizar los demás controles.

5.1.4.5.3.3 CÓDIGO C# PARA OBTENCIÓN DE ÓRDENES DE COMPRA GENERALES

El siguiente código *C#* permite tratar todo lo relacionado el adecuado comportamiento de la vista:

```
private void VerOrdenRibbonButton_Executed(object sender, ExecutedRoutedEventArgs e) {
    try {
        // Usando el procedimiento almacenado obtenemos todo lo relacionado con productos
        ClienteWCF.Sp_Orden sp_ordenes =
            InstanceClienteWCF.Instance.ServicioWCF.Sp_GetOrden();

        // Llenamos las listas genéricas
        Colecciones.Ordenes = sp_ordenes.ListaOrden.ToList();
        Colecciones.OrdenesDetalle = sp_ordenes.ListaDetalleOrden.ToList();
        Colecciones.OrdenesMetodoEnvio = sp_ordenes.ListaMetodosEnvioOrden.ToList();
        Colecciones.OrdenesFormaPago = sp_ordenes.ListaFormasPagoOrden.ToList();
        Colecciones.OrdenesStatus = sp_ordenes.ListaStatusOrden.ToList();

        // Obtenemos el DataGrid
        OrdenesDataGridWPF dgWPF =
            (OrdenesDataGridWPF)this.GridOrdenesDataGrid.Children[0];
        // Vinculamos las listas genéricas con los controles
        dgWPF.ordenesDataGrid.ItemsSource = Colecciones.Ordenes;
        // Limpiar el estado previo de la barra de estado
        this.EstadoTextBlock.Text = String.Empty;
    } catch (NullReferenceException) {
        MessageBox.Show(this, "No está autentificado.", "Prohibido",
            MessageBoxButton.OK, MessageBoxImage.Exclamation);
    } catch (Exception ex) {
        MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
            MessageBoxImage.Exclamation);
    }
}
```

En resumen, el código anterior realiza es lo siguiente: *1)* Obtener desde el método *Sp_GetOrden* toda la información pertinente a las órdenes de compra que se han hecho a la empresa, *2)* Asignar los valores obtenidos a listas genéricas locales y *3)* Asignar a la propiedad *ItemsSource* de los controles que muestren información al usuario la lista genérica local correspondiente.

5.1.4.5.3.4 CÓDIGO C# PARA OBTENCIÓN DE ÓRDENES DE COMPRA PARTICULARES

El siguiente código *C#* permite tratar todo lo relacionado el adecuado comportamiento de la vista:

```
private void DetalleButton_Click(object sender, RoutedEventArgs e) {
    DataGridRow row =
        YnnafSoftwareClientWCFWPFCSharp.Utilidades.UtilidadesGraficas.GetRow(
        this.ordenesDataGrid,
        this.ordenesDataGrid.Items.IndexOf(this.ordenesDataGrid.CurrentItem));
    // Convertir el renglón al tipo adecuado
    Orden_Join orden = (Orden_Join)row.Item;

    IEnumerable<OrdenDetalle_Join> match = from q in Colecciones.OrdenesDetalle
```

```
                                        where q.OrdenId == orden.OrdenId
                                        select q;

    RibbonWindow rb = (RibbonWindow)((Grid)((Grid)this.Parent).Parent).Parent;
    if (rb.EditarOrdenRibbonToggleButton.IsChecked == true) {
        DetalleWindow detalleWindow = new DetalleWindow(rb.Background, orden,
            match.ToList(),
            YnnafSoftwareClientWCFWPFCSharp.Data.ProcedimientoOrden.Update);
        detalleWindow.Show();
    } else if (rb.EliminarOrdenRibbonToggleButton.IsChecked == true) {
        DetalleWindow detalleWindow = new DetalleWindow(rb.Background, orden,
            match.ToList(),
            YnnafSoftwareClientWCFWPFCSharp.Data.ProcedimientoOrden.Delete);
        detalleWindow.Show();
    } else {
        DetalleWindow detalleWindow = new DetalleWindow(rb.Background, orden,
            match.ToList(),
            YnnafSoftwareClientWCFWPFCSharp.Data.ProcedimientoOrden.Select);
        detalleWindow.Show();
    }
}
```

En resumen, el código anterior realiza es lo siguiente: *1)* Obtener la clase que está vinculada con el DataGrid, 2) Buscar en las colecciones locales el detalle de la orden seleccionada y 3) Mostrar una ventana con la información de la orden de compra

5.1.4.5.3.5 CÓDIGO C# PARA INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN DE ÓRDENES DE COMPRA PARTICULARES

```
private void GuardarOrdenButton Click(object sender, RoutedEventArgs e) {
    // Obtener el renglón actual
    DataGridRow row =
        YnnafSoftwareClientWCFWPFCSharp.Utilidades.UtilidadesGraficas.GetRow(
        this.ordenDataGrid, 0);
    // Convertir el renglón al tipo adecuado
    ClienteWCF.Orden_Join orden = (ClienteWCF.Orden_Join)row.Item;
    String patronProductos = String.Empty;

    if (this.procedimientoTextBlock.Text.Equals("Insert")) {
        // Obtener la cantidad y el Id del producto
        int i = 0;
        foreach (OrdenDetalle Join ord in this.ordenDetalleDataGrid.Items) {
            patronProductos += ord.Cantidad.ToString();
            patronProductos += ",";
            patronProductos += ord.ProductoId.ToString();
            if (i != this.ordenDetalleDataGrid.Items.Count - 1)
                patronProductos += ",";
            i++;
        }

        // Crear la orden
        int ordenId = 0;
        String descripcionResultado = String.Empty;
        InstanceClienteWCF.Instance.ServicioWCF.Sp_CreateOrden(
            new Orden {
                CargoExtra = orden.CargoExtra,
                ClienteId = orden.ClienteId,
                FechaAtencion = orden.FechaAtencion,
                FechaEntrega = orden.FechaEntrega,
                FechaEnvio = orden.FechaEnvio,
                FechaModificacion = DateTime.Now,
                FechaOrden = orden.FechaOrden,
                FormaPagoOrdenId = orden.FormaPagoOrdenId,
                Iva = orden.Iva,
                MetodoEnvioOrdenId = orden.MetodoEnvioOrdenId,
                NumeroCuenta = orden.NumeroCuenta,
                OrdenId = 0,
                StatusOrdenId = orden.StatusOrdenId
```

```
            }, patronProductos, ref ordenId, ref descripcionResultado);
        this.ordenIdTextBlock.Text = ordenId.ToString();
        this.EstadoTextBlock.Text = descripcionResultado;
    } else if (this.procedimientoTextBlock.Text.Equals("Update")) {
        // Obtener los datos de la orden
        InstanceClienteWCF.Instance.ServicioWCF.UpdateOrden(
            new Orden {
                CargoExtra = orden.CargoExtra,
                ClienteId = orden.ClienteId,
                FechaAtencion = orden.FechaAtencion,
                FechaEntrega = orden.FechaEntrega,
                FechaEnvio = orden.FechaEnvio,
                FechaModificacion = DateTime.Now,
                FechaOrden = orden.FechaOrden,
                FormaPagoOrdenId = orden.FormaPagoOrdenId,
                Iva = orden.Iva,
                MetodoEnvioOrdenId = orden.MetodoEnvioOrdenId,
                NumeroCuenta = orden.NumeroCuenta,
                OrdenId = Convert.ToInt32(this.ordenIdTextBlock.Text),
                StatusOrdenId = orden.StatusOrdenId
            });

        // Obtener el detalle de la orden
        List<OrdenDetalle> listaDetalleOrden =
            new List<OrdenDetalle>();
        foreach (OrdenDetalle_Join ordDet in this.ordenDetalleDataGrid.Items) {
            listaDetalleOrden.Add(new OrdenDetalle {
                DetalleOrdenId = ordDet.DetalleOrdenId,
                Cantidad = ordDet.Cantidad,
                FechaModificacion = DateTime.Now,
                OrdenId = Convert.ToInt32(this.ordenIdTextBlock.Text),
                PrecioUnitario = ordDet.PrecioUnitario,
                ProductoId = ordDet.ProductoId
            });
        }
        int editados =
            InstanceClienteWCF.Instance.ServicioWCF.UpdateDetalleOrden(
            listaDetalleOrden.ToArray());
        this.EstadoTextBlock.Text = "Orden editada exitosamente, se editaron: " +
            editados.ToString();
    }
    else if (this.procedimientoTextBlock.Text.Equals("Delete")) {
        int idOrden = Convert.ToInt32(this.ordenIdTextBlock.Text);
        InstanceClienteWCF.Instance.ServicioWCF.DeleteOrden(idOrden);
        this.ordenDataGrid.ItemsSource = new List<Orden_Join>();
        this.ordenDetalleDataGrid.ItemsSource = new List<OrdenDetalle_Join>();
        this.EstadoTextBlock.Text = "Orden eliminada exitosamente";
    }
}
```

En resumen lo que el código anterior efectúa es lo siguiente: ***Para insertar:*** Se obtiene la clase *Orden_Join* que está enlazada con el primer *DataGrid* y las clases *OrdenDetalle_Join* que están enlazadas en el segundo *DataGrid*, las clases del segundo *DataGrid* se utilizan para generar el patrón *Cantidad, ProductoId_n*. Una vez que se generó el patrón, se llama al contrato de operación *CreateOrden* y se le pasa como parámetro la orden creada (obtenida del primer *DataGrid* y convertida de *Orden_Join* a *Orden*), la cadena que contiene el patrón *{Cantidad, ProductoId_1, Cantidad, ProductoId_2, …}* y los parámetros por referencia necesarios. ***Para actualizar:*** *Paso 1)* Se obtiene la clase *Orden_Join* que está enlazada con el primer *DataGrid, esta clase se convierte a la clase Orden.* Una vez hecho esto, se manda a llamar el contrato de operación *UpdateOrden* pasándole como parámetro la clase Orden. *Paso 2)* Se obtiene cada una de las clases *OrdenDetalle_Join* que están enlazadas en el segundo *DataGrid*, cada una de estas clases es descompuesta para generar la clase *OrdenDetalle* y se agrega a una lista genérica de este último tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *UpdateDetalleOrden* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método

de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.* ***Para eliminar:*** Se obtiene el número de orden de compra que se desea eliminar desde el *TextBlock* que lo contiene y se manda a llamar al contrato de operación *DeleteOrden*, el cual recibe como parámetro el número de orden a eliminar.

5.1.4.5.4 OBTENCIÓN, INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN DE CLIENTES.

Se debe notar que el enlace de datos necesario para el *DataGrid* (vista en *MVC*) es único para los cuatro procedimientos (obtención, inserción, actualización y eliminación de clientes). Para definir el funcionamiento del control *DataGrid* de *WPFControlToolkit.dll* que almacenará los datos, vamos a hacerlo vía código *XAML* y *C#*, el primero nos va servir para tratar todo lo relacionado con la vista mientras que el segundo nos servirá para mantener un correcto funcionamiento en la vista.

5.1.4.5.4.1 CÓDIGO XAML

El siguiente código *XAML* permite tratar todo lo relacionado con la vista:

```
<!--Inicialización del Data Grid-->
<toolkit:DataGrid x:Name="clientesDataGrid" ItemsSource="{Binding}"
             AlternationCount="2"
             AutoGenerateColumns="False"
             BorderBrush="Transparent"
             Background="Transparent"
             ColumnHeaderStyle="{DynamicResource dgHeaderStyleAzul}"
             RowStyle="{DynamicResource dgRowStyleAzul}"
             CellStyle="{DynamicResource dgCellStyle}"
             SelectionMode="Extended"
             SelectionUnit="FullRow"
             GridLinesVisibility="All"
             VerticalGridLinesBrush="DarkGray"
             RowDetailsVisibilityMode="VisibleWhenSelected"
             CanUserAddRows="False"
             CanUserDeleteRows="False"
             BeginningEdit="clientesDataGrid_BeginningEdit"
             RowEditEnding="clientesDataGrid_RowEditEnding"
             PreviewKeyDown="clientesDataGrid_PreviewKeyDown">

    <!-- Definición de columnas -->
    <toolkit:DataGrid.Columns>
        <!--Nombre-->
        <toolkit:DataGridTemplateColumn x:Name="NombreClienteDataGridTemplateColumn"
            Header="Nombre o Suc." MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Nombre}" MinHeight="23"
                        MinWidth="100" Padding="0,5,0,0" />
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
            <toolkit:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate x:Name="NombreClienteDataTemplate">
                    <TextBox Style="{StaticResource TextBoxInError}">
                        <Binding Path="Nombre" UpdateSourceTrigger="PropertyChanged"
                            Mode="TwoWay">
                        </Binding>
                    </TextBox>
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellEditingTemplate>
        </toolkit:DataGridTemplateColumn>
        <!--Username-->
        <toolkit:DataGridTemplateColumn x:Name="UsernameDataGridTemplateColumn"
            Header="Username" MinWidth="100">
            <toolkit:DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Username}" MinHeight="23"
                    MinWidth="100" Padding="0,5,0,0" />
```

```
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellTemplate>
            <toolkit:DataGridTemplateColumn.CellEditingTemplate>
                <DataTemplate x:Name="UsernameDataTemplate">
                    <TextBox Style="{StaticResource TextBoxInError}">
                        <Binding Path="Username" UpdateSourceTrigger="PropertyChanged"
                            Mode="TwoWay">
                        </Binding>
                    </TextBox>
                </DataTemplate>
            </toolkit:DataGridTemplateColumn.CellEditingTemplate>
        </toolkit:DataGridTemplateColumn>
        <!—Otras definiciones de columnas-->
        <!—...-->
    </toolkit:DataGrid.Columns>
</toolkit:DataGrid>
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *DataGrid* que servirá para mostrar información sobre productos obtenida desde el código de *C#*. *2)* Definición de las columnas, generalmente la propiedad del control que permita visualizar datos al usuario (*Text*, *TextBinding*, *SelectedItemBinding*, etc.), está enlazada (*Binding*) con la propiedad de la fuente de datos que se desea mostrar. Por ejemplo, para mostrar el nombre de usuario (*username*), la propiedad *Text* del control *TextBlock* está enlazada con la propiedad *Username* de la lista genérica que sirve como fuente de datos.

5.1.4.5.2.1.2 CÓDIGO C# PARA OBTENCIÓN DE CLIENTES

El siguiente código *C#* permite tratar todo lo relacionado al modelo:

```
private void VerClienteRibbonButton Executed(object sender, ExecutedRoutedEventArgs e) {
    try {
        // Llenamos las listas genéricas
        Colecciones.Clientes =
            InstanceClienteWCF.Instance.ServicioWCF.GetClientes().ToList();
        Colecciones.TiposCliente =
            InstanceClienteWCF.Instance.ServicioWCF.GetTiposCliente().ToList();

        // Obtenemos el DataGrid
        ClientesDataGridWPF dgWPF =
            (ClientesDataGridWPF)this.GridClientesDataGrid.Children[0];

        // Vinculamos las listas genéricas con los controles
        dgWPF.TipoClienteDataGridComboBoxColumn.ItemsSource =
            Colecciones.TiposCliente.Select(p => p.TipoCliente);
        dgWPF.clientesDataGrid.ItemsSource = Colecciones.Clientes;

        // Limpiar el estado previo de la barra de estado
        this.EstadoTextBlock.Text = String.Empty;
    } catch (NullReferenceException) {
        MessageBox.Show(this, "No está autentificado.", "Prohibido", MessageBoxButton.OK,
            MessageBoxImage.Exclamation);
    } catch (Exception ex) {
        MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
            MessageBoxImage.Exclamation);
    }
}
```

En resumen lo que el código anterior efectúa es lo siguiente: *1)* Obtener desde el método *GetClientes* toda la información referente a los clientes de la empresa y desde el método *GetTiposCliente* la información referente a los tipos de clientes permitidos por la empresa, *2)* Asignar los valores obtenidos a listas genéricas locales y *3)* Asignar a la propiedad *ItemsSource* de los controles que muestren información al usuario la lista genérica local correspondiente.

5.1.4.5.2.1.3 CÓDIGO C# PARA INSERCIÓN, ACTUALIZACIÓN Y ELIMINACIÓN DE CLIENTES

Conociendo que el código *XAML* es el mismo que el utilizado en la obtención de clientes, el código *C#* para insertar, actualizar y eliminar clientes es el siguiente:

```
private void GuardarClienteRibbonButton_Executed(object sender,
    ExecutedRoutedEventArgs e) {
    if (this.ribbon.SelectedTab.Label == "Clientes") {
        #region INSERT Clientes
        if (this.AgregarClienteRibbonToggleButton.IsChecked == true &&
            this.EditarClienteRibbonToggleButton.IsChecked == false &&
            this.EliminarClienteRibbonToggleButton.IsChecked == false) {
            try {
                List<ClienteWCF.Cliente> lista = new
                    List<YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Cliente>();
                ClientesDataGridWPF dgWPF =
                    (ClientesDataGridWPF)this.GridClientesDataGrid.Children[0];

                dgWPF.clientesDataGrid.CanUserAddRows = false;
                dgWPF.clientesDataGrid.CanUserDeleteRows = false;

                foreach (ClienteWCF.Cliente_Join p in dgWPF.clientesDataGrid.Items) {
                    lista.Add(new YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Cliente {
                        Nombre_o_Sucursal = p.Nombre,
                        ApellidoPaterno_o_NumSucursal = p.ApellidoPaterno,
                        ApellidoMaterno = p.ApellidoMaterno,
                        Username = p.Username,
                        Password = Encriptacion.EncriptarSha512(p.Password),
                        FechaAlta = p.FechaAlta,
                        TipoClienteId = p.TipoClienteId,
                        Calle = p.Calle,
                        Numero = p.Numero,
                        Pais = p.Pais,
                        Estado = p.Estado,
                        Ciudad = p.Ciudad,
                        CodigoPostal = p.CodigoPostal,
                        Telefono = p.Telefono,
                        Fax = p.Fax,
                        Email = p.Email,
                        FechaModificacion = DateTime.Now
                    });
                }
                this.EstadoTextBlock.Text = "Se insertaron: " +
                    InstanceClienteWCF.Instance.ServicioWCF.
                        InsertClientes(lista.ToArray()) + " clientes";
            } catch (NullReferenceException) {
                MessageBox.Show(this, "No está autentificado.", "Prohibido",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            } catch (Exception ex) {
                MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
                    MessageBoxImage.Exclamation);
            }
        }
        #endregion
        #region UPDATE Clientes
        else if (this.AgregarClienteRibbonToggleButton.IsChecked == false &&
            this.EditarClienteRibbonToggleButton.IsChecked == true &&
            this.EliminarClienteRibbonToggleButton.IsChecked == false) {
            try {
                List<ClienteWCF.Cliente> lista = new
                    List<YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Cliente>();
                ClientesDataGridWPF dgWPF =
                    (ClientesDataGridWPF)this.GridClientesDataGrid.Children[0];

                foreach (ClienteWCF.Cliente_Join p in dgWPF.ClientesEditados) {
                    if (this.PasswordRibbonToggleButton.IsChecked == true) {
                        lista.Add(new YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Cliente {
                            ClienteId = p.ClienteId,
                            Nombre_o_Sucursal = p.Nombre,
                            ApellidoPaterno_o_NumSucursal = p.ApellidoPaterno,
                            ApellidoMaterno = p.ApellidoMaterno,
```

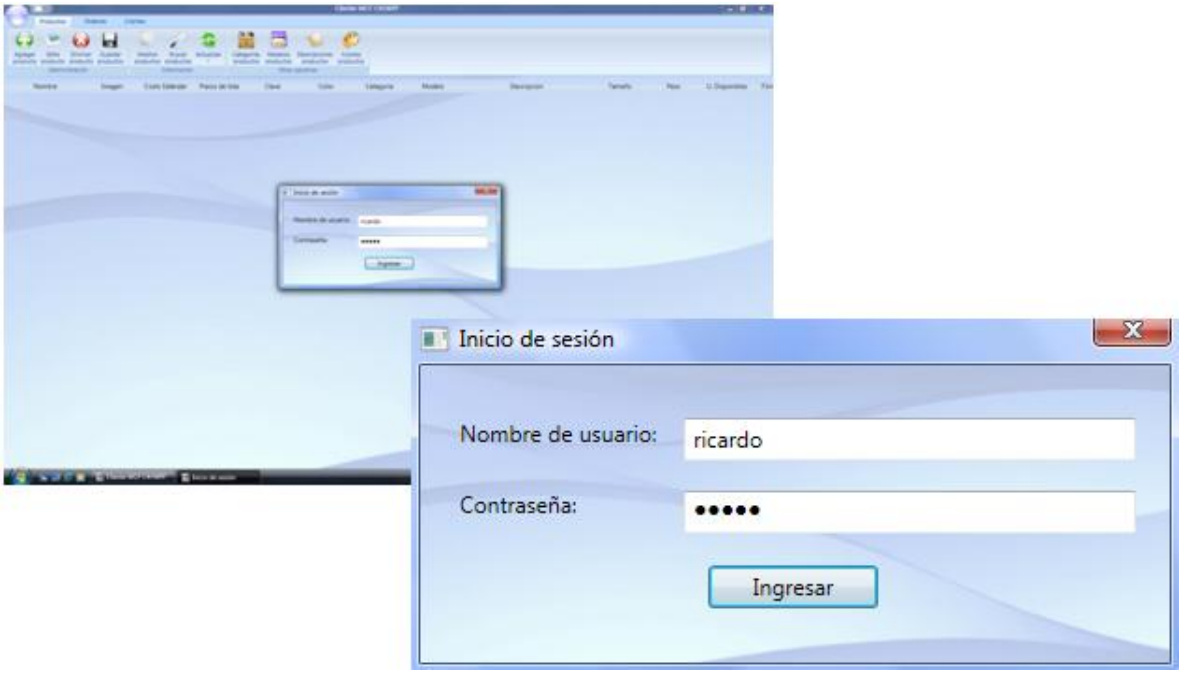
```
                            Username = p.Username,
                            Password = Encriptacion.EncriptarSha512(p.Password),
                            FechaAlta = p.FechaAlta,
                            TipoClienteId = p.TipoClienteId,
                            Calle = p.Calle,
                            Numero = p.Numero,
                            Pais = p.Pais,
                            Estado = p.Estado,
                            Ciudad = p.Ciudad,
                            CodigoPostal = p.CodigoPostal,
                            Telefono = p.Telefono,
                            Fax = p.Fax,
                            Email = p.Email,
                            FechaModificacion = DateTime.Now
                        });
                    } else {// Si no esta seleccionado que el password es modificable
                        lista.Add(new YnnafSoftwareClientWCFWPFCSharp.ClienteWCF.Cliente {
                            ClienteId = p.ClienteId,
                            Nombre_o_Sucursal = p.Nombre,
                            ApellidoPaterno_o_NumSucursal = p.ApellidoPaterno,
                            ApellidoMaterno = p.ApellidoMaterno,
                            Username = p.Username,
                            Password = p.Password,
                            FechaAlta = p.FechaAlta,
                            TipoClienteId = p.TipoClienteId,
                            Calle = p.Calle,
                            Numero = p.Numero,
                            Pais = p.Pais,
                            Estado = p.Estado,
                            Ciudad = p.Ciudad,
                            CodigoPostal = p.CodigoPostal,
                            Telefono = p.Telefono,
                            Fax = p.Fax,
                            Email = p.Email,
                            FechaModificacion = DateTime.Now
                        });
                    }
                }
                this.EstadoTextBlock.Text = "Se editaron: " +
                    InstanceClienteWCF.Instance.ServicioWCF.UpdateClientes(
                        lista.ToArray()) + " clientes";
            } catch (NullReferenceException) {
                MessageBox.Show(this, "No está autentificado.", "Prohibido",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            } catch (Exception ex) {
                MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
                    MessageBoxImage.Exclamation);
            }
        }
        #endregion
        #region DELETE Clientes
        else if (this.AgregarClienteRibbonToggleButton.IsChecked == false &&
            this.EditarClienteRibbonToggleButton.IsChecked == false &&
            this.EliminarClienteRibbonToggleButton.IsChecked == true) {
            try {
                List<int> lista = new List<int>();
                ClientesDataGridWPF dgWPF =
                    (ClientesDataGridWPF)this.GridClientesDataGrid.Children[0];

                foreach (int n in dgWPF.ClientesEliminados) {
                    lista.Add(n);
                }
                this.EstadoTextBlock.Text = "Se eliminaron: " +
                    InstanceClienteWCF.Instance.ServicioWCF.DeleteClientes(
                        lista.ToArray()) + " clientes";
            } catch (NullReferenceException) {
                MessageBox.Show(this, "No está autentificado.", "Prohibido",
                    MessageBoxButton.OK, MessageBoxImage.Exclamation);
            } catch (Exception ex) {
                MessageBox.Show(this, ex.Message, "Error", MessageBoxButton.OK,
                    MessageBoxImage.Exclamation);
```

```
            }
        }
        #endregion
        else {
            this.EstadoTextBlock.Text = "No ha seleccionado alguna operación a realizar.";
        }
    }
}
```

En resumen lo que el código anterior efectúa es lo siguiente, ***Para insertar:*** Se obtienen las clases *Cliente_Join* que están enlazadas con el *DataGrid,* cada una de estas clases es descompuesta para generar la clase *Cliente* y se agrega esta nueva clase a una lista genérica de este último tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *InsertClientes* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.* ***Para actualizar:*** Se obtienen las clases *Cliente_Join* desde una propiedad que almacena los productos que fueron modificados por el usuario, cada una de estas clases es descompuesta para generar la clase *Producto* y se agrega esta nueva clase a una lista genérica de este último tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *UpdateClientes* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.* ***Para eliminar:*** Se obtienen las estructuras *Int32 (*contiene el número entero del Id del cliente a eliminar*)* desde una propiedad que almacena los productos que fueron eliminados por el usuario, cada una de estas estructuras es agregada a una lista genérica de este tipo. Una vez que se generó todo el contenido de la lista genérica, se llama al contrato de operación *DeleteClientes* y se le pasa como parámetro la lista genérica pero como arreglo genérico usando el método de extensión *ToArray* porque así se pide en la definición del *WSDL* importado del servicio *WCF.*

5.1.5 DESPLIEGUE

AUTENTICACIÓN DE USUARIO



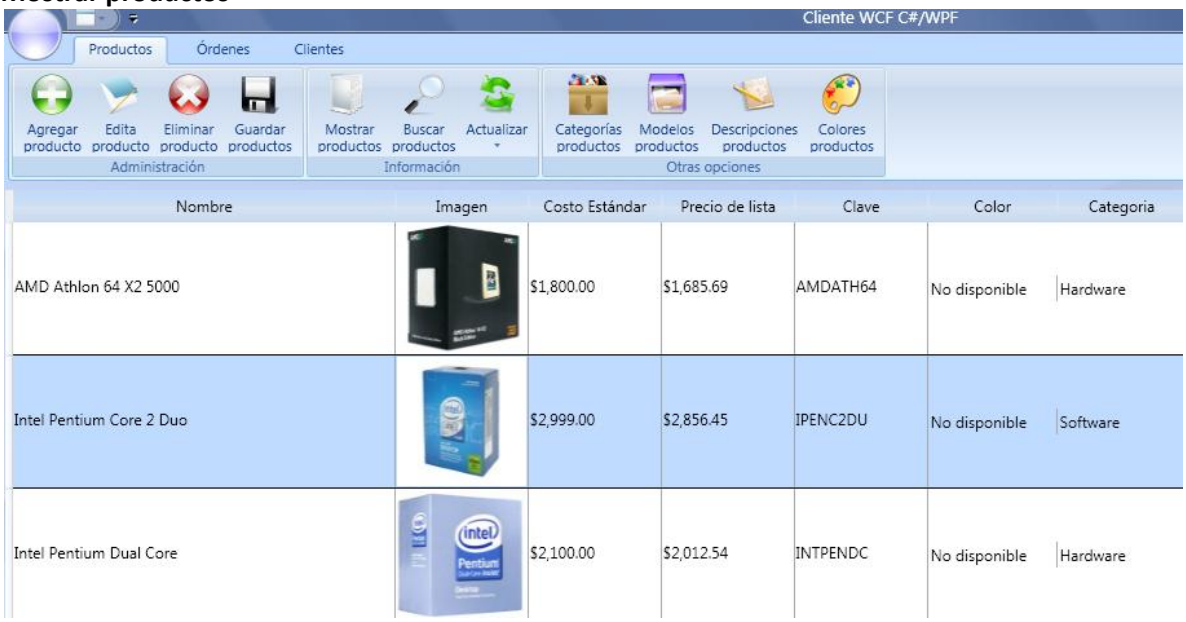
Inserción de nombre de usuario y contraseña. Si tanto el nombre de usuario como la contraseña son correctos, el usuario podrá utilizar las funciones del programa en su totalidad, de lo contrario el programa no podrá ser utilizado en ninguna de sus funciones.

MANIPULACIÓN DE PRODUCTOS

Tabulación productos

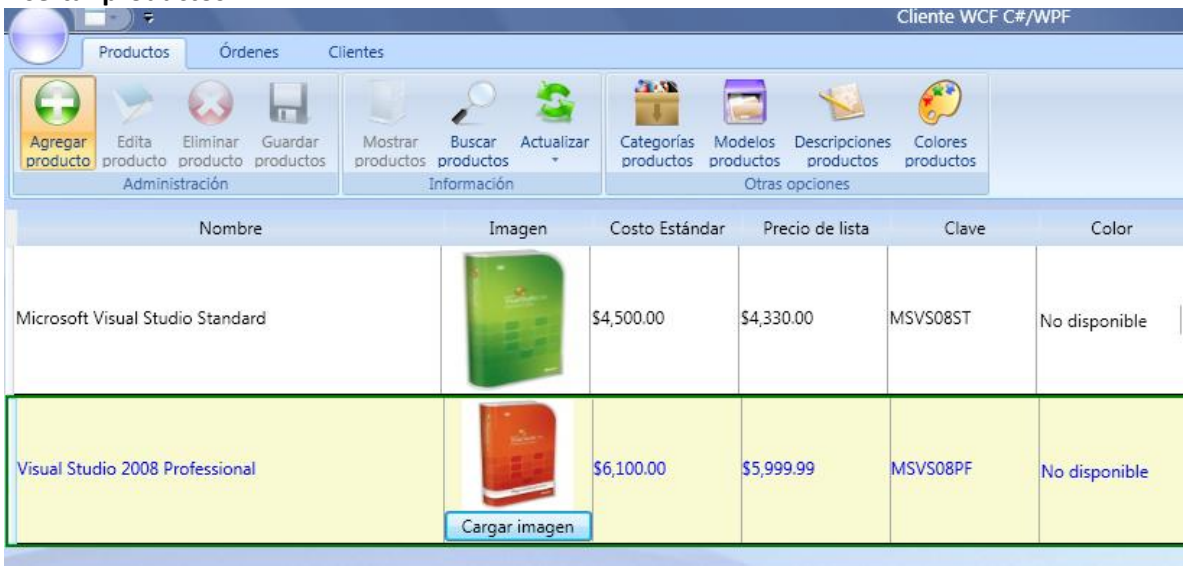
Tabulación Productos. Donde: El botón ***Agregar producto*** habilita al usuario a agregar un nuevo producto a la base de datos (RAM), el botón ***Editar producto*** habilita al usuario a modificar un producto existente en la base de datos (RAM), el botón ***Eliminar producto*** habilita al usuario a eliminar un producto existente en la base de datos (RAM), el botón ***Guardar producto*** permite al usuario guardar las modificaciones hechas por cualquiera de los tres botones anteriores de forma permanente, el botón ***Mostrar productos*** permite al usuario ver todos los productos del sistema, elbotón ***Buscar producto*** permite al usuario buscar productos específicos de acuerdo a su nombre, el botón ***Categorias productos*** permite manipular todo lo referente a categorías nuevas o existentes de productos, el botón ***Modelos productos*** permite manipular todo lo referente a modelos nuevos o existentes de productos, el botón ***Descripciones productos*** permite manipular todo lo referente a descripciones nuevas o existentes de productos, el botón ***Colores productos*** permite manipular todo lo referente a colores nuevos o existentes de productos y el botón ***Actualizar*** permite al usuario actualizar en la tabulación *Productos* los cambios hechos en *Categorías*, *Modelos*, *Descripciones* y *Colores* de productos.

Mostrar productos



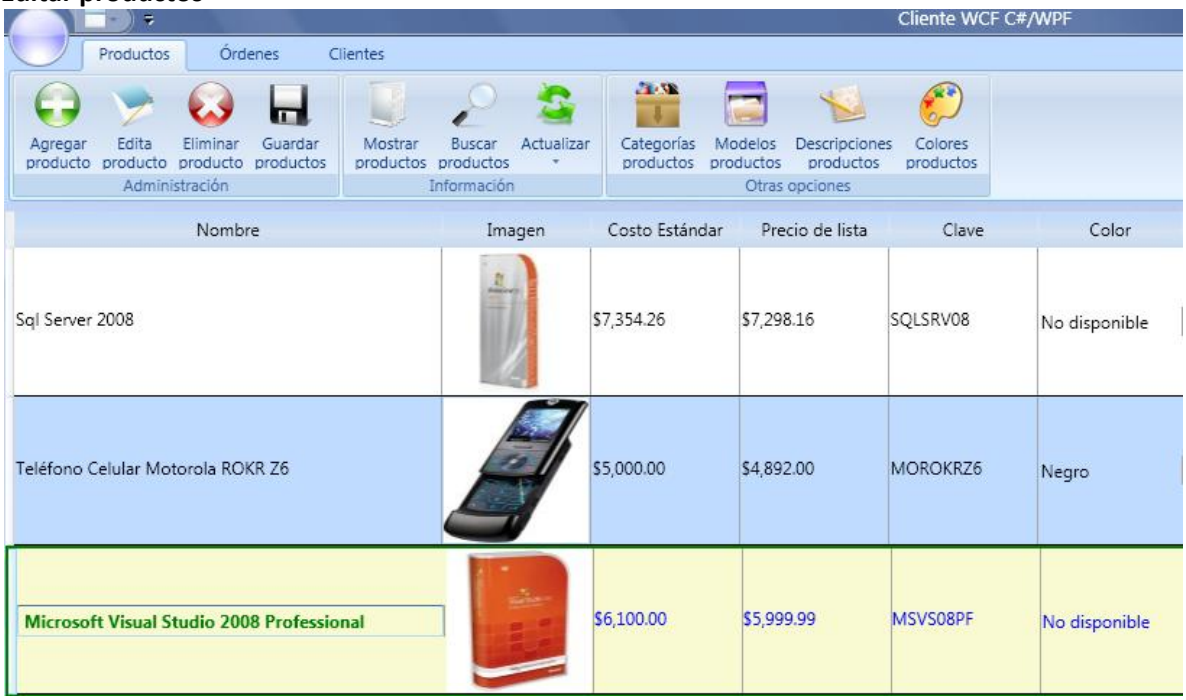
Mostrar productos. Si el usuario presiona el botón ***Mostrar Productos***, el programa visualizará todos los productos contenidos en la base de datos del sistema. A través de esta ventana se pueden manipular todos los productos.

Insertar productos



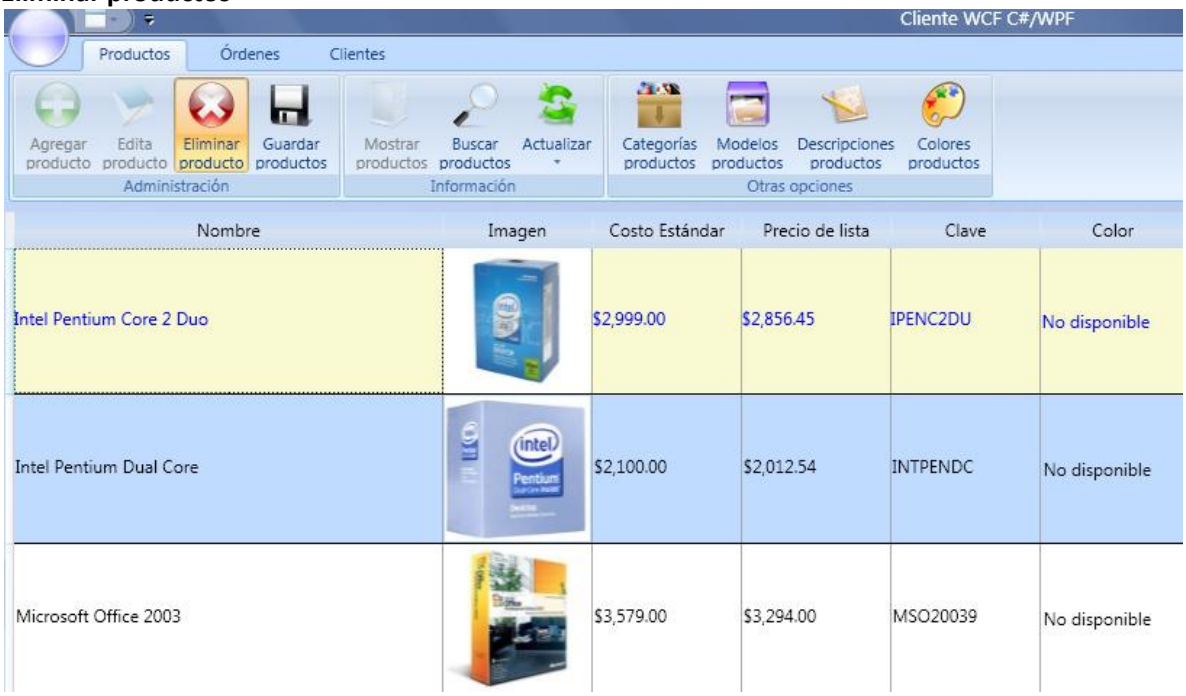
Inserción de productos: 1) Se presiona el botón ***Insertar producto***, 2) Se agregan cuantos productos sean necesarios, una vez que se insertaron los productos deseados, 3) Se presiona el botón ***Guardar productos.*** En el ejemplo anterior se insertaron los productos: “Microsoft Visual Studio 2008 Standard” y “Visual Studio 2008 Professional”

Editar productos



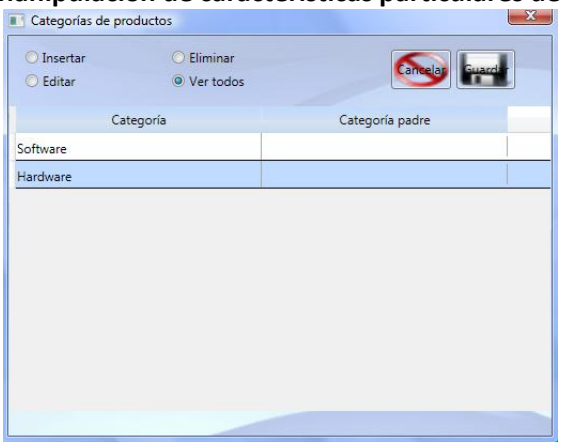
Edición de productos: 1) Se presiona el botón ***Editar producto***, 2) Cualquiera de los productos que se visualicen al usuario puede ser modificado, una vez que se hicieron las modificaciones necesarias, 3) Se presiona el botón ***Guardar productos***. En el ejemplo anterior se modificó el nombre del producto "Visual Studio 2008 Professional" (puesto en el ejemplo anterior) por "Microsoft Visual Studio 2008 Professional"

Eliminar productos



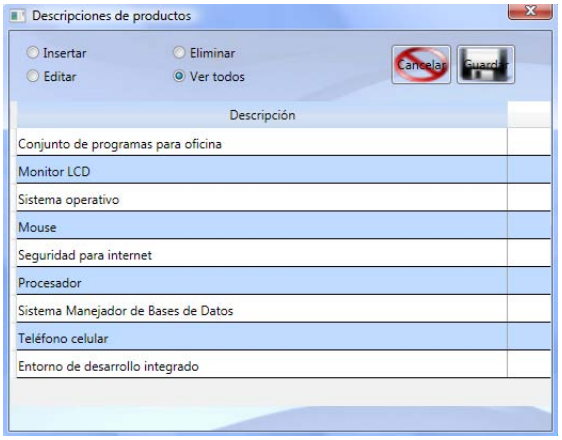
Eliminación de productos: 1) Se presiona el botón ***Eliminar productos***, 2) Cualquiera de los productos que se visualicen al usuario puede ser eliminado, seleccionándolo y presionando la tecla *Suprimir*, una vez que se eliminaron los productos, 3) Se presiona el botón ***Guardar productos***. En el ejemplo anterior se eliminó el producto "AMD Athlon 64 X2 5000" (Comparar con la pantalla *Insertar productos*)

Manipulación de características particulares de productos



Ventana que permite manipular todo lo relacionado a categorías de productos

Ventana que permite manipular todo lo relacionado a modelos de productos



Ventana que permite manipular todo lo relacionado a descripciones de productos

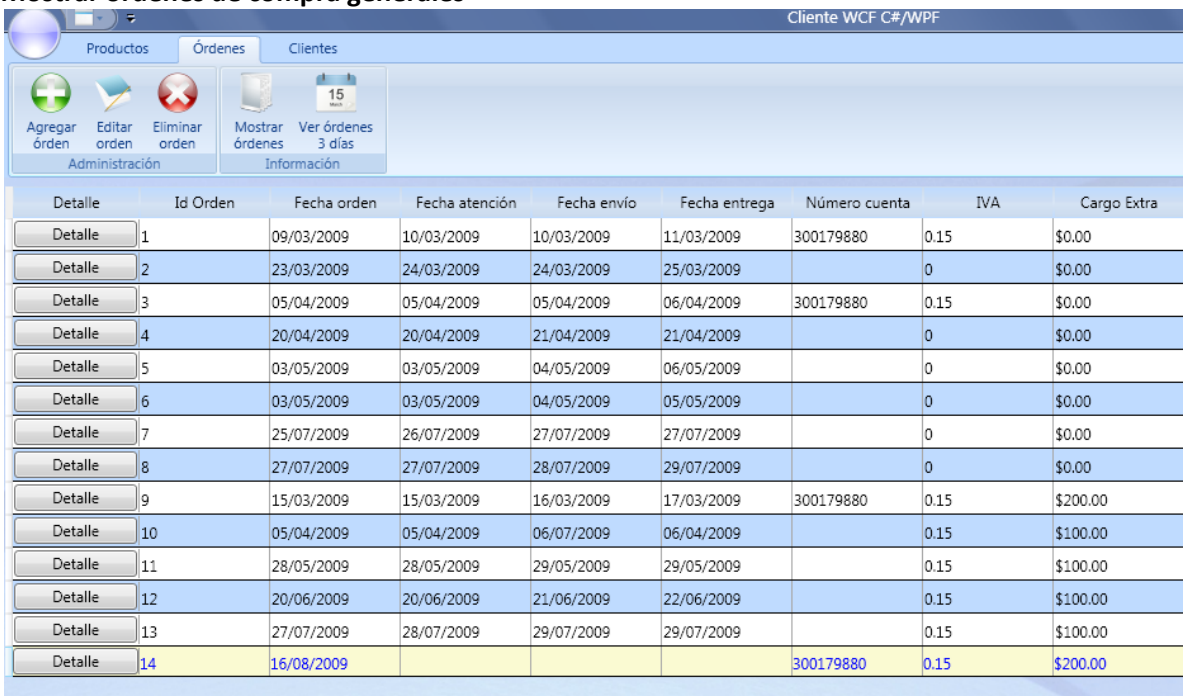
Ventana que permite manipular todo lo relacionado a colores de productos

MANIPULACIÓN DE ÓRDENES

Tabulación órdenes

Tabulación Órdenes. Donde: El botón ***Agregar orden*** habilita al usuario a agregar una nueva orden de compra a la base de datos (RAM), el botón ***Editar orden*** habilita al usuario a modificar una orden de compra existente en la base de datos (RAM), el botón ***Eliminar orden*** habilita al usuario a eliminar una orden de compra existente en la base de datos (RAM), el botón ***Mostrar órdenes*** permite al usuario ver todas las órdenes de compra del sistema y el botón ***Ver órdenes 3 días*** permite al usuario ver todas las órdenes hechas en los últimos tres días.

Mostrar órdenes de compra generales



Detalle	Id Orden	Fecha orden	Fecha atención	Fecha envío	Fecha entrega	Número cuenta	IVA	Cargo Extra
Detalle	1	09/03/2009	10/03/2009	10/03/2009	11/03/2009	300179880	0.15	$0.00
Detalle	2	23/03/2009	24/03/2009	24/03/2009	25/03/2009		0	$0.00
Detalle	3	05/04/2009	05/04/2009	05/04/2009	06/04/2009	300179880	0.15	$0.00
Detalle	4	20/04/2009	20/04/2009	21/04/2009	21/04/2009		0	$0.00
Detalle	5	03/05/2009	03/05/2009	04/05/2009	06/05/2009		0	$0.00
Detalle	6	03/05/2009	03/05/2009	04/05/2009	05/05/2009		0	$0.00
Detalle	7	25/07/2009	26/07/2009	27/07/2009	27/07/2009		0	$0.00
Detalle	8	27/07/2009	27/07/2009	28/07/2009	29/07/2009		0	$0.00
Detalle	9	15/03/2009	15/03/2009	16/03/2009	17/03/2009	300179880	0.15	$200.00
Detalle	10	05/04/2009	05/04/2009	06/07/2009	06/04/2009		0.15	$100.00
Detalle	11	28/05/2009	28/05/2009	29/05/2009	29/05/2009		0.15	$100.00
Detalle	12	20/06/2009	20/06/2009	21/06/2009	22/06/2009		0.15	$100.00
Detalle	13	27/07/2009	28/07/2009	29/07/2009	29/07/2009		0.15	$100.00
Detalle	14	16/08/2009				300179880	0.15	$200.00

Mostrar órdenes generales. Si el usuario presiona el botón ***Mostrar Órdenes***, el programa visualizará todas las órdenes de compra contenidas en la base de datos del sistema. Si el usuario presiona el botón ***Ver órdenes 3 días*** el programa visualizará las órdenes de compra contenidas en la base de datos del sistema efectuadas los últimos 3 días. A través de esta ventana no se pueden manipular órdenes de compra, únicamente se pueden consultar.

Mostrar orden de compra particular a partir de la general

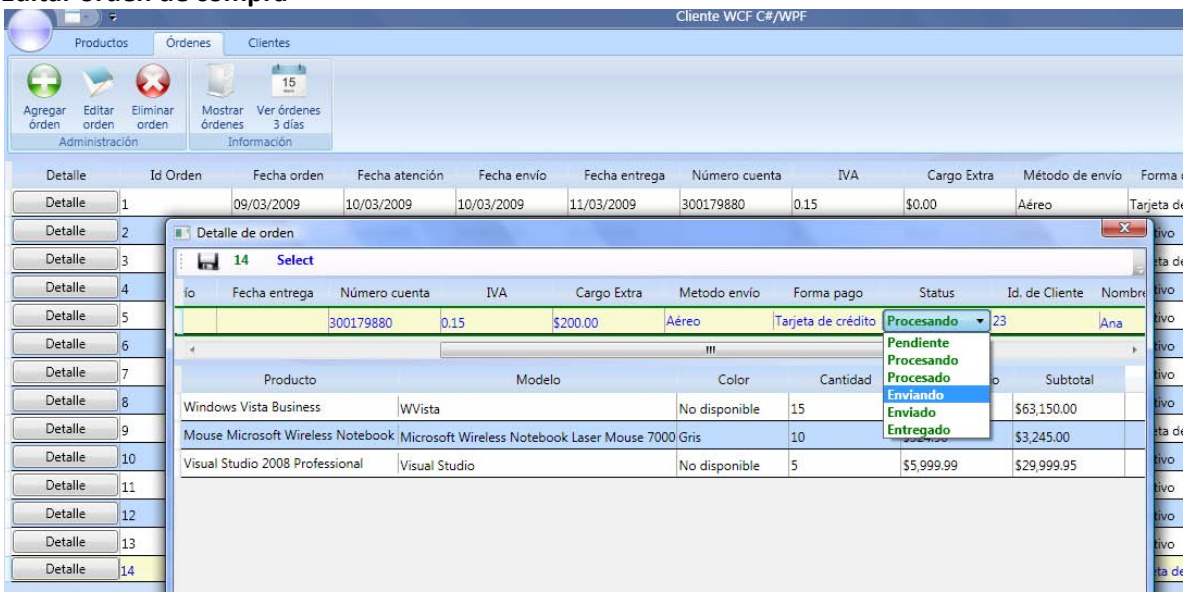
Producto	Modelo	Color	Cantidad	Precio unitario	Subtotal
Windows Vista Ultimate	WVista	No disponible	20	$4,799.00	$95,980.00
Sql Server 2008	Sql Server	No disponible	30	$7,298.16	$218,944.80
Microsoft Visual Studio 2008 Standar	Visual Studio	No disponible	10	$4,330.00	$43,300.00
Intel Pentium Dual Core	Intel Pentium	No disponible	15	$2,012.54	$30,188.10
Teléfono Celular Motorola ROKR Z6	Motorola ROKR Z6	Negro	50	$4,892.00	$244,600.00

Mostrar orden de compra particular y general. Si el usuario presiona el botón ***Detalle***, se mostrará una ventana que visualizará el detalle de la orden de compra en la cual se dio clic al botón. A través de esta ventana se pueden manipular todas las órdenes de compra. En la imagen anterior se pueden ver los productos solicitados a través del cliente WSIT de Java de esta misma tesis en el siguiente apartado de este capítulo.

Insertar orden de compra

Inserción de orden de compra: 1) Se presiona el botón ***Insertar orden***, 2) Se agregan los datos de la orden de compra (tanto los generales como los particulares), 3) Se presiona el botón ***Guardar.***

Editar orden de compra



Edición de orden de compra: 1) Se presiona el botón ***Editar orden***, 2) Se presiona el botón Detalle de la orden de compra a editar, una vez que se hicieron las modificaciones necesarias en la ventana, 3) Se presiona el botón ***Guardar***. En el ejemplo anterior se **modificó el status de la orden de compra *14*** que se crea en el ejemplo del ***cliente ASP.NET*** de esta tesis (en el último apartado de este quinto capítulo); se cambia el status de la orden de "Procesado" a "Enviando". En el ***cliente ASP.NET*** se debe ver reflejada esta modificación al hacer un seguimiento de esta orden de compra. [Ver imagen correspondiente en el despliegue de ese cliente en esta tesis].

Eliminar orden de compra

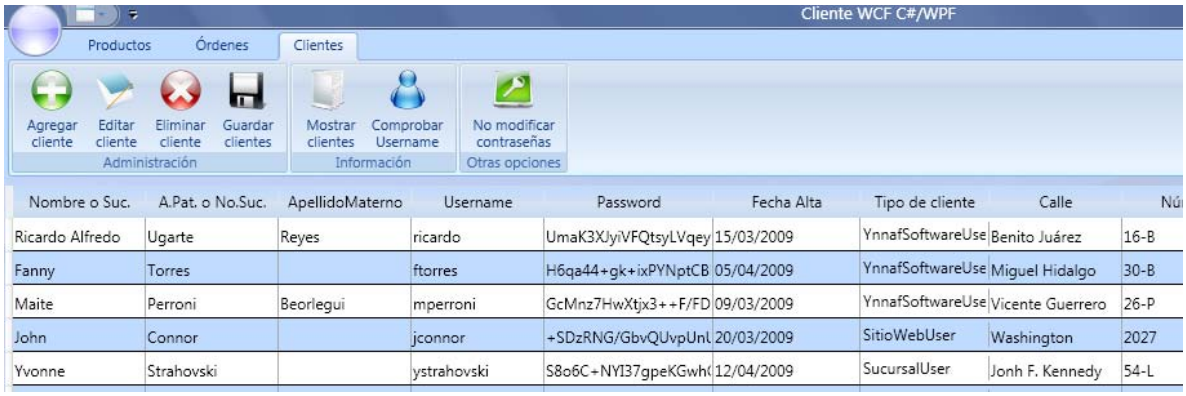
Eliminación de orden de compra: 1) Se presiona el botón ***Eliminar orden***, 2) Se presiona el botón Detalle de la orden de compra a eliminar, 3) Se presiona el botón ***Guardar***. En el ejemplo anterior se está por eliminar la orden de compra con el id de orden 4.

MANIPULACIÓN DE CLIENTES

Tabulación Clientes

Tabulación Clientes. Donde: El botón ***Agregar cliente*** habilita al usuario a agregar un nuevo cliente a la base de datos (RAM), el botón ***Editar cliente*** habilita al usuario a modificar un cliente existente en la base de datos (RAM), el botón ***Eliminar orden*** habilita al usuario a eliminar un cliente existente en la base de datos (RAM), el botón ***Guardar cliente*** permite al usuario guardar las modificaciones hechas por cualquiera de los tres botones anteriores de forma permanente, el botón ***Mostrar clientes*** permite al usuario ver todos los clientes del sistema, el botón ***ComprobarUsername*** permite al usuario si el nombre de usuario ya existe en la base de datos y el botón ***No modificar contraseña*** indica si la contraseña del o los clientes a modificar también debe ser modificada juntos con sus datos.

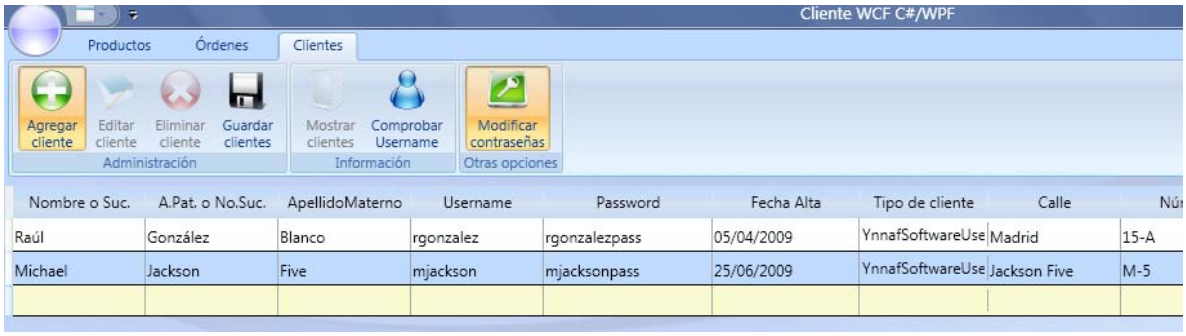
Mostrar clientes



Nombre o Suc.	A.Pat. o No.Suc.	ApellidoMaterno	Username	Password	Fecha Alta	Tipo de cliente	Calle	Nú
Ricardo Alfredo	Ugarte	Reyes	ricardo	UmaK3XJyiVFQtsyLVqey	15/03/2009	YnnafSoftwareUse	Benito Juárez	16-B
Fanny	Torres		ftorres	H6qa44+gk+ixPYNptCB	05/04/2009	YnnafSoftwareUse	Miguel Hidalgo	30-B
Maite	Perroni	Beorlegui	mperroni	GcMnz7HwXtjx3++F/FD	09/03/2009	YnnafSoftwareUse	Vicente Guerrero	26-P
John	Connor		jconnor	+SDzRNG/GbvQUvpUnl	20/03/2009	SitioWebUser	Washington	2027
Yvonne	Strahovski		ystrahovski	S8o6C+NYI37gpeKGwh(	12/04/2009	SucursalUser	Jonh F. Kennedy	54-L

Mostrar clientes. Si el usuario presiona el botón ***Mostrar Clientes***, el programa visualizará todos los productos contenidos en la base de datos del sistema. A través de esta ventana se pueden manipular todos los clientes.

Insertar clientes



Nombre o Suc.	A.Pat. o No.Suc.	ApellidoMaterno	Username	Password	Fecha Alta	Tipo de cliente	Calle	Nú
Raúl	González	Blanco	rgonzalez	rgonzalezpass	05/04/2009	YnnafSoftwareUse	Madrid	15-A
Michael	Jackson	Five	mjackson	mjacksonpass	25/06/2009	YnnafSoftwareUse	Jackson Five	M-5

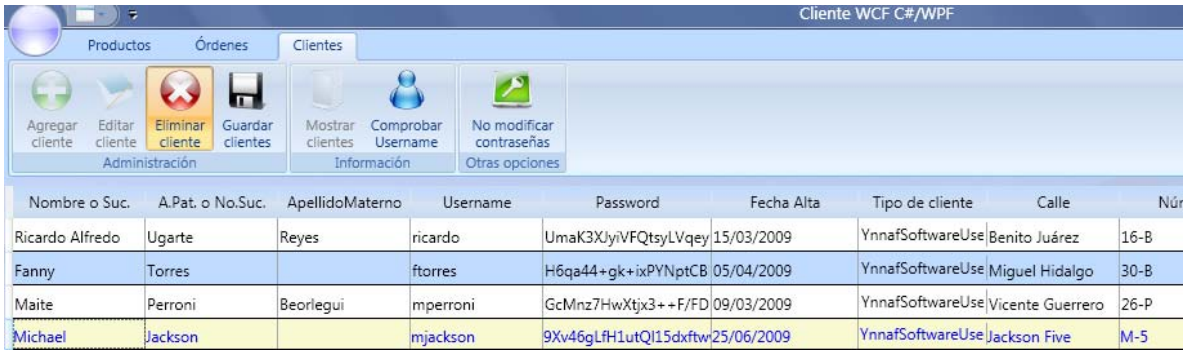
Inserción de clientes: 1) Se presiona el botón ***Insertar cliente***, 2) Se agregan cuantos clientes sean necesarios, una vez que se insertaron los clientes deseados, 3) Se presiona el botón ***Guardar cliente.*** En el ejemplo anterior se insertaron los clientes: "Raúl González con el nombre de usuario rgonzalez" y "Michael Jackson con el nombre de usuario mjackson"

Editar clientes

Nombre o Suc.	A.Pat. o No.Suc.	ApellidoMaterno	Username	Password	Fecha Alta	Tipo de cliente	Calle	Nú
Ricardo Alfredo	Ugarte	Reyes	ricardo	UmaK3XJyiVFQtsyLVqey	15/03/2009	YnnafSoftwareUse	Benito Juárez	16-B
Fanny	Torres		ftorres	H6qa44+gk+ixPYNptCB	05/04/2009	YnnafSoftwareUse	Miguel Hidalgo	30-B
Maite	Perroni	Beorlegui	mperroni	GcMnz7HwXtjx3++F/FD	09/03/2009	YnnafSoftwareUse	Vicente Guerrero	26-P
Raúl	González	Blanco	rgonzalez	IUR5m2fbPiJy95tekmeP\	05/04/2009	YnnafSoftwareUse	Madrid	15-A
Michael	Jackson		mjackson	9Xv46gLfH1utQl15dxftw	25/06/2009	YnnafSoftwareUse	Jackson Five	M-5

Edición de clientes: 1) Se presiona el botón ***Editar cliente***, 2) Cualquiera de los clientes que se visualicen al usuario puede ser modificado, una vez que se hicieron las modificaciones necesarias, 3) Se presiona el botón ***Guardar clientes***. En el ejemplo anterior se modificó el apellido materno del nombre de usuario del usuario *mjackson* de "Five" a ""

Eliminar clientes



Eliminación de clientes: 1) Se presiona el botón ***Eliminar cliente***, 2) Cualquiera de los clientes que se visualicen al usuario puede ser eliminado, seleccionándolo y presionando la tecla *Suprimir*, una vez que se eliminaron los clientes, 3) Se presiona el botón ***Guardar clientes***. En el ejemplo anterior se eliminó el cliente "rgonzalez"

5.2 CREACIÓN DE UN CLIENTE WSIT EN JAVA. SUCURSAL FUSIÓN. (APLICACIÓN DE ESCRITORIO)

A continuación se muestra como un servicio *Windows Communication Foundation (WCF)* obtiene una completa interoperabilidad con un cliente *Web Services Interoperability Technologies (WSIT).* Ambos usando características diferentes a la especificación *Web Services Interoperability (WS-I) Basic Profile.* El primero construido bajo *.NET Framework* el segundo bajo *Java,* el cliente *WSIT* es una aplicación de escritorio.

5.2.1 REQUERIMIENTOS

Se requiere construir un cliente en *Java* para acceder al servicio *WCF* que expone la lógica del negocio de la empresa *YnnafSoftware*. Esta aplicación debe cumplir las siguientes condiciones:

1. Establecer comunicación con el servicio *WCF* utilizando el certificado *X.509* provisto por la empresa para acceder a la lógica del negocio.
2. Crear órdenes de acuerdo a las necesidades de la sucursal fusión.
3. Hacer seguimiento a cualquiera de las órdenes creadas previamente por la sucursal. Los seguimientos deben ser independientes entre empleados.

5.2.2 ANÁLISIS

Cuando se va a poner en marcha la construcción del cliente *Java* para consumir el servicio *WCF*, se debe tener en cuenta que *Java* no permite la completa interoperabilidad con servicios *WCF* que implementan características complejas, como autenticación usando certificados *X.509* usando enlaces *WsHttpBinding*. Esto se puede ver claramente en el proceso de importación del *WSDL* del servicio *WCF* utilizando *wsimport.exe,* esta aplicación muestra el siguiente mensaje de advertencia: *[WARNING] Ignoring SOAP port "WsHttpBinding_YnnafSoft-wareService": it uses non-standard SOAP 1.2 binding.* Si no se soluciona esta advertencia, la aplicación cliente en tiempo de ejecución puede realizar una petición al servicio *WCF* pero la respuesta nunca es retornada.

Después de analizar el problema anterior, se llega a la conclusión que *Java* de forma nativa, únicamente permite la interoperabilidad con un servicio *web* a través del *Web Services Interoperability (WS-I) Basic Profile.* Es decir, la única forma en que el cliente *Java* podría interoperar con el ser-

vicio *WCF* sería que, el servicio utilizase un enlace sin seguridad como *BasicHttpBinding* en lugar de un enlace *WsHttpBinding*.

Dado que la empresa no está dispuesta a cambiar su enlace (*WsHttpBinding*) porque lo más importante para ella es proporcionar seguridad a todos sus clientes, se debe buscar una solución a este problema de interoperabilidad entre *.NET Framework* y *Java*. Después de una investigación sobre qué tecnología solventaría este inconveniente, se encontró que utilizar *Web Services Interoperability Technologies (WSIT) es la solución.*

5.2.2.1 WEB SERVICES INTEROPERABILITY TECHNOLOGIES (WSIT)

Web Services Interoperability Technologies (WSIT) es un componente de software obtenido como resultado del trabajo conjunto realizado entre *Sun Microsystems* y *Microsoft* para lograr la completa interoperabilidad entre los servicios web actuales. *WSIT* se ha probado extensivamente con *Windows Communication Foundation (WCF)* y proporciona una manera confiable para que las aplicaciones basadas en la tecnología *Java* se integren e interoperen con *.NET Framework* (más allá de *Web Services Interoperability (WS-I) Basic Profile*). *WSIT* Incluye las implementaciones de:

- ✓ WS-Trust
- ✓ WS-SecureConversation
- ✓ WS-SecurityPolicy
- ✓ WS-ReliableMessaging
- ✓ WS-AtomicTransactions/Coordination
- ✓ WS-MetadataExchange
- ✓ WS-Policy
- ✓ WS-Security
- ✓ SOAP sobre TCP

La tecnología *WSIT* es parte de *Metro*. *Metro* es un componente que permite manejar servicios web con un alto rendimiento, extensibilidad y facilidad. Contiene todo lo que un servicio web necesite. Se puede crear desde el más simple servicio web como el clásico “Hola mundo” hasta interoperar con servicios web que permiten confiabilidad, seguridad, y transacciones, es decir, interoperar con servicios *WCF* de *.NET Framework 3.0* y *3.5*. Las versiones *1.0 ~ 1.2* de *Metro* implementan las mismas especificaciones que *WCF* para *.NET Framework 3.0*, a partir de la versión *1.3* de *Metro* también se implementan las mismas especificaciones de *WCF* para *.NET Framework 3.5*.

5.2.3 DISEÑO

A continuación se muestran los casos de uso detallados que se deben seguir para los diferentes procedimientos que se utilizarán a lo largo del programa

5.2.3.1 CASOS DE USO DETALLADOS

Caso de uso:	**Inicio de sesión con el servicio WCF.**
Tipo:	Básico
Actor principal:	Cliente *WSIT* escrito en *Java 6.0* utilizando *Swing*
Descripción:	Se describe como se obtiene instancia con el servicio *WCF*.
Pre-condiciones:	El cliente WSIT debe poseer los certificados *X.509* adecuados para el servicio *WCF*
Post-condiciones:	El cliente WSIT obtiene instancia y sesión con el servicio *WCF*.
Versión:	0.0.1
Fecha de creación	13-02-2009
Fecha de actualización:	
FLUJOS	
Flujo básico:	Inicio de sesión con nombre de usuario y contraseña válidos.
ACTOR	**SISTEMA**

Paso	Acciones	Paso	Acciones	Excepción
1	El usuario decide ejecutar cualquiera de las dos tareas para las que sirve el programa. Se debe iniciar una sesión con el servicio *WCF* llamando al contrato de operación *IniciarSesion (username, password)*	2	Se retorna un valor booleano verdadero si el nombre de usuario y contraseña se encuentran en la base de datos. Como consecuencia se crea una instancia y sesión del servicio *WCF*.	
Flujo alternativo:	Inicio de sesión con nombre de usuario y contraseña no válidos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide ejecutar cualquiera de las dos tareas para las que sirve el programa. Se debe iniciar una sesión con el servicio *WCF* llamando al contrato de operación *IniciarSesion (username, password)*	2	Se retorna una excepción al cliente indicando que no se encontró el nombre de usuario o contraseña. Esta excepción debe ser interceptada por el cliente *WSIT*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Seleccionar una lista con todos los productos.**
Tipo:	Básico
Actor principal:	Cliente *WSIT* escrito en *Java 6.0* utilizando *Swing*
Descripción:	Se describe como se obtiene una lista de todos los productos desde el servicio *WCF*.
Pre-condiciones:	Se debe tener un nivel de acceso que permita obtener datos desde la base de datos.
Post-condiciones:	El cliente obtiene los datos que solicitó.
Versión:	0.0.1
Fecha de creación	13-02-2009
Fecha de actualización:	

FLUJOS				
Flujo básico:	Selección de datos con nivel de acceso correcto			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide iniciar una orden de compara, para ello necesita obtener una lista de todos los productos llamando al contrato de operación adecuado.	2	Dado que hay una instancia y sesión creada en el servicio *WCF* si se tiene el nivel de acceso necesario para seleccionar datos, se retornan los datos al cliente *WCF*.	
Flujo alternativo:	Selección de datos con nivel de acceso no válido.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide iniciar una orden de compara, para ello necesita obtener una lista de todos los productos llamando al contrato de operación adecuado.	2	Si no hay una instancia y sesión creada en el servicio *WCF* o no se tiene el nivel de acceso necesario, se retorna al cliente una excepción de referencia nula de objeto Esta excepción debe ser interceptada por el cliente *WSIT*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Enviar orden de compra.**
Tipo:	Básico
Actor principal:	Cliente *WSIT* escrito en *Java 6.0* utilizando *Swing*
Descripción:	Se describe como se envía una nueva orden al servicio *WCF*
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*
Post-condiciones:	El cliente obtiene el número de la orden creada exitosamente.
Versión:	0.0.1
Fecha de creación	13-02-2009
Fecha de actualización:	

FLUJOS				
Flujo básico:	Envío de la orden con cantidad de productos mayor a cero			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide finalizar su orden enviando los datos de los productos que necesita la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que la cantidad de productos sea mayor a cero y que se tenga en existencia los productos a comprar. Si esto se cumple, se retorna el número de orden creada exitosamente.	
Flujo alternativo:	Envío de la orden con cantidad de productos igual a cero			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide finalizar su orden enviando los datos de los productos que necesita la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que la cantidad de productos sea mayor a cero. Si esto no se cumple, se retorna una excepción indicando que la cantidad de productos a comprar debe ser mayor a cero. Esta excepción debe ser interceptada por el cliente *WSIT*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Seguimiento de orden de compra hecha con anterioridad.**			
Tipo:	Básico			
Actor principal:	Cliente *WSIT* escrito en *Java 6.0* utilizando *Swing*			
Descripción:	Se describe como se hace un seguimiento a las órdenes de compra hechas con anterioridad.			
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*			
Post-condiciones:	El cliente obtiene información del estado de su orden de compra.			
Versión:	0.0.1			
Fecha de creación	13-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Seguimiento de orden de compra existente			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide hacer un seguimiento de una orden de compra específica realizada con anterioridad por la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que exista el número de la orden de compra, al existir, se retorna al llamador todos los datos existentes relacionados con esa orden de compra.	
Flujo alternativo:	Seguimiento de orden de compra inexistente			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide hacer un seguimiento de una orden de compra específica realizada con anterioridad por la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que exista el número de la orden de compra, al no existir, se retorna al llamador una *FaultException* (*OrdenFault*). Esta excepción debe ser interceptada por el cliente *WSIT*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

5.2.4 IMPLEMENTACIÓN

A continuación se muestra la implementación del programa en *Java 6.0/Swing*:

5.2.4.1 INSTALACIÓN DE METRO (WEB SERVICES INTEROPERABILITY TECHNOLOGIES) Y SU INCORPORACIÓN CON NETBEANS 5.5.1.

Una vez que se sabe cuál es la solución para lograr la interoperabilidad entre *Java* y *.NET Framework*, se procede a instalar los componentes necesarios para su ejecución como se explica a continuación:

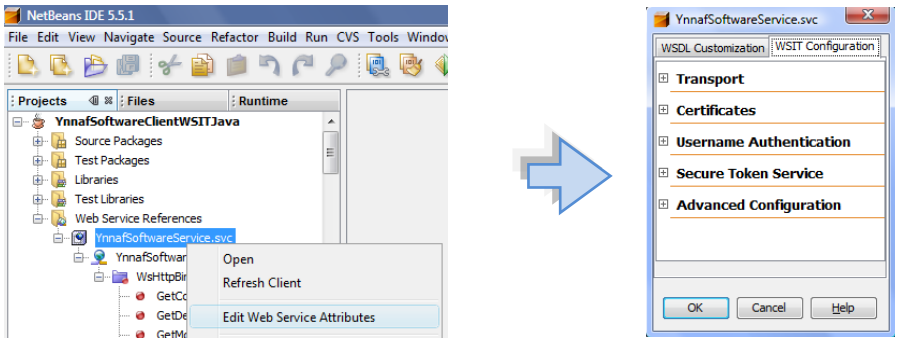
5.2.4.1.1 INSTALACIÓN DE METRO (WSIT)

Lo primero es bajar la versión más reciente y estable de *Metro* (en esta tesis se utilizará la versión *1.4*) desde la siguiente dirección de Internet: *https://metro.dev.java.net/1.4/metro-1_4.jar*. Una vez descargado, debemos instalar el archivo *JAR* con el siguiente comando: *java -jar metro-1_4.jar*. Esto instalará (descomprimirá) todos los archivos de *Metro* en el mismo directorio donde se encuentra el archivo *JAR* descargado previamente. Los archivos (*JAR*) que nos interesan para poder efectuar la interoperabilidad entre *.NET Framework* (*WCF*) y *Java* (*WSIT*) serán: *.\metro\lib\webservices-rt.jar* y *.\metro\lib\webservices-tools.jar* sobre *Windows; ./metro/lib/webservices-rt.jar* y *./metro/lib/webservices-tools.jar sobre Linux.*

5.2.4.1.2 INSTALACIÓN DE LOS MÓDULOS DE WSIT EN EL IDE NETBEANS.

1. Descargar las versiones estables de los 2 módulos *WSIT NetBeans* para *WSIT Runtime Milestone 5* desde la dirección: *http://websvc.netbeans.org/servlets/ProjectDocumentList?folderID=184&ex-pandFolder=184&folderID=191.*
   1.1 El módulo *WSIT Support* (*org-netbeans-modules-websvc-wsitconf.nbm*) y el módulo *WSIT WSDL extensions* (*org-netbeans-modules-websvc-wsitmodelext.nbm*), ambos permiten trabajar con *WSIT M5* en *Glassfish* y *Netbeans 5.5.1* o posterior.
2. Abrir el *IDE NetBeans*.
   2.1 Ir a *Tools -> Update Center*, seleccionar *Install Manually Downloaded Modules (.nbm files)* y dar clic en el botón *Next.*
   2.2 En la siguiente ventana dar clic en el botón *Add*. Buscar los archivos *.nbm*, seleccionarlos y dar clic en el botón *OK*.
   2.3 Una vez que se han seleccionado para su agregado, dar clic en el botón *Next*.
   2.4 Ir a través de las siguientes ventanas de instalación asegurándose de seleccionar estos dos módulos para su instalación.
   2.5 Una vez finalizada la integración de los módulos, reiniciar *NetBeans.*

   Nota: Otra alternativa es buscar los módulos en *Tool -> Update Center* de *NetBeans*, seleccionar *Check the Web for Available Updates and New Modules*, buscar los 2 módulos mencionados anteriormente, seleccionarlos e instalarlos.



Integración de Web Services Interoperability Technologies (WSIT) con NetBeans 5.5.1

5.2.4.2 CONVERSIÓN DEL CERTIFICADO X.509 (PFX - JKS) PARA EL USO DEL SERVICIO WCF

Un archivo *PFX (**P**ersonal In**f**ormation E**x**change)* es un certificado *PKCS #12* (certificado *X.509*) que puede contener tanto la clave pública como la privada. Estos archivos son utilizados tanto por el servicio *WCF* como por los clientes creados en *.NET Framework* sin cambio alguno. Para usar estos certificados *X.509* en un programa *Java*, se necesita convertirlos a archivos *Java Keystore (JKS)*. Para hacer esto de una manera fácil, se utilizará una aplicación que viene junto con *Jetty* (Servidor *HTTP* gratuito escrito en *Java*). Para este cliente *WSIT* (a diferencia de los cliente *WCF*) se necesitan los dos archivos que contienen los certificados *X.509* (*YnnafSoftwareClient.pfx* y *YnnafSoftwareService.pfx*). Entonces, los pasos a seguir para convertir los archivos *.PFX* a archivos *.JKS* son los siguientes:

1. Obtener tanto el certificado del servicio como el del cliente provistos por la empresa y colocarlos en *C:\YnnafSoftwareClient.pfx y C:\YnnafSoftwareService.pfx*
2. Descargar la versión más estable del servidor desde: *http://www.mortbay.org*. Para la creación de los *certificados X.509* se utilizó la versión 6.1.15 del servidor (*http://dist.codehaus.org/jetty/jetty-6.1.15/jetty-6.1.15.rc3.zip*)
3. Extraer el archivo *jetty-6.1.15.rc3.zip* en la unidad raíz del sistema operativos (*C:* en *Windows*).
4. Abrir una terminal (*cmd.exe*) y cambiar de directorio donde se extrajo el archivo del paso anterior (El directorio es: *C:\jetty-6.1.15.rc3*)
5. Insertar el siguiente comando de Java: *java -classpath lib/jetty-6.1.15.rc3.jar org.mortbay.jetty.se-curity.PKCS12Import C:\YnnafSoftwareClient.pfx C:\YnnafSoftwareClient.jks*
6. Insertar las contraseña (*YnnafSoftware*) cuando aparezcan los mensajes: *Enter input keystore passphrase:* y *Enter output keystore passphrase:*
7. De acuerdo a la información proporcionada después de realizar el proceso satisfactoriamente, tener a cuenta el nombre del alias asignado (*547f680e-7c4a-4d44-93a2-4a7c6698dd70*). La información proporcionada es: *Alias 0: 547f680e-7c4a-4d44-93a2-4a7c6698dd70* y *Adding key for alias 547f680e-7c4a-4d44-93a2-4a7c6698dd70*
8. Insertar el siguiente comando de Java: *java -classpath lib/jetty-6.1.15.rc3.jar org.mortbay.jetty.se-curity.PKCS12Import C:\YnnafSoftwareService.pfx C:\YnnafSoftwareService.jks*
9. Insertar la contraseña (*YnnafSoftware*) cuando aparezcan los mensajes: *Enter input keystore passphrase:* y *Enter output keystore passphrase:*
10. De acuerdo a la información proporcionada después de realizar el proceso satisfactoriamente, tener a cuenta el nombre del alias asignado (*2c555cac-a323-4f51-93d4-*

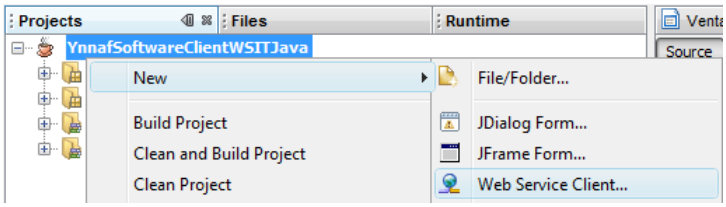
a9bf35a8cabd). La información proporcionada es: *Alias 0: 2c555cac-a323-4f51-93d4-a9bf35a8cabd* y *Adding key for alias 2c555cac-a323-4f51-93d4-a9bf35a8cabd*

Para conocer información acerca de los *certificados X.509* convertidos recientemente a archivos *.JSK,* abrir una terminal y cambiarse al directorio *C:\Archivos de programa\Java\jre1.6.0_04\bin* e insertar las instrucciones Java siguiente: *keytool -list -keystore C:\YnnafSoftwareClient.jks -v* y *keytool -list -keystore C:\YnnafSoftwareService.jks -v*.

5.2.4.3 IMPORTACIÓN DEL WSDL DEL SERVICIO WCF

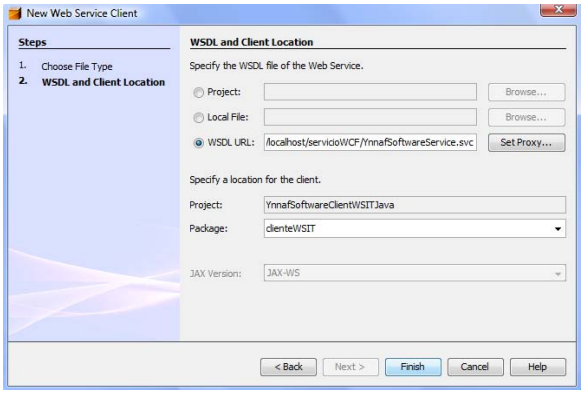
Lo que se debe hacer para importar el código *WSDL* del servicio *WCF* es crear un cliente a partir del servicio. Esto lo haremos apoyándonos del *IDE NetBeans 5.5.1* siguiendo los siguientes pasos:

1. Dar clic derecho sobre el proyecto de la aplicación *Java* y seleccionar *New -> Web Service Client…*



Creación de cliente de servicio web

2. Introducir en la ventana *New Web Service Client,* la *URL* del servicio *WCF* y el nombre del paquete que contendrá al cliente *WSIT* del servicio *WCF* y dar clic en el botón *Finish*



Ventana New Web Service Client

Esto provocará que el importador de código *WSDL* de *Java* (*wsimport.exe*) empiece a trabajar para generar el cliente *WSIT*. Lo que podemos ver en el *IDE* (*Retriever Output*) es algo como lo siguiente:

```
14/02/2009 06:38:08 PM : Retrieving Location:
http://localhost/servicioWCF/YnnafSoftwareService.svc?WSDL
    Retrieved :   http://localhost/servicioWCF/YnnafSoftwareService.svc?WSDL
    Saved at: F:\Tesis WCF por Fanny\YnnafSoftwareClientWSITJava\xml-resources\web-service-
      references\YnnafSoftwareService.svc\wsdl\localhost\servicioWCF\
      YnnafSoftwareService.svc.wsdl
    Retrieving Location: http://piezadesol/servicioWCF/YnnafSoftwareService.svc?wsdl=wsdl1
    Found in document: http://localhost/servicioWCF/YnnafSoftwareService.svc?WSDL
    Retrieved :   http://piezadesol/servicioWCF/YnnafSoftwareService.svc?wsdl=wsdl1
```

```
      Saved at: F:\Tesis WCF por Fanny\YnnafSoftwareClientWSITJava\xml-resources\web-service-
        references\YnnafSoftwareService.svc\wsdl\piezadesol\servicioWCF\
        YnnafSoftwareService.svc.wsdl_wsdl1.wsdl
       Retrieving Location: http://piezadesol/servicioWCF/YnnafSoftwareService.svc?wsdl=wsdl0
      Found in document: http://piezadesol/servicioWCF/YnnafSoftwareService.svc?wsdl=wsdl1
      Retrieved :    http://piezadesol/servicioWCF/YnnafSoftwareService.svc?wsdl=wsdl0
      Saved at: F:\Tesis WCF por Fanny\YnnafSoftwareClientWSITJava\xml-resources\web-service-
        references\YnnafSoftwareService.svc\wsdl\piezadesol\servicioWCF\
        YnnafSoftwareService.svc.wsdl_wsdl0.wsdl
//Más WSDL obtenido
```

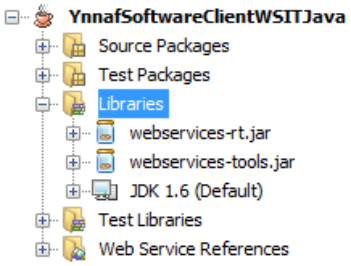
Obtención del WSDL del servicio WCF

Una vez que se terminó de obtener el *WSDL, la aplicación* procede de forma automática a generar las clases del cliente y a compilarlas. Lo que podemos ver en el *IDE* (*YnnafSoftwareClientWSITJava-Impl (wsimport-client-compile)*) es algo como lo siguiente:

```
init:
wsimport-init:
wsimport-client-check-YnnafSoftwareService.svc:
wsimport-client-YnnafSoftwareService.svc:
Consider using <depends>/<produces> so that wsimport won't do unnecessary compilation
parsing WSDL...

[WARNING] Ignoring SOAP port "WsHttpBinding_YnnafSoftwareService": it uses non-standard
SOAP 1.2 binding.
  line 1 of file:/F:/Tesis%20WCF%20por%20Fanny/YnnafSoftwareClientWSITJava/xml-resources/
    web-service-references/YnnafSoftwareService.svc/wsdl/localhost_8080/servicioWCF/
    YnnafSoftwareService.svc.wsdl

generating code...
compiling code...

BUILD SUCCESSFUL (total time: 21 seconds)
```

Generación de cliente WSIT

3. Habilitar *WSIT* en el proyecto.
   Para permitir la interoperabilidad entre servicios *WCF* seguros de *.NET Framework* con clientes *WSIT* de *Java*, necesitamos referenciar en el proyecto 2 archivos *.JAR* pertenecientes a *Metro* y que son los correspondientes a la implementación de *Web Services Interoperability Technologies (WSIT)*. Estos archivos son:
   - ✓ .\metro\lib\webservices-rt.jar en *Windows* o ./metro/lib/webservices-rt.jar en *Linux*
   - ✓ .\metro\lib\webservices-tools.jar en *Windows* o ./metro/lib/webservices-tools.jar en *Linux*



Librerías de Metro referenciadas para usar WSIT

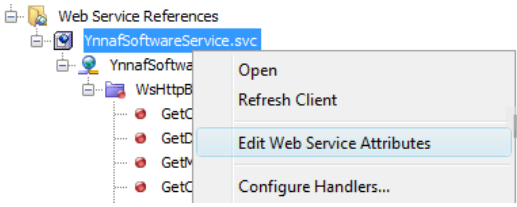
4. Usando los módulos de *WSIT* para *NetBeans* asignaremos los certificados *X.509* a través de los archivos *.JKS* generados en el tema *5.2.4 Creación del certificado X.509 provisto para el uso del servicio WCF.* Esto lo haremos de la siguiente manera:

   4.1 Dar clic derecho sobre la referencia del servicio web y seleccionar:

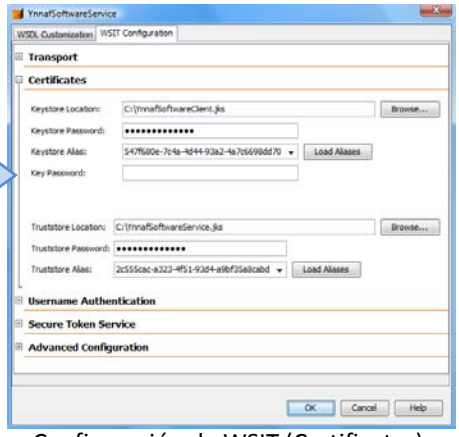
   Edit Web Service Attributes

   4.2 Seleccionar la tabulación: *WSIT Configuration -> Certificates* de la ventana: *YnnafSoftwareService* y seguir los siguientes pasos y dar clic en el botón *OK*:

   ✓ Insertar en *Keystore Location*: *C:\YnnafSoftwareClient.jks*
   ✓ Insertar en *Keystore Password*: *YnnafSoftware*
   ✓ Presionar el botón Load Aliases (se cargará el alias *547f680e-7c4a-4d44-93a2-4a7c6698dd70*)
   ✓ Insertar en *Truststore Location*: *C:\YnnafSoftwareService.jks*
   ✓ Insertar en *Truststore Password*: *YnnafSoftware*
   ✓ Presionar el botón Load Aliases (se cargará el alias *2c555cac-a323-4f51-93d4-a9bf35a8cabd*)



Edición de atributos del cliente (WSIT) del servicio web (WCF)



Configuración de WSIT (Certificates)

Esta generará un archivo *XML* donde podemos ver cómo se va a extraer información desde los archivos *JKS* que se usarán para identificar al cliente y al servicio. Deberá lucir como el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="YnnafSoftwareService"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:i0="http://www.ynnafSoftware.com">
  <!-- Otras definiciones del WSDL -->
  <wsp:Policy wsu:Id="WsHttpBinding_YnnafSoftwareServicePolicy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsa10:UsingAddressing/>
        <sc:KeyStore wspp:visibility="private" storepass="YnnafSoftware"
                     type="JKS" location="C:\YnnafSoftwareClient.jks"
                     alias="547f680e-7c4a-4d44-93a2-4a7c6698dd70"/>
        <sc:TrustStore wspp:visibility="private" storepass="YnnafSoftware"
                       type="JKS" location="C:\YnnafSoftwareService.jks"
                       peeralias="2c555cac-a323-4f51-93d4-a9bf35a8cabd"/>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
</wsdl:definitions>
```

Código generado por *WSIT Configuration* (en el archivo *YnnafSoftwareService.xml*)

Este archivo a su vez, es importado junto con por el archivo *XML* generado por *wsimport.exe*. Para poder entonces generar la importación correctamente (que permita usar *WSIT*) habremos de actualizar el cliente (*Refresh Client*). Una vez finalizada la generación del código del servicio *WCF*, podemos ver que los códigos tanto en el servicio como en el cliente son similares, el modo de uso depende del lenguaje de programación:

DEFINICIÓN DEL CONTRATO DE OPERACIÓN SP_CREATEORDEN EN EL SERVICIO WCF

```
[OperationContract(Name = "Sp_CreateOrden", IsInitiating = false, IsTerminating = false)]
[FaultContract(typeof(OrdenFault))]
int Sp_CreateOrden(Orden orden, String listaProductos, ref int ordenId,
ref String DescripcionResultado);
```

EL MISMO CONTRATO DE OPERACIÓN EN EL CLIENTE WSIT CONSTRUIDO EN JAVA 6.0 TRAS LA IMPORTACIÓN (ESTE CÓDIGO ES GENERADO AUTOMÁTICAMENTE)

```
@WebMethod(operationName = "Sp_CreateOrden",
    action = "http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrden")
@RequestWrapper(localName = "Sp_CreateOrden", targetNamespace =
    "http://www.ynnafSoftware.com", className = "clienteWSIT.SpCreateOrden")
@ResponseWrapper(localName = "Sp_CreateOrdenResponse", targetNamespace =
    "http://www.ynnafSoftware.com", className = "clienteWSIT.SpCreateOrdenResponse")
@Action(input = "http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrden",
    output =
    "http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenResponse",
    fault = {
     @FaultAction(className =
     YnnafSoftwareServiceWCFSpCreateOrdenOrdenFaultFaultFaultMessage.class,
     value =
     "http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenOrdenFaultFault")
    })
public void spCreateOrden(
    @WebParam(name = "orden", targetNamespace = "http://www.ynnafSoftware.com")
    Orden orden,
    @WebParam(name = "listaProductos", targetNamespace = "http://www.ynnafSoftware.com")
    String listaParametros,
    @WebParam(name = "ordenId", targetNamespace = "http://www.ynnafSoftware.com",
        mode = WebParam.Mode.INOUT)
    Holder<Integer> ordenId,
    @WebParam(name = "descripcionResultado", targetNamespace =
        "http://www.ynnafSoftware.com", mode = WebParam.Mode.INOUT)
    Holder<String> descripcionResultado,
    @WebParam(name = "Sp_CreateOrdenResult", targetNamespace =
        "http://www.ynnafSoftware.com", mode = WebParam.Mode.OUT)
    Holder<Integer> spCreateOrdenResult)
    throws YnnafSoftwareServiceWCFSpCreateOrdenOrdenFaultFaultFaultMessage;
```

5.2.4.4 INSTANCIA SINGLETON PARA EL CLIENTE WSIT

Puesto que el servicio *WCF* maneja instancias que guardan valores ayudándose de sesiones, es necesario mantener desde el cliente *WSIT* una instancia única con el servicio *WCF* para obtener un correcto funcionamiento Para lograr esto, utilizaremos el patrón *Singleton*. A continuación se muestra el código fuente escrito en *Java 6.0* que llevará a cabo esta tarea.

```
/** Permite manipular una única instancia del servicio WCF
 */
public final class InstanceClienteWSIT {

    /** Mantiene la instancia única
     */
    private static volatile InstanceClienteWSIT INSTANCE;

    /** Evita la construcción de un objeto de esta clase de forma externa
     */
    private InstanceClienteWSIT() { }

    /** Intenta construir la instancia única si no existe
```

```
     */
    private static synchronized InstanceClienteWSIT crearInstancia() {
        if (INSTANCE == null) {
            INSTANCE = new InstanceClienteWSIT();
        }
        return INSTANCE;
    }

    /** Obtiene la instancia única
     */
    public static InstanceClienteWSIT getInstance() {
        InstanceClienteWSIT instancia = INSTANCE;
        if (instancia == null) {
            instancia = crearInstancia();
        }
        return instancia;
    }
    // Campo estático para almacenar los valores del servicio WCF
    private clienteWSIT.YnnafSoftwareServiceWCF servicioWCF = null;

    /** Obtiene los valores del servicio WCF
     */
    public clienteWSIT.YnnafSoftwareServiceWCF getServicioWCF() {
        return servicioWCF;
    }

    /** Establece los valores del servicio WCF
     * @param instanciaServicioWCF La instancia del servicio WCF
     */
    public void setServicioWCF(
        clienteWSIT.YnnafSoftwareServiceWCF instanciaServicioWCF) {
        servicioWCF = instanciaServicioWCF;
    }
}
```

Con este código fuente (contenido en el paquete *instanciaWSIT*) aseguramos tener una y solo una instancia del cliente *WSIT* incluso en entornos multihilos. La forma en que se llamará a esta instancia será a través de la instrucción: *InstanceClienteWSIT.getInstance().*

5.2.4.5 ENCRIPTACIÓN DE CONTRASEÑAS EN SHA-512.

Dado que el servicio *WCF* requiere que la contraseña necesaria para autorizar el acceso del usuario a la lógica del negocio esté encriptada en el algoritmo de seguridad *SHA-512* usando la codificación *UTF-8*, se debe hacer de la siguiente forma usando *Java 6.0*:

```
public static String EncriptarSha512(String cadenaSinEncriptar) {
     try{
         byte[] cadenaSinEncriptarArray = cadenaSinEncriptar.getBytes("UTF-8");
         MessageDigest sha512 = MessageDigest.getInstance("SHA-512");
         sha512.update(cadenaSinEncriptarArray);
         byte[] cadenaEncriptadaArray = sha512.digest();
         // Es necesario hacer el reemplazo de \n\r para que la cadena resultante
         // sea igual a la generada con .NET Framework.
         return new BASE64Encoder().encode(cadenaEncriptadaArray).
                 replace("\n", "").replace("\r", "");
     }catch(Exception ex){
         System.out.println(ex.getMessage());
         return null;
     }
}
```

Hay que destacar que para que este método funcione adecuadamente se deben importar las clases: *java.security.MessageDigest* y *sun.misc.BASE64Encoder*. Otra cosa importante a destacar es que, para que la cadena encriptada resultante en *Java 6.0* sea compatible con las cadenas encrip-

tadas generadas por *.NET Framework 2.0,* es necesario reemplazar en la cadena resultante de *Java* todos los caracteres "\n" y"\r" que son agregados de forma innecesaria.

NOTA: Al compilar el proyecto, aparecerá el siguiente mensaje de advertencia: *L:\Tesis WCF por Fanny\YnnafSoftwareClientWSITJava\src\encrypt\Encriptar.java:4: warning: sun.misc.BASE64Encoder is Sun proprietary API and may be removed in a future release. import sun.misc.BASE64Encoder;*. Este mensaje indica que la clase con la que encriptamos la cadena es propietaria de *Sun Microsystems* y que podría ser removida en versiones futuras de *Java*. Esto es algo a tomar en cuenta para cuestiones de compatibilidad.

5.2.4.6 LLAMADAS A CONTRATOS DE OPERACIONES DEL SERVICIO WCF

A continuación se muestra el código fuente utilizado para llamar a los contratos de operación del servicio *WCF* necesarios para la ejecución del programa. Se debe notar que no se coloca el código para inicializar la interfaz gráfica de usuario.

5.2.4.6.1 INICIO DE SESIÓN

Lo primero que necesitamos hacer para utilizar el programa es autenticar el cliente *WSIT* ante el servicio *WCF*. Para lograr esto, se crea una instancia de *YnnafSoftwareServiceWCF*, se establecen los certificados *X.509* y se asigna esta instancia a la instancia *Singleton*. Una vez hecho lo anterior se debe obtener autorización de acceso a la lógica del negocio del sistema, debemos entonces llamar al contrato de operación *IniciarSesion(username, password)* del servicio *WCF*. Si el nombre de usuario y contraseña son correctos se asigna el campo encapsulado *sesionIniciada* con el valor devuelto por el servicio (*true*), de lo contrario se caerá en el bloque *catch*. A partir de la instancia *Singleton* accederemos, en toda la ejecución del programa al servicio *WCF*. En el código fuente siguiente, podemos ver cómo se realiza el procedimiento descrito anteriormente:

```
private void AceptarJButtonActionPerformed(java.awt.event.ActionEvent evt){
     try {
         clienteWSIT.YnnafSoftwareService service = new
             clienteWSIT.YnnafSoftwareService();
         clienteWSIT.YnnafSoftwareServiceWCF port =
             service.getWsHttpBindingYnnafSoftwareService();
         InstanceClienteWSIT.getInstance().setServicioWCF(port);
         this.sesionIniciada = InstanceClienteWSIT.getInstance().
             getServicioWCF().iniciarSesion(
                 this.UsernameJTextField.getText(),
                 Encriptacion.EncriptarSha512(new
                     String(this.PasswordJPasswordField.getPassword())));
         this.dispose();
     } catch (Exception ex) {
         JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
             JOptionPane.ERROR_MESSAGE);
     }
}
```

5.2.4.6.2 OBTENCIÓN DE PRODUCTOS

La obtención y visualización de todos los productos que contiene la empresa se hace después que el cliente realizó un inicio de sesión exitoso para obtener acceso a la lógica del negocio del sistema (ver código anterior). El procedimiento a seguir para obtener todos los productos es: *1)* Llamar al contrato de operación *getProductoJoin* que obtiene todos los productos desde la base de datos y asignarlo a una lista genérica y *2)* Llenar el modelo del *JComboBox* con los Id de los productos de la lista genérica. El código fuente siguiente ejemplifica los pasos anteriores:

```
private void InicioSesionJMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
    try {
         // Obtener las dimensiones de la pantalla
         Dimension pantalla = Toolkit.getDefaultToolkit().getScreenSize();
         // Crear instancia del cuadro de diálogo de Login
         LoginJDialog loginJDialog = new LoginJDialog(this, true);
         // Obtener las dimensiones del cuadro de diálogo y centrarlo en la pantalla
         Dimension dialogo = loginJDialog.getSize();
         loginJDialog.setLocation((pantalla.width - dialogo.width) / 2,
                 (pantalla.height - dialogo.height) / 2);
         loginJDialog.setVisible(true);
         // Verificar si se inició una sesión
         if(loginJDialog.isSesionIniciada()) {
             // Obtener todos los productos desde el servicio WCF
             listaProductos = InstanceClienteWSIT.getInstance().
                 getServicioWCF().getProductos().getProductoJoin();
             // Llenar el JComboBox con los id de los productos a
             // través del modeloOrdenJTable
             for(clienteWSIT.ProductoJoin producto : listaProductos)
                 this.modeloProductoJComboBox.addElement(producto.getProductoId());
             // Hacer visibles los JLayeredPane
             mostrarJLayeredPane(true);
             // Habilitar / inhabilitar controles
             this.InicioSesionJMenuItem.setEnabled(false);
             this.CerrarSesionJMenuItem.setEnabled(true);
             this.OrdenesJMenuItem.setEnabled(true);
             loginJDialog.dispose();
         } else {
             JOptionPane.showMessageDialog(this, "Sesión no iniciada", "No iniciado",
                 JOptionPane.WARNING_MESSAGE);
         }
     }catch (Exception ex){
         JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
             JOptionPane.ERROR_MESSAGE);
     }
}
```

Nota: A partir de la selección del Id del producto en el *JComboBox*, se podrán mostrar detalles del producto en otros controles del *JFrame* como *JTextField* y *JLabel*.

5.2.4.6.3 CREACIÓN DE ORDEN

Dado que los productos de la orden de compra se agregan a un *TableModel* (*ModeloOrdenJTable*) para un *JTable*, se debe obtener el patrón {*Cantidad, ProductoId_1, Cantidad, ProductoId_2, …*} requerido por el procedimiento almacenado del servicio *WCF* a partir de él, eso se hace en la primera parte del método. Lo segundo que se debe hacer es inicializar las otras cuatro variables que se pasarán como parámetros al contrato de operación del servicio, es decir, se debe construir (e inicializar) la clase *Orden* y las clases *Holder<T> para ordenId*, *descripcionResultado* y *resultado*, estas últimas almacenarán el valor devuelto por el contrato de operación y los valores obtenidos vía los parámetros por referencia del mismo. El tercer paso es mandar a llamar al contrato de operación *spCreateOrden* del servicio *WCF* pasándole los 5 parámetros necesarios y mostrar al usuario los posibles resultados devueltos tras su ejecución (éxito o error). En el código fuente siguiente, podemos ver cómo se realiza el procedimiento descrito anteriormente:

```
private void EnviarJButtonActionPerformed(java.awt.event.ActionEvent evt) {
     try {
         if(this.modeloOrdenJTable.getRowCount() == 0) {
             JOptionPane.showMessageDialog(this, "No hay nunguna orden a enviar.",
                 "Sin orden", JOptionPane.WARNING MESSAGE);
         } else {
             // Crear el patrón { Cantidad, ProductoId } requerido por el procedimiento
             // almacenado para crear la orden
             String listaProductos = "";
             for (int i = 0; i < this.modeloOrdenJTable.getRowCount(); i++){
                 listaProductos += String.valueOf(this.modeloOrdenJTable.getValueAt(i,1));
                 listaProductos += ",";
```

```
                listaProductos += String.valueOf(this.modeloOrdenJTable.getValueAt(i,0));
                if(i != this.modeloOrdenJTable.getRowCount() - 1)
                    listaProductos += ",";
            }

            // Crear una instancia XMLGregorianCalendar para los campos que lo requieran.
            // Este valor no importa cuál sea porque el valor será insertado por el
            // servicio WCF
            XMLGregorianCalendar fechaXmlDummy = javax.xml.datatype.
                DatatypeFactory.newInstance().newXMLGregorianCalendar();
            fechaXmlDummy.setYear(1979);
            fechaXmlDummy.setMonth(04);
            fechaXmlDummy.setDay(05);

            // Crear la instancia de la clase Orden e insertar los valores necesarios.
            clienteWSIT.Orden orden = new clienteWSIT.Orden();
            orden.setCargoExtra(null);
            orden.setClienteId(0); // Un valor cualquiera
            orden.setFechaAtencion(null);
            orden.setFechaEntrega(null);
            orden.setFechaEnvio(null);
            orden.setFechaModificacion(fechaXmlDummy);
            orden.setFechaOrden(fechaXmlDummy);
            orden.setFormaPagoOrdenId(1); // Un valor cualquiera pero existente (FK)
            orden.setIva(null);
            orden.setMetodoEnvioOrdenId(1); // Un valor cualquiera pero existente (FK)
            orden.setNumeroCuenta(null);
            orden.setStatusOrdenId(0); // Un valor cualquiera

            // Valores que almacenan los tipos por referencia del servicio WCF
            javax.xml.ws.Holder<Integer> ordenId = new javax.xml.ws.Holder<Integer>();
            javax.xml.ws.Holder<String> descripcionResultado = new
                javax.xml.ws.Holder<String>();
            javax.xml.ws.Holder<Integer> resultado = new javax.xml.ws.Holder<Integer>();

            // Crear la orden llamando al procedimiento almacenado del servicio
            InstanceClienteWSIT.getInstance().getServicioWCF().spCreateOrden(
                    orden, listaProductos, ordenId, descripcionResultado, resultado);

            // Comprobar si ocurrió un error al momento de crear la orden, si es así,
            // mostrar el mensaje de error y salir del método.
            if(resultado.value == 0){
                JOptionPane.showMessageDialog(this, descripcionResultado,
                    "Error", JOptionPane.ERROR_MESSAGE);
                return; // Salir del método
            }

            // Si no ocurrió ningún error, mostrar el Id de la orden.
            JOptionPane.showMessageDialog(this, "Orden creada con el Id de orden: " +
                String.valueOf(ordenId.value), "Orden creada exitosamente",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }catch (Exception ex){
        JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE);
    }
}
```

5.2.4.6.4 SEGUIMIENTO DE ORDEN

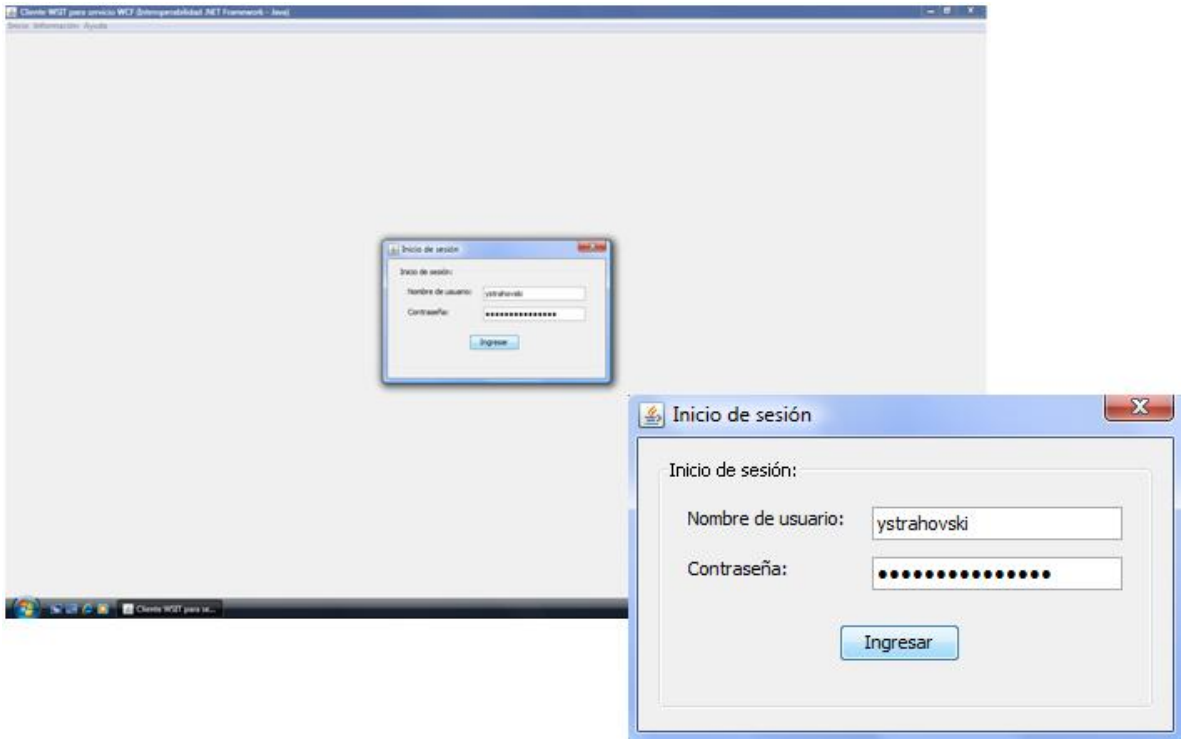
Para hacer seguimientos de las órdenes hechas por las *sucursales fusión*, se debe buscar por el número de la orden de compra. Los pasos que se deben seguir para realizar un seguimiento son los siguientes: *1)* Comprobar que haya un número de orden a buscar, *2)* Llamar al contrato de operación *spGetOrdenById* el cual obtiene la información de la orden de compra y almacenar el resultado en el contrato de datos *SpOrdenId* a través de *value* y *3)* Mostrar los datos generales de la orden en los *JTextField* y los datos particulares en el *modelo del JTable* del programa. El código fuente que realiza el procedimiento anterior es el que se muestra a continuación:

```
private void BuscarJButtonActionPerformed(java.awt.event.ActionEvent evt) {
     // Verificar que el JTextField del Id de orden no esté vacío
     if(this.OrdenIdJTextField.getText().equals("") ||
        this.OrdenIdJTextField.getText().equals(null)) {
         JOptionPane.showMessageDialog(this, "El id de orden a buscar no puede estar vacío",
             "Error", JOptionPane.INFORMATION_MESSAGE);
     } else {
         try {
             javax.xml.ws.Holder<SpOrdenId> resultado = new
                 javax.xml.ws.Holder<SpOrdenId>();

             // Obtener la orden de acuerdo al Id de la orden, la respuesta del servicio
             // WCF se almacena en la instancia 'resultado'
             resultado.value = InstanceClienteWSIT.getInstance().getServicioWCF().
                 spGetOrdenById(Integer.parseInt(this.OrdenIdJTextField.getText()));

             // Llenar los controles con los datos generales de la orden
             this.IdOrdenJTextField.setText(resultado.value.getOrden().
                 getOrdenId().toString());
             this.FechaOrdenJTextField.setText(fechas.FormatoFechas.
                 ConvertirXmlGregorianAStringFull(
                     resultado.value.getOrden().getFechaOrden()));
             this.FechaAtencionJTextField.setText(fechas.FormatoFechas.
                 ConvertirXmlGregorianAStringFull(
                     resultado.value.getOrden().getFechaAtencion()));
             this.FechaEnvioJTextField.setText(fechas.FormatoFechas.
                 ConvertirXmlGregorianAStringFull(
                     resultado.value.getOrden().getFechaEnvio()));
             this.FechaEntregaJTextField.setText(fechas.FormatoFechas.
                 ConvertirXmlGregorianAStringFull(
                     resultado.value.getOrden().getFechaEntrega()));
             this.StatusOrdenJTextField.setText(
                 resultado.value.getOrden().getStatusOrden());

             // Asignar la lista genérica con los datos del detalle de la orden
             List<clienteWSIT.OrdenDetalleJoin> listaOrdenDetalle =
                 resultado.value.getListaDetalleOrden().getOrdenDetalleJoin();

             // Instanciar y asignar el modelo al JTable
             modeloOrdenJTable = new OrdenTableModel();
             this.OrdenJTable.setModel(modeloOrdenJTable);

             // Iterar con la lista genérica y crear los objetos de la
             // clase Productos del paquete 'clasesLocales'
             for(clienteWSIT.OrdenDetalleJoin ordenDetalle : listaOrdenDetalle){
                 Productos producto = new Productos();
                 producto.setProductoId(ordenDetalle.getProductoId());
                 producto.setCantidad(ordenDetalle.getCantidad());
                 producto.setProducto(ordenDetalle.getProducto());
                 producto.setImagen(imagenes.ConversorImagenes.
                     convertirBytesAImagen(ordenDetalle.getImagen().getBytes()));
                 producto.setUnidadesDisponibles(ordenDetalle.getUnidadesDisponibles());
                 producto.setClaveProducto(ordenDetalle.getClaveProducto());
                 producto.setCategoria(ordenDetalle.getCategoria());
                 producto.setDescripcion(ordenDetalle.getDescripcion());
                 producto.setColor(ordenDetalle.getColor());
                 producto.setCostoEstandar(ordenDetalle.getCostoEstandar());
                 producto.setPrecioLista(ordenDetalle.getPrecioUnitario());
                 producto.setTamaño(ordenDetalle.getTamaño());
                 producto.setPeso(ordenDetalle.getPeso());
                 producto.setFechaInicioVenta(fechas.FormatoFechas.
                     ConvertirXmlGregorianAString(ordenDetalle.getFechaInicioVenta()));
                 producto.setFechaFinVenta(fechas.FormatoFechas.
                     ConvertirXmlGregorianAString(ordenDetalle.getFechaFinVenta()));
                 // Agregar el objeto al modelo
                 modeloOrdenJTable.agregar(producto);
             }
         }catch(clienteWSIT.YnnafSoftwareServiceWCFSpGetOrdenByIdOrdenFaultFaultFaultMessage ex){
             JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
                 JOptionPane.ERROR_MESSAGE);
         }catch(Exception ex){
```

```
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this, ex.getMessage(), "Error",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
    }
```

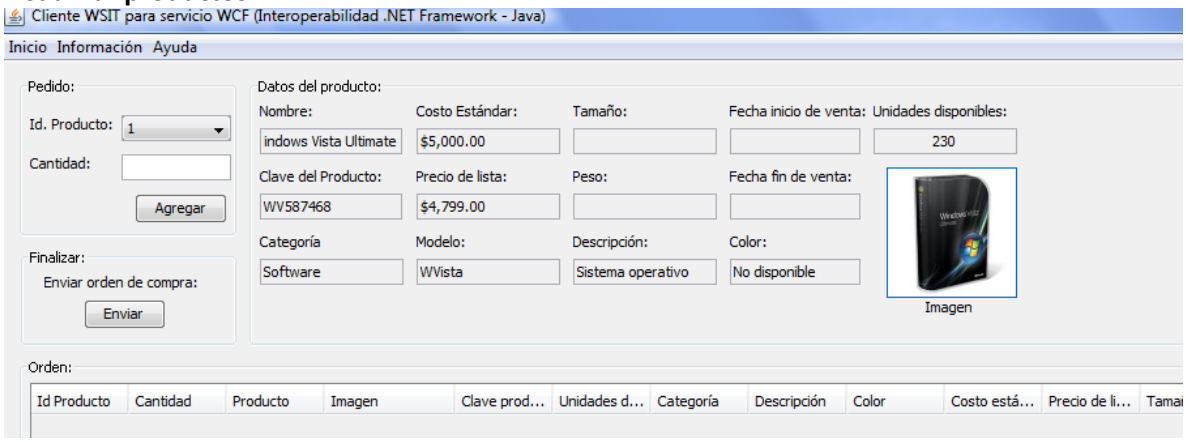
5.2.5 DESPLIEGUE

Autenticación de usuario



Inserción de nombre de usuario y contraseña. Si tanto el nombre de usuario como la contraseña son correctos, el usuario podrá utilizar las funciones del programa en su totalidad, de lo contrario el programa no podrá ser utilizado en ninguna de sus funciones.

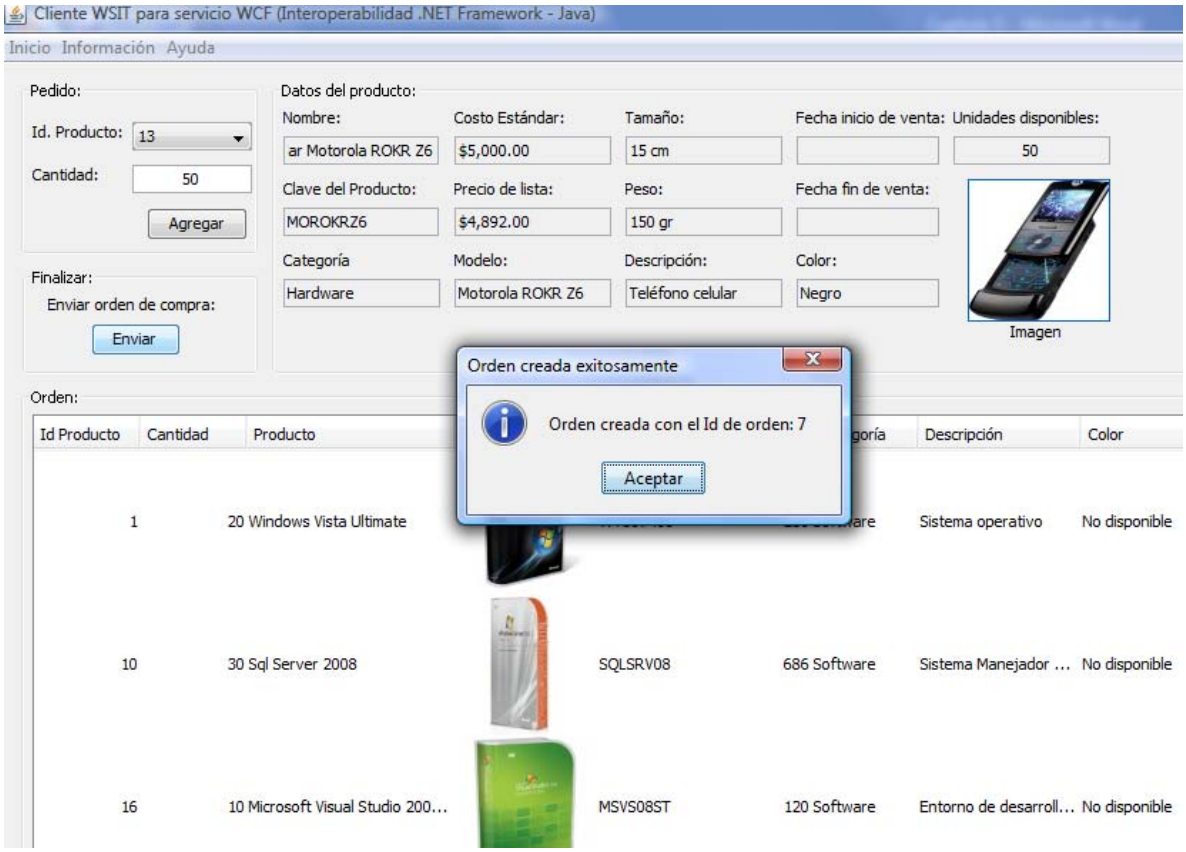
Visualizar productos



Visualización de productos. El usuario puede ver la lista de productos (ordenados por el id del producto en un ***JComboBox***) que se encuentran almacenados en la base de datos del sistema. Cada vez que el usuario selecciona un nuevo id producto, puede ver todas las características del productos seleccionado.

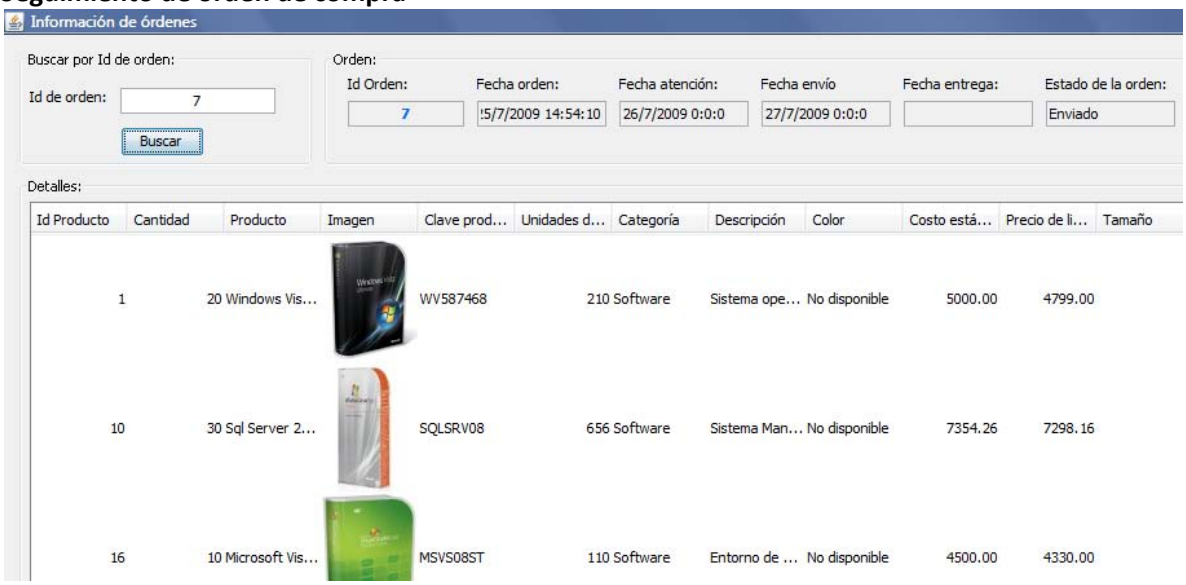
Crear orden de compra

Creación de orden de compra 1. 1) El usuario selecciona el ***producto*** que desea la sucursal e inserta la ***cantidad*** requerida y presiona el botón ***Agregar***. Los productos se van agregando en la lista inferior del programa. Si se desea modificar la cantidad de productos a comprar se deberá dar doble clic sobre la cantidad de un producto específico y modificar la cantidad previa.



Creación de orden de compra 2. 1) El usuario selecciona el ***producto*** que desea la sucursal e inserta la ***cantidad*** requerida, una vez que se agregaron todos los productos necesarios para la orden de compra, 2) Se presiona el botón ***Enviar*** para finalizar la orden de compra. Se muestra al usuario el número de orden de compra que le asignó. En el ejemplo anterior fue: 7.

Seguimiento de orden de compra



Seguimiento de orden de compra. 1) El usuario abre la ventana Seguimiento de órdenes desde el menú ***Información*** -> ***Órdenes…***, 2) Se inserta el ***id de la orden*** de la cual se desea obtener su estado, 3) Se presiona el botón ***Buscar***, Si la orden de compra existe, se muestra en la ventana la información de su orden de compra. En el ejemplo se muestra el estado de la orden número 7.

5.3 CREACIÓN DE UN CLIENTE WCF EN VB.NET SUCURSAL PROPIA (APLICACIÓN DE ESCRITORIO)

A continuación se muestra como un servicio *Windows Communication Foundation* obtiene una completa interoperabilidad con un cliente *Windows Communication Foundation.* Ambos construidos bajo *.NET Framework* pero el primero escrito en *C# 3.0* y el segundo escrito en *Visual Basic 9.0,* el cliente *WCF* es una aplicación de escritorio.

5.3.1 REQUERIMIENTOS

Se requiere construir un cliente sobre *Visual Basic* para consumir el servicio *WCF* que expone la lógica del negocio de la empresa. Esta aplicación debe cumplir las siguientes condiciones:

1. Establecer comunicación con el servicio *WCF* utilizando el certificado *X.509* provisto por la empresa para acceder a la lógica del negocio.
2. Crear órdenes de acuerdo a las necesidades de la sucursal.
3. Hacer seguimiento a cualquiera de las órdenes creadas previamente por la sucursal.

5.3.2 ANÁLISIS

Dado que el servicio *WCF* está construido sobre la plataforma *.NET Framework*, y se requiere que el cliente *WCF* esté construido en *Visual Basic*, se opta por utilizar *Visual Basic 9.0* de *.NET Framework 3.5*. La razón de utilizar esta versión de *Visual Basic* en lugar de la versión *6.0* (última versión de *Visual Basic* no enfocada a *.NET Framework* y muy usado anteriormente por muchas empresas) es la de explotar toda la potencialidad otorgada a este lenguaje a través de *.NET Framework*. La versión *6.0* de *Visual Basic* no permitiría obtener todos los beneficios que proporciona *Windows Communication Foundation*.

5.3.3 DISEÑO

A continuación se muestran los casos de uso detallados que se deben seguir para los diferentes procedimientos que se utilizarán a lo largo del programa

5.3.3.1 CASOS DE USO DETALLADOS

Caso de uso:	**Inicio de sesión con el servicio WCF.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *Visual Basic 9.0* utilizando *Windows Forms*			
Descripción:	Se describe como se obtiene instancia con el servicio *WCF*.			
Pre-condiciones:	El cliente *WCF* debe poseer los certificados *X.509* adecuados para el servicio *WCF*			
Post-condiciones:	El cliente *WCF* obtiene instancia y sesión con el servicio *WCF*.			
Versión:	0.0.1			
Fecha de creación	25-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Inicio de sesión con nombre de usuario y contraseña válidos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide ejecutar cualquiera de las dos tareas para las que sirve el programa. Se debe iniciar una sesión con el servicio WCF llamando al contrato de	2	Se retorna un valor booleano verdadero si el nombre de usuario y contraseña se encuentran en la base de datos y son correctos. Como consecuencia se crea una	

	operación *IniciarSesion (username, password)*		instancia y sesión del servicio *WCF*.	
Flujo alternativo:	Inicio de sesión con nombre de usuario y contraseña no válidos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide ejecutar cualquiera de las dos tareas para las que sirve el programa. Se debe iniciar una sesión con el servicio WCF llamando al contrato de operación *IniciarSesion (username, password)*	2	Se retorna una excepción al cliente indicando que no se encontró el nombre de usuario o contraseña. Esta excepción debe ser interceptada por el cliente *WCF* escrito en *Visual Basic 9.0.*	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Seleccionar una lista con todos los productos.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *Visual Basic 9.0* utilizando *Windows Forms*			
Descripción:	Se describe como se obtiene una lista de todos los productos desde el servicio *WCF*.			
Pre-condiciones:	Se debe tener un nivel de acceso que permita obtener datos desde la base de datos.			
Post-condiciones:	El cliente obtiene los datos que solicitó.			
Versión:	0.0.1			
Fecha de creación	25-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Selección de datos con nivel de acceso correcto			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide iniciar una orden de compara, para ello necesita obtener una lista de todos los productos llamando al contrato de operación adecuado.	2	Dado que hay una instancia y sesión creada en el servicio *WCF* si se tiene el nivel de acceso necesario para seleccionar datos, se retornan los datos al cliente *WSIT*.	
Flujo alternativo:	Selección de datos con nivel de acceso no válido.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide iniciar una orden de compara, para ello necesita obtener una lista de todos los productos llamando al contrato de operación adecuado.	2	Si no hay una instancia y sesión creada en el servicio *WCF* o no se tiene el nivel de acceso necesario, se retorna al cliente una excepción de referencia nula de objeto Esta excepción debe ser interceptada por el cliente *WCF* escrito en *Visual Basic 9.0.*	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Enviar orden de compra.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *Visual Basic 9.0* utilizando *Windows Forms*			
Descripción:	Se describe como se envía una nueva orden al servicio *WCF*			
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*			
Post-condiciones:	El cliente obtiene el número de la orden creada exitosamente.			
Versión:	0.0.1			
Fecha de creación	25-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Envío de la orden con cantidad de productos mayor a cero			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**

1	El usuario decide finalizar la orden de compra enviando los datos de los productos que necesita la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que la cantidad de productos sea mayor a cero y que se tenga en existencia los productos a comprar. Si esto se cumple, se retorna el número de orden creada exitosamente.	
Flujo alternativo:	Envío de la orden con cantidad de productos igual a cero			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide finalizar la orden de compra enviando los datos de los productos que necesita la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que la cantidad de productos sea mayor a cero. Si esto no se cumple, se retorna una excepción indicando que la cantidad de productos a comprar debe ser mayor a cero. Esta excepción debe ser interceptada por el cliente *WCF* escrito en *Visual Basic 9.0*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

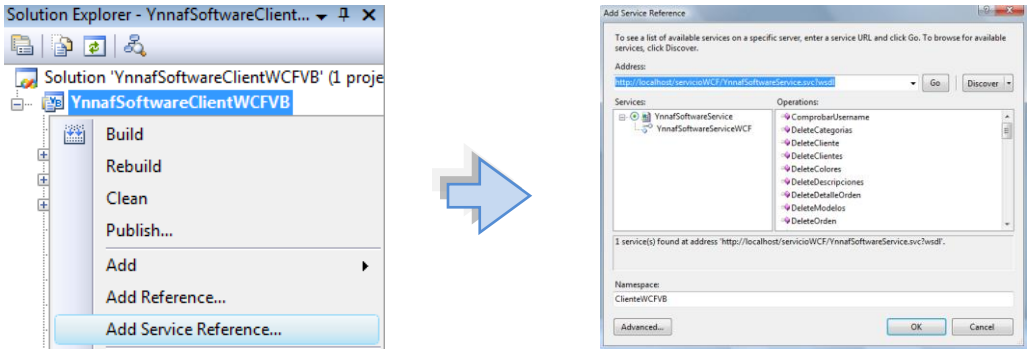
Caso de uso:	**Seguimiento de orden de compra hecha con anterioridad.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *Visual Basic 9.0* utilizando *Windows Forms*			
Descripción:	Se describe como se hace un seguimiento a las órdenes de compra hechas con anterioridad.			
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*			
Post-condiciones:	El cliente obtiene información del estado de su orden de compra.			
Versión:	0.0.1			
Fecha de creación	25-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Seguimiento de orden de compra existente			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide hacer un seguimiento de una orden de compra específica realizada con anterioridad por la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio *WCF*, se verifica que exista el número de la orden de compra, al existir, se retorna al llamador todos los datos existentes relacionados con esa orden de compra.	
Flujo alternativo:	Seguimiento de orden de compra inexistente			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide hacer un seguimiento de una orden de compra específica realizada con anterioridad por la sucursal al servicio *WCF*.	2	Dado que hay una instancia y sesión creada en el servicio WCF, se verifica que exista el número de la orden de compra, al no existir, se retorna al llamador una *FaultException* (*OrdenFault*). Esta excepción debe ser interceptada por el cliente *WCF*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

5.3.4 IMPLEMENTACIÓN

A continuación se muestra la implementación del programa en *Visual Basic 9.0/Windows Forms*:

5.3.4.1 IMPORTACIÓN DEL WSDL DEL SERVICIO WCF

Para poder generar el código que nos permita acceder al servicio WCF debemos seguir los siguientes pasos: *1)* Crear una referencia de servicio usando la dirección provista por la empresa *YnnafSoftware* para el servicio *WCF*: *http://localhost/servicioWCF/YnnafSoftwareService.svc*. 2) El espacio de nombres (*Namespace*) para acceder al servicio en el cliente será: *ClienteWCFVB*



Referencia al servicio WCF

Una vez finalizada la generación del código del servicio *WCF*, podemos ver que los códigos tanto en el servicio como en el cliente son similares, el modo de uso depende del lenguaje de programación:

DEFINICIÓN DEL CONTRATO DE OPERACIÓN SP_CREATEORDEN EN EL SERVICIO WCF

```
[OperationContract(Name = "Sp_CreateOrden", IsInitiating = false, IsTerminating = false)]
[FaultContract(typeof(OrdenFault))]
int Sp_CreateOrden(Orden orden, String listaProductos, ref int ordenId,
ref String DescripcionResultado);
```

EL MISMO CONTRATO DE OPERACIÓN EN EL CLIENTE WCF CONSTRUIDO EN VISUAL BASIC 9.0 TRAS LA IMPORTACIÓN (ESTE CÓDIGO ES GENERADO AUTOMÁTICAMENTE)

```
<System.ServiceModel.OperationContractAttribute(IsInitiating:=false,
Action:="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrden",
ReplyAction:="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenResponse"
), _
System.ServiceModel.FaultContractAttribute(GetType(ClienteWCFVB.OrdenFault),
Action:="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenOrdenFaultFault", _
Name:="OrdenFault", [Namespace]:="http://www.ynnafSoftware.com/DataContract/Faults")>  _
Function Sp_CreateOrden(ByVal orden As ClienteWCFVB.Orden, ByVal listaProductos As String, _
    ByRef ordenId As Integer, ByRef descripcionResultado As String) As Integer
```

5.3.4.2 INSTANCIA SINGLETON PARA EL CLIENTE WCF

Puesto que el servicio *WCF* maneja instancias que guardan valores ayudándose de sesiones, es necesario mantener en el cliente *WCF* una instancia única con el servicio *WCF* para obtener un correcto funcionamiento. Para lograr esto, utilizaremos el patrón *Singleton*. A continuación se muestra el código fuente escrito en *Visual Basic 9.0* que llevará a cabo esta tarea.

```
Public NotInheritable Class InstanceClienteWCF
    ' Mantiene la instancia única
    Private Shared _instance As InstanceClienteWCF = Nothing

    Private _servicioWCF As YnnafSoftwareServiceWCFClient

    ''' <summary>
    ''' Evita la construcción de un objeto de esta clase de forma externa
    ''' </summary>
    Private Sub New()
```

```
    End Sub

    ''' <summary>
    ''' Obtiene la instancia única del cliente WCF
    ''' </summary>
    Public Shared ReadOnly Property Instance() As InstanceClienteWCF
        Get
            If _instance Is Nothing Then
                _instance = New InstanceClienteWCF()
            End If
            Return _instance
        End Get
    End Property

    ''' <summary>
    ''' Obtiene o establece la instancia del servicio WCF
    ''' </summary>
    Public Property ServicioWCF() As YnnafSoftwareServiceWCFClient
        Get
            Return _servicioWCF
        End Get
        Set(ByVal value As YnnafSoftwareServiceWCFClient)
            Me._servicioWCF = value
        End Set
    End Property
End Class
```

Con este código fuente (contenido en el espacio de nombres *YnnafSoftwareClientWCFVB*) aseguramos tener una y solo una instancia del cliente *WCF*. La forma en que llamaremos a esta instancia será usando: *InstanceClienteWCFVB.Instance.*

5.3.4.3 ENCRIPTACIÓN DE CONTRASEÑAS.

Dado que el servicio *WCF* requiere que la contraseña enviada para autorizar el acceso del usuario a la lógica del negocio esté encriptada en el algoritmo de seguridad *SHA-512* usando la codificación *UTF-8*, el código siguiente muestra la forma de hacerlo usando *Visual Basic 9.0*:

```
Public Shared Function EncriptarSha512(ByVal cadenaSinEncriptar As String) As String
    Try
        Dim cadenaSinEncriptarArray As Byte() = _
            Encoding.UTF8.GetBytes(cadenaSinEncriptar)
        Dim sha512 As New SHA512Managed()
        Dim cadenaEncriptadaArray As Byte() = sha512.ComputeHash( _
            cadenaSinEncriptarArray, 0, cadenaSinEncriptarArray.Length)
        Return Convert.ToBase64String( _
            cadenaEncriptadaArray, 0, cadenaEncriptadaArray.Length)
    Catch ex As Exception
        Console.WriteLine(ex.Message)
        Return Nothing
    End Try
End Function
```

Hay que destacar que para que este método funcione adecuadamente se deben importar los espacios de nombres: *System.Text* y *System.Security.Cryptography.*

5.3.4.4 LLAMADAS A CONTRATOS DE OPERACIONES DEL SERVICIO WCF

A continuación se muestra el código fuente utilizado para llamar a los contratos de operación del servicio *WCF* necesarios para la ejecución del programa. Se debe notar que no se coloca el código para inicializar la interfaz gráfica de usuario.

5.3.4.4.1 INICIO DE SESIÓN.

Lo primero que necesitamos hacer para utilizar el programa es autenticar el cliente *WSIT* ante el servicio *WCF*. Para lograr esto, se crea una instancia de *YnnafSoftwareServiceWCF*, se establecen los certificados *X.509* y se asigna esta instancia a la instancia *Singleton*. Una vez hecho lo anterior se debe obtener autorización de acceso a la lógica del negocio del sistema, debemos entonces llamar al contrato de operación *IniciarSesion(username, password)* del servicio *WCF*. Si el nombre de usuario y contraseña son correctos se asigna el campo encapsulado *_sesionIniciada* con el valor devuelto por el servicio (*True*), de lo contrario se caerá en el bloque *Catch*. A partir de la instancia *Singleton* accederemos, en toda la ejecución del programa al servicio *WCF*. En el código fuente siguiente, podemos ver cómo se realiza el procedimiento descrito anteriormente:

```
Private Sub AceptarButton_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles AceptarButton.Click
    Try
        Dim settingsPath As ConnectionStringSettings
        settingsPath = ConfigurationManager.ConnectionStrings("PathCertificado")
        Dim settingsPass As ConnectionStringSettings
        settingsPass = ConfigurationManager.ConnectionStrings("PassCertificado")

        Dim instanciaClienteWCF As New ClienteWCFVB.YnnafSoftwareServiceWCFClient()
        instanciaClienteWCF.ClientCredentials.ClientCertificate.Certificate = New _
            X509Certificate2(settingsPath.ConnectionString, settingsPass.ConnectionString)
        InstanceClienteWCF.Instance.ServicioWCF = instanciaClienteWCF

        Me._sesionIniciada = InstanceClienteWCF.Instance.ServicioWCF.IniciarSesion( _
            Me.usernameTextBox.Text, Encriptar.EncriptarSha512(Me.passwordTextBox.Text))
        If Me._sesionIniciada = True Then
            Me.DialogResult = Windows.Forms.DialogResult.OK
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)
    End Try
End Sub
```

NOTA: La ruta física y contraseña del certificado *X.509* para autenticar al cliente ante el servicio se deben obtener desde el archivo de configuración *app.config.* La razón de esto es para prevenir que sean obtenidas por usuarios malintencionados al ofuscar el ensamblado.

5.3.4.4.2 OBTENCIÓN DE PRODUCTOS

La obtención y visualización de todos los productos que contiene la empresa se hace después que el cliente realizó un inicio de sesión exitoso para obtener acceso a la lógica del negocio del sistema (ver código anterior). El procedimiento a seguir para obtener todos los productos es: *1)* Llamar al contrato de operación *GetProductos* que obtiene todos los productos desde la base de datos y asignarlo a una lista genérica y *2)* Llenar el *ComboBox* con los Id de los productos de la lista genérica. El código fuente siguiente ejemplifica los pasos anteriores:

```
Private Sub IniciarSesiónToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles IniciarSesiónToolStripMenuItem.Click
    Try
        Dim dialogo As New LoginForm()
        If dialogo.ShowDialog(Me) = Windows.Forms.DialogResult.OK Then
            ' Verificar si se inició una sesión
            If dialogo.SesionIniciada = True Then
                ' Obtener todos los productos desde el servicio WCF
                listaProductos = InstanceClienteWCF.Instance.ServicioWCF. _
                    GetProductos().ToList()
                ' Llenar el ComboBox con los id de los productos
                For Each producto As ClienteWCFVB.Producto Join In listaProductos
                    Me.ProductoIdComboBox.Items.Add(producto.ProductoId)
```

```
                Next
                ' Mostrar los GroupBox
                MostrarGroupBox(True)
                ' Habilitar / inhabilitar controles
                Me.IniciarSesiónToolStripMenuItem.Enabled = False
                Me.CerrarSesiónToolStripMenuItem.Enabled = True
                Me.ÓrdenesToolStripMenuItem.Enabled = True
                ' Seleccionar automáticamente el primer elemento del ComboBox
                If Me.ProductoIdComboBox.Items.Count > 0 Then
                    Me.ProductoIdComboBox.SelectedIndex = 0
                    ProductoIdComboBox_SelectionChangeCommitted(Nothing, Nothing)
                End If
            End If
        Else
            MessageBox.Show(Me, "Sesión no iniciada", "No iniciado", _
                MessageBoxButtons.OK, MessageBoxIcon.Warning)
        End If
    Catch ex As Exception
        MessageBox.Show(Me, ex.Message, "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error)
    End Try
End Sub
```

Nota: A partir de la selección del Id del producto en el *ComboBox*, se podrán mostrar detalles del producto en otros controles del *Form* como *TextBox* y *PictureBox*.

5.3.4.4.3 CREACIÓN DE ORDEN.

Dado que los productos de la orden de compra se agregan a un *ListView* (*OrdenListView*), se debe obtener el patrón {*Cantidad, ProductoId_1, Cantidad, ProductoId_2, …*} requerido por el procedimiento almacenado del servicio *WCF* a partir de él, eso se hace en la primera parte del método. . Lo segundo que se debe hacer es inicializar las otras cuatro variables que se pasarán como parámetros al contrato de operación del servicio, es decir, se debe construir (e inicializar) la clase *Orden* e inicializar las variables para *ordenId*, *descripcionResultado* y *resultado*, estas últimas almacenarán el valor devuelto por el contrato de operación y los valores obtenidos vía los parámetros por referencia del mismo. El tercer paso es mandar a llamar al contrato de operación *Sp_CreateOrden* del servicio *WCF* pasándole los 5 parámetros necesarios y mostrar al usuario los posibles resultados devueltos tras su ejecución (éxito o error). En el código fuente siguiente, podemos ver cómo se realiza el procedimiento descrito anteriormente:

```
Private Sub EnviarButton_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles EnviarButton.Click
    Try
        ' Obtención del patrón: Cantidad, ProductoId_1, Cantidad, ProductoId_2, …
        Dim listaProductos As String = ""
        Dim i As Integer
        For Each lvi As ListViewItem In Me.OrdenListView.Items
            listaProductos += lvi.SubItems(2).Text
            listaProductos += ","
            listaProductos += lvi.SubItems(1).Text
            If Not i = Me.OrdenListView.Items.Count - 1 Then
                listaProductos += ","
            End If
            i += 1
        Next

        ' Creación del objeto Orden
        Dim orden As New ClienteWCFVB.Orden()
        orden.CargoExtra = Nothing
        orden.ClienteId = 0 ' Un valor cualquiera
        orden.FechaAtencion = Nothing
        orden.FechaEntrega = Nothing
        orden.FechaEnvio = Nothing
        orden.FechaModificacion = DateTime.Now
        orden.FechaOrden = DateTime.Now
        orden.FormaPagoOrdenId = 1 ' Un valor cualquiera pero existente (FK)
```

```
        orden.Iva = Nothing
        orden.MetodoEnvioOrdenId = 1 ' Un valor cualquiera pero existente (FK)
        orden.NumeroCuenta = Nothing
        orden.StatusOrdenId = 0 ' Un valor cualquiera

        ' Variables para los parámetros de salida
        Dim ordenId As Integer
        Dim descripcionResultado As String = Nothing
        Dim resultado As Integer

        ' Llamado al contrato de operación
        resultado = InstanceClienteWCF.Instance.ServicioWCF.Sp_CreateOrden( _
            orden, listaProductos, ordenId, descripcionResultado)

        ' Mostrar error si ocurrió alguno
        If resultado = 0 Then
            MessageBox.Show(Me, descripcionResultado, "Error", _
                MessageBoxButtons.OK, MessageBoxIcon.Error)
            Exit Sub ' Salir
        End If

        ' Mostrar número de orden creada
        MessageBox.Show(Me, "Orden creada con el Id de orden: " & _
            ordenId.ToString(), "Orden creada exitosamente", _
            MessageBoxButtons.OK, MessageBoxIcon.Information)
    Catch ex As Exception
        MessageBox.Show(Me, ex.Message(), "Error", MessageBoxButtons.OK, _
            MessageBoxIcon.Error)
    End Try
End Sub
```

5.3.4.4.4 SEGUIMIENTO DE ORDEN

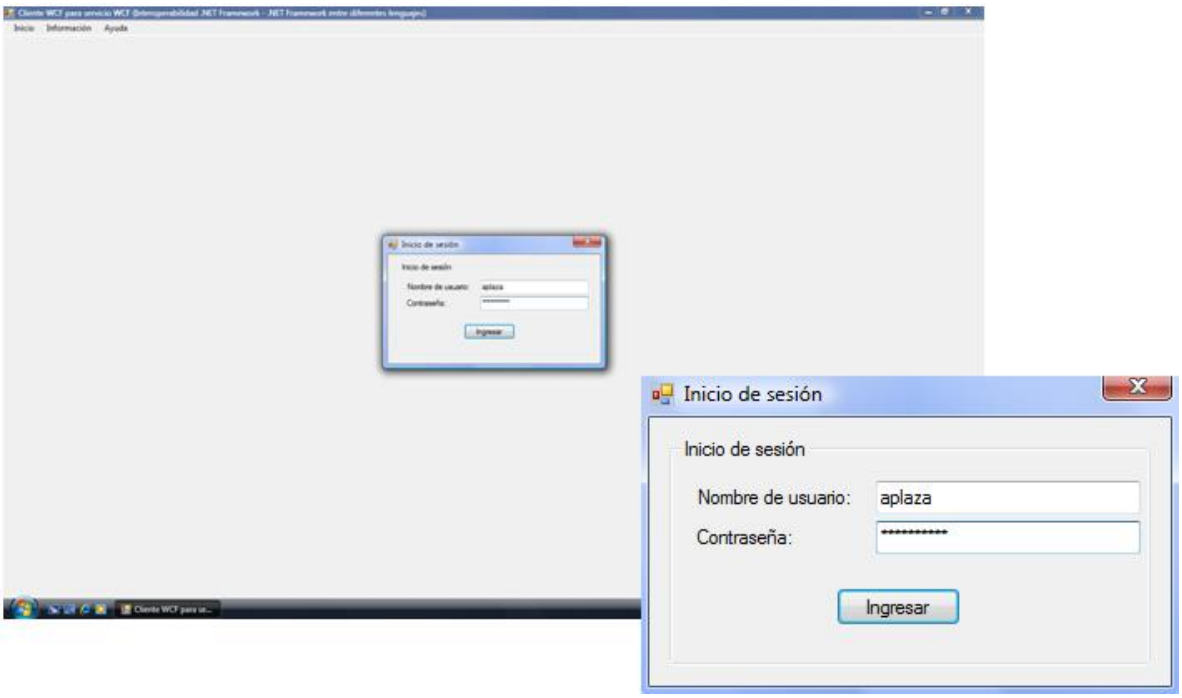
Para hacer seguimientos de las órdenes hechas por las *sucursales*, se debe buscar por el número de la orden de compra. Los pasos que se deben seguir para realizar un seguimiento son los siguientes: *1)* Comprobar que haya un número de orden a buscar, *2)* Llamar al contrato de operación *Sp_GetOrdenById* el cual obtiene la información de la orden de compra y almacenar el resultado en el contrato de datos *Sp_Orden_Id* y *3)* Mostrar los datos generales de la orden en los *TextBox* y los datos particulares en el *ListView* del programa. El código fuente que realiza el procedimiento anterior es el que se muestra a continuación:

```
Private Sub BuscarButton_Click(ByVal sender As System.Object, ByVal e _
    As System.EventArgs) Handles BuscarButton.Click
    ' Verificar que el TextBox del Id de orden no esté vacío
    If String.IsNullOrEmpty(Me.BuscarTextBox.Text) Then
        MessageBox.Show(Me, "El id de orden a buscar no puede estar vacío", "Error", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)
    Else
        Try
            Me.OrdenListView.Items.Clear() ' Limpiar el ListView
            Dim resultado As New ClienteWCFVB.Sp_Orden_Id()
            ' Obtener la orden de acuerdo al Id de la orden, la respuesta del servicio
            ' WCF se almacena en la instancia "resultado"
            resultado = InstanceClienteWCF.Instance.ServicioWCF.Sp_GetOrdenById(
                Convert.ToInt32(Me.BuscarTextBox.Text))

                ' Llenar los controles con los datos generales de la orden
            Me.OrdenIdTextBox.Text = resultado.Orden.OrdenId.ToString()
            Me.FechaOrdenTextBox.Text = resultado.Orden.FechaOrden.ToString()
            Me.FechaAtencionTextBox.Text = resultado.Orden.FechaAtencion.ToString()
            Me.FechaEnvioTextBox.Text = resultado.Orden.FechaEnvio.ToString()
            Me.FechaEntregaTextBox.Text = resultado.Orden.FechaEntrega.ToString()
            Me.EstadoOrdenTextBox.Text = resultado.Orden.StatusOrden

            ' Asignar la lista genérica con los datos del detalle de la orden
            Dim listaOrdenDetalle As List(Of ClienteWCFVB.OrdenDetalle_Join)
            listaOrdenDetalle = resultado.ListaDetalleOrden.ToList()
```

```
            ' Iterar con la lista genérica y mostrar los datos en el ListView
            For Each ordenDetalle As ClienteWCFVB.OrdenDetalle_Join In listaOrdenDetalle
                Me.listaImagenes.Images.Add(ordenDetalle.ProductoId.ToString(), _
                    ConversorImagenes.ConvertirBytesAImagen(ordenDetalle.Imagen.Bytes))
                Dim lvi As New ListViewItem()
                lvi.ImageKey = ordenDetalle.ProductoId.ToString()
                lvi.SubItems.Add(ordenDetalle.ProductoId.ToString())
                lvi.SubItems.Add(ordenDetalle.Cantidad.ToString())
                lvi.SubItems.Add(ordenDetalle.Producto)
                lvi.SubItems.Add(ordenDetalle.UnidadesDisponibles.ToString())
                lvi.SubItems.Add(ordenDetalle.Producto)
                lvi.SubItems.Add(ordenDetalle.Categoria)
                lvi.SubItems.Add(ordenDetalle.Descripcion)
                lvi.SubItems.Add(ordenDetalle.Color)
                lvi.SubItems.Add(ordenDetalle.CostoEstandar.ToString())
                lvi.SubItems.Add(ordenDetalle.PrecioUnitario.ToString())
                lvi.SubItems.Add(ordenDetalle.Tamaño)
                lvi.SubItems.Add(ordenDetalle.Peso)
                lvi.SubItems.Add(ordenDetalle.FechaInicioVenta.ToString())
                lvi.SubItems.Add(ordenDetalle.FechaFinVenta.ToString())
                Me.OrdenListView.Items.Add(lvi)
            Next
        Catch ex As ServiceModel.FaultException(Of ClienteWCFVB.OrdenFault)
            MessageBox.Show(Me, "No se encontró ningún resultado", "Error", _
                MessageBoxButtons.OK, MessageBoxIcon.Error)
        Catch ex As Exception
            MessageBox.Show(Me, ex.GetType().ToString(), "Error", MessageBoxButtons.OK, _
                MessageBoxIcon.Error)
        End Try
    End If
End Sub
```

5.3.5 DESPLIEGUE

Autenticación de usuario

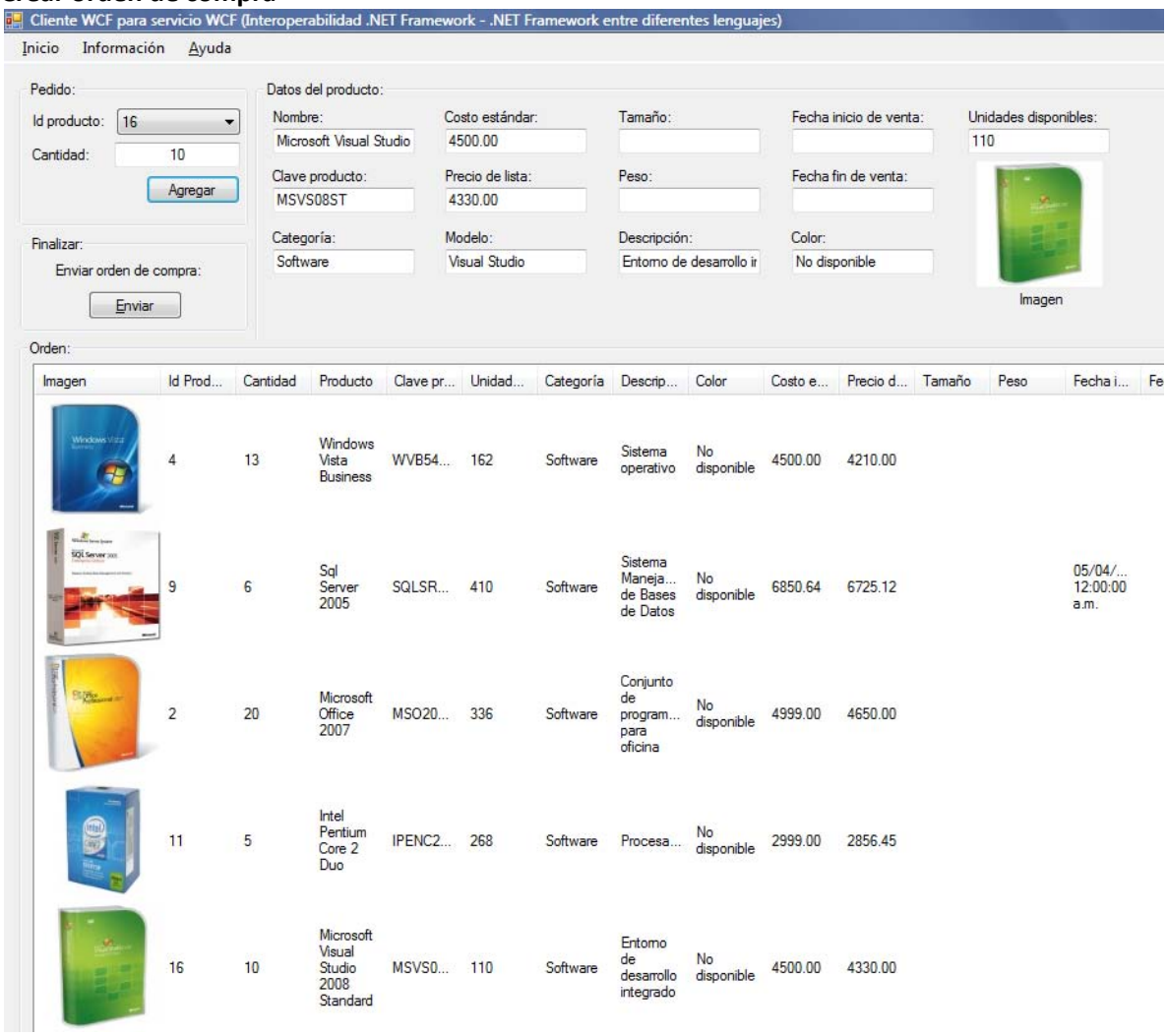


Inserción de nombre de usuario y contraseña. Si tanto el nombre de usuario como la contraseña son correctos, el usuario podrá utilizar las funciones del programa en su totalidad, de lo contrario el programa no podrá ser utilizado en ninguna de sus funciones.

Visualizar productos

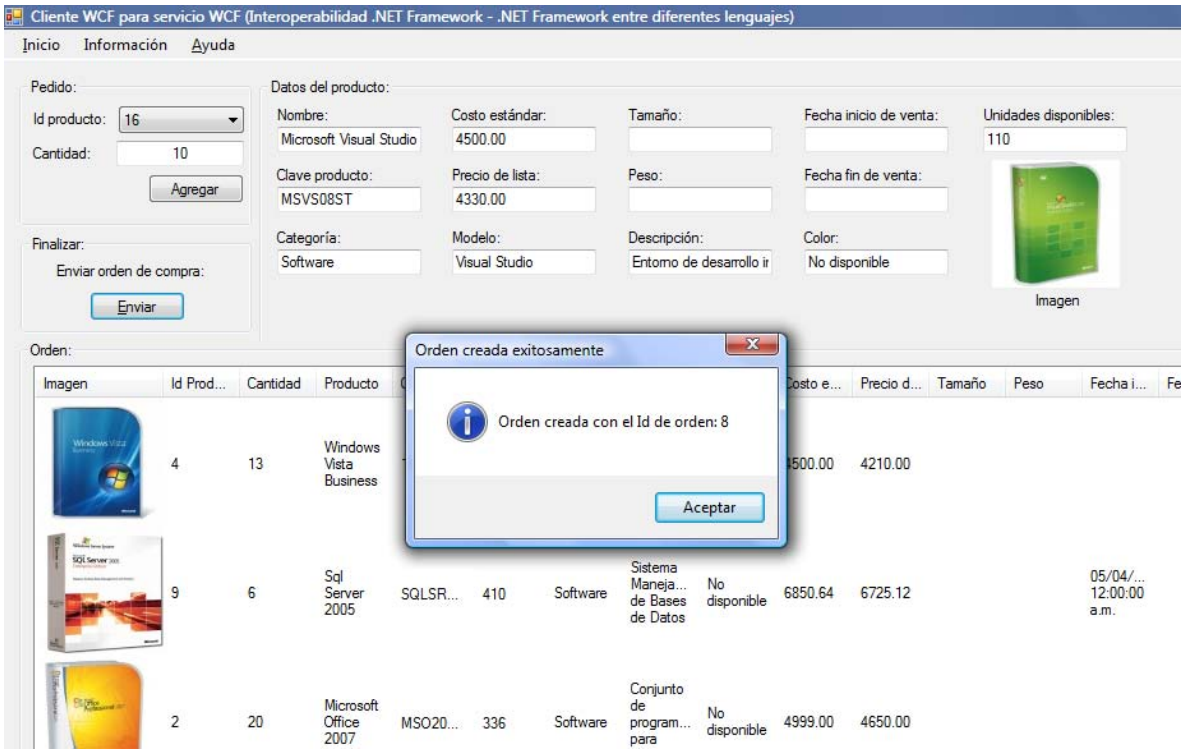
Visualización de productos. El usuario puede ver la lista de productos (ordenados por el id del producto en un ***JComboBox***) que se encuentran almacenados en la base de datos del sistema. Cada vez que el usuario selecciona un nuevo id producto, puede ver todas las características del productos seleccionado.

Crear orden de compra



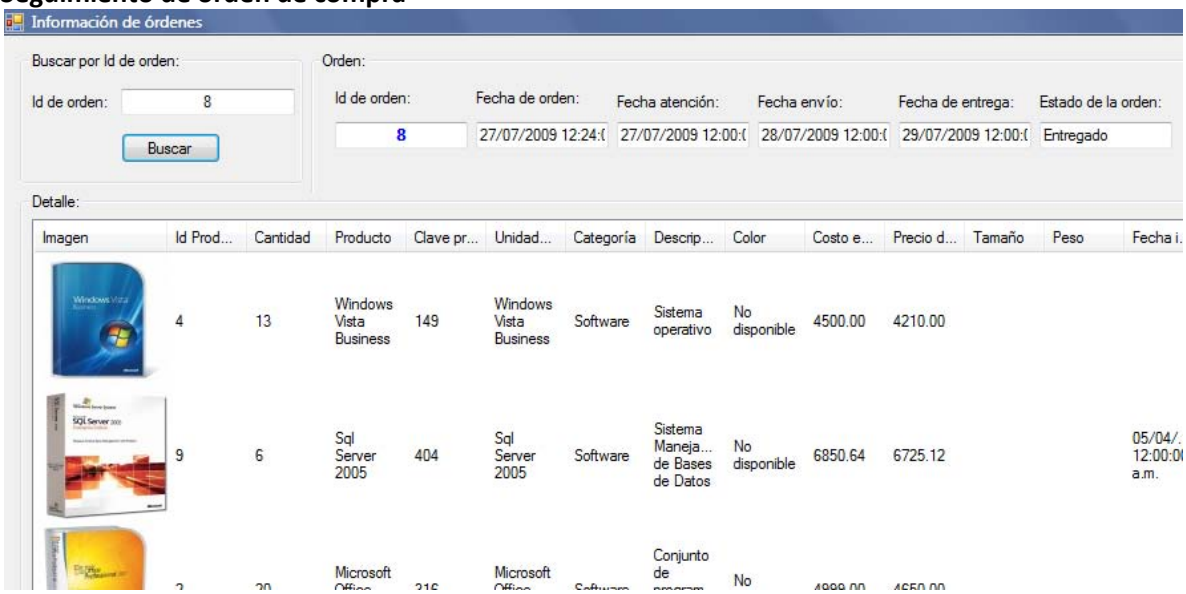
Creación de orden de compra 1. 1) El usuario selecciona el ***producto*** que desea la sucursal e inserta la ***cantidad*** requerida y presiona el botón ***Agregar***. Los productos se van agregando en la lista inferior del programa. Si se desea modificar la

cantidad de productos a comprar se deberá dar doble clic en un producto específico y modificar la cantidad en el cuadro de diálogo que aparecerá.



Creación de orden de compra 2. 1) El usuario selecciona el ***producto*** que desea la sucursal e inserta la ***cantidad*** requerida, una vez que se agregaron todos los productos necesarios para la orden de compra, 2) Se presiona el botón ***Enviar*** para finalizar la orden de compra. En el ejemplo anterior se creó la orden de compra número 8.

Seguimiento de orden de compra



Seguimiento de orden de compra. 1) El usuario abre la ventana Seguimiento de órdenes desde el menú ***Información*** -> ***Órdenes…***, 2) Se inserta el ***id de la orden*** de la cual se desea obtener su estado, 3) Se presiona el botón ***Buscar***, Si la orden de compra existe, se muestra en la ventana la información de su orden de compra.

5.4 CREACIÓN DE UN CLIENTE WCF EN C# - ASP.NET. APLICACIÓN WEB

A continuación se muestra como *Windows Communication Foundation* además de tener una completa compatibilidad con los clientes construidos anteriormente (aplicaciones de escritorio), también se pueden construir aplicaciones web que sean 100% compatibles con el servicio *WCF* sin ningún problema. La aplicación cliente se construirá utilizando *ASP.NET, C#, HTML, JS, CSS,* etc.

5.4.1 REQUERIMIENTOS

Se requiere construir una aplicación web que permita realzar compras vía Internet. Esta aplicación debe cumplir las siguientes condiciones:

1. Establecer comunicación con el servicio *WCF* utilizando el certificado *X.509* provisto por la empresa para acceder a la lógica del negocio.
2. El sitio web debe ser visualmente atractivo para el cliente, el código fuente resultante debe ser fácil de mantener y actualizar a futuro; y las páginas deben ser ejecutadas a una rápida velocidad.
3. Se debe poder hacer búsquedas de productos por medio de su nombre.
4. El usuario puede comprar cualquier cantidad de productos, él debe insertar el número si la cantidad de productos es mayor a uno.
5. El usuario debe estar registrado y autentificado para poder finalizar una orden de compra.
6. Una vez que el usuario seleccionó la forma de pago, método de envío y se finalizó la compra, se le indicará cuál es el número de su orden de compra para que a través de ella pueda hacer seguimientos generales y particulares de sus órdenes.

5.4.2 ANÁLISIS

Puesto que se requiere que el sitio de Internet sea visualmente atractivo y fácil de mantener, se usará *ASP.NET 3.5* (escribiendo el *CodeBehind* en *C# 3.0*), *CSS* y *Java Script*. Para que el sitio web se ejecute rápidamente, se deberá crear éste de forma totalmente pre-compilada, usando un proyecto de Aplicación Web (*Web Application*). El uso de *ASP.NET 3.5* con *C# 3.0* brindará un mejor rendimiento en comparación con otras tecnologías como *Java Servlets*, *Java Struts*, *Java Server Faces*, *PHP*, *Active Server Pages (ASP)*, etc. dado que la forma en que se desarrollan aplicaciones web es *MUY* similar a la forma en que se desarrollan aplicaciones de escritorio, esto trae como consecuencia que no se tenga que cambiar la forma en que los programadores de la empresa crean aplicaciones de software.

5.4.3 DISEÑO

A continuación se muestran los casos de uso detallados que se deben seguir para los diferentes procedimientos que se utilizarán a lo largo de la aplicación web.

5.4.3.1 CASOS DE USO DETALLADOS

Caso de uso:	**Inicio de sesión con el servicio WCF**
Tipo:	Básico
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*
Descripción:	Se describe como se obtiene instancia con el servicio *WCF*.
Pre-condiciones:	El cliente *WCF* debe poseer los certificados *X.509* adecuados para el servicio *WCF*
Post-condiciones:	El cliente *WCF* obtiene instancia y sesión con el servicio *WCF*.

Versión:	0.0.2			
Fecha de creación	01-03-2009			
Fecha de actualización:	15-08-2009			
FLUJOS				
Flujo básico:	Inicio de sesión con nombre de usuario y contraseña.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	Al cargarse la página se debe iniciar una sesión llamando al contrato de operación *IniciarSesion(username, password)*. El nombre de usuario y contraseña dependen de la autenticación del usuario con el sitio web de *ASP.NET*.	2	Se obtiene un valor booleano indicando si el usuario existe en la base de datos.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Mostrar productos.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se obtiene una lista de todos los productos desde el servicio *WCF*.			
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*			
Post-condiciones:	El cliente obtiene una lista de todos los productos.			
Versión:	0.0.1			
Fecha de creación	01-03-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Muestra los productos de la empresa en la página principal.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario accede a la página principal del sitio web. El cliente *WCF* inicia sesión con el servicio *WCF* y obtiene los productos que se mostrarán al usuario.	2	Se muestran los productos de la empresa en la página principal (*default.aspx*) y se finaliza y destruye la sesión y el objeto del servicio *WCF*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Buscar productos.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se buscan tipos de productos específicos desde el servicio *WCF*.			
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*			
Post-condiciones:	El cliente obtiene una lista de los productos que coincidieron con su búsqueda			
Versión:	0.0.1			
Fecha de creación	01-03-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Muestra los productos de la empresa en la página inicial.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario accede a la página principal del sitio web. El cliente *WCF* inicia sesión con el servicio *WCF* y obtiene los productos buscados que se mostrarán al usuario.	2	Se muestran los productos de la empresa en la página de búsqueda (*buscar.aspx*) y se finaliza y destruye la sesión y el objeto del servicio *WCF*.	

EXCEPCIONES		
Identificador	**Nombre**	**Respuesta del sistema**

Caso de uso:	**Autenticación del usuario (Login)**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se autentica un usuario ante el sistema.			
Pre-condiciones:	El cliente debe estar registrado en la base de datos del sistema			
Post-condiciones:	El cliente obtiene la autenticación por formularios.			
Versión:	0.0.1			
Fecha de creación	01-03-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Inicio de sesión con nombre de usuario y contraseña válidos.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide acceder a un directorio restringido, para hacerlo necesita autenticarse ante el sistema. Se inicia sesión con el servicio WCF.	2	Si el nombre de usuario y contraseña son correctos, se le asigna al cliente una autenticación por formularios y se le da permiso de acceso a todas las páginas restringidas.	
Flujo alternativo:	Inicio de sesión con nombre de usuario y contraseña inválidos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide acceder a un directorio restringido, para hacerlo necesita autenticarse ante el sistema. Se inicia sesión con el servicio WCF.	2	Se retorna una excepción, lo cual indica que el servicio no está disponible. Esta excepción debe ser interceptada por la página web y se le debe mostrar al usuario que su contraseña o nombre de usuario no son válidos	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Registro de usuario ante la página**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se registra un usuario ante el sistema.			
Pre-condiciones:				
Post-condiciones:	El usuario obtiene un nombre de usuario y contraseña con el cual puede comprar productos.			
Versión:	0.0.1			
Fecha de creación	01-03-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Registro con datos válidos			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide registrarse ante el sistema para poder comprar productos vía Internet.	2	Si todos los datos son correctos y el nombre de usuario no existe previamente, el sistema registra en la base de datos al usuario.	
Flujo alternativo:	Registro con datos inválidos.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide registrarse ante el sistema para poder comprar productos vía Internet.	2	Si el nombre de usuario ya existe en la base de datos, se informa al usuario que alguno (o algunos) de los campos es (son) requerido (s) y que se debe completar.	

Flujo alternativo:	Registro con nombre de usuario existente.			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide registrarse ante el sistema para poder comprar productos vía Internet.	2	Se retorna una excepción, la cual indica que el nombre de usuario ya existe. Esta excepción debe ser interceptada por la página web y se le debe informar al usuario que el nombre ya existe.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Administración del carrito de compra**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se autentica un usuario ante el sistema.			
Pre-condiciones:				
Post-condiciones:	El cliente mantiene el estado de sus productos.			
Versión:	0.0.1			
Fecha de creación	01-03-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Administración de productos vía instancia Singleton.			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide agregar productos, modificar la cantidad de productos o eliminar productos de su carrito de compra.	2	El sistema administra los productos del carrito de compra a través de la clase *ShoppingCart*.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Crear orden de compra.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se crea una orden al servicio *WCF*			
Pre-condiciones:	El cliente debe tener productos en su carrito de compra y debe estar autenticado ante el sistema.			
Post-condiciones:	El cliente obtiene el número de la orden creada exitosamente.			
Versión:	0.0.1			
Fecha de creación	01-03-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Creación de la orden con cantidad de productos mayor a cero			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide finalizar la orden de compra enviando los datos de los productos que necesita al servicio *WCF*.	2	Dado que el usuario se autenticó ante el sistema y tiene productos en su carrito de compra, se retorna el número de orden creada exitosamente al cliente.	
Flujo alternativo:	Creación de la orden con cantidad de productos mayor a cero			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide finalizar la orden de compra enviando los datos de los productos que necesita al servicio *WCF*.	2	Si la cantidad de productos es igual a cero, se retorna una excepción al cliente. Se le debe mostrar un mensaje en *JavaScript* indicándole que no hay productos a comprar.	

Flujo alternativo:	Creación de la orden sin autenticación previa del usuario			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide finalizar la orden de compra enviando los datos de los productos que necesita al servicio *WCF*.	2	Si el usuario no se ha autenticado, se le redirecciona a la página de *Login* para que lo haga.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

Caso de uso:	**Seguimiento de orden de compra hecha con anterioridad.**			
Tipo:	Básico			
Actor principal:	Cliente *WCF* escrito en *C# 3.0* utilizando *ASP.NET 3.5*			
Descripción:	Se describe como se hace un seguimiento a las órdenes de compra hechas con anterioridad.			
Pre-condiciones:	El cliente debe haber realizado un inicio de sesión con el servicio *WCF*			
Post-condiciones:	El cliente obtiene información del estado de sus órdenes de compra.			
Versión:	0.0.1			
Fecha de creación	25-02-2009			
Fecha de actualización:				
FLUJOS				
Flujo básico:	Seguimiento de todas las órdenes de compra hechas por el cliente			
ACTOR		**SISTEMA**		
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide hacer un seguimiento de una orden de compra realizada con anterioridad.	2	Dado que el usuario se autenticó ante el sistema, se retorna al cliente la información de todas las órdenes de compra hechas por él desde que se dio de alta en el sistema.	
Flujo alternativo:	Seguimiento de orden de compra sin autenticación previa del usuario			
Paso	**Acciones**	**Paso**	**Acciones**	**Excepción**
1	El usuario decide hacer un seguimiento de una orden de compra realizada con anterioridad.	2	Si el usuario no se ha autenticado, se le redirecciona a la página de Login para que lo haga.	
EXCEPCIONES				
Identificador	**Nombre**	**Respuesta del sistema**		

5.4.3.2 DIRECTORIOS Y NOMBRES DE PÁGINAS DEL SITIO WEB

El sitio web se dividirá en 5 directorios: el directorio *raíz (/)*, el directorio *compra*, el directorio *css*, el directorio *js* y el directorio *imagenes*. A continuación se lista el contenido de cada uno de ellos:

DIRECTORIO RAÍZ (/) (DIRECTORIO PÚBLICO)				
ARCHIVOS	buscar.aspx	carrito.aspx	default.aspx	ImagenesHandler.ashx
	login.aspx	masInformacion.aspx	registro.aspx	

DIRECTORIO COMPRA (DIRECTORIO ACCESIBLE ÚNICAMENTE CON ANTENTICACIÓN)				
ARCHIVOS	carrito2.aspx	compraSatisfactoria.aspx		
	seguimiento.aspx			

DIRECTORIO CSS (DIRECTORIO PARA CASCADE STYLE SHEET)				
ARCHIVOS	Body.css	GridView.css	GridViewDiv.css	Login.css
	MenuNavegacion.css	TituloDiv.css	WaterMark.css	From.css

DIRECTORIO IMÁGENES (DIRECTORIO PARA IMÁGENES)				
ARCHIVOS	Logo.png			

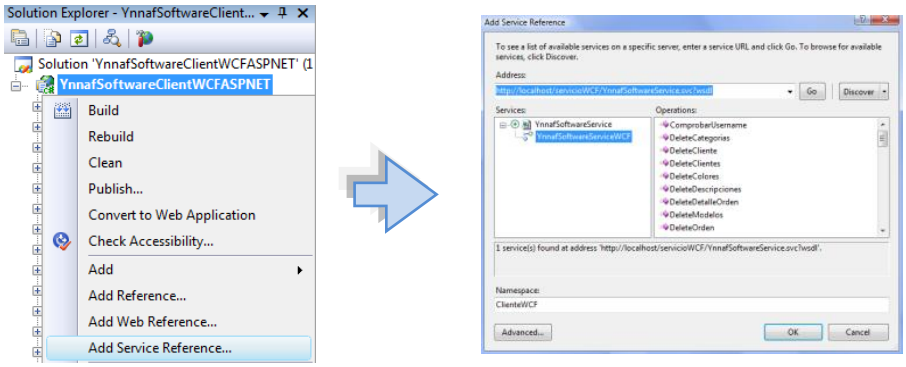
DIRECTORIO JS (DIRECTORIO PARA JAVASCRIPT)				
ARCHIVOS	jsSHA/sha.js	centrarPopup.js	mantenerUsernamePassword.js	
	textBoxBuscar.js	verificarCampos.js	verificarCompra.js	

5.4.4 IMPLEMENTACIÓN

A continuación se muestra la implementación del programa en *C#3.0/ASP.NET*:

5.4.4.1 IMPORTACIÓN DEL WSDL DEL SERVICIO WCF

Para poder generar el código que nos permita acceder al servicio WCF debemos seguir los siguientes pasos: *1)* Crear una referencia de servicio usando la dirección provista por la empresa *YnnafSoftware* para el servicio *WCF*: *http://localhost/servicioWCF/YnnafSoftwareService.svc*. *2)* El espacio de nombres (*Namespace*) para acceder al servicio en el cliente será: *ClienteWCF*



Referencia al servicio WCF

Una vez finalizada la generación del código del servicio *WCF*, podemos ver que los códigos tanto en el servicio como en el cliente son similares, el modo de uso depende del lenguaje de programación; aquí al ser el mismo lenguaje, el modo de uso es el mismo:

DEFINICIÓN DEL CONTRATO DE OPERACIÓN SP_CREATEORDEN EN EL SERVICIO WCF

```
[OperationContract(Name = "Sp_CreateOrden", IsInitiating = false, IsTerminating = false)]
[FaultContract(typeof(OrdenFault))]
int Sp_CreateOrden(Orden orden, String listaProductos, ref int ordenId,
ref String DescripcionResultado);
```

EL MISMO CONTRATO DE OPERACIÓN EN EL CLIENTE WCF CONSTRUIDO EN ASP.NET/C# 3.0 TRAS LA IMPORTACIÓN (ESTE CÓDIGO ES GENERADO AUTOMÁTICAMENTE)

```
[System.ServiceModel.OperationContractAttribute(IsInitiating=false,
Action="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrden",
ReplyAction="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenResponse")]
[System.ServiceModel.FaultContractAttribute(typeof(YnnafSoftwareClientWCFASPNET.ClienteWCF
.OrdenFault),
Action="http://www.ynnafSoftware.com/YnnafSoftwareServiceWCF/Sp_CreateOrdenOrdenFaultFault
", Name="OrdenFault", Namespace="http://www.ynnafSoftware.com/DataContract/Faults")]
int Sp_CreateOrden(YnnafSoftwareClientWCFASPNET.ClienteWCF.Orden orden, string
listaProductos, ref int ordenId, ref string descripcionResultado);
```

5.4.4.2 ENCRIPTACIÓN DE CONTRASEÑAS EN SHA-512.

Dado que el servicio *WCF* requiere que la contraseña necesaria para autorizar el acceso del usuario a la lógica del negocio esté encriptada en el algoritmo de seguridad *SHA-512* usando la codificación *UTF-8.* La definición del método (estático) necesario para realizar esto es el mismo que se utiliza en el cliente *C#/WPF*. Este método y su clase se encentran en el ensamblado: *YnnafSoftwareClientWCFASPNET_Encrypt.dll* referenciado al proyecto.

5.4.4.3 AJAX CONTROL TOOLKIT

Para el propósito facilitar el desarrollo de este sitio web como *RIA* (*Rich Internet Application*) se va a hacer uso del ensamblado *AjaxControlToolkit.dll* (referenciándolo al proyecto) disponible desde la página: *http://www.codeplex.com/Wiki/-View.aspx?ProjectName=AjaxControlToolkit.* El *AJAX Control Toolkit* es un proyecto conjunto entre la *comunidad* y *Microsoft*, construido para las *Extensiones de ASP.NET 3.5 AJAX*, el *Toolkit* es la más grande y mejor colección de controles para aplicaciones web cliente disponible.

Para que todas las páginas de nuestro sitio web puedan hacer uso de este ensamblado debemos registrarlo en cada una de las páginas que vayan a hacer uso de él como se muestra a continuación.

```
<%@ Register Assembly="AjaxControlToolkit"
Namespace="AjaxControlToolkit" TagPrefix="cc1" %>
```

5.4.4.4 AUTENTICACIÓN POR FORMULARIOS

Como se indicó en los requerimientos, necesitaremos restringir el acceso de los usuarios a ciertas páginas (contenidas en el directorio *compra*) si no están previamente autorizados a verlas (si no han hecho *login*). El archivo *Web.config* de la aplicación deberá tener las siguientes líneas de código *XML* entre las etiquetas *<configuration> <system.web>*

```
<?xml version="1.0"?>
<configuration>
  <!-- Otras configuraciones -->
  <system.web>
    <!--Configurar que la autenticación sea por formularios y no por Windows-->
    <authentication mode="Forms">
    <forms name="YnnafSoftwareCookie" loginUrl="login.aspx"
         defaultUrl="default.aspx" timeout="30" protection="All" path="/"/>
    </authentication>
    <authorization>
    <!--Permitir que cualquier usuario pueda acceder a las
        páginas del directorio raíz-->
      <allow users="*"/>
    </authorization>
  </system.web>
  <!-- Otras configuraciones -->
</configuration>
```

Con el código anterior estamos indicando que el directorio raíz es accesible para cualquier usuario que se encuentre navegando a través de nuestro sitio web. Además del código anterior, el archivo *Web.config* deberá tener las siguientes líneas de código *XML* entre las etiquetas *<configuration> <location>*

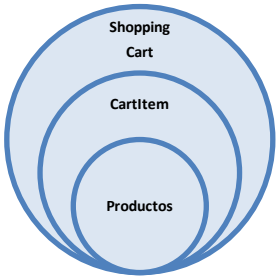
```
<?xml version="1.0"?>
<configuration>
 <!-- Otras configuraciones -->
```

```
  <!-- Impedir el acceso a ciertos directorios -->
  <!-- Todas las páginas contenidas en el directorio 'compra' requieren
       Autenticación -->
  <location path="compra">
   <system.web>
    <authorization>
      <!--Exigir autenticación a los usuarios no autenticados-->
      <deny users="?"/>
    </authorization>
   </system.web>
  </location>
  <!-- Otras configuraciones -->
</configuration>
```

En el código anterior se está indicando que el directorio *compra* es accesible solo para los usuarios que hayan hecho previamente un Inicio de sesión exitoso con el servicio *WCF* (y se le haya autenticado en la página de *ASP.NET* a través del código *C#*). Si un usuario trata de entrar a las páginas almacenadas en el directorio sin estar autenticado, se le redireccionará a la página *~/login.aspx*.

5.4.4.5 IMPLEMENTACIÓN DEL CARRITO DE COMPRA

Para poder implementar el carrito de compra vamos a basarnos en el uso de 3 clases; la clase *Productos*, *CartItem* y *ShoppingCart*.



El diagrama de la izquierda muestra la relación entre las tres clases. La clase *Productos* contiene información sobre los productos que se venden, la clase *CartItem* hace uso de la clase *Productos* y sirve para obtener la información un producto específico a comprar y finalmente la clase *ShoppingCart* la cual lleva el control de los productos agregados al carrito de compra. A continuación se muestra la descripción y definición de estas clases.

5.4.4.5.1 LA CLASE PRODUCTOS

Las propiedades de esta clase son iguales a las de la clase *Producto_Join* (proveniente del servicio *WCF*). Esta clase se encargará de regresar al llamador (la clase *CartItem*) las características específicas del producto (de acuerdo al *Id* del producto pasado como parámetro al constructor) después de haberlo buscado en la base de datos del sistema por medio del servicio de *Windows Communication Foundation*. La definición de la clase *Productos* es la siguiente:

```
/// <summary>
/// Permite obtener desde una fuente de datos
/// las características del producto a agregar
/// </summary>
public class Productos {
    #region Fields
    // Almacena los productos obtenidos desde el servicio WCF
    private List<Producto_Join> productos = new List<Producto_Join>();
    #endregion

    #region Properties
    // Propiedades para esta clase, son iguales en tipo y nombre a las
    // propiedades del origen de datos (servicio WCF)
    public int ProductoId { get; set; }
    public String Producto { get; set; }
    public String ClaveProducto { get; set; }
    public decimal? CostoEstandar { get; set; }
    public decimal PrecioLista { get; set; }
    public String Tamaño { get; set; }
```

```
        public String Peso { get; set; }
        public int UnidadesDisponibles { get; set; }
        public Binary Imagen { get; set; }
        public DateTime? FechaInicioVenta { get; set; }
        public DateTime? FechaFinVenta { get; set; }
        public int CategoriaProductoId { get; set; }
        public String Categoria { get; set; }
        public int ModeloProductoId { get; set; }
        public String Modelo { get; set; }
        public String Descripcion { get; set; }
        public int ColorProductoId { get; set; }
        public String Color { get; set; }
        public DateTime FechaModificacion { get; set; }
        #endregion

        #region Constructor
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="productoId">El id del producto a obtener
        /// desde el origen de datos (Servicio WCF)</param>
        public Productos(int productoId) {
            // Usuario (predeterminado) y certificado que accede al servicio WCF.
            // Se obtienen desde el archivo XML (web.config) por seguridad
            ConnectionStringSettings settingsUser = null;
            settingsUser = ConfigurationManager.ConnectionStrings[
                "SitioWebUserConnectionString"];
            ConnectionStringSettings settingsPass = null;
            settingsPass = ConfigurationManager.ConnectionStrings[
                "SitioWebPassConnectionString"];
            ConnectionStringSettings settingsPathCert;
            settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
            ConnectionStringSettings settingsPassCert;
            settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

            // Conectar con el servicio WCF. Presentar certificados X.509
            // e iniciar sesión
            ClienteWCFShoppingChart.YnnafSoftwareServiceWCFClient cliente =
                new YnnafSoftwareServiceWCFClient();
            cliente.ClientCredentials.ClientCertificate.Certificate =
                new X509Certificate2(settingsPathCert.ConnectionString,
                    settingsPassCert.ConnectionString);
            cliente.IniciarSesion(settingsUser.ConnectionString,
                Encriptacion.EncriptarSha512(settingsPass.ConnectionString));

            // Obtener todos los productos de la base de datos desde el servicio
            // WCF, iterar con ellos y asignar una vez que se encontró el producto las
            // propiedades locales con las propiedades del origen de datos, finalizar
            // sesión y destruir el objeto del servicio WCF.
            productos = cliente.GetProductos().ToList();
            foreach (Producto_Join p in productos) {
                if (p.ProductoId == productoId) {
                    this.ProductoId = productoId;
                    this.Producto = p.Producto;
                    this.ClaveProducto = p.ClaveProducto;
                    this.CostoEstandar = p.CostoEstandar;
                    this.PrecioLista = p.PrecioLista;
                    this.Tamaño = p.Tamaño;
                    this.Peso = p.Peso;
                    this.UnidadesDisponibles = p.UnidadesDisponibles;
                    this.Imagen = p.Imagen;
                    this.FechaInicioVenta = p.FechaInicioVenta;
                    this.FechaFinVenta = p.FechaFinVenta;
                    this.CategoriaProductoId = p.CategoriaProductoId;
                    this.Categoria = p.Categoria;
                    this.ModeloProductoId = p.ModeloProductoId;
                    this.Modelo = p.Modelo;
                    this.Descripcion = p.Descripcion;
                    this.ColorProductoId = p.ColorProductoId;
                    this.Color = p.Color;
                    this.FechaModificacion = p.FechaModificacion;
```

```
                cliente.FinalizarSesion();
                cliente.Close();

                break; // Salir del foreach una vez que se encontró
                       // el producto en la colección (lista genérica)
            }
        }
    }
    #endregion
```

5.4.4.5.2 LA CLASE CARTITEM

Las propiedades de esta clase son iguales a las de la clase *Productos* (a excepción de la propiedad *Cantidad* que mantiene el estado del número de productos a comprar). Esta clase se encargará de mantener la información de los productos que se agregarán al carrito de compra (clase *Shopping-Cart*). La definición de la clase *CartItem* es la siguiente:

```
/// <summary>
/// Permite controlar los productos que se agregan en el carrito de compra
/// </summary>
public class CartItem : IEquatable<CartItem> {
    #region Fields
    private int productoId;
    private Productos producto = null;
    #endregion

    #region Properties y método GetProducto()

    // Propiedades para esta clase, son iguales en tipo y nombre a las
    // propiedades de la clase Productos. Su valor (a excepción de Cantidad)
    // depende del valor de la propiedad equivalente en la clase Productos.
    public int Cantidad { get; set; }

    public int ProductoId {
        get { return this.productoId; }
        set {
            // Asegurar que el correcto funcionamiento del método GetProducto()
            this.producto = null;
            this.productoId = value;
        }
    }

    public Productos GetProducto() {
        // Lazy initialization - el objeto no será creado hasta que sea necesario.
        if (this.producto == null) {
            this.producto = new Productos(this.productoId);
        }
        return this.producto;
    }

    public String Producto { get { return GetProducto().Producto; } }
    public String ClaveProducto { get { return GetProducto().ClaveProducto; } }
    public decimal? CostoEstandar { get { return GetProducto().CostoEstandar; } }
    public String Tamaño { get { return GetProducto().Tamaño; } }
    public String Peso_kg { get { return GetProducto().Peso; } }
    public int UnidadesDisponibles { get { return GetProducto().UnidadesDisponibles; } }
    public Binary Imagen { get { return GetProducto().Imagen; } }
    public DateTime? FechaInicioVenta { get { return GetProducto().FechaInicioVenta; } }
    public DateTime? FechaFinVenta { get { return GetProducto().FechaFinVenta; } }
    public int CategoriaProductoId { get { return GetProducto().CategoriaProductoId; } }
    public String Categoria { get { return GetProducto().Categoria; } }
    public int ModeloProductoId { get { return GetProducto().ModeloProductoId; } }
    public String Modelo { get { return GetProducto().Modelo; } }
    public String Descripcion { get { return GetProducto().Descripcion; } }
    public int ColorProductoId { get { return GetProducto().ColorProductoId; } }
    public String Color { get { return GetProducto().Color; } }
    public DateTime FechaModificacion { get { return GetProducto().FechaModificacion; } }
    public decimal PrecioLista { get { return GetProducto().PrecioLista; } }
```

```
    public decimal Total { get { return PrecioLista * Cantidad; } }
    #endregion

    #region Constructor
    /// <summary>
    /// Constructor CartItem solo necesita un productoId
    /// </summary>
    /// <param name="productoId">El Id del producto</param>
    public CartItem(int productoId) {
        this.ProductoId = productoId;
    }
    #endregion

    #region IEquatable<CartItem> Members
    /// <summary>
    /// Método para implementar la interface IEquatable(T).
    /// Prueba si el item es igual o no al parámetro.
    /// Este método es llamado por el método Contains() de la clase
    /// List(T). Este método se usa en el método AddItem() de ShoppingCart
    /// </summary>
    /// <param name="item">El CartItem a comparar</param>
    /// <returns></returns>
    public bool Equals(CartItem item) {
        return item.ProductoId == this.ProductoId;
    }
    #endregion
}
```

5.4.4.5.3 LA CLASE SHOPPINGCART

Esta es la clase que mantiene el control de los productos que el usuario va agregando o eliminando en el carrito de compra. Esta clase contiene los métodos para agregar productos, eliminar productos, cambiar la cantidad de productos y obtener el subtotal (sin IVA) y total (IVA + Costo de envío) de los productos a comprar. La definición de la clase *ShoppingCart* es la siguiente:

```
/// <summary>
/// Permite controlar los CartItems en el carrito de compra
/// </summary>
public class ShoppingCart {
    #region Fields y Properties

    private List<CartItem> items = new List<CartItem>();

    /// <summary>
    /// Obtiene una lista con los CartItems del carrito de compra
    /// </summary>
    public List<CartItem> Items { get { return this.items; } }

    #endregion

    #region Manejo de sesiones de ASP.NET y constructor privado

    private static ShoppingCart instance = null;

    /// <summary>
    /// Obtiene una instancia inicial o una desde la sesión existente
    /// </summary>
    public static ShoppingCart Instance {
        get {
            // Si el carro no esta en la sesión, crear uno y ponerlo allí.
            // De lo contrario, obtenerlo de la sesión.
            if (HttpContext.Current.Session["YnnafSoftwareShoppingCart"] == null) {
                instance = new ShoppingCart();
                HttpContext.Current.Session["YnnafSoftwareShoppingCart"] = instance;
            } else {
                instance =
                    (ShoppingCart)HttpContext.Current.Session["YnnafSoftwareShoppingCart"];
            }
            return instance;
```

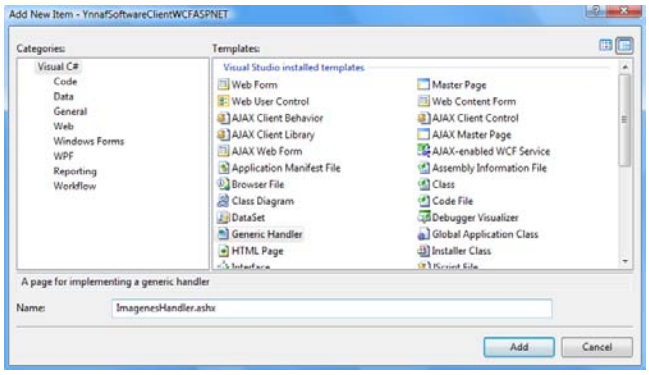
```
        }
    }

    // El constructor privado impide la construcción de la clase
    private ShoppingCart() { }

    #endregion

    #region Métodos para modificación de Item

    /// <summary>
    /// Agrega un item al carrito de compra
    /// </summary>
    /// <param name="productId">El Id del producto</param>
    public void AddItem(int productId) {
        // Crear un nuevo item para agregarlo al carrito
        CartItem newItem = new CartItem(productId);

        // Si el item ya existe en la lista, incrementar la cantidad en 1 (++)
        // de lo contrario, agregar uno nuevo y asignar la cantidad en 1
        if (Items.Contains(newItem)) {
            foreach (CartItem item in Items) {
                if (item.Equals(newItem)) {
                    item.Cantidad++;
                    return;
                }
            }
        } else {
            newItem.Cantidad = 1;
            Items.Add(newItem);
        }
    }


    /// <summary>
    /// Cambia la cantidad de un item en el carrito
    /// </summary>
    /// <param name="productoId">El Id del producto</param>
    /// <param name="cantidad">La cantidad del producto</param>
    public void SetItemQuantity(int productoId, int cantidad) {
        // Si se coloca la cantidad de 0, remover el item completamente
        if (cantidad == 0) {
            RemoveItem(productoId);
            return;
        }

        // Encontrar el item y actualizar la cantidad
        CartItem updatedItem = new CartItem(productoId);

        foreach (CartItem item in Items) {
            if (item.Equals(updatedItem)) {
                item.Cantidad = cantidad;
                return;
            }
        }
    }

    /// <summary>
    /// Remueve un item del carrito de compra
    /// </summary>
    /// <param name="productoId">El Id del producto</param>
    public void RemoveItem(int productoId) {
        CartItem removedItem = new CartItem(productoId);
        Items.Remove(removedItem);
    }
    #endregion

    #region Métodos de reporte

    /// <summary>
    /// Retorna el total (Sin IVA) de todos los items antes
```

```
    /// del IVA, envío, etc.
    /// </summary>
    /// <returns>El subtototal</returns>
    public decimal GetSubTotal() {
        decimal subTotal = 0;
        foreach (CartItem item in Items)
            subTotal += item.Total;

        return subTotal;
    }

    /// <summary>
    /// Retorna el total (con costo de envío e IVA) de todos los items
    /// antes de los impuestos, envío, etc.
    /// </summary>
    /// <param name="costoEnvio">El costo de envío</param>
    /// <returns>El total</returns>
    public decimal GetTotal(decimal costoEnvio) {
        decimal subTotal = 0M;
        decimal iva = 0M;
        decimal total = 0M;
        const decimal IvaPorc = 0.15M;

        // Obtenemos el subtotal sin costo de envío e IVA
        foreach (CartItem item in Items)
            subTotal += item.Total;

        // Sumamos el costo del envío
        subTotal += costoEnvio;
        // Calculamos el IVA
        iva = subTotal * IvaPorc;
        // Sumamos el IVA al subtotal
        total = (subTotal += iva);

        return total;
    }
    #endregion
}
```

NOTA: Estas tres clases se encuentran en un ensamblado llamado (*YnnafSoftwareClientWCFASP-NET_ShoppingCart.dll*) y referenciado en el proyecto principal.

5.4.4.6 CARGA DINÁMICA DE IMÁGENES A PARTIR DEL TIPO BINARY UTILIZADO EN EL SERVICIO WCF

Un control *GridView* de *ASP.NET* permite cargar directamente una imagen siempre y cuando esta sea recibida como un arreglo de bytes, sin embargo en la implementación del servicio *WCF* se usa *Linq to Sql* y la imagen está contenida en un tipo *System.Data.Linq.Binary* Por lo tanto para poder cargar la imagen dinámicamente se necesita crear un *Http Handler*. Esto lo hacemos agregando un *Generic Handler* a nuestro proyecto de la siguiente forma: *Project -> New Item… -> Generic Handler -> Name* = "*ImagenesHandler.ashx*".



Inserción de Manejador genérico.

Una vez que agregamos el manejador genérico, creamos una clase que implemente las interfaces *IHttpHandler*. Para la primera interfaz se debe implementar tanto el método *void ProcessRequest(HttpContext context)*[22] como la propiedad *bool IsReusable*[23].

```
public class ImagenesHandler : IHttpHandler {
    /// <summary>
    /// Para almacenar los productos del Servicio WCF
    /// </summary>
    private List<Producto_Join> productos = new List<Producto_Join>();

    public void ProcessRequest(HttpContext context) {
        // Usuario (predeterminado) y certificado que accede al servicio WCF.
        ConnectionStringSettings settingsUser = null;
        settingsUser = ConfigurationManager.ConnectionStrings[
            "SitioWebUserConnectionString"];
        ConnectionStringSettings settingsPass = null;
        settingsPass = ConfigurationManager.ConnectionStrings[
            "SitioWebPassConnectionString"];
        ConnectionStringSettings settingsPathCert;
        settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
        ConnectionStringSettings settingsPassCert;
        settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

        // Conectar con el servicio WCF. Presentar certificados.
        YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient
            cliente = new
            YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
        cliente.ClientCredentials.ClientCertificate.Certificate =
            new X509Certificate2(settingsPathCert.ConnectionString,
                settingsPassCert.ConnectionString);

        // Iniciar sesión.
        cliente.IniciarSesion(settingsUser.ConnectionString,
            Encriptacion.EncriptarSha512(settingsPass.ConnectionString));

        // Obtener el Id del producto desde el Request
        int imagenProductoId =
            Convert.ToInt32(context.Request.QueryString["ProductoId"]);

        // Obtener la lista de productos desde el servicio WCF y obtener
        // las características del producto.
        productos = cliente.GetProductos().ToList();
        Producto_Join pr = productos.Find(p => p.ProductoId == imagenProductoId);

        // Finalizar la sesión y el objeto
        cliente.FinalizarSesion();
        cliente.Close();
```

[22] IHttpHandler.ProcessRequest(HttpContext context): Permite el procesamiento de solicitudes web *HTTP* mediante un *HttpHandler* personalizado que implementa la interfaz *IHttpHandler*.
[23] IHttpHandler.IsReusable: Obtiene un valor que indica si otra solicitud puede utilizar la instancia de *IHttpHandler*.

```
        // Escribir en la Response el arreglo de bytes que contiene la imagen
        context.Response.ContentType = "image/jpeg";
        context.Response.BinaryWrite(pr.Imagen.ToArray());
    }

    public bool IsReusable { get { return true; } }
}
```

En resumen, lo que esta clase es: *1)* Establecer comunicación con el servicio WCF, *2)* Obtener el Id del producto desde *Request*, *2)* Obtener la lista de todos los productos, *3)* Buscar en la lista utilizando expresiones lambda el producto y *4)* Escribir en la respuesta del contexto de *HTTP* el arreglo de bytes de la imagen obtenida desde la clase *System.Data.Linq.Binary*.

El código *XML* de este *HTTP Handler* debe ser el siguiente:

```
<%@ WebHandler Language="C#" CodeBehind="ImagenesHandler.ashx.cs"
Class="ImagenesHandler" %>
```

5.4.4.7 LLAMADAS A CONTRATOS DE OPERACIÓN DEL SERVICIO WCF

A continuación se muestra el código fuente utilizado para llamar a los contratos de operación del servicio WCF necesarios para la ejecución del programa:

5.4.4.7.1 LOGIN

Si el cliente necesita finalizar su orden de compra necesitará forzosamente obtener una autenticación, esto lo hará vía la página *~/login.aspx*. El botón que envía los datos de autenticación del usuario dispara 2 eventos: *OnClientClick* y *OnClick*. El primero genera una cookie en *JavaScript* con el nombre de usuario y password (encriptado en *SHA-512*) y el segundo otorga autorización de acceso a la página vía código que se ejecuta en servidor. El siguiente fragmento de código muestra como se deben declara estos dos eventos en un control de botón de *ASP.NET*.

```
<!-- Otro código fuente en HTML -->
<asp:Button ID="IngresarButton" runat="server" Style="text-align: center;"
            Text="Ingresar" OnClientClick="setCookie(
                    'YnnafSotfwareCookie ' +
                    document.getElementById('UserNameTextBox').value,
                    'user=' + document.getElementById('UserNameTextBox').value + '&' +
                    'pass=' +
                    encriptarSha512(document.getElementById('PasswordTextBox')), 1)"
            OnClick="IngresarButton_Click" />
<!-- Otro código fuente en HTML -->
```

Nota: El evento *OnClick* se ejecuta antes que el evento *OnClientClick*.

El siguiente código fuente de *JavaScript* encripta contiene los métodos para encriptar una cadena en *SHA-512* e insertar la cookie en el navegador del cliente:

```
/// <reference path="jsSHA/sha.js" />
// sha.js es un script Open Source para encriptar cadenas en SHA 1 y SHA 2

/**
* Encripta el valor (value) del control pasado como parámetro
* @param {Element} control El control que contiene el valor
*/
function encriptarSha512(control) {
    var shaObj = new jsSHA(control.value);
    var hash = shaObj.getHash("SHA-512", "B64"); // Secure Hash Algorithm (SHA-512)
                                                 // y codificado en BASE 64
    hash += "=="; //Necesario para compatibilidad con .NET Framework
```

```
    return hash;
}

/**
* Coloca la cookie en el navegador del cliente
* @param {String} nombreCookie El nombre de la cookie
* @param {String} valor El valor de la cookie
* @param {Number} diasExpira Días en que expira la cookie
*/
function setCookie(nombreCookie, valor, diasExpira) {
    var fechaExpiracion = new Date();
    fechaExpiracion.setDate(fechaExpiracion.getDate() + diasExpira);
    document.cookie = nombreCookie + "=" + valor +
    ";expires=" + fechaExpiracion.toGMTString();
}
```

El siguiente código fuente es el que se ejecuta al disparar el evento *OnClick*. En este código se conecta al servicio *WCF* a través del contrato de operación *IniciarSesion(username, password)*. Si el nombre de usuario y contraseña son correctos y no se produce una excepción, el cliente tendrá acceso a las páginas que antes no pues ahora está autenticado, de lo contrario se caerá en el bloque *catch* que mostrará un mensaje indicando que el nombre de usuario o contraseña no se encontró y se eliminará la cookie que se insertó con los datos erróneos. Partiendo del hecho que el código *HTML* que mantiene la vista contiene 2 controles *asp:TextBox* para insertar el nombre y contraseña del usuario, el código fuente siguiente (*CodeBehind*) en *C#* muestra cómo se realiza el procedimiento descrito anteriormente al pulsar el botón *IngresarButton* de ingreso:

```
protected void IngresarButton_Click(object sender, EventArgs e) {
    String username = this.UserNameTextBox.Text;
    ClienteWCF.YnnafSoftwareServiceWCFClient cliente = null;
    try {
        // Obtener el nombre del usuario
        String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
            User.Identity.Name : null;

        // Obtener la cookie del usuario (Creada previamente en JavaScript)
        String nombreCookie = "YnnafSotfwareCookie_" + nombreUsuario;
        HttpCookie httpCookie = Request.Cookies.Get(nombreCookie);

        // Usuario (predeterminado si no se está autenticado) y certificado
        // que accede al servicio WCF.
        ConnectionStringSettings settingsUser = null;
        settingsUser = ConfigurationManager.ConnectionStrings[
            "SitioWebUserConnectionString"];
        ConnectionStringSettings settingsPass = null;
        settingsPass = ConfigurationManager.ConnectionStrings[
            "SitioWebPassConnectionString"];
        ConnectionStringSettings settingsPathCert;
        settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
        ConnectionStringSettings settingsPassCert;
        settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

        // Conectar con el servicio WCF. Presentar certificados.
        cliente = new
            YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
        cliente.ClientCredentials.ClientCertificate.Certificate =
            new X509Certificate2(settingsPathCert.ConnectionString,
                settingsPassCert.ConnectionString);

        // Iniciar sesión. El usuario con el que se iniciará sesión con el servicio WCF
        // dependerá de que exista o no la cookie en su PC.
        if (httpCookie == null) {
            cliente.IniciarSesion(this.UserNameTextBox.Text,
                Encriptacion.EncriptarSha512(this.PasswordTextBox.Text));
            // Otorgar autenticación al usuario
            FormsAuthentication.RedirectFromLoginPage(this.UserNameTextBox.Text, false);
        } else {
```

```
            cliente.IniciarSesion(httpCookie.Values["user"], httpCookie.Values["pass"]);
        }
    }
    catch (Exception) {
        this.ErrorLabel.Text = "No se encontró su nombre de usuario o contraseña";
        try {
            // Remover la cookie insertada por JavaScript
            Response.Cookies.Remove("YnnafSotfwareCookie_" + username);
        }
        catch (Exception) { }
    }
    // Finalizar la sesión y el objeto
    cliente.FinalizarSesion();
    cliente.Close();
}
```

5.4.4.7.2 OBTENCIÓN DE PRODUCTOS

La obtención y visualización de los productos que contiene la empresa se hará en la página *~/default.aspx*. Lo anterior se realiza en dos pasos: *1)* Generar el código *HTML* (apariencia de la página) y *2)* Generar el código *C#* que provee los datos al código *HTML*.

5.4.4.7.2.1 ENLACE DE DATOS EN GRIDVIEW

ASP.NET provee controles *HTML* que facilitan la manipulación de datos. El control *GridView (asp:GridVew)* permite que de una manera fácil se puedan mostrar datos a un usuario como si se tratara de una Tabla (*<table>*) de *HTML*. El siguiente código permite definir un *GridView* que mostrará los datos asignados a su propiedad *DataSource* vía *C#*.

```
<!-- Otro código fuente en HTML -->
<asp:GridView runat="server" ID="ProductosGridView" CssClass="GridViewStyle"
    RowStyle-CssClass="GridViewRowStyle" PagerStyle-CssClass="GridViewPagerStyle"
    AlternatingRowStyle-CssClass="GridViewAlternatingRowStyle"
    HeaderStyle-CssClass="GridViewHeaderStyle" AllowPaging="True"
    OnPageIndexChanging="ProductosGridView_PageIndexChanging"
    AutoGenerateColumns="False" GridLines="None"
    OnRowCommand="ProductosGridView_RowCommand" HorizontalAlign="Center">
    <Columns>
        <asp:TemplateField HeaderText="Imagen" ItemStyle-Width="100px">
            <ItemTemplate>
                <asp:Image runat="server" ImageUrl='<%#
                    "ImagenesHandler.ashx?ProductoId=" + Eval("ProductoId") %>' />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:BoundField DataField="Producto" HeaderText="Nombre"
            ItemStyle-Width="150px">
        </asp:BoundField>
        <asp:BoundField DataField="PrecioLista" HeaderText="Precio"
            ItemStyle-Width="100px" DataFormatString="{0:C}">
        </asp:BoundField>
        <asp:BoundField DataField="Categoria" HeaderText="Categoria"
            ItemStyle-Width="100px">
        </asp:BoundField>
        <asp:BoundField DataField="Modelo" HeaderText="Modelo"
            ItemStyle-Width="100px">
        </asp:BoundField>
        <asp:BoundField DataField="Color" HeaderText="Color" ItemStyle-Width="100px">
        </asp:BoundField>
        <asp:TemplateField HeaderText="Detalles" ItemStyle-Width="100px"
            ItemStyle-HorizontalAlign="Center">
            <ItemTemplate>
                <a href="javascript:VerDetalles('<%# Eval("ProductoId") %>')">
                    Ver más</a>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Carrito" ItemStyle-Width="100px"
```

```
            ItemStyle-HorizontalAlign="Center">
            <ItemTemplate>
                <asp:LinkButton ID="AgregarLinkButton" runat="server" Text="Agregar"
                  CommandName="Agregar" CommandArgument='<%# Eval("ProductoId") %>' />
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
<!-- Otro código fuente en HTML -->
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *GridView* que servirá para mostrar información sobre productos obtenida desde el código de *C#* y *2)* Definición de las columnas, generalmente la propiedad *DataField* permite visualizar los datos al usuario. Por ejemplo, para mostrar el nombre del producto, la propiedad *DataField* del control *asp:BoundField* está enlazada con la propiedad *Producto* de la lista genérica que sirve como fuente de datos.

Método Page_Load

El siguiente método se ejecuta al iniciar la carga de la página (*~/default.aspx*) y es el que se encarga de obtener los datos que se muestran en el control *GridView*. A continuación se muestra la definición del método:

```
protected void Page_Load(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;

    // Obtner la cookie del usuario
    String nombreCookie = "YnnafSotfwareCookie_" + nombreUsuario;
    HttpCookie httpCookie = Request.Cookies.Get(nombreCookie);

    // Usuario (predeterminado si no se está autenticado) y certificado
    // que accede al servicio WCF.
    ConnectionStringSettings settingsUser = null;
    settingsUser = ConfigurationManager.ConnectionStrings["SitioWebUserConnectionString"];
    ConnectionStringSettings settingsPass = null;
    settingsPass = ConfigurationManager.ConnectionStrings["SitioWebPassConnectionString"];
    ConnectionStringSettings settingsPathCert;
    settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
    ConnectionStringSettings settingsPassCert;
    settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

    // Conectar con el servicio WCF. Presentar certificados.
    ClienteWCF.YnnafSoftwareServiceWCFClient cliente =
        new YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
    cliente.ClientCredentials.ClientCertificate.Certificate =
        new X509Certificate2(settingsPathCert.ConnectionString,
            settingsPassCert.ConnectionString);

    // Iniciar sesión. El usuario con el que se conectará dependerá de
    // que exista o no la cookie en su PC
    if (httpCookie == null) {
        cliente.IniciarSesion(settingsUser.ConnectionString,
            Encriptacion.EncriptarSha512(settingsPass.ConnectionString));
    } else {
        cliente.IniciarSesion(httpCookie.Values["user"], httpCookie.Values["pass"]);
    }

    // Buscar si el usuario está autenticado ante la página.
    this.BienvenidoLabel.Text = !String.IsNullOrEmpty(nombreUsuario) ? "Bienvenido: " +
        nombreUsuario : "Bienvenido: Anónimo";

    // Habilitar o inhabilitar el Logout
    this.LogoutLinkButton.Enabled = !String.IsNullOrEmpty(nombreUsuario) ? true : false;
```

```
    // Obtener todos los productos
    this.productos = cliente.GetProductos().ToList();

    // Mostrar los productos en el GridView
    this.ProductosGridView.DataSource = productos;
    this.ProductosGridView.DataBind();

    // Finalizar la sesión y el objeto
    cliente.FinalizarSesion();
    cliente.Close();
}
```

En resumen, lo que hace el método es el siguiente: *1)* Obtener el nombre del usuario (depende de la autenticación previa del usuario), *2)* Obtener la *cookie* del usuario, *3)* Conectar al Servicio *WCF* iniciando sesión con el usuario adecuado (depende de la existencia de la cookie), *4)* Mostrar un mensaje de bienvenida al usuario, *5)* Habiitar o deshabilitar el botón de logout (depende de la autenticación del usuario), *6)* Obtener los productos desde el servicio *WCF*, *7)* Enlazar los datos en el *GridView* y *8)* Finalizar la sesión y destruir el servicio con el servicio *WCF*.

5.4.4.7.2.2 MÉTODOS AUXILIARES EN GRIDVIEW

Los dos métodos que se indican a continuación sirven para agregar funcionalidades al GridView o para optimizar su funcionamiento:

Método OnRowCommand (ProductosGridView_RowCommand)

Este método está íntimamente ligado con la columna *Carrito* del *GridView* código *HTML*. Cuando se presiona cualquiera de los *LinkButton* contenidos en la columna, este método recibe el Id del producto (como se indicó en el código *HTML* en la propiedad *CommandArgument*), agrega el producto al carrito de compra (clase *ShoppingCart*) y redirecciona al usuario a la página que muestra la información de su carrito de compra (*~/carrito.aspx*). A continuación se muestra la definición del método:

```
protected void ProductosGridView_RowCommand(object sender,
    GridViewCommandEventArgs e) {
    ShoppingCart sc = null;
    if (e.CommandName == "Agregar") {
        // Asignar la referencia y agregar el producto
        sc = ShoppingCart.Instance;
        sc.AddItem(Convert.ToInt32(e.CommandArgument));

        // Redirigir a la página del carrito de compra
        Response.Redirect("carrito.aspx");
    }
}
```

Método OnPageIndexChanging (ProductosGridView_PageIndexChanging)

Este método está ligado al a la propiedad *AllowPaging=”True”* del *GridView*. Cada vez que se da clic en la paginación del *GridView* (en su parte inferior) se muestra una nueva cantidad de productos al usuario (en intervalos de 10 productos). La propiedad *PageIndex* debe estar asignada a la propiedad *NewPageIndex* del evento *GridViewPageEventArgs*, una vez que se hizo la asignación, se debe volver a enlazar los datos al *GridView*. A continuación se muestra la definición del método:

```
protected void ProductosGridView_PageIndexChanging(object sender,
    GridViewPageEventArgs e) {
    // Código necesario para permitir la paginación del GridView
    this.ProductosGridView.PageIndex = e.NewPageIndex;
    this.ProductosGridView.DataBind();
}
```

5.4.4.7.3 REGISTRO

Si el cliente necesita finalizar su orden de compra necesitará forzosamente estar registrado en el sistema de la empresa, esto lo hará vía la página *~/registro.aspx*. Para que un usuario pueda registrarse en la base de datos de la empresa como un cliente, se debe llamar al contrato de operación *Sp_CreateClienteASPNET* del servicio *WCF*. Partiendo del hecho que el código *HTML* que mantiene la vista contiene varios controles *<asp:TextBox>* y *<asp:DropDownList>* para recibir los datos del usuario (controlando a través de *AJAX* la validación de los mismos), el código fuente siguiente (*CodeBehind*) en *C#* muestra cómo se registra un usuario al pulsar el botón *RegistrarButton*:

```
protected void RegistrarButton_Click(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;

    // Usuario (predeterminado si no se está autenticado) y certificado
    // que accede al servicio WCF.
    ConnectionStringSettings settingsUser = null;
    settingsUser = ConfigurationManager.ConnectionStrings["SitioWebUserConnectionString"];
    ConnectionStringSettings settingsPass = null;
    settingsPass = ConfigurationManager.ConnectionStrings["SitioWebPassConnectionString"];
    ConnectionStringSettings settingsPathCert;
    settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
    ConnectionStringSettings settingsPassCert;
    settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

    // Conectar con el servicio WCF. Presentar certificados.
    ClienteWCF.YnnafSoftwareServiceWCFClient cliente =
        new YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
    cliente.ClientCredentials.ClientCertificate.Certificate =
        new X509Certificate2(settingsPathCert.ConnectionString,
            settingsPassCert.ConnectionString);

    // Iniciar sesión. El usuario con el que se conectará es el predeterminado
    cliente.IniciarSesion(settingsUser.ConnectionString,
        Encriptacion.EncriptarSha512(settingsPass.ConnectionString));

    // Variables que almacenan el resultado del registro
    int existeUserName = 0;
    int numeroError = 0;
    cliente.Sp_CreateClienteASPNET(new Cliente {
        // Propiedades de la clase Cliente (asignadas a los valores de los controles)
        Nombre_o_Sucursal = this.NombreTextBox.Text,
        ApellidoPaterno_o_NumSucursal = this.ApellidoPaternoTextBox.Text,
        ApellidoMaterno = this.ApellidoMaternoTextBox.Text,
        Username = this.UserNameTextBox.Text,
        Password = this.PasswordTextBox.Text,
        FechaAlta = new DateTime(),
        TipoClienteId = 3, // Tipo de cliente de sitio web
        Calle = this.CalleTextBox.Text,
        Numero = this.NumeroTextBox.Text,
        Ciudad = this.CiudadTextBox.Text,
        Estado = this.EstadoTextBox.Text,
        Pais = this.PaisDropDownList.Text,
        CodigoPostal = this.CodigoPostalTextBox.Text,
        Telefono = this.TelefonoTextBox.Text,
        Fax = this.FaxTextBox.Text,
        Email = this.EmailTextBox.Text,
        FechaModificacion = new DateTime()
    },
        ref existeUserName, ref numeroError);

    // Finalizar la sesión y el objeto
    cliente.FinalizarSesion();
    cliente.Close();

    if (existeUserName == 1) { // El nombre de usuario existe en la base de datos
        this.ErrorLabel.Text = "Lo sentimos, el nombre de usuario ya existe en" +
```

```
                "nuestra base de datos intente con otro";
            this.ErrorLabel.ForeColor = Color.Red;
            this.UserNameTextBox.BackColor = Color.Red;
            this.UserNameTextBox.BorderColor = Color.Yellow;
        }
        else if (existeUserName == 2) { // Ocurrió un error diferente
            this.ErrorLabel.Text = "Un error ocurrió al tratar de registrarlo, " +
                "por favor contáctenos para resolver el problema";
        } else {
            // Se le da autenticación directa al usuario si la variable
            // existeUsername es diferente de 1 ó 2 (cero)
            FormsAuthentication.RedirectFromLoginPage(this.UserNameTextBox.Text, false);
        }
    }
```

En resumen, lo que hace el método es el siguiente: *1)* Verificar que las dos contraseñas introducidas por el usuario sean iguales, si no son iguales se muestra al usuario un *Alert* de *JavaScript* indicándoselo, *2)* Llamar al método *Sp_CreateClienteASPNET* y pasarle los parámetros necesarios, *3)* Verificar que no haya ocurrido un error al crear el usuario (el nombre de usuario ya existe o error desconocido) y *4)* Si no ocurrió ningún error, realizar la autenticación vía formularios para que el cliente pueda realizar compras sin restricciones.

5.4.4.7.4 CARRITO DE COMPRA

El cliente necesita mantener un control sobre los productos que desea comprar, esto se hará a través de su carrito de compra por medio de la página ~*carrito.aspx* y ~*compra\carrito2.aspx*

5.4.4.8.4.1 ENLACE DE DATOS EN GRIDVIEW

ASP.NET provee controles *HTML* que facilitan la manipulación de datos. El control *GridView (asp:GridVew)* permite que de una manera fácil se puedan mostrar datos a un usuario como si se tratara de una Tabla (*<table>*) de *HTML*. El siguiente código permite definir un *GridView* que mostrará los datos asignados a su propiedad *DataSource* vía *C#*.

```
<!-- Otro código fuente en HTML -->
<asp:GridView runat="server" ID="ShoppingCartGridView"
    CssClass="GridViewStyle" RowStyle-CssClass="GridViewRowStyle"
    PagerStyle-CssClass="GridViewPagerStyle" AlternatingRowStyle-
    CssClass="GridViewAlternatingRowStyle"
    FooterStyle-CssClass="GridViewPagerStyle"
    HeaderStyle-CssClass="GridViewHeaderStyle"
    AllowPaging="False" AutoGenerateColumns="false" GridLines="None"
    ShowFooter="true" DataKeyNames="ProductId"
    EmptyDataText="No hay ningún producto en tu carrito de compra."
    OnRowDataBound="ShoppingCartGridView_RowDataBound"
    OnRowCommand="ShoppingCartGridView_RowCommand"
    HorizontalAlign="Center">
    <Columns>
        <asp:BoundField DataField="Producto" HeaderText="Nombre"
            ItemStyle-Width="150px" />
        <asp:TemplateField HeaderText="Imagen" ItemStyle-Width="120px">
            <ItemTemplate>
                <asp:Image ID="Image1" runat="server"
                    ImageUrl='<%# "ImagenesHandler.ashx?ProductoId=" +
                        Eval("ProductoId") %>' />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Cantidad" ItemStyle-Width="100px">
            <ItemTemplate>
                <asp:TextBox runat="server" ID="CantidadTextBox" Columns="5"
                   Text='<%# Eval("Cantidad") %>'></asp:TextBox><br />
                <asp:LinkButton runat="server" ID="RemoverLinkButton"
                   Text="Remover" CommandName="Remover"
                   CommandArgument='<%# Eval("ProductoId") %>'
                   Style="font-size: 12px;"></asp:LinkButton>
```

```
            </ItemTemplate>
        </asp:TemplateField>
        <asp:BoundField DataField="PrecioLista" HeaderText="Precio"
            ItemStyle-HorizontalAlign="Right"
            ItemStyle-Width="100px" HeaderStyle-HorizontalAlign="Right"
            DataFormatString="{0:C}" />
        <asp:BoundField DataField="Total" HeaderText="Subtotal"
            ItemStyle-HorizontalAlign="Right"
            ItemStyle-Width="100px" HeaderStyle-HorizontalAlign="Right"
            DataFormatString="{0:C}" />
    </Columns>
</asp:GridView>
<table style="width: 400px; background-color: White;" align="center">
    <tr>
        <td style="text-align: center">
            <asp:Button runat="server"
                ID="UpdateShoppingCartButton" Text="Actualizar carrito"
                OnClick="UpdateShoppingCartButton_Click" />
        </td>
    </tr>
</table>
<!-- Otro código fuente en HTML -->
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *GridView* que servirá para mostrar información sobre los productos agregados al carrito de compra a través del código de *C#, 2)* Definición de las columnas, generalmente la propiedad *DataField* permite visualizar los datos al usuario. Por ejemplo, para mostrar el precio de lista del producto, la propiedad *DataField* del control *asp:BoundField* está enlazada con la propiedad *PrecioLista* de la lista genérica que sirve como fuente de datos y *3)* Definición del botón que permite actualizar el carrito de compra con los nuevos valores de los productos a comprar.

Método Page_Load

Este es el método que se ejecuta al iniciar la carga de la página del carrito de compra, se encarga de llenar con datos el *GridView* de la página. A continuación se muestra la definición del método:

```
protected void Page_Load(object sender, EventArgs e){
    if (!IsPostBack) // Si se carga en respuesta a una redirección
        BindData();
}
```

Lo que hace el método anterior es lo siguiente: *1)* Al cargar la página (las veces que sean necesarias en respuesta a una redirección) llama al método *BindData*, el cual se explica a continuación:

```
protected void BindData() {
    // Enlazar los datos al GridView
    // El GridView tomará nuestros cart items uno por uno y usará las
    // propiedades que declaramos como nombres de columnas (DataFields)
    ShoppingCartGridView.DataSource = ShoppingCart.Instance.Items;
    ShoppingCartGridView.DataBind();
}
```

La función del método anterior es la siguiente: *1)* Asignar a la propiedad *DataSource* del *GridView* la colección genérica que mantiene todos los productos del carrito de compra y *2)* Mostrar al usuario estos productos en el *GridView.*

5.4.4.7.4.2 MÉTODOS AUXILIARES EN GRIDVIEW

Los dos métodos que se indican a continuación sirven para agregar funcionalidades al *GridView* o para optimizar su funcionamiento:

Método OnRowCommand (ProductosGridView_RowCommand)

Este método está íntimamente ligado con la columna *Cantidad* del *GridView* del código *HTML*. Cuando se presiona cualquiera de los *LinkButton* contenidos en esta columna, este método recibe el *Id* del producto (como se indicó en el código *HTML* en la propiedad *CommandArgument*), elimina el producto del carrito de compra y reenlaza los datos del *GridView* para mostrar los datos actuales. A continuación se muestra la definición del método:

```
protected void ShoppingCartGridView_RowCommand(object sender,
    GridViewCommandEventArgs e) {
    ShoppingCart sc = null;
    if (e.CommandName == "Remover") {
        int productId = Convert.ToInt32(e.CommandArgument);
        sc = ShoppingCart.Instance;
        sc.RemoveItem(productId);
    }

    // Tenemos que reiniciar los datos en el GridView, esto no
    // muestra los datos antiguos.
    BindData();
}
```

Método OnRowDataBound (ShoppingCartGridView_RowDataBound)

Este método se ejecuta cada vez que se enlazan o reenlazan los datos en el *GridView* (al llamar al método *GridView.DataBind()*). La funcionalidad de este método es la de mostrar el total a pagar cada vez que se actualiza el contenido del carrito de compra.

```
protected void ShoppingCartGridView_RowDataBound(object sender,
    GridViewRowEventArgs e) {
    ShoppingCart sc = null;
    // Si estamos enlazando el renglón footer, agregamos aquí el total
    if (e.Row.RowType == DataControlRowType.Footer) {
        sc = ShoppingCart.Instance;
        e.Row.Cells[4].Text = "Total (Sin IVA): " + sc.GetSubTotal().ToString("C");
    }
}
```

Actualización del carrito de compra

Este es el método que se ejecuta al presionar el botón (*UpdateShoppingCartButton*) de actualización de los productos del carrito de compra. A continuación se muestra la definición del método:

```
protected void UpdateShoppingCartButton_Click(object sender, EventArgs e) {
    ShoppingCart sc = null;
    foreach (GridViewRow row in this.ShoppingCartGridView.Rows) {
        if (row.RowType == DataControlRowType.DataRow) {
            // Usaremos un bloque try catch en caso de que se haya tipado algo
            // diferente a un número. Si es así, solo lo ignoraremos.
            try {
                // Obtener el productoId desde el datakeys del GridView
                int productId = Convert.ToInt32(
                    this.ShoppingCartGridView.DataKeys[row.RowIndex].Value);
                // Encontrar el TextBox de la Cantidad y recibir el valor
                int quantity = int.Parse(((TextBox)row.Cells[1].FindControl(
                    "CantidadTextBox")).Text);

                // Establecer la nueva cantidad de productos al Id del producto
                sc = ShoppingCart.Instance;
                sc.SetItemQuantity(productId, quantity);
            }
            catch (FormatException) { }
        }
    }
    BindData(); //  Revincular datos
}
```

Este método ejecuta los siguientes procesos: *1)* Obtener el *Id* de cada uno de los productos almacenados en el *GridView*, *2)* Obtener la nueva cantidad de productos deseada por el cliente, *3)* Actualizar los miembros de la clase *ShoppingCart* con los nuevos valores y *4)* Una vez finalizada la iteración, mostrar los datos actualizados en el *GridView*.

Finalización de la orden de compra

Una vez que se decidió completar la orden de compra, se deberá dar clic en el vínculo *Finalizar compra >>* ubicado en la parte de debajo de la página *~/carrito.aspx.* Este vínculo llevará al cliente a la página *~/compra/carrito2.aspx,* a través de esta página se finalizará la orden de compra. Esta página no es muy diferente a la página *~/carrito.aspx* en su funcionamiento. Sin embargo, aquí es donde el cliente escoge la forma de pago y el método de envío de su orden de compra.

Una vez que el cliente seleccionó la forma de envío y pago, acepta los términos (cuánto va a pagar en total), y da clic en el botón *Comprar*, se deberá llamar al método *Sp_CreateOrden* del servicio *WCF* para levantar su pedido. El código siguiente muestra los procesos que se ejecutan:

```
protected void ComprarButton_Click(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;

    // Obtner la cookie del usuario
    String nombreCookie = "YnnafSotfwareCookie_" + nombreUsuario;
    HttpCookie httpCookie = Request.Cookies.Get(nombreCookie);

    // Usuario (predeterminado si no se está autenticado) y certificado
    // que accede al servicio WCF.
    ConnectionStringSettings settingsUser = null;
    settingsUser = ConfigurationManager.ConnectionStrings["SitioWebUserConnectionString"];
    ConnectionStringSettings settingsPass = null;
    settingsPass = ConfigurationManager.ConnectionStrings["SitioWebPassConnectionString"];
    ConnectionStringSettings settingsPathCert;
    settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
    ConnectionStringSettings settingsPassCert;
    settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

    // Conectar con el servicio WCF. Presentar certificados.
    ClienteWCF.YnnafSoftwareServiceWCFClient cliente =
        new YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
    cliente.ClientCredentials.ClientCertificate.Certificate =
        new X509Certificate2(settingsPathCert.ConnectionString,
            settingsPassCert.ConnectionString);

    // Verificar datos correctos
    // Verificar que haya productos a comprar, sino mostrar un mensaje de error
    if (ShoppingCart.Instance.Items.Count == 0) {
        this.ErrorLabel.Text = "No hay ningún producto que comprar";
        return; // Salir del método
    }

    // Verificar que se haya seleccionado una forma de pago, sino mostrar el error
    if (this.FormaPagoDropDownList.SelectedValue == Convert.ToString(0)) {
        this.ErrorLabel.Text = "Debe seleccionar una forma de pago";
        return; // Salir del método
    } else if (this.FormaPagoDropDownList.SelectedValue == Convert.ToString(2) ||
        this.FormaPagoDropDownList.SelectedValue == Convert.ToString(3) ||
        this.FormaPagoDropDownList.SelectedValue == Convert.ToString(4))
    {
    // Verificar que se haya insertado un número de cuenta si la forma de pago
    // seleccionada corresponde a los índices 2, 3 ó 4, sino mostrar un mensaje de error
        if (String.IsNullOrEmpty(this.NumeroCuentaTextBox.Text)) {
            this.ErrorLabel.Text = "Debe insertar su número de tarjeta / cheque";
```

```
            return; // Salir del método
        }
    }
    // Verificar que se haya seleccionado un método de envío, sino mostrar el error
    if (this.MetodoEnvioDropDownList.SelectedValue == Convert.ToString(0)) {
        this.ErrorLabel.Text = "Debe seleccionar un método de envío";
        return; // Salir del método
    }

    // Obtener los productos desde la clase ShoppingCart
    List<CartItem> productosCart = ShoppingCart.Instance.Items;
    String listaParametros = null;
    String descripcionResultado = null;
    int i = 0;
    int ordenId = 0;
    int resultado = 0;

    // Generar el patrón {cantidad, productoId, ... n) para el procedimiento almacenado
    foreach (CartItem cartItem in productosCart) {
        listaParametros += cartItem.Cantidad.ToString() + ", " +
            cartItem.ProductoId.ToString();
        if (i < productosCart.Count - 1)
            listaParametros += ", ";
        i++;
    }

    // Iniciar sesión. El usuario con el que se conectará dependerá de
    // que exista o no la cookie en su PC
    if (httpCookie == null) {
        cliente.IniciarSesion(settingsUser.ConnectionString,
            Encriptacion.EncriptarSha512(settingsPass.ConnectionString));
    } else {
        cliente.IniciarSesion(httpCookie.Values["user"], httpCookie.Values["pass"]);
    }

    // Crear la orden de compra
    resultado = cliente.Sp_CreateOrden(new Orden {
        FechaOrden = DateTime.Now,
        FechaAtencion = DateTime.Now,
        FechaEnvio = DateTime.Now,
        FechaEntrega = DateTime.Now,
        NumeroCuenta = this.NumeroCuentaTextBox.Text,
        Iva = 0.15M,
        CargoExtra = Convert.ToDecimal(this.CostoTextBox.Text.Replace("$", "")),
        ClienteId = 100,
        StatusOrdenId = 1,
        MetodoEnvioOrdenId = Convert.ToInt32(this.MetodoEnvioDropDownList.SelectedValue),
        FormaPagoOrdenId = Convert.ToInt32(this.FormaPagoDropDownList.SelectedValue),
        FechaModificacion = DateTime.Now
    }, listaParametros, ref ordenId, ref descripcionResultado);

    if (resultado == 1) {
        // Mantener en una sesión el resultado exitoso de la orden y redirigir
        // a la página compraSatisfactoria.aspx para mostrarlo al usuario
        HttpContext.Current.Session.Add("YnnafCompraSatisfactoria",
            "Orden con el número " + ordenId.ToString() + " creada exitosamente");
        Response.Redirect("compraSatisfactoria.aspx");
    } else {
        this.ErrorLabel.Text = descripcionResultado; // Mostrar el error
    }

    // Finalizar la sesión y el objeto
    cliente.FinalizarSesion();
    cliente.Close();
}
```

En resumen, los pasos que realiza el código anterior son los siguientes: *1)* Verificar que haya productos a comprar y que la forma de pago y método de envío se hayan llenado correctamente, *2)* Generar el patrón {*Cantidad, ProductoId_1, Cantidad, Producto Id_2, …*}, como lo solicita el servi-

cio *WCF 3)* Conectar con el servicio WCF (con el usuario adecuado) y llamar al contrato de operación *Sp_CreateOrden* pasándole los parámetros requeridos y *4)* Verificar si se efectúo correctamente la creación de la orden, si no fue así, mostrar el mensaje de error al cliente. De lo contrario, almacenar en una sesión los datos del número de orden creada y redireccionar a la página *~/compra/compraSatisfactoria.aspx* para mostrar al cliente los datos de su orden de compra.

5.4.4.7.5 SEGUIMIENTO DE ORDEN

Para hacer seguimientos de las órdenes hechas por un cliente vía sitio web, se debe hacer una búsqueda de acuerdo al número del cliente (asignado en la base de datos y controlado por el servicio *WCF*). La página mostrará *TODAS* las órdenes que el cliente que solicitó el seguimiento ha efectuado con la empresa desde que se dio de alta. El código siguiente (*HTML* y *C#*) muestra como se hace el procedimiento:

5.4.4.7.5.1 ENLACE DE DATOS PARA ORDEN DE COMPRA GENERAL EN GRIDVIEW

ASP.NET provee controles *HTML* que facilitan la manipulación de datos. El control *GridView (asp:GridVew)* permite que de una manera fácil se puedan mostrar datos a un usuario como si se tratara de una Tabla (*<table>*) de *HTML*. El siguiente código permite definir un *GridView* que mostrará los datos asignados a su propiedad *DataSource* vía *C#* (Órdenes de forma general)

```
<!-- Otro código fuente en HTML -->
<asp:GridView runat="server" ID="OrdenGridView" CssClass="GridViewStyle"
RowStyle-CssClass="GridViewRowStyle" PagerStyle-CssClass="GridViewPagerStyle"
AlternatingRowStyle-CssClass="GridViewAlternatingRowStyle"
FooterStyle-CssClass="GridViewPagerStyle" HeaderStyle-CssClass="GridViewHeaderStyle"
AllowPaging="False" AutoGenerateColumns="false" GridLines="None" ShowFooter="true"
DataKeyNames="OrdenId" EmptyDataText="No hay ningúna orden a seguir."
HorizontalAlign="Center" OnRowCommand="OrdenGridView_RowCommand">
    <Columns>
        <asp:BoundField DataField="OrdenId" HeaderText="Número" ItemStyle-Width="80px" />
        <asp:BoundField DataField="FechaOrden" HeaderText="Fecha orden"
            ItemStyle-Width="100px" />
        <asp:BoundField DataField="FechaAtencion" HeaderText="Fecha atención"
            ItemStyle-Width="120px" />
        <asp:BoundField DataField="FechaEnvio" HeaderText="Fecha envío"
            ItemStyle-Width="100px" />
        <asp:BoundField DataField="FechaEntrega" HeaderText="Fecha entrega"
            ItemStyle-Width="100px" />
        <asp:BoundField DataField="StatusOrden" HeaderText="Status"
            ItemStyle-Width="100px" />
        <asp:TemplateField HeaderText="Detalles" ItemStyle-Width="100px"
            ItemStyle-HorizontalAlign="Center">
            <ItemTemplate>
                <asp:LinkButton ID="VerLinkButton" runat="server" CommandName="Ver"
                    CommandArgument='<%# Eval("OrdenId") %>' Text="Ver" />
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
<!-- Otro código fuente en HTML -->
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *GridView* que servirá para mostrar información sobre las órdenes de compra del cliente obtenida a través del código de *C#, 2)* Definición de las columnas, generalmente la propiedad *DataField* permite visualizar los datos al usuario. Por ejemplo, para mostrar el número de orden de compra, la propiedad *DataField* del control *asp:BoundField* está enlazada con la propiedad *OrdenId* de la lista genérica que sirve como fuente de datos.

Método Page_Load

Este es el método que se ejecuta al iniciar la carga de la página del carrito de compra, se encarga de llenar con datos el *GridView* de la página. A continuación se muestra la definición del método:

```
protected void Page_Load(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;

    // Mostrar su nombre, esto depende de si está autenticado o no
    this.BienvenidoLabel.Text = !String.IsNullOrEmpty(nombreUsuario) ?
        "Bienvenido: " + User.Identity.Name : "Bienvenido: Anónimo";

    // Habilitar o inhabilitar el Logout
    this.LogoutLinkButton.Enabled = !String.IsNullOrEmpty(nombreUsuario) ? true : false;
    // Obtner la cookie del usuario
    String nombreCookie = "YnnafSotfwareCookie_" + nombreUsuario;
    HttpCookie httpCookie = Request.Cookies.Get(nombreCookie);

    // Usuario (predeterminado si no se está autenticado) y certificado
    // que accede al servicio WCF.
    ConnectionStringSettings settingsUser = null;
    settingsUser = ConfigurationManager.ConnectionStrings[
        "SitioWebUserConnectionString"];
    ConnectionStringSettings settingsPass = null;
    settingsPass = ConfigurationManager.ConnectionStrings[
        "SitioWebPassConnectionString"];
    ConnectionStringSettings settingsPathCert;
    settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
    ConnectionStringSettings settingsPassCert;
    settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

    // Conectar con el servicio WCF. Presentar certificados.
    ClienteWCF.YnnafSoftwareServiceWCFClient cliente =
        new YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
    cliente.ClientCredentials.ClientCertificate.Certificate =
        new X509Certificate2(settingsPathCert.ConnectionString,
            settingsPassCert.ConnectionString);

    // Iniciar sesión. El usuario con el que se conectará dependerá de
    // que exista o no la cookie en su PC
    if (httpCookie == null) {
        cliente.IniciarSesion(settingsUser.ConnectionString,
            Encriptacion.EncriptarSha512(settingsPass.ConnectionString));
    } else {
        cliente.IniciarSesion(httpCookie.Values["user"], httpCookie.Values["pass"]);
    }

    this.BienvenidoLabel.Text = !String.IsNullOrEmpty(nombreUsuario) ? "Bienvenido: " +
        nombreUsuario : "Bienvenido: Anónimo";

    ordenClienteId = cliente.Sp_GetOrdenByClienteId();
    this.OrdenGridView.DataSource = ordenClienteId.ListaOrden.ToList();
    this.OrdenGridView.DataBind();

    // Finalizar la sesión y el objeto
    cliente.FinalizarSesion();
    cliente.Close();
}
```

Lo que el código anterior efectúa es lo siguiente: *1)* Obtener el nombre del usuario (depende de la autenticación previa del usuario), *2)* Obtener la *cookie* del usuario, *3)* Conectar al Servicio *WCF* iniciando sesión con el usuario adecuado (depende de la existencia de la cookie), *4)* Mostrar un mensaje de bienvenida al usuario, *5)* Habiitar o deshabilitar el botón de logout (depende de la autenticación del usuario), *6)* Obtiene todas las órdenes que ha realizado el cliente a lo largo de la historia con la empresa y se asigna al contrato de datos *OrdenCliente* que almacena el resultado

arrojado por el método, *7)* Enlazar los datos en el *GridView* y *8)* Finalizar la sesión y destruir el servicio con el servicio *WCF*.

5.4.4.7.5.2 ENLACE DE DATOS PARA ORDEN DE COMPRA PARTICULAR EN GRIDVIEW

ASP.NET provee controles *HTML* que facilitan la manipulación de datos. El control *GridView (asp:GridVew)* permite que de una manera fácil se puedan mostrar datos a un usuario como si se tratara de una Tabla (*<table>*) de *HTML*. El siguiente código permite definir un *GridView* que mostrará los datos asignados a su propiedad *DataSource* vía *C#* (Órdenes de forma particular)

```
<!-- Otro código fuente en HTML -->
<asp:GridView runat="server" ID="OrdenDetalleGridView" CssClass="GridViewStyle"
    RowStyle-CssClass="GridViewRowStyle" PagerStyle-CssClass="GridViewPagerStyle"
    AlternatingRowStyle-CssClass="GridViewAlternatingRowStyle"
    FooterStyle-CssClass="GridViewPagerStyle" HeaderStyle-CssClass="GridViewHeaderStyle"
    AllowPaging="False" AutoGenerateColumns="false" GridLines="None" ShowFooter="true"
    DataKeyNames="OrdenId" EmptyDataText="" HorizontalAlign="Center">
    <Columns>
        <asp:TemplateField HeaderText="Imagen" ItemStyle-Width="100px">
            <ItemTemplate>
                <asp:Image ID="ProductoImagenImage" runat="server"
                    ImageUrl='<%#
                        "../ImagenesHandler.ashx?ProductoId=" + Eval("ProductoId") %>' />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:BoundField DataField="Producto" HeaderText="Producto"
            ItemStyle-Width="150px" />
        <asp:BoundField DataField="Categoria" HeaderText="Producto"
            ItemStyle-Width="150px" />
        <asp:BoundField DataField="Modelo" HeaderText="Producto"
            ItemStyle-Width="150px" />
        <asp:BoundField DataField="Cantidad" HeaderText="Cantidad"
            ItemStyle-Width="100px" ItemStyle-HorizontalAlign="Center" />
    </Columns>
</asp:GridView>
<!-- Otro código fuente en HTML -->
```

En resumen lo que podemos ver en el código anterior es lo siguiente: *1)* Definición visual del control *GridView* que servirá para mostrar información detallada de la orden de compra del cliente obtenida a través del código de *C#, 2)* Definición de las columnas, generalmente la propiedad *DataField* permite visualizar los datos al usuario. Por ejemplo, para mostrar la cantidad de productos a comprar, la propiedad *DataField* del control *asp:BoundField* está enlazada con la propiedad *Cantidad* de la lista genérica que sirve como fuente de datos.

Método OnRowCommand (OrdenGridView_RowCommand)

Este método está íntimamente ligado con la columna *Detalles* del *GridView* (*OrdenGridView*) del código *HTML*. Cuando se presiona cualquiera de los *LinkButton* contenidos en la columna, este método recibe el Id de la orden de compra (como se indicó en el código *HTML* en la propiedad *CommandArgument*), y muestra los detalles (particulares) de la orden de compra general.

```
protected void OrdenGridView_RowCommand(object sender, GridViewCommandEventArgs e) {
    if (e.CommandName == "Ver") {
        // Usando LINQ To Objects buscamos el detalle de la orden
        var match = from q in this.ordenClienteId.ListaDetalleOrden
                    where q.OrdenId == Convert.ToInt32(e.CommandArgument)
                    select q;

        // Mostramos el Id de la orden que se está mostrando.
        this.DetalleLabel.Text = "Productos de la orden: " + e.CommandArgument.ToString();

        // La mostramos en el GridView
        this.OrdenDetalleGridView.DataSource = match.ToList();
```

```
        this.OrdenDetalleGridView.DataBind();

        // Obtener el total a pagar en la orden
        decimal total = 0M;
        decimal subtotal = 0M;
        decimal iva = 0M;
        const decimal ivaPorc = 0.15M;
        // Obtener el subtotal por los productos
        foreach (OrdenDetalle_Join p in match) {
            subtotal += (p.PrecioUnitario * p.Cantidad);
        }

        var match2 = from q in this.ordenClienteId.ListaOrden
                     where q.OrdenId == Convert.ToInt32(e.CommandArgument)
                     select q;

        // Agregar el cargo extra al subtotal
        subtotal += match2.Single().CargoExtra;
        // Obtener el IVA
        iva = (subtotal * ivaPorc);
        // Obtener el total
        total = (subtotal + iva);

        // Mostrar el total a pagar en la parte de abajo de la página
        this.CostoLabel.Text = String.Format("Costo: {0:C}", total);
    }
}
```

En resumen lo que el código anterior efectúa es lo siguiente: *1)* Obtener de la lista genérica el detalle de la orden seleccionada de acuerdo al Id de la orden, *2)* Enlazar la información con el *GridView* y *3)* Calcular la cantidad que se pagó por esa orden de compra y mostrarla al cliente.

5.4.4.7.6 BUSCAR PRODUCTOS

El siguiente código muestra como se puede buscar un producto a través de cualquiera de las ventanas de la aplicación web. Este método se encuentra en todas las páginas de la aplicación. El siguiente código en *C#* ejemplifica el proceso que se ejecuta en la ventana origen (donde se inició la búsqueda):

```
protected void BuscarButton_Click(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;

    // Obtner la cookie del usuario
    String nombreCookie = "YnnafSotfwareCookie_" + nombreUsuario;
    HttpCookie httpCookie = Request.Cookies.Get(nombreCookie);

    // Usuario (predeterminado si no se está autenticado) y certificado
    // que accede al servicio WCF.
    ConnectionStringSettings settingsUser = null;
    settingsUser = ConfigurationManager.ConnectionStrings[
        "SitioWebUserConnectionString"];
    ConnectionStringSettings settingsPass = null;
    settingsPass = ConfigurationManager.ConnectionStrings[
        "SitioWebPassConnectionString"];
    ConnectionStringSettings settingsPathCert;
    settingsPathCert = ConfigurationManager.ConnectionStrings["PathCertificado"];
    ConnectionStringSettings settingsPassCert;
    settingsPassCert = ConfigurationManager.ConnectionStrings["PassCertificado"];

    // Conectar con el servicio WCF. Presentar certificados.
    ClienteWCF.YnnafSoftwareServiceWCFClient cliente =
        new YnnafSoftwareClientWCFASPNET.ClienteWCF.YnnafSoftwareServiceWCFClient();
    cliente.ClientCredentials.ClientCertificate.Certificate =
        new X509Certificate2(settingsPathCert.ConnectionString,
```

```
            settingsPassCert.ConnectionString);

    // Iniciar sesión. El usuario con el que se conectará dependerá de
    // que exista o no la cookie en su PC
    if (httpCookie == null) {
        cliente.IniciarSesion(settingsUser.ConnectionString,
            Encriptacion.EncriptarSha512(settingsPass.ConnectionString));
    } else {
        cliente.IniciarSesion(httpCookie.Values["user"], httpCookie.Values["pass"]);
    }

    // Obtener los productos y asignarlos a la sesión que los mantendrá
    // persistentes para poder mostrarse en la página buscar.aspx
    this.productos = cliente.GetProductosByNombre(this.BuscarTextBox.Text).ToList();
    HttpContext.Current.Session["ProductosBuscadosYnnafSoftwareSession"] = productos;

    // Finalizar la sesión y el objeto
    cliente.FinalizarSesion();
    cliente.Close();

    // Redirigir a la página de búsqueda
    Response.Redirect("buscar.aspx");
}
```

En resumen lo que el código anterior hace es lo siguiente: *1)* Obtener el nombre del usuario (depende de la autenticación previa del usuario), *2)* Obtener la *cookie* del usuario, *3)* Conectar al Servicio *WCF* iniciando sesión con el usuario adecuado (depende de la existencia de la cookie), *4)* Mostrar un mensaje de bienvenida al usuario, *5)* Habiitar o deshabilitar el botón de *Logout* (depende de la autenticación del usuario), *6)* Obtener una lista genérica con todos los productos que coincidan con el nombre del producto a buscar, *7)* Asignar la lista genérica a una sesión de *HTTP* llamada *ProductosBuscadosYnnafSoftwareSession*, *8)* Redirigir al usuario a la página donde se verán los resultados y *9)* Finalizar la sesión y destruir el servicio con el servicio *WCF*.

El siguiente código en *C#* muestra los procesos que se ejecutan para mostrar al usuario los resultados de la búsqueda (la página *~/buscar.aspx* contiene un *GridView* para mostrar los datos):

```
protected void Page_Load(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;

    // Mostrar su nombre, esto depende de si está autenticado o no
    this.BienvenidoLabel.Text = !String.IsNullOrEmpty(nombreUsuario) ?
        "Bienvenido: " + User.Identity.Name : "Bienvenido: Anónimo";

    // Habilitar o inhabilitar el Logout
    this.LogoutLinkButton.Enabled = !String.IsNullOrEmpty(nombreUsuario) ? true : false;

    // Obtener los productos desde la sesión y mostrarlos en el GridView
    this.productos = (List<Producto_Join>)
        HttpContext.Current.Session["ProductosBuscadosYnnafSoftwareSession"];
    this.ProductosGridView.DataSource = productos;
    this.ProductosGridView.DataBind();
}
```

En resumen lo que el código anterior hace es lo siguiente: *1)* Obtener el nombre del usuario (depende de la autenticación previa del usuario), *2)* Habiitar o deshabilitar el botón de *Logout* (depende de la autenticación del usuario), *3)* Recuperar el valor contenido en la sesión y asignarlo a una lista genérica y *4)* Mostrar los datos obtenidos al usuario a través del *GridView*.

5.4.4.7.7 LOGOUT

El siguiente código muestra como finalizar una autenticación del cliente cuando éste pulsa el botón (*<asp:LinkButton>*) *Logout* del menú de navegación. Este método se encuentra en todas las páginas de la aplicación.

```
protected void LogoutLinkButton_Click(object sender, EventArgs e) {
    // Obtener el nombre del usuario
    String nombreUsuario = !String.IsNullOrEmpty(User.Identity.Name) ?
        User.Identity.Name : null;
    // Obtener la cookie del usuario
    String nombreCookie = "YnnafSotfwareCookie " + nombreUsuario;
    // Remover la cookie y quitar autenticación
    Response.Cookies.Remove(nombreCookie);
    FormsAuthentication.SignOut();
    // Ocultar el botón
    this.LogoutLinkButton.Enabled = false;
    // Redirigirlo a la página principal
    Response.Redirect("default.aspx");
}
```

En resumen lo que este código hace es: *1)* Obtener el nombre del usuario (depende de la autenticación previa del usuario), *2)* Generar el nombre de la cookie insertada en *JavaScript* en el proceso de *Login*, *3)* Remover la cookie, *4)* Destruir la autenticación, *5)* Deshabilitar el botón de *Logout* (depende de la autenticación del usuario), *6)* Redirigir al usuario a la página principal.

5.4.5 DESPLIEGUE

Una vez que el sitio web está publicado en *IIS 7.0 (la forma en que publica la aplicación web es similar a la usada para publicar el servicio WCF en el capítulo 4)* con el alias *website* (dirección *http://localhost/website*) el usuario puede navegar en la página de la siguiente manera:

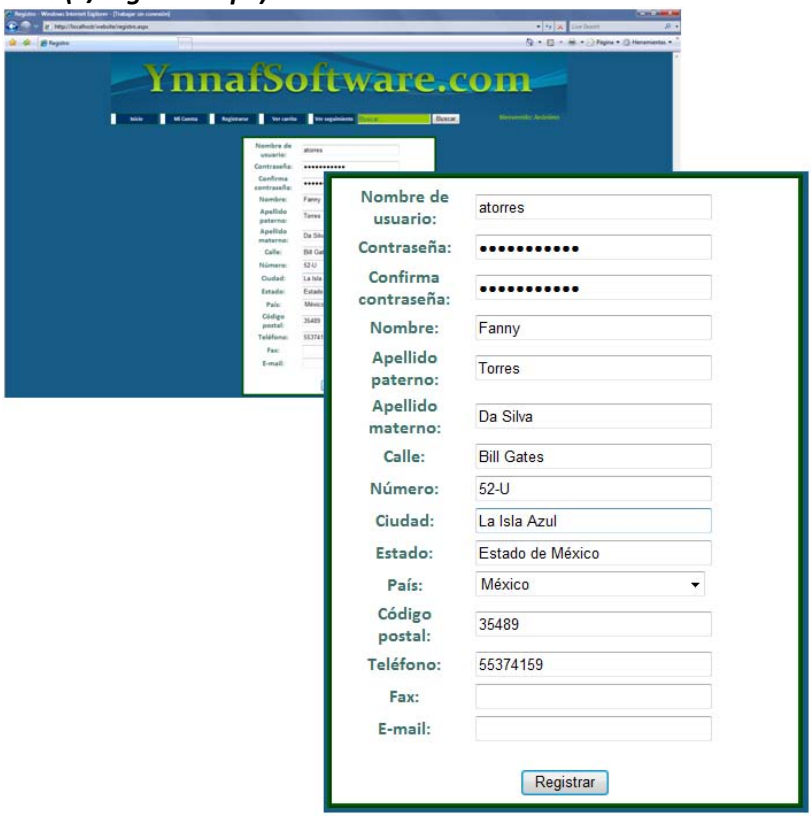
Página inicial. *(~/default.aspx)*

Página inicial: http://localhost/website o http://localhost/website/default.aspx. Es esta página se muestran productos que puede comprar un cliente que accede al sitio web. En el menú (de izquierda a derecha): *Inicio* (Lleva a la página inicial); *Mi Cuenta -> Login* (Lleva a la página de autenticación); *Mi Cuenta -> Logout* (Finaliza una autenticación previa); *Registrarse* (Lleva a la página donde se puede dar de alta un nuevo cliente; *Ver seguimiento* (Lleva a la página donde se puede hacer seguimiento de órdenes de compra anteriores hechas por un cliente); *Buscar* (Busca en la base de datos el producto insertado en la caja de texto); *Bienvenido: [usuario]* (Muestra el nombre de usuario del cliente). En la imagen se puede observar que el funcionamiento de la página es compatible entre diferentes navegadores, de izquierda a derecha: ***Microsoft Internet Explorer 8.0***, ***Google Chrome 2.0*** y ***Mozilla FireFox 3.5***.

Login de usuario. *(~/login.aspx)*

Página de login: http://localhost/website/login.aspx. En esta página se obtiene autorización para finalizar una orden de compra. El usuario debe introducir su nombre de usuario y su contraseña y presionar el botón *Ingresar*. Si no está registrado, puede ir a la página de registro desde esta página.

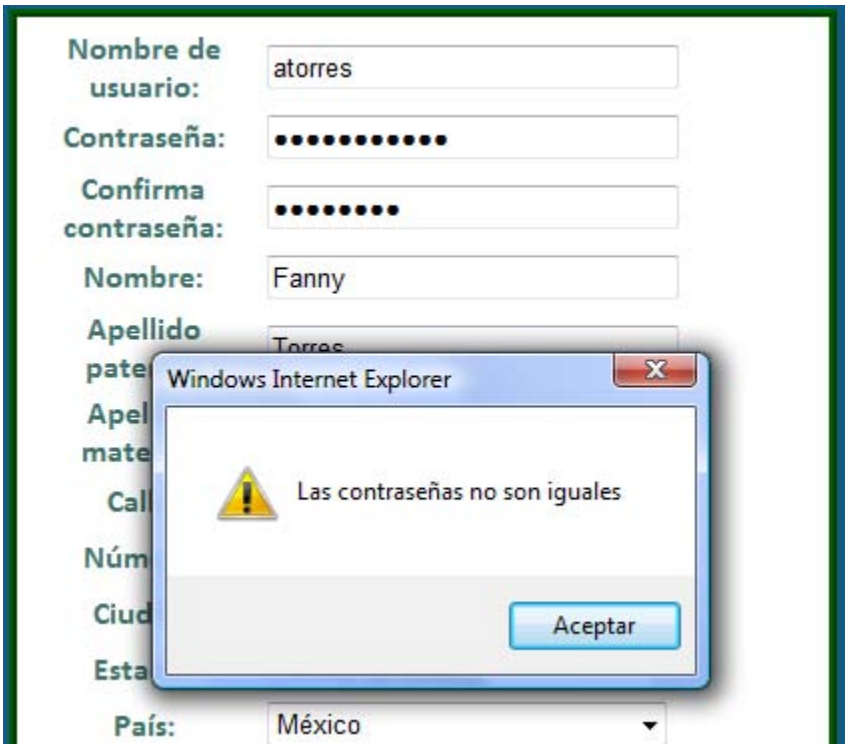
Registro de usuario. *(~/registro.aspx)*



Página de Registro: http://localhost/website/registro.aspx. Esta página permite al usuario darse de alta ante el sistema. Al estar registrado, el usuario puede realizar compras. Si los datos introducidos son correctos se redirige al usuario a la página inicial del sitio teniendo ya la autenticación con el sistema.

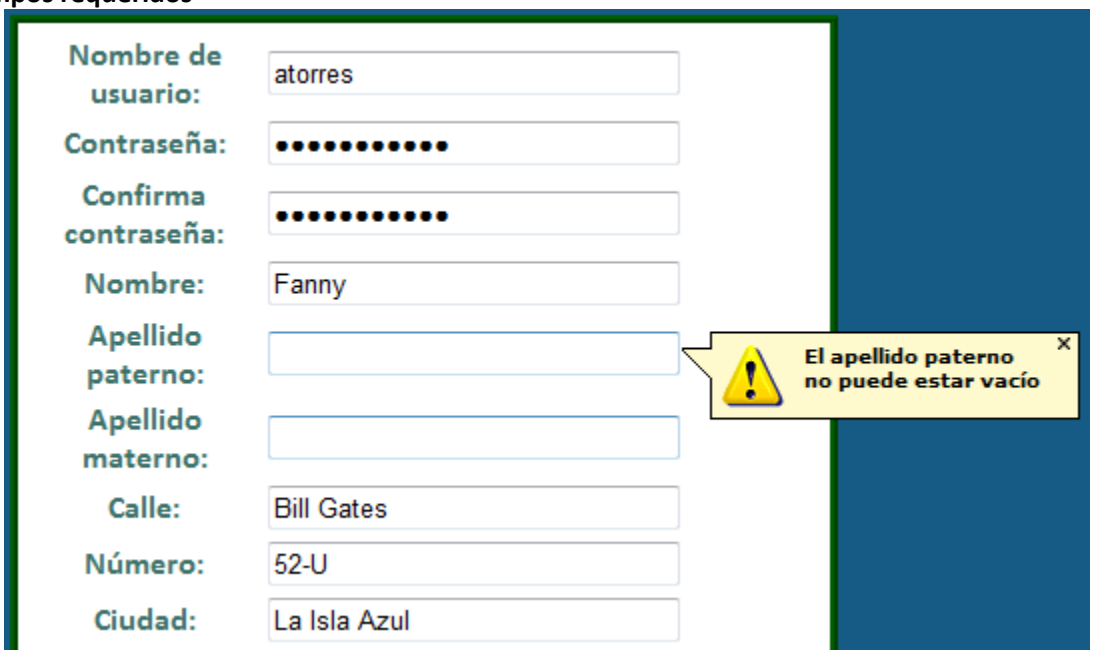
Registro de usuario -> Validaciones

Contraseñas



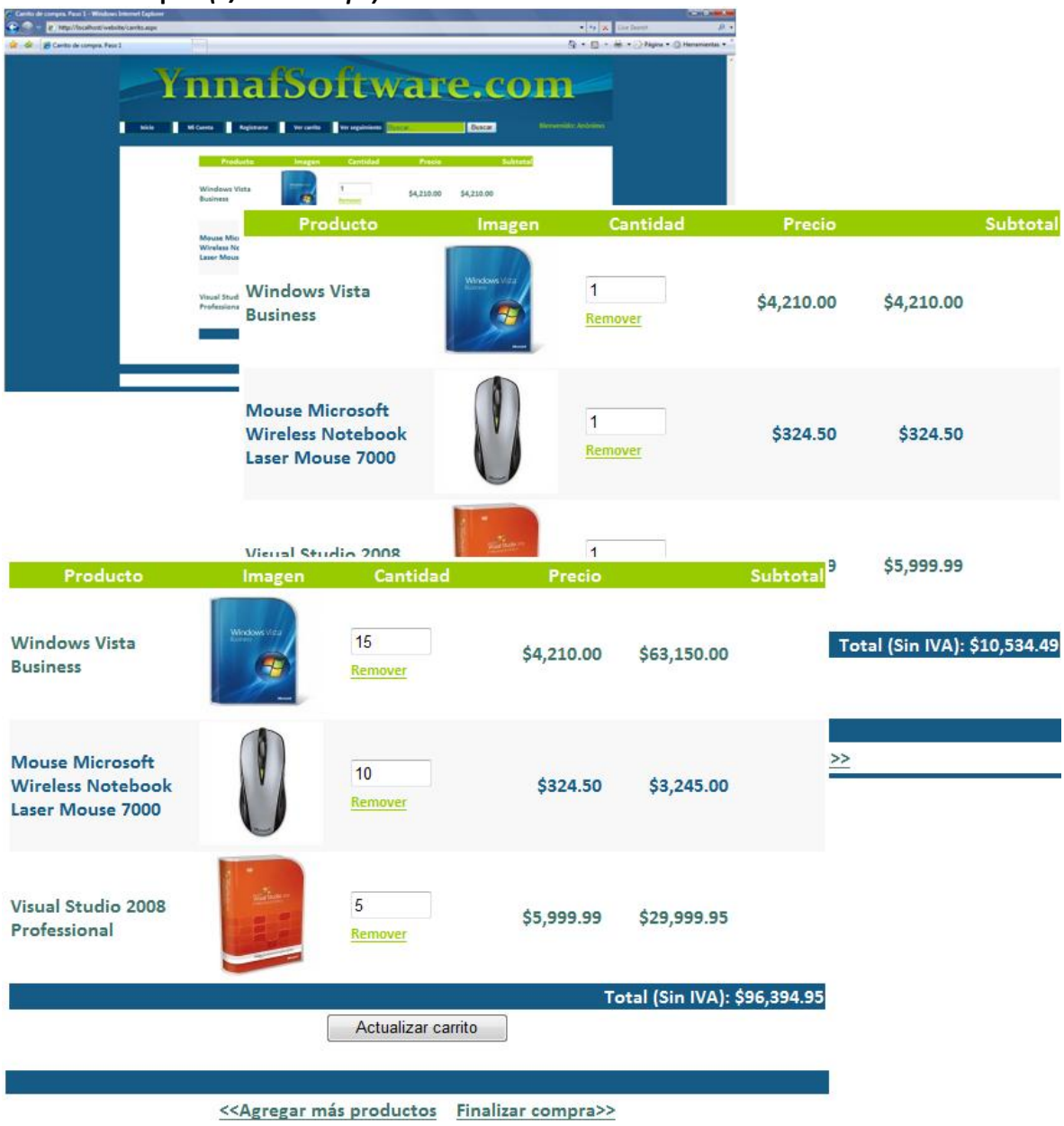
Si las contraseñas no son iguales, se le muestra al usuario un mensaje (creado en *JavaScript*) y se impide el envío de datos al servidor hasta que el usuario verifique las contraseñas.

Campos requeridos



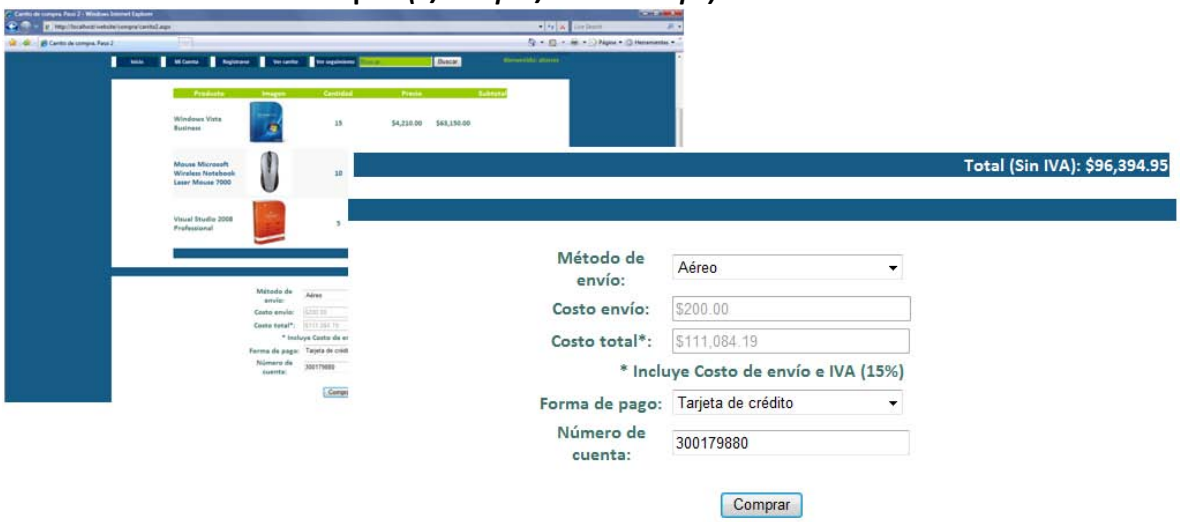
Si hay un campo obligatorio y éste no se ha llenado, se le muestra al usuario un mensaje (creado en *AJAX Control Toolkit*) y se impide el envío de datos al servidor hasta que el usuario complete los campos.

Carrito de compra. ***(~/carrito.aspx)***



Página de Carrito de compra Paso 1: http://localhost/website/carrito.aspx. Esta página permite a un usuario ver los productos que ha agregado a su carrito de compra desde las diferentes páginas del sitio web.

Finalización de orden de compra. *(~/compra/carrito2.aspx)*



Página de Carrito de compra Paso 2: http://localhost/website/compra/carrito2.aspx. Esta página permite a un usuario: 1) Ver los productos que va a comprar, 2) Seleccionar la forma de envío, 3) Seleccionar la forma de pago (e insertar el número de cuenta si la forma de pago así lo requiere) y 4) Finalizar su orden de compra presionando el botón *Comprar*

Orden de compra satisfactoria. *(~/compra/ordenSatisfactoria.aspx)*

Página de Compra satisfactoria: http://localhost/website/compra/ordenSatisfactoria.aspx. Si no ocurrió ningún error al crear la orden de compra, esta página muestra al usuario el número de su orden de compra. En esta imagen el número de orden de compra asignado es *14*.

Seguimiento general de la orden de compra. *(~/compra/seguimiento.aspx)*

Número	Fecha orden	Fecha atención	Fecha envío	Fecha entrega	Status	Detalles
9	15/03/2009 12:00:00 a.m.	15/03/2009 12:00:00 a.m.	16/03/2009 12:00:00 a.m.	17/03/2009 12:00:00 a.m.	Entregado	Ver
10	05/04/2009 12:00:00 a.m.	05/04/2009 12:00:00 a.m.	06/07/2009 12:00:00 a.m.	06/04/2009 12:00:00 a.m.	Entregado	Ver
11	28/05/2009 12:00:00 a.m.	28/05/2009 12:00:00 a.m.	29/05/2009 12:00:00 a.m.	29/05/2009 12:00:00 a.m.	Entregado	Ver
12	20/06/2009 12:00:00 a.m.	20/06/2009 12:00:00 a.m.	21/06/2009 12:00:00 a.m.	22/06/2009 12:00:00 a.m.	Entregado	Ver
13	27/07/2009 03:06:10 p.m.	28/07/2009 12:00:00 a.m.	29/07/2009 12:00:00 a.m.	29/07/2009 12:00:00 a.m.	Entregado	Ver
14	16/08/2009 08:20:23 p.m.	17/08/2009 12:00:00 a.m.			Enviando	Ver

Página de Seguimiento general: http://localhost/website/compra/seguimiento.aspx. Esta página muestra todas las órdenes de compra que ha hecho el usuario a la empresa. Únicamente se muestra información general de la orden, pero también se da la posibilidad de ver el detalle de la orden pulsando el vínculo ***Ver.*** En la imagen anterior se ve que el **estado de la orden de compra *14*** (creada previamente) es **"*Enviando*"**, **éste fue afectado por** el primer programa (***C#/WPF***) construido en esta tesis.

Seguimiento particular de la orden de compra. *(~/compra/seguimiento.aspx)*

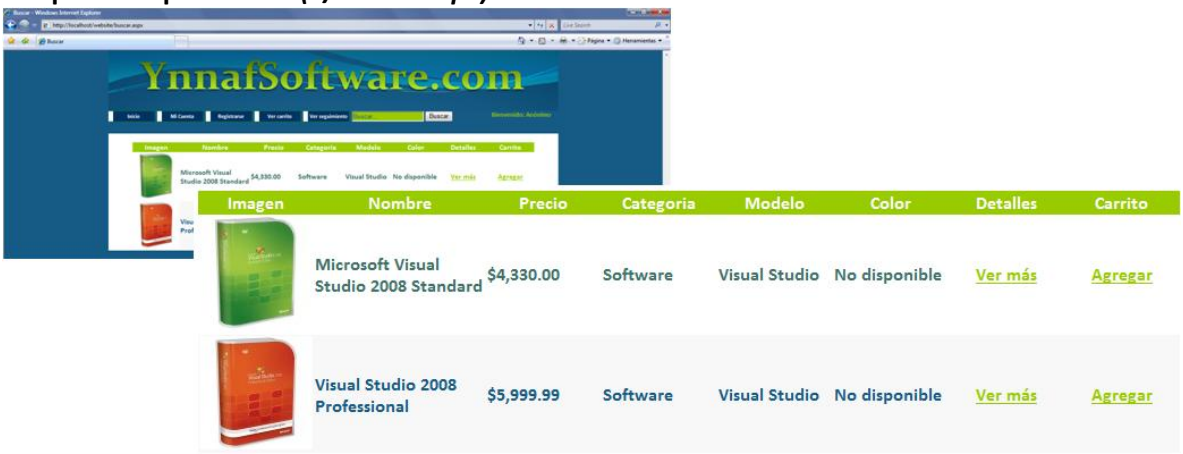
Productos de la orden: 14

Imagen	Producto	Producto	Producto	Cantidad
	Windows Vista Business	Software	WVista	15
	Mouse Microsoft Wireless Notebook Laser Mouse 7000	Hardware	Microsoft Wireless Notebook Laser Mouse 7000	10
	Visual Studio 2008 Professional	Software	Visual Studio	5

Costo: $111,084.19

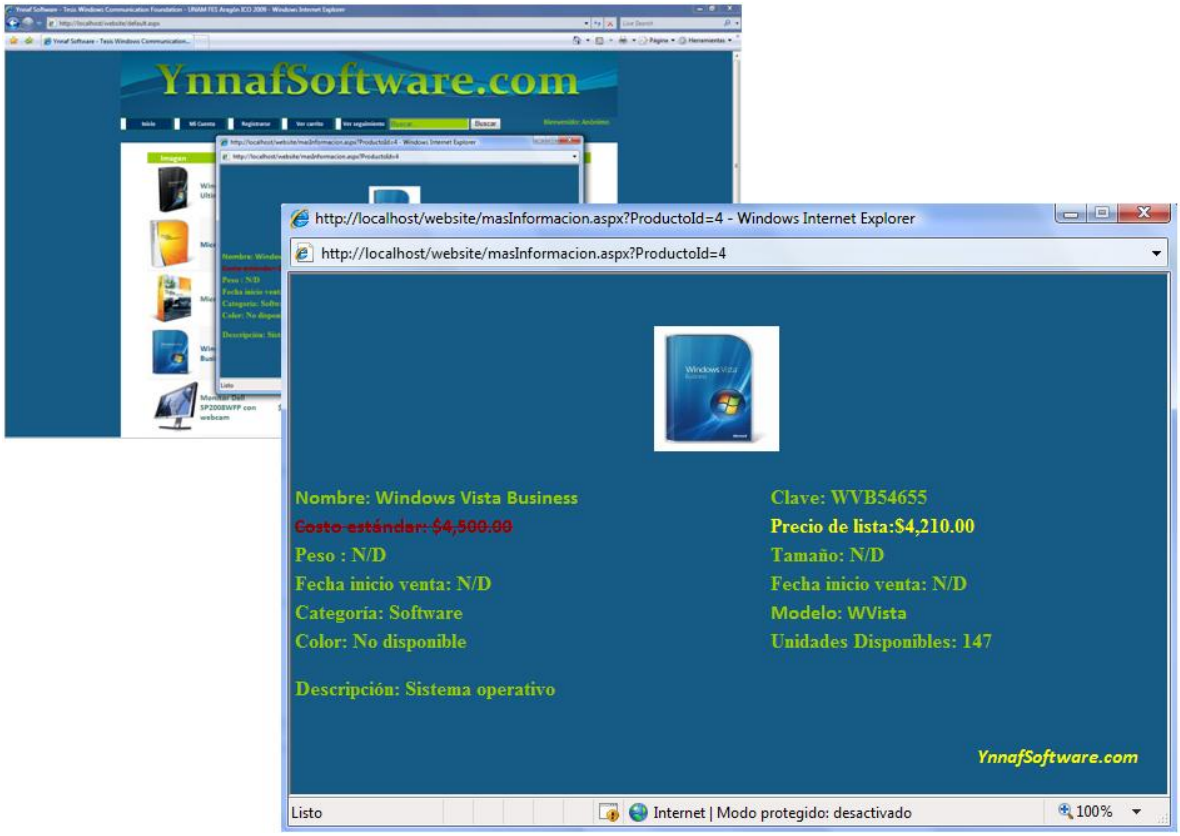
Página de Seguimiento particular: http://localhost/website/compra/seguimiento.aspx. Esta página muestra información detallada de la orden de compra. La información detallada depende del vínculo ***Ver*** presionado en la orden de compra general. En la imagen se ve el detalle de la orden de compra número *14* (creada anteriormente).

Búsqueda de productos. *(~/buscar.aspx)*



Página de búsqueda: http://localhost/website/buscar.aspx. En esta ventana podemos ver los productos que se encontraron si el usuario buscó productos que contuvieran el nombre "*Visual*". En la búsqueda anterior los resultados son: *Microsoft Visual Studio 2008 Standard* y *Professional*.

Información detallada del producto. *(~/masInformacion.aspx?ProductoId=[ProductoId])*



Página de detalle del producto: http://localhost/website/masInformacion.aspx?ProductoId=[ProductoId]. En esta ventana se muestra al usuario información particular del producto si presionó el vínculo *Ver más.* En la imagen se presionó el vínculo *Ver más* del producto *Windows Vista Business* en la página *default.aspx.*

CONCLUSIÓN

A través de este trabajo de tesis se concluye que *Windows Communication Foundation (WCF)* es una alternativa eficaz y eficiente que permite la interoperabilidad entre diferentes aplicaciones construidas tanto en diversos lenguajes de programación y plataformas de programación de software como en iguales.

Con *Windows Communication Foundation* de *.NET Framework 3.0* y *3.5* se ofrece un modelo estándar de programación que engloba las tecnologías de comunicaciones de *.NET Framework 2.0* como *Web Services Enhancements (WSE)*, *Enterprise Services*, *.NET Remoting* o *Microsoft Message Queuing* y además implementa las especificaciones *WS-**. Así, se pueden crear servicios web de una forma fácil, rápida y, sobretodo, adaptable, esto hace a esta tecnología superior a todas las existentes anteriormente (como los servicios web de *ASP.NET* que permiten crear aplicaciones distribuidas usando *XML* y *SOAP.*) y perfectamente compatible con tecnologías similares y actuales como se mostró en esta tesis al hacer compatible a *Windows Communication Foundation* con *Web Services Interoperability Technologies* de *Java*.

Como se vio a lo largo de este trabajo, *Windows Communication Foundation* permite escribir, publicar, implementar y consumir servicios no solo con la interoperabilidad que proporcionan los servicios web estándar, sino también utilizando diferentes formas de transporte, seguridad y confiabilidad de forma transparente al programador. La programación orientada a servicios en la que se basa *Windows Communication Foundation* facilita el desarrollo de sistemas distribuidos que se apoyan en procesos de negocio facilitando la tarea de combinar dichos procesos dentro de un sistema obteniendo un alto rendimiento en el resultado final.

Un punto importante a destacar es el beneficio económico que pueden obtener las empresas a partir del uso de *Windows Communication Foundation*, puesto que muchas empresas hoy en día utilizan *.NET Framework* o *Java* para desarrollar sus aplicaciones, en esta tesis se demuestra que *Windows Communication Foundation* interopera perfectamente con *Web Services Interoperability Technologies.* Esto evita que las empresas se vean obligadas a migrar de una tecnología a otra cuando se presentan problemas de interoperabilidad, disminuyendo notablemente los gastos de operación de las empresas cuando éstas requieren cubrir sus necesidades de interoperabilidad de software.

Se debe observar que aunque todos las aplicaciones de software expuestas en esta tesis se ejecutan sobre *Windows*, esto no quiere decir que la interoperabilidad entre aplicaciones sea exclusiva para este sistema operativo pues gracias a que las plataformas de programación de software utilizadas son multiplataforma, la interoperabilidad de *Windows Communication Foundation* / *Web Services Interoperability Technologies* se extiende también a otros sistemas operativos como *Linux* o *MAC OS*.

Se espera que este trabajo de tesis sea de gran utilidad para todas aquellas personas que necesiten utilizar herramientas de programación poderosas que le permitan construir aplicaciones distribuidas interoperables escritas o construidas con diferentes tecnologías de programación que se desempeñen con un alto rendimiento.

BIBLIOGRAFÍA

- ✓ Hejlsberg, Anders et al. **The C# Programming Language**. Pearson Education. United States 2004.
- ✓ Lowy, Juval. **Programming WCF Services**. O'Reilly Media First edition. United States 2007
- ✓ Rattz, Joseph C. Jr. **Pro LINQ: Language Integrated Query in C# 2008**. Apress First Edition. United States 2008.
- ✓ MacDonald, Matthew **Pro WPF in C# 2008: Windows Presentation Foundation with .NET 3.5**. Apress Second Edition. United States 2008.
- ✓ Moldes, Francisco Javier, **JAVA SE 6**, Anaya multimedia. México 2008.
- ✓ Troelsen, Andrew. **Pro C# with .NET 3.0**. Apress Special Edition. United States 2007.
- ✓ Solis, Daniel. **Illustrated C# 2005**. Apress First Edition. United States 2006.
- ✓ Charte Ojeda, Francisco. **Visual C#**. Anaya multimedia. México 2002.
- ✓ Jones, Allen and MacDonald, Matthew. **Visual C# 2005 Recipes. A problem-solution approach**. Apress First Edition. United States 2006.
- ✓ Ceballos, Francisco Javier. **Enciclopedia de Microsoft C#**. Alfaomega Ra-Ma Segunda edición. México 2007.
- ✓ Sierra, Antonio Martín. **Desarrollo de aplicaciones web con ASP.NET 2.0**. Alfaomega Primera edición. México 2007.
- ✓ Lhotka, Rockford. **Expert VB 2005 Bussiness Objects**. Apress Second edition, United States 2007.
- ✓ Charte Ojeda, Francisco. **Visual Basic 2008**. Anaya multimedia. México 2008.

REFERENCIAS

1. **MSDN: Microsoft Developer Network. Biblioteca de clases de .NET Framework**
   http://msdn.microsoft.com/es-mx/library/ms229335.aspx
2. **Detalles de las características de WCF**
   http://msdn.microsoft.com/es-mx/library/ms733103.aspx
3. **Java Platform, Standard Edition 6 API Specification**
   http://java.sun.com/javase/6/docs/api
4. **Web Services Interoperability Technologies (WSIT)**
   https://wsit.dev.java.net