Universidad Central “Marta Abreu” de Las Villas
Facultad de Matemática-Física-Computación

TITULO: Un modelo de intercambio de datos basado en XML.

Autor: Lic. Yoan Pacheco Cárdenas

Tutora: Dra. Ana María García Pérez

Tesis en opción al Título de Master en Ciencia de la Computación

2006
“Año de la Revolución Energética en Cuba”

RESUMEN

En esta tesis de maestría se hace un estudio comparativo entre las diferentes tecnologías de intercambio de datos existentes en la actualidad, determinando la necesidad del establecimiento de guías para construir soluciones efectivas a las demandas crecientes de las empresas en el mundo para la interconexión de servicios de negocio.

El presente trabajo propone un modelo de intercambio de datos basado en XML. El mismo aporta una serie de conceptos novedosos a los ya existentes en este campo, entre los más notables se encuentra la utilización de patrones de diseño aplicables a arquitecturas orientadas a servicios Web.

En el modelo se propone además la integración de conceptos para lograr agilidad en la interacción entre servicios Web.

A manera de validación se muestra la aplicación del modelo al diseño de una arquitectura orientada a servicios que permite intercambiar datos en una Agencia de Viajes con sus proveedores y sucursales.

ABSTRACT

In this master degree we offer a comparative study among the different data interchange technologies that exist nowadays, taking into consideration the need to establish some guidelines to create solutions directed to satisfy the increasing requirements of growing entities in the world to satisfy the interconnection of business services.

This project proposes a data interchange model based on XML. It brings to light certain genuine concepts in addition to the existing ones, where we can find the use of design patterns applied to oriented architectures, to web services.

It also suggests the integration of concepts to better the interaction among web services.

As a way to validate our proposal a sample of the model applied to the design of a service oriented architecture that will allow data interchange with a travel agency, its providers and branches is shown.

ÍNDICE

INTRODUCCIÓN

Las aplicaciones de software para la gestión demandan cada vez más el intercambio de información entre puntos distribuidos, por ejemplo, entre proveedores de servicios y clientes de lugares geográficos distantes o entre sucursales de una misma empresa. El explosivo crecimiento de la Web, la creciente popularidad de las PC y los avances en las redes de alta velocidad han llevado la computación distribuida a su máxima expresión.

Internet permite que los usuarios accedan a servicios y ejecuten aplicaciones sobre un conjunto heterogéneo de redes y computadores. Esta heterogeneidad (es decir, variedad y diferencia) se aplica a los siguientes elementos:

- Redes.
- Hardware de computadoras.
- Sistemas operativos.
- Lenguajes de programación.
- Implementaciones de diferentes desarrolladores.

A pesar de que Internet consta de muchos tipos de redes diferentes sus diferencias se encuentran enmascaradas debido a que todas las computadoras conectadas a ésta utilizan los protocolos de Internet para comunicarse una con otra. Por ejemplo, un computador conectado a Ethernet tiene una implementación de los protocolos de Internet sobre Ethernet, así un computador en un tipo de red diferente necesitará una implementación de los protocolos de Internet para esa red.

Los tipos de datos, como los enteros, pueden representarse de diferente forma en diferentes clases de hardware, por ejemplo, hay dos alternativas para ordenar los bytes en el caso de los enteros.

Hay que tratar con estas diferencias de representación si se van a intercambiar mensajes entre programas que se ejecutan en diferente hardware. Aunque los sistemas

operativos de todas las computadoras de Internet necesitan incluir una implementación de los protocolos de Internet, no todas presentan necesariamente la misma interfaz de programación para estos protocolos. Por ejemplo, las llamadas para intercambiar mensajes en UNIX son diferentes de las llamadas en Windows.

Lenguajes de programación diferentes emplean representaciones diferentes para caracteres y estructuras de datos como cadenas de caracteres y registros. Hay que tener en cuenta estas diferencias si queremos que los programas escritos en diferentes lenguajes de programación sean capaces de comunicarse entre ellos.

Los programas escritos por diferentes programadores no podrán comunicarse entre sí a menos que utilicen estándares comunes, por ejemplo para la comunicación en red y la representación de datos elementales y estructuras de datos en mensajes. Para que esto ocurra es necesario concertar y adoptar estándares (como así lo son los protocolos de Internet).

Algunos de estos estándares especifican una capa intermedia o middleware. El término middleware se aplica a la capa de software que provee una abstracción de programación, así como un enmascaramiento de la heterogeneidad subyacente de las redes, hardware, sistemas operativos y lenguajes de programación. CORBA [1] es un ejemplo de ello. Algún middleware, como Java RMI [2] sólo se soporta en un único lenguaje de programación. La mayoría de los middleware se implementan sobre protocolos de Internet, enmascarando éstos la diversidad de redes existentes. Aun así cualquier middleware trata con las diferencias de sistema operativo y hardware.

Además de soslayar los problemas de heterogeneidad, el middleware proporciona un modelo computacional uniforme al alcance de los programadores de servidores y aplicaciones distribuidas. Los posibles modelos incluyen invocación sobre objetos remotos, notificación de eventos remotos, acceso remoto mediante SQL y procesamiento distribuido de transacciones. Por ejemplo, CORBA proporciona llamadas a procedimientos remotos, lo que permite que un objeto en un programa en ejecución en un computador invoque un método de un objeto de un programa que se ejecuta en otro computador. La implementación oculta el hecho de que los mensajes se transmiten en red en cuanto al envío de la petición de invocación y su respuesta.

Por tanto el diseño de sistemas distribuidos presenta los siguientes desafíos:

EXTENSIBILIDAD

La extensibilidad de un sistema de cómputo es la característica que determina si el sistema puede ser extendido y reimplementado en diversos aspectos. La extensibilidad de los sistemas distribuidos se determina en primer lugar por el grado en el cual se pueden añadir nuevos servicios de compartición de recursos y ponerlos a disposición para el uso por una variedad de programas clientes.

No es posible obtener extensibilidad a menos que la especificación y la documentación de las interfaces software clave de los componentes de un sistema estén disponibles para los desarrolladores de software.

Los sistemas diseñados de este modo para dar soporte a la compartición de recursos se etiquetan como sistemas distribuidos abiertos (open distributed systems) para remarcar el hecho de ser extensibles.

SEGURIDAD

Entre los recursos de información que se ofrecen y se mantienen en los sistemas distribuidos, muchos tienen un alto valor intrínseco para sus usuarios. Por esto su seguridad es de considerable importancia. La seguridad de los recursos de información tiene tres componentes:

1. Confidencialidad (protección contra el descubrimiento por individuos no autorizados);
2. Integridad (protección contra la alteración o corrupción);
3. Disponibilidad (protección contra interferencia con los procedimientos de acceso a los recursos).

ESCALABILIDAD

Los sistemas distribuidos operan efectiva y eficientemente en muchas escalas diferentes, desde pequeñas intranets a Internet. Se dice que un sistema es escalable si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios.

TRATAMIENTO DE FALLOS.

Cualquier proceso, computador o red puede fallar independientemente de los otros. En consecuencia cada componente necesita estar al tanto de las formas posibles en que pueden fallar los componentes de los que depende y estar diseñado para tratar apropiadamente con cada uno de estos fallos.

CONCURRENCIA

La presencia de múltiples usuarios en un sistema distribuido es una fuente de peticiones concurrentes a sus recursos. Cada recurso debe estar diseñado para ser seguro en un entorno concurrente.

TRANSPARENCIA

El objetivo es que ciertos aspectos de la distribución sean invisibles al programador de aplicaciones de modo que sólo necesite ocuparse del diseño de su aplicación particular. Por ejemplo, no debe ocuparse de su ubicación o los detalles sobre cómo se accede a sus operaciones por otros componentes, o si será replicado o migrado. Incluso los fallos de las redes y los procesos pueden presentarse a los programadores de aplicaciones en forma de excepciones, aunque deban de ser tratados.

La presente tesis de maestría tiene el siguiente

Objetivo General:

Mostrar guías para desarrolladores, que permitan la construcción de mecanismos de intercambio de datos entre aplicaciones distribuidas en diferentes plataformas, subredes y dispositivos, utilizando el Standard XML y las tecnologías asociadas a éste.

Como Objetivos Específicos:

1. Caracterizar el estado del arte en el empleo de las tecnologías existentes para intercambio de datos entre aplicaciones.
2. Proponer un modelo de intercambio de datos para aplicaciones en diferentes plataformas, subredes y dispositivos.
3. Mostrar el desarrollo de mecanismos obtenidos por el autor empleando el modelo anterior.

Como **hipótesis** de nuestra investigación, consideramos que:

Un modelo de intercambio de datos basado en XML, y ejemplos de su uso, permitirá a los desarrolladores crear conexiones entre aplicaciones que se ejecuten en diferentes plataformas, subredes y dispositivos.

La **novedad** de la misma radica en que la tesis presentará formas novedosas de abordar la construcción de software empleando la interconexión de varias aplicaciones, característica que se requiere en la actualidad.

Como **valor práctico**: La tesis contribuirá a ahorrar tiempo de desarrollo de software donde se requiera el intercambio de datos, lo cual es grandemente valorado en la industria de software actual, pues se esclarece cómo usar con este propósito el Standard XML y sus tecnologías asociadas.

La estructura en Capítulos es la siguiente:

- Capitulo 1: El estado del arte en tecnologías para el intercambio de datos: Se realiza un estudio de las soluciones para la implementación de sistemas distribuidos para intercambiar datos, abordando los sockets, CORBA, DCOM, Java RMI, .Net Remoting, XML RPC y Servicios Web. Finalmente se analizan los retos de diseñar una arquitectura orientada a servicios (SOA), en particular servicios Web, despejando la necesidad de un modelo para el intercambio de datos basado en XML en empresas que deseen implementar sistemas distribuidos de este tipo.
- Capítulo 2: Un modelo de intercambio de datos para aplicaciones en diferentes plataformas, subredes y dispositivos: Se plantean las premisas y los requisitos del modelo. Se aborda el modelo y se explican sus etapas, combinando al final un conjunto de patrones de diseño que pueden ser aplicados en problemas del mismo tipo.
- Capítulo 3: Guías para el uso del modelo propuesto: Se aplica el modelo propuesto a un caso particular de modo que esto ayude a los desarrolladores y arquitectos de software a construir aplicaciones con estas guías.

CAPÍTULO I

A continuación se realiza un estudio del estado del arte en las tecnologías de intercambio de datos entre aplicaciones distribuidas, con esto se pretende comprender la evolución de los sistemas distribuidos a lo largo del tiempo, sus ventajas y deficiencias hasta llegar al estado actual.

1.1 Sockets

Una de las primeras soluciones a la implementación de sistemas distribuidos para el intercambio de datos fueron los sockets, que precisamente tienen la capacidad de comunicar dos procesos, ya sea mediante datagramas o flujos de datos (streams). Sin embargo, los sockets requieren que las aplicaciones implementen sus propios protocolos para codificar y decodificar los mensajes que intercambian, lo que introduce una problemática diferente a la naturaleza del problema a resolver y aumenta la posibilidad de errores durante la ejecución.

1.2 Remote Procedure Call (RPC)

Una de las primeras alternativas que surgió al empleo de los sockets fue RPC. RPC es una abreviatura para Remote Procedure Call y una especificación diseñada por el Open Group [3]. RPC es el pilar arquitectónico de muchas de las tecnologías de acceso remoto.

La cual incluye:

- Proxies y stubs: Código presente en clientes y servidores cuyo propósito es hacer y recibir llamadas remotas como si fueran locales.
- Marshaling de datos: El proceso de empaquetar una cadena de datos que contiene el nombre de una llamada y sus parámetros.
- Lenguaje de definición de interfaz (IDL): Un tipo de documento que lista el nombre y los parámetros de los procedimientos que los clientes remotos pueden invocar.

La figura 1.1 muestra una estructura RPC típica. Para invocar una función remota, el cliente hace llamadas al proxy del cliente. El proxy empaqueta los parámetros de la llamada en un mensaje de solicitud e invoca el protocolo de transporte para que lleve el mensaje al servidor. En el lado del servidor, el protocolo de transporte entrega el mensaje al stub del servidor, este entonces desempaca el mensaje de solicitud y llama la función correcta en el objeto.

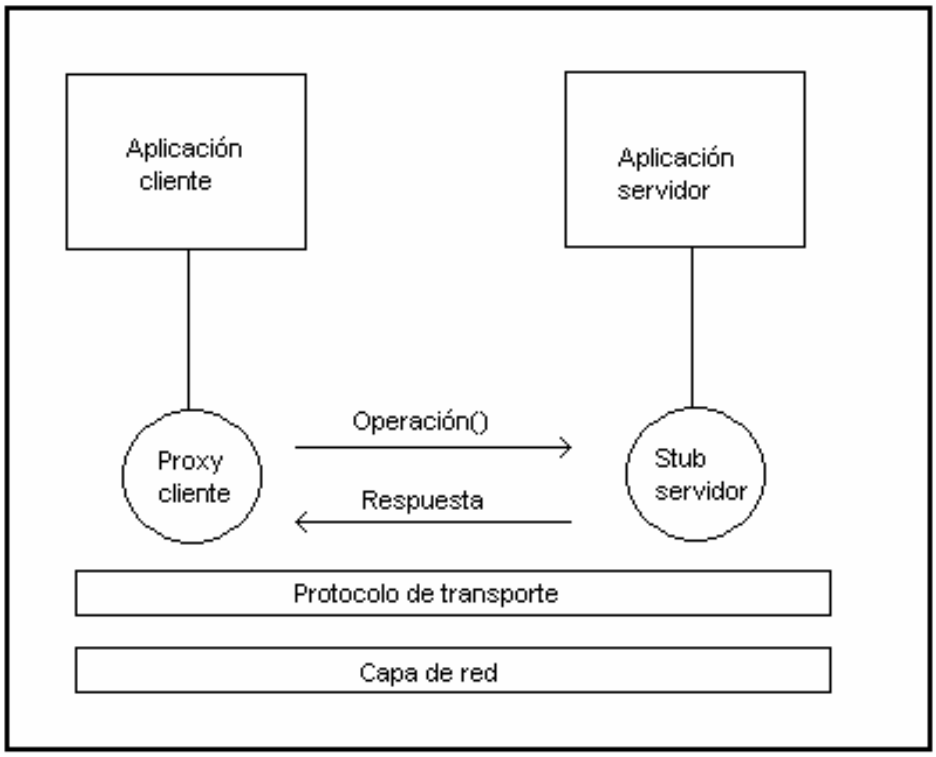


Figura 1.1. Arquitectura RPC Básica

Durante varios años CORBA y DCOM fueron los estándares que competían y marcaban pautas en el desarrollo de aplicaciones distribuidas. En ambos, la interacción entre los procesos cliente y servidor de objetos son implementados como comunicaciones, estilo RPC, orientadas a objeto [4]

1.3 Distributed COM (DCOM)

DCOM es la extensión distribuida del COM de Microsoft (Component Object Model) [5], la cual construye una capa de llamadas procedurales a objetos remotos (ORPC) sobre

DCE RPC [6] para soportar objetos remotos. Un servidor COM puede crear instancias de objetos de múltiples clases. Un objeto COM puede soportar múltiples interfaces, cada una representando una vista o comportamiento diferente del objeto. Un cliente COM interactúa con un objeto COM adquiriendo un puntero a una de las interfaces del objeto e invocando métodos a través de ese puntero, como si el objeto residiera en el espacio de direcciones del cliente. COM específica que cualquier interfaz debe seguir un esquema de memoria estándar, que es el mismo que el de la tabla virtual de funciones de C++[7]. Como la especificación es a nivel binario, permite la integración de componentes binarios escritos en diferentes lenguajes de programación como C++, Java, y Visual Basic.

1.4 Common Object Request Broker Architecture (CORBA)

CORBA es una plataforma de objetos distribuidos propuesta por un consorcio de más de 700 compañías, llamado Object Management Group (OMG) [8]. Su objetivo fue definir una arquitectura que permitiera la comunicación entre ambientes heterogéneos a nivel de objeto, sin importar quien diseñase los puntos finales de la aplicación distribuida.

Consta de un Lenguaje de Definición de Interfaz (IDL) y un API que posibilita la interacción entre el cliente y el servidor a nivel de objeto dentro de una implementación específica de un Object Request Broker (ORB). El ORB es el núcleo de esta arquitectura y no es más que una capa intermedia que establece la relación cliente-proveedor entre objetos distribuidos. Versiones posteriores de CORBA especificaron como ORBs de diferentes vendedores podían interoperar.

Un objeto CORBA es representado para el mundo exterior como una interfaz con un conjunto de métodos. Una instancia particular de un objeto es identificada por una referencia al mismo. El cliente de un objeto CORBA adquiere la referencia a su objeto y la usa como un manipulador (handle) para hacer llamadas a métodos, como si el objeto estuviese en el espacio de direcciones del cliente. El ORB es responsable de todos los mecanismos requeridos para encontrar la implementación del objeto, prepararlo para recibir un pedido, comunicarle el pedido, y enviar la respuesta (si existe) de regreso al cliente.

1.5 DCOM y CORBA en el diseño de sistemas distribuidos.

Aunque DCOM y CORBA han sido implementadas en varias plataformas, la realidad es que cualquier solución construida sobre estos protocolos será dependiente de la implementación de un simple vendedor. Por ejemplo, en el desarrollo de una aplicación usando DCOM, todos los nodos participantes deberán correr sobre alguna versión del Sistema Operativo Windows, pues DCOM es propietaria de Microsoft y no se ha implementado sobre otros sistemas operativos, lo cual limita la extensibilidad de una solución basada en esta tecnología.

En el caso de CORBA no ocurre esto, pues aplicaciones en varios sistemas operativos pueden comunicarse a través del ORB, pero al ser CORBA una especificación que consta de diferentes implementaciones dependiendo del vendedor, cualquier nodo en el ambiente de la aplicación necesitará correr usando el mismo producto ORB. Existen implementaciones que permiten la interoperabilidad entre ORBs CORBA de diferentes vendedores. Sin embargo esta interoperabilidad no es fácil de lograr por parte de los programadores. Además, cualquier optimización específica de un vendedor de ORB se perdería en esa situación.

Ambos protocolos dependen de un ambiente fuertemente administrado. DCOM y CORBA son ambos protocolos razonables para comunicaciones en el ambiente para el cual fueron diseñados. Por ejemplo, si todas las aplicaciones que quieren comunicarse están en la misma una intranet sobre Windows entonces tendría sentido usar DCOM para lograr la interoperabilidad entre ellas. O si se quiere lograr interoperabilidad entre aplicaciones sobre diferentes sistemas operativos en una intranet podría usarse CORBA. Sin embargo, cuando las máquinas clientes están distribuidas a través de Internet la comunicación debería hacerse usando otro estándar pues el paso de mensajes a través de firewalls usando estas tecnologías muchas veces limita la efectividad del firewall, lo cual no es deseable pues se comprometería la seguridad de todo el sistema. Debería utilizarse otro estándar más seguro y escalable.

1.6 Java Remote Method Invocation (Java RMI)

Java Remote Method Invocation (Java RMI [9]) es la versión orientada a objetos de Remote Procedure Calls (RPC). Mientras RPC permite llamar procedimientos a través de la red, Java RMI invoca un método de un objeto a través de la red.

En el modelo RMI, el servidor define los objetos que el cliente puede usar remotamente. El cliente puede entonces invocar métodos de este objeto remoto como si fuera un objeto local corriendo en su misma máquina virtual.

RMI oculta el mecanismo subyacente mediante el cual se transportan los argumentos de los métodos y sus valores de retorno a través de la red. El tipo de un argumento o valor de retorno puede ser cualquiera de los tipos primitivos de Java o cualquier otro objeto de Java serializable.

1.6.1 Definición de interfaz

El lenguaje Java permite la definición de interfaces, las cuales aíslan al programador de los detalles de la implementación y establecen un contrato entre la clase que implementa la interfaz y el objeto que invoca los métodos de la misma.

RMI hace uso de las interfaces para declarar las funciones que exporta un objeto remoto. Luego estas funciones son implementadas por una clase en el cliente y sobre esta implementación se ejecuta la herramienta **rmic** que genera un stub y un skeleton para las funciones de la interfaz. El stub se coloca en el cliente y el skeleton en el servidor.

1.6.2 Arquitectura de Java RMI

RMI utiliza el mismo mecanismo que los sistemas RPC para implementar la comunicación con los objetos remotos, el basado en stubs y skeletons. En RMI al Proxy cliente se le llama stub y al stub servidor se le denomina skeleton.

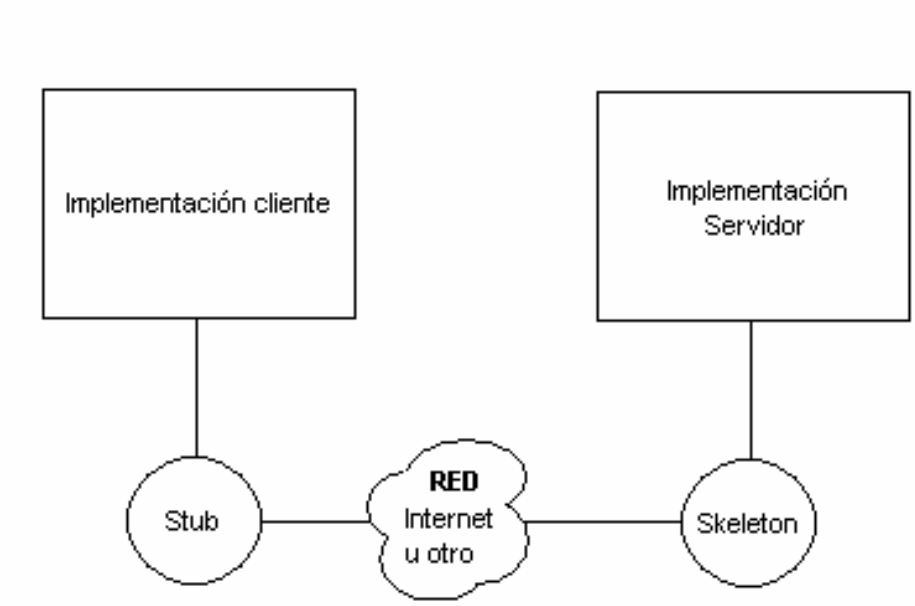


Figura 1.2. Arquitectura de Java RMI

Los stubs actúan como representantes de los objetos remotos ante sus clientes. En el cliente se invocan los métodos del stub, quien es el responsable de invocar de manera remota al código que implementa el objeto remoto.

Cuando se invoca algún método de un stub, este realiza las siguientes acciones:

1. Inicia una conexión con la Máquina virtual que contiene al objeto remoto.
2. Empaqueta (marshaling) y transmite los parámetros de la invocación a la Máquina Virtual remota.
3. Espera por el resultado de la invocación.
4. Desempaqueta (unmarshals) y devuelve el valor de retorno o la excepción.
5. Devuelve el valor a quien lo llamó.

Los stubs se encargan de ocultar el empaquetado de los parámetros, así como los mecanismos de comunicación empleados. En la Máquina Virtual remota, cada objeto debe poseer su esqueleto. El esqueleto es responsable de despachar la invocación al objeto remoto.

Cuando un esqueleto recibe una invocación, realiza las siguientes acciones:

1. Desempaqueta los parámetros necesarios para la ejecución del método remoto.
2. Invoca el método de la implementación del objeto remoto.
3. Empaqueta los resultados y los envía de vuelta al cliente.

1.6.3 RMI Registry

Hasta este punto no se ha mencionado como se lleva acabo la conectividad entre las funciones remotas, esto al igual que cualquier otra comunicación de Red se lleva acabo mediante un puerto TCP/IP. Esta comunicación requiere definir lo que es denominado RMI Registry, el RMI Registry se establece en un puerto TCP/IP (por defecto 1099) y mantiene un mapa de las funciones remotas que serán utilizadas, esto es, cuando arriba la petición para una función remota en el puerto conocido, el RMI Registry será encargado de redireccionar a la implementación apropiada.

La utilización de un puerto TCP/IP implica que de alguna manera la función remota debe ser ubicada a través de un mecanismo, esto generalmente se hace mediante un RMI URL (Universal Resource Locator) como rmi://java.sun.com/transaccion (Note la similitud con un HTTP URL usado en Páginas Web), sin embargo, la utilización de RMI URL tiene una grave deficiencia: Estos deben ser definidos explícitamente (Hard-Coded) en las funciones remotas , por lo tanto si la función remota cambia de servidor requerirá cambiar el código fuente, la alternativa de RMI URLs es emplear JNDI "Java Naming Directory Interface" lo cual permite utilizar información que resida en un directorio distribuido LDAP el cual resulta más fácil modificar que código fuente.

Finalmente cabe mencionar que bajo TCP/IP, RMI emplea el protocolo llamado JRMP ("Java Remote Method Protocol") para ejecutar métodos/funciones en los diversos objetos del sistema, esto a diferencia de CORBA que utiliza IIOP.

1.6.4 Utilización de RMI a través de firewalls por medio de Proxies

Normalmente, la capa de transporte de RMI intenta abrir sockets directamente a puertos de los servidores a los que se desea conectar, pero en el caso de que los clientes se encuentren detrás de algún firewall que no permita este tipo de conexiones, la capa de transporte estándar de RMI proporciona un mecanismo alterno basado en el protocolo HTTP (confiable para el firewall), para permitir a los clientes que se encuentran detrás del firewall, invocar métodos de objetos que se encuentren del otro lado de él.

Para atravesar el firewall, la capa de transporte de RMI incluye la llamada remota dentro del protocolo HTTP, como el cuerpo de una solicitud POST, mientras que los valores de retorno se reciben en el cuerpo de la respuesta HTTP. La capa de transporte de RMI puede formular la solicitud POST, de alguna de las dos maneras siguientes:

1. Si el proxy firewall permite entregar una solicitud HTTP directamente sobre cualquier puerto de la máquina destino, la solicitud se dirige directamente al puerto donde el servidor RMI se encuentra escuchando. La capa de transporte RMI en la máquina destino escucha con un socket que es capaz de entender y decodificar llamadas RMI, que se encuentren dentro de una solicitud POST.
2. Si el proxy firewall sólo entrega solicitudes HTTP a ciertos puertos HTTP bien conocidos, la llamada se envía a un servidor HTTP que se encuentre escuchando en el puerto 80, donde se puede ejecutar un guión (script) CGI que se encargue de enviar la llamada RMI al puerto específico del servidor.

El sistema de transporte de RMI especializa la clase java.rmi.server.RMISocketFactory, para proporcionar una implementación por defecto de una fábrica de sockets que se encargue de proveer tanto a clientes como servidores de dichos recursos. La fábrica de sockets por defecto, crea sockets que proporcionan el mecanismo para hacer transparente el paso por firewalls.

- Los sockets clientes automáticamente intentan conexiones HTML cuando el servidor no puede contestar directamente.
- Los sockets del lado del servidor, automáticamente detectan si es que una nueva conexión se trata de una solicitud POST de HTTP. En ese caso, devuelven un socket que únicamente expone el cuerpo de la solicitud al sistema de transporte y da formato a sus salidas, como una respuesta HTML.

1.6.5 RMI en el diseño de sistemas distribuidos.

La invocación remota de métodos en Java parte del hecho de correr sobre una plataforma homogénea, por tanto RMI forma parte de todo Java Developer Kit (JDK) , por ende, cualquier plataforma que tenga acceso a un JDK también tendrá acceso a estos procedimientos.

RMI está diseñada para tomar ventaja de esta característica, lo que le permite presentar propiedades que otros modelos de objetos como CORBA no poseen. RMI posee todas las características de seguridad que hereda de la plataforma Java misma como la recolección de basura distribuida y el manejo de hilos. Brinda soluciones para traspasar firewalls sin comprometer la efectividad de los mismos. Pero a diferencia de CORBA, que posee una arquitectura que proporciona independencia del lenguaje de programación, RMI está diseñada exclusivamente para Java lo cual hace más recomendable su uso cuando los sistemas que se quieren implementar están basados en tecnología Java.

1.7 .Net Remoting

Con el advenimiento de la plataforma .NET, Microsoft introdujo un conjunto de nuevas tecnologías, .NET remoting [10] es una de ellas, la cual provee un sistema genérico que usan diferentes aplicaciones para comunicarse entre ellas. Los objetos .NET son expuestos a procesos remotos, permitiendo por tanto la comunicación interprocesos. Las aplicaciones pueden estar localizadas en la misma computadora, en diferentes computadoras, en la misma red o incluso en diferentes computadoras en diferentes redes.

.NET Remoting utiliza una arquitectura flexible y extensible y usa el concepto de .NET llamado Dominio de Aplicación (AppDomain) para determinar su actividad. Un AppDomain es una construcción abstracta para asegurar el aislamiento de datos y código, sin tener que depender de conceptos específicos del sistema operativo como procesos o hilos. Un proceso puede contener múltiples AppDomains pero un AppDomain puede solo existir en exactamente un proceso. Si una llamada desde la lógica del programa cruza la frontera de un AppDomain entonces .NET Remoting entra en acción. Un objeto es considerado local si reside en el mismo AppDomain del que lo

invoca. Si el objeto no está en el mismo AppDomain del invocador entones es considerado remoto.

1.7.1 Diferentes tipos de objetos remotos.

La infraestructura de .Net Remoting permite crear dos tipos distintos de objetos:

1. **Objetos activados por el cliente:** Un objeto activado por el cliente es un objeto del lado del servidor cuya creación y destrucción es controlada por la aplicación cliente. Una instancia del objeto remoto es creada cuando el cliente llama el operador **new** en el objeto del servidor. Esta instancia vive mientras el cliente la necesite, y vive a través de una o varias llamadas a sus métodos. El objeto estará sujeto a la recolección de basura una vez que se determine que ningún cliente lo necesita.
2. **Objetos activados por el servidor:** El tiempo de vida de un objeto activado por el servidor es manejado por el servidor remoto, no por el cliente que instancia el objeto. Esto difiere de los objetos activados por el cliente, donde el cliente decide cuando no necesita más un objeto. Es importante entender que los objetos activados por el servidor no se crean cuando el cliente llama a **New** o **Activator.GetObject**. Ellos se crean cuando el cliente invoca un método en el proxy. Hay dos tipos de objetos activados por el servidor. Estos son:
    - **Llamada simple:** Los objetos de llamada simple manipulan un y solo un pedido del cliente. Cuando el cliente llama un método en un objeto de llamada simple, el objeto mismo se construye, realiza la acción por la cual fue llamado su método, y entonces el objeto puede ser destruido por el mecanismo de recolección de basura. No se mantiene el estado entre llamadas, y para cada llamada (no importa de que cliente viene) se crea una nueva instancia de un objeto.
    - **Singleton:** La diferencia entre singleton y llamada simple radica en el manejo del tiempo de vida. Mientras los objetos de llamada simple por naturaleza no mantienen estados, los singletons si lo hacen, lo cual significa que pueden ser utilizados para mantener estados a través de múltiples llamadas. Una instancia de un objeto singleton atiende a múltiples clientes, permitiendo a esos clientes que puedan compartir datos entre ellos.

1.7.2 Arquitectura de .Net Remoting

En la arquitectura de .Net Remoting [11] el proceso de una llamada remota es el siguiente:

Cuando un cliente llama el método remoto, el proxy recibe la llamada, codifica o empaqueta el mensaje usando un formateador apropiado y entonces envía la llamada sobre un canal al proceso del servidor. Un canal de escucha en el AppDomain del servidor recibe el pedido y se lo reenvía al sistema remoto del servidor, el cual localiza e invoca los métodos en el objeto solicitado.

Una vez completada la ejecución, el proceso se hace a la inversa y los resultados son retornados al cliente.

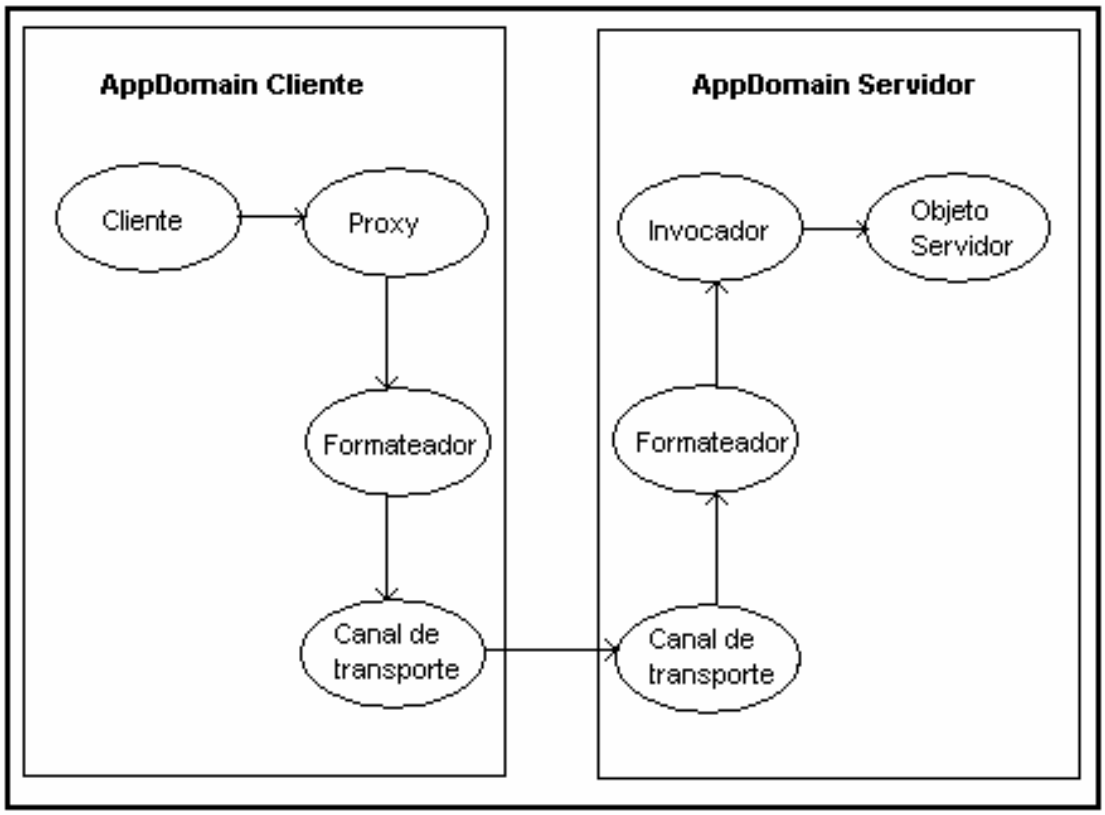


Figura 1.3. Arquitectura de .Net Remoting.

La plataforma Remoting viene con dos formateadores: un formateador binario y un formateador SOAP. El formateador binario es muy rápido, y codifica las llamadas a los métodos en un formato binario propietario. El formateador SOAP es más lento, pero permite a los desarrolladores codificar los mensajes remotos en formato SOAP. Si

ningún formateador de la plataforma cumple con las necesidades de desarrollo, los desarrolladores son libres de escribir sus propios formateadores.

Los sistemas remotos son accedidos a través de canales. Los canales transportan físicamente los mensajes hacia y desde los objetos remotos. Normalmente existen dos canales, TcpChannel y HttpChannel. Sus nombres se deben al protocolo que usan. Adicionalmente, los canales TcpChannel o HttpChannel pueden ser extendidos, o un nuevo canal puede ser creado si se determina que los canales existentes no satisfacen las necesidades del proyecto.

1.7.3 Utilización de .Net Remoting a través de firewalls por medio de Proxies

.Net Remoting evolucionó de tecnologías como DCOM. DCOM está basada en un protocolo binario que no todos los modelos de objeto soportan y que limita la interoperabilidad entre plataformas. La comunicación de DCOM se realiza a través de un rango de puertos que tradicionalmente están bloqueados por el servidor.

.NET Remoting elimina estas dificultades pues soporta diferentes formatos para los protocolo de transporte y comunicaciones. Lo cual permite que sea adaptable al ambiente de red en el cual esta siendo usado. Concretamente usando el formateador SOAP y el canal de comunicaciones HttpChannel se puede lograr la comunicación entre aplicaciones detrás de firewalls y sobre diferentes sistemas operativos.

1.7.4 .Net Remoting en el diseño de aplicaciones distribuidas

.Net Remoting es una fuerte tecnología que provee una cómoda plataforma para el desarrollo de aplicaciones distribuidas. Para las aplicaciones que requieran comunicación con otros componentes de la plataforma .NET probablemente esta sea la mejor opción pues se puede integrar con los servidores y otros servicios de la misma.
.NET Remoting garantiza la interoperabilidad con otras soluciones a través del uso de formateador SOAP y el canal de comunicación HTTP lo cual le permite atravesar firewalls. Sin embargo existen otras tecnologías, como los servicios Web que son más adecuados para lograr interoperabilidad entre diferentes estándares. Podrían combinarse ambas soluciones utilizando .NET Remoting para la comunicación con clientes .NET de una intranet y servicios Web para la comunicación con clientes externos en otras plataformas y así tomar ventajas de lo mejor de ambas propuestas.

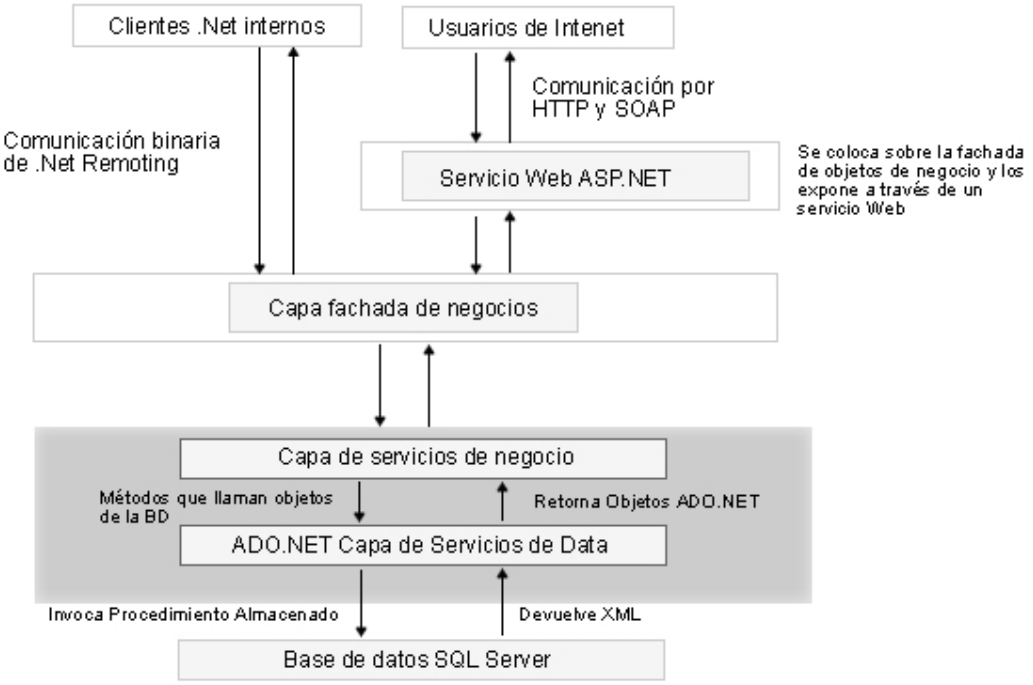


Figura 1.4. Combinación de .Net Remoting y servicios Web.

1.8 XML-RPC

XML-RPC es una especificación y un conjunto de implementaciones que permiten que software corra en diversos sistemas operativos y ambientes, hacer llamadas a procedimientos a través de Internet.

Esto es llamadas a procedimientos remotos (RPC) usando HTTP como transporte y XML como codificador. XML-RPC fue diseñado para ser tan simple como sea posible, y que a su vez permita que estructuras de datos complejas sean transmitidas, procesadas y retornadas. [12]

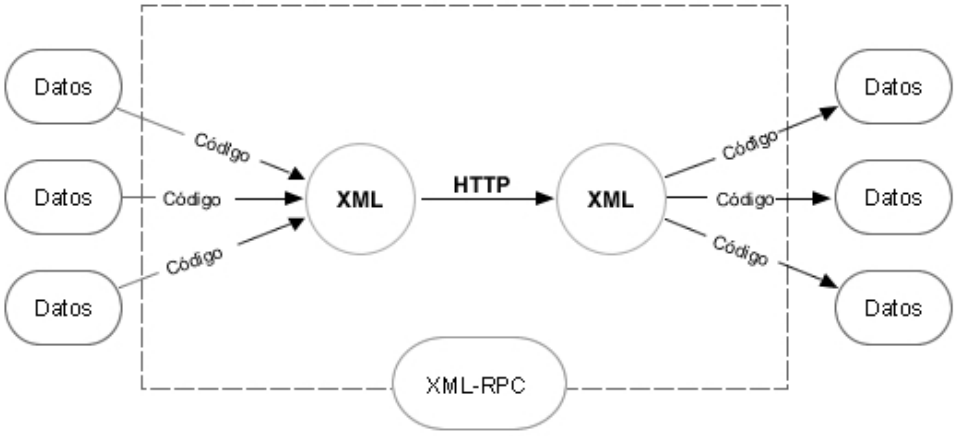


Figura 1.5. Arquitectura XML-RPC

1.8.1 Objetivos específicos de XML-RPC

Paso a través de firewalls. El objetivo de este protocolo es establecer una plataforma compatible a través de diferentes ambientes, de modo que el software del firewall espere mensajes POST cuyo contenido sea texto o XML.

Fácil Comprensión. Plantear un formato extensible que es muy simple. Debe ser posible para un programador Web, de modo fácil, revisar un archivo conteniendo una llamada XML-RPC, entender que es lo que hace, y ser capaz de modificarlo y ponerlo a punto en el primer intento.

Fácil implementación. Se pensó en un protocolo fácil de implementar que pudiera ser rápidamente adaptado para que corra en otro ambiente o sistema operativo.

1.8.2 XML-RPC en el diseño de aplicaciones distribuidas.

XML-RPC presenta un número de limitaciones que hay que tener en cuenta a la hora de diseñar sistemas distribuidos complejos, como por ejemplo:

Llamadas a métodos: XML-RPC llama los métodos remotos a través de su propiedad methodName, la cual solo puede contener caracteres como letras mayúsculas y minúsculas y los caracteres numéricos desde el 0-9, guión bajo (_), punto, dos puntos y

slash. Para la mayoría de los problemas esto está bien, sin embargo las cosas se complican cuando se trata de pasar un objeto como argumento.

Nombramiento de estructuras de datos: Las estructuras y arreglos son siempre anónimos. Cuando se pasan múltiples estructuras o arreglos los programadores dependen del orden de los parámetros para diferenciar entre cada estructura o arreglo. Esto no es un problema demasiado complicado pero hay situaciones donde ayudaría mucho poder nombrar las estructuras y los arreglos.

Simplicidad: La simplicidad de XML-RPC es también su gran limitación. Aunque la mayoría de los problemas relacionados con llamadas a procedimientos remotos pueden resolverse usando esta tecnología, hay cosas que simplemente no se pueden hacer de manera sencilla, como pasar un objeto como argumento de una función, o especificar a que parte específica de la aplicación receptora va dirigido un mensaje.

En resumen XML-RPC es una tecnología simple, fácil de entender, basada en peticiones y respuestas, que posibilita la interoperabilidad entre aplicaciones corriendo sobre diferentes sistemas operativos y a través de firewalls. De esta forma se pueden implementar muchas soluciones de forma muy sencilla. Ahora bien, si los conjuntos de información que se quieren transferir son más complejos, convendría analizar propuestas que utilicen otros protocolos, como por ejemplo SOAP.

SOAP, requiere una profusa especificación de marcas para los atributos, espacios de nombres (namespaces), y otros aspectos más complejos para describir exactamente que es lo que se está enviando. Lo cual tiene sus ventajas y desventajas. SOAP trae consigo una mayor sobrecarga pero adiciona mucha más información sobre lo que se envía. Si lo que se requiere es la posibilidad de enviar tipos de datos complejos definidos por el usuario y la habilidad de que cada mensaje defina como debe ser procesado entonces SOAP es una mejor solución que XML-RPC. Pero si tipos de datos estándar y llamadas a métodos simples es lo que se necesita entonces XML-RPC le dará al desarrollador aplicaciones más rápidas con menos trabajo.

1.9 Servicios Web (Web Services)

Recientemente, los Servicios Web han estado emergiendo como una plataforma sistemática y extensible para interacciones entre aplicaciones, construida sobre protocolos Web existentes y estándares XML abiertos [13].

Los servicios Web son un nuevo tipo de aplicación Web. Son aplicaciones modulares auto contenidas y auto descritas, que pueden ser publicadas, localizadas e invocadas a través de la Web. Pueden realizar funciones sencillas como simples llamadas solicitando información o crear y ejecutar complejos procesos de negocio. Una vez que el servicio Web se ha desplegado, este puede ser descubierto e invocado por otras aplicaciones (u otros servicios Web)

La ventaja clave de los servicios Web es la habilidad de crear aplicaciones al vuelo a través del uso de componentes de software reusables y débilmente acoplados. Esto tiene implicaciones importantes en las tecnologías y aplicaciones de negocios. El software puede ser entregado y pagado como una cadena de servicios a diferencia de la manera tradicional, productos empaquetados.

Es posible lograr una interoperabilidad dinámica y automática entre sistemas para realizar tareas de negocio. Los servicios de negocio pueden estar completamente descentralizados y distribuidos a través de Internet y pueden ser accedidos por una variedad de dispositivos de comunicación. Los negocios pueden ser liberados de la incómoda tarea que es la integración de software, y en ves de esto enfocarse en el valor de las ofertas y otras misiones más importantes.

La plataforma de los servicios Web está dividida en tres áreas:

1. Protocolos de comunicación (SOAP).
2. Descripción del servicio (WSDL).
3. Descubrimiento del servicio (UDDI)

1.9.1 SOAP

Simple Object Access Protocol (SOAP [14]), posibilita la comunicación entre servicios Web y es fundamentalmente un paradigma sin estados, para el intercambio de mensajes en un solo sentido que posibilita a las aplicaciones la creación de patrones de

interacción más complejos (por ejemplo. pedido/respuesta, pedido/múltiples respuestas, etc.) combinando intercambios en un solo sentido con características brindadas por otro protocolo subyacente y/o información específica de aplicación.

En su núcleo un mensaje SOAP tiene una estructura muy simple: Un elemento XML con dos elementos hijos, uno contiene el encabezamiento y el otro el cuerpo. El contenido del encabezamiento y el cuerpo son representados también en XML.

Los mensajes SOAP pueden ser transportados sobre HTTP. EL protocolo HTTP hace el papel de puente en las interacciones entre sistemas de computadoras.

1.9.2 Web Services Description Language (WSDL)

El lenguaje de descripción de servicios Web (WSDL [15]) brinda un lenguaje entendible por las computadoras para la descripción de los servicios Web.

WSDL provee un modelo y un formato XML para describir servicios Web. WSDL define los servicios como una colección de puertos o puntos finales en una red. En WSDL, la definición abstracta de puntos finales y mensajes no está atada a un tipo concreto de red o formato de datos. Esto permite el reuso de definiciones abstractas de mensajes que describen los datos que se intercambian y los puertos que representan colecciones de operaciones.

1.9.3 Universal Description Discovery & Integration (UDDI)

UDDI [16] brinda un mecanismo para que los clientes encuentren los servicios Web [17]. Los servicios Web son importantes solo si usuarios potenciales pueden encontrar información suficiente para permitir su ejecución.

UDDI se enfoca en la definición de un conjunto de servicios que dan soporte a la descripción y descubrimiento de negocios, organizaciones y otros proveedores de servicios Web, también describe y descubre los propios servicios de que UDDI dispone y las interfaces técnicas necesarias para accederlos. Basado en un conjunto de estándares de la industria, que incluyen HTTP, XML, XML Schema, y SOAP, UDDI provee una infraestructura fundacional e interoperable para un ambiente de software

basado en servicios Web ya sea para servicios públicamente disponibles o para servicios solo expuestos dentro de una organización.

Un registro UDDI puede ser considerado como un servicio DNS para aplicaciones de negocio. Un registro UDDI tiene dos tipos de clientes: negocios que quieren publicar una descripción de un servicio (y sus interfaces de uso) y clientes que quieren obtener descripciones de servicios de cierto tipo y enlazarse al servicio por programación (usando SOAP). La información UDDI contiene cuatro niveles. El nivel superior es la entidad de negocio que provee la información general sobre una compañía, tal como su dirección, una corta descripción, información de contacto y otros identificadores generales.

Este tipo de información puede ser visto como las páginas blancas de UDDI. Asociada a cada entidad de negocio esta una lista de servicios de negocios, incluyendo la descripción de cada servicio y las categorías del servicio, por ejemplo, inversión, embarque, etc. Esto puede ser considerado como las páginas amarillas de UDDI. Dentro de un servicio de negocio, una o más plantillas de enlace definen las páginas verdes que brindan más información técnica sobre el servicio Web.

1.9.4 Los servicios Web en el diseño de aplicaciones distribuidas.

Los servicios Web fueron diseñados para atacar el problema de la integración de fuentes heterogéneas y hacer interoperar sistemas heterogéneos. Tecnologías como CORBA y RPC tenían los mismos objetivos, pero cada una con su propia infraestructura. Por tanto, las soluciones al problema de la integración de sistemas heterogéneos eran muy caras.

Una dificultad al usar con tecnologías distribuidas antes de CORBA es que había que lidiar con diferentes lenguajes. La solución de CORBA a este problema fue la creación de un único lenguaje de definición de interfaz (IDL) [18]. Aunque esta es una buena idea, el lenguaje IDL se parece mucho a C++ y los desarrolladores deben también entender aspectos del mismo para usar CORBA. Si se reemplazaran los lenguajes como IDL con especificaciones de alto nivel, seguro la comunicación se simplificaría usando mensajes más significativos. Los servicios Web, definidos como aplicaciones modulares auto contenidas y auto descritas, que pueden ser publicadas, localizadas e

invocadas a través de la Web han logrado este objetivo y pueden automáticamente entrar en el ambiente Internet, a diferencia de CORBA que fue primeramente usado en el ambiente intraempresarial (al menos inicialmente).

Los servicios Web son una buena opción para arquitecturas débilmente acopladas [19]. La arquitectura CORBA es mas adecuada para ambientes intra-empresariales, mientras que las características técnicas de los servicios Web los hacen más reusables y por tanto más apropiados para ambientes globales e inter-empresariales.

Se puede entonces explicar el éxito de los servicios Web mirándolos como una tecnología basada en un desacople máximo y por tanto máxima reusabilidad, disponible sobre la infraestructura económica existente (la Internet). Su poder no es tanto por la tecnología usada pues la idea de RPC no es nueva, sino porque ofrecen una solución Web-nativa basada en XML [20], de modo que podemos rápidamente diseñar, implementar y desplegar servicios Web en Internet.

SOAP es un protocolo ligero, dedicado al intercambio de información estructurada en un ambiente descentralizado y distribuido. Ha sido diseñado para ser independiente de cualquier modelo de programación en particular y otros aspectos semánticos específicos de implementación. Al estar basado en el protocolo HTTP en conjunto con XML como sintaxis de los mensajes, la frontera de un sistema de computadora es fácil de traspasar y la noción de un mejor RPC o incluso una mejor tecnología de manejo de componentes distribuidos salta a la vista.

1.9.5 Algunos problemas y soluciones de los servicios Web

SOAP, WSDL, y UDDI son importantes tecnologías que hacen funcionar a los servicios Web. Sin embargo, para satisfacer completamente los requerimientos de las aplicaciones de negocio, las tecnologías actuales tienen debilidades. La seguridad, la composición y la semántica de los servicios Web son algunos de estos problemas.

1.9.5.1 El problema de la seguridad

Básicamente, los problemas de seguridad que afectan a los servicios Web son los mismos que afectaron a los sistemas Web convencionales.

En este contexto esto quiere decir que el receptor del mensaje debe ser capaz de verificar la integridad del mismo para asegurarse que no ha sido modificado, debe haber recibido el mensaje confidencialmente, de modo que usuarios no autorizados no puedan leerlo o conocer la identidad del que lo envía. Esto normalmente se hace a través de mensajes encriptados [21].

Han surgido algunos estándares para aliviar el problema de la seguridad del mensaje, incluyendo WS Security y varias otras iniciativas, posibilitando la firma digital de mensajes y transacciones.

1.9.5.2 El problema de la composición

Cuando existen complejas interacciones de negocios se requiere de un soporte para un nivel alto de funcionalidades de negocio. Las interacciones entre negocios son típicamente largos procesos de ejecución e involucran múltiples interacciones entre socios.

Para desplegar y usar efectivamente estos tipos de servicios, debemos ser capaces representar procesos de negocios y estados de servicios de modo que creemos servicios compuestos (agregaciones complejas) en una forma estandarizada y sistemática. Varias propuestas para ejecutar esta tarea existen; por ejemplo, Web Services Flow Language [22], XLANG [23] and BPEL4WS [24], otra forma es que el propio servicio maneje internamente la composición de los servicios.

Recientemente, los términos orquestación y coreografía han sido usados para describir esto también. La orquestación describe como los servicios Web pueden interactuar entre ellos a nivel de mensaje, incluyendo lógica de negocio y orden de ejecución de las interacciones. Estas interacciones pueden agrupar múltiples aplicaciones y/o organizaciones, y resultar en un modelo de larga vida, transaccional y multiproceso.

1.9.5.3 Los problemas semánticos en los servicios Web

La tecnología actual de servicios Web básicamente provee una solución sintáctica y sigue careciendo de la parte semántica. Un servicio Web es descrito en un WSDL, señalando que entrada espera el servicio y que salida devuelve. Para lograr la orquestación podemos utilizar The Web Service Flow Language (WSFL), que deja al usuario decidir que servicio Web combinar y en que orden. Sin embargo, seguimos necesitando un ambiente que semánticamente describa el servicio, de modo los agentes de software puedan localizarlo, identificarlo, invocarlo y combinar estos servicios.

1.10 Arquitecturas orientadas a servicios.

La computación está entrando en una nueva era marcada fundamentalmente por el desarrollo de Internet, las aplicaciones móviles, la programación distribuida y los servicios Web. Ya hoy en día se cuenta con tecnologías sólidas que hacen que se puedan desarrollar aplicaciones que puedan ser consumidas por una gran variedad de dispositivos, sistemas operativos sin importar la ubicación geográfica. En esta nueva era de la computación también aparece un nuevo concepto de arquitectura para enfrentar los más diversos problemas: Las arquitecturas orientadas a servicios [25]

Una Arquitectura Orientada a Servicios es un paradigma para organizar y utilizar capacidades distribuidas bajo el control de diferentes dominios. La cual brinda una manera uniforme para ofrecer, descubrir e interactuar y utiliza estas capacidades para producir los efectos requeridos con precondiciones y expectativas medibles.

Cualquier arquitectura orientada a servicios tiene que ser capaz de manejar una serie de características:

- Independencia de la plataforma: Como uno de los objetivos fundamentales de los servicios es permitir la integración de estos con otros servicios o con aplicaciones, la independencia del sistema operativo es una de las características imprescindibles para lograr este objetivo.
- Estar basado en estándares: Este es un requisito indispensable para lograr la interoperabilidad entre aplicaciones basadas en servicios. Ejemplos de estándares son XML, HTTP. XML Schema, etc.

- Otras características importantes de SOA son el reuso, la granularidad, la modularidad, y la posibilidad de división en componentes, así como la composición de los servicios.

Muchos defensores de los servicios, en particular los servicios Web comienzan con la oración obligatoria de que esta es una solución de negocio que siempre provee agilidad. Sin embargo, hay una gran distancia entre esta necesidad de agilidad y la tecnología de servicios Web que es presentada como la solución. Mientras los servicios Web esencialmente ayudan a esto, lograr soluciones ágiles es más una consecuencia de un análisis bien pensado de las necesidades de negocio con la agilidad en mente y un diseño cuidadoso de la solución; que es arquitectónicamente diseñada para lograr agilidad.

Muchas empresas adoptan esta tecnología sin tener en cuenta que si no se realiza un diseño correcto de los servicios Web y sus relaciones con otros servicios Web internos o externos pueden dejar de ser útiles para sus clientes al cambiar el proceso de negocio o las necesidades de los clientes. Un modelo para el intercambio de datos en arquitecturas de este tipo ayudaría mucho a las empresas a diseñar soluciones orientadas a servicios que cumplan con los requerimientos de arquitecturas de este tipo.

Conclusiones parciales

1. Se demostró que los servicios Web brindan una solución adecuada cuando se quieren implementar arquitecturas desacopladas entre aplicaciones que corren sobre diferentes sistemas operativos y donde haya la necesidad de atravesar firewalls, sin embargo para sistemas intraempresariales otras tecnologías son más apropiadas.
2. El uso de servicios Web llevará a la Internet a convertirse en una plataforma global común donde las organizaciones e individuos se comunican para llevar a cabo diversas actividades comerciales y proveer servicios de valor agregado. Las barreras para proveer nuevas ofertas y entrar a nuevos mercados serán menores y posibilitarán un acceso más fácil a pequeñas y medianas empresas.
3. Los servicios Web son una tecnología que llegó para quedarse pero todavía tiene aspectos qué hay que pulir, como son la seguridad, la orquestación de servicios y la semántica de los servicios.
4. Se han analizado 7 tecnologías para la construcción de sistemas distribuidos y el intercambio de datos entre los mismos, 4 utilizan formatos propietarios para el intercambio de datos y 3 el estándar XML. Se demostró que el uso del formato XML facilita la construcción de aplicaciones que pueden intercambiar datos independientemente del vendedor, plataforma y lenguaje de programación, logrando así una verdadera interoperabilidad.
5. Cuando se utilizan múltiples servicios deben utilizarse arquitecturas orientadas a servicios (SOA) lo cual implica consideraciones adicionales de diseño.
6. El diseño de una arquitectura orientada a servicios Web no es cosa trivial, un modelo de intercambio de datos sería de gran ayuda para diseñar arquitecturas de este tipo pues guiaría a las empresas en la definición de los servicios Web que necesita y las interacciones de estos con otros servicios Web.

CAPÍTULO II

MODELO DE INTERCAMBIO DE DATOS PARA APLICACIONES EN DIFERENTES PLATAFORMAS, SUBREDES Y DISPOSITIVOS.

En este capítulo se expone una solución al problema científico planteado en la introducción de esta investigación, que toma como base el estado del arte plasmado en el marco teórico referencial y que consiste en un método para la implementación de un modelo de intercambio de datos para aplicaciones en diferentes plataformas, subredes y dispositivos. Primero se exponen las premisas generales para la construcción y luego los principios en que se sustenta el modelo. Finalmente se describe el modelo en cada una de sus etapas. El modelo consta de 4 etapas y propone la definición y coordinación de servicios y sus interfaces basado en el proceso de negocio de la empresa o cliente que desea implementar una arquitectura SOA. Hace uso además de patrones de diseño para resolver problemas comunes que se presentan en aplicaciones de este tipo.

2.1 Premisas generales para la construcción del modelo

El modelo hace énfasis en la agilidad que debe tener una arquitectura orientada a servicios (SOA). La construcción del modelo se realizó teniendo en cuenta las premisas de los consumidores y los proveedores de servicios Web pues esta es la tecnología que mayores potencialidades brinda en la construcción de sistemas desacoplados y distribuidos en diferentes plataformas. Se hace énfasis en el logro de un modelo ágil.

Premisas de los consumidores:

- Reducir el esfuerzo para usar nuevos servicios sobre soluciones basadas en servicios existentes
- Seleccionar instancias alternativas de un tipo de servicio.

Premisas de los proveedores:

- Reducir las demandas de nuevas funcionalidades por parte de nuevos consumidores.
- Componer nuevos servicios a partir de otros existentes..
- Reducir el impacto de los cambios en la implementación del servicio.
- Brindar servicio en un contexto nuevo e imprevisto.

- Brindar servicio a un rango de consumidores tan ancho como sea posible.

2.2 Principios en los que se sustenta el modelo

Los principios que sustentan el modelo desarrollado son los siguientes:

1. **Abstracción**: El principio de la abstracción se usa normalmente para asegurar que un servicio sea independiente de una implementación específica y otros detalles. Uno de los principios de la orientación a servicios es enfocarse en lo que un servicio hace no en como lo hace.

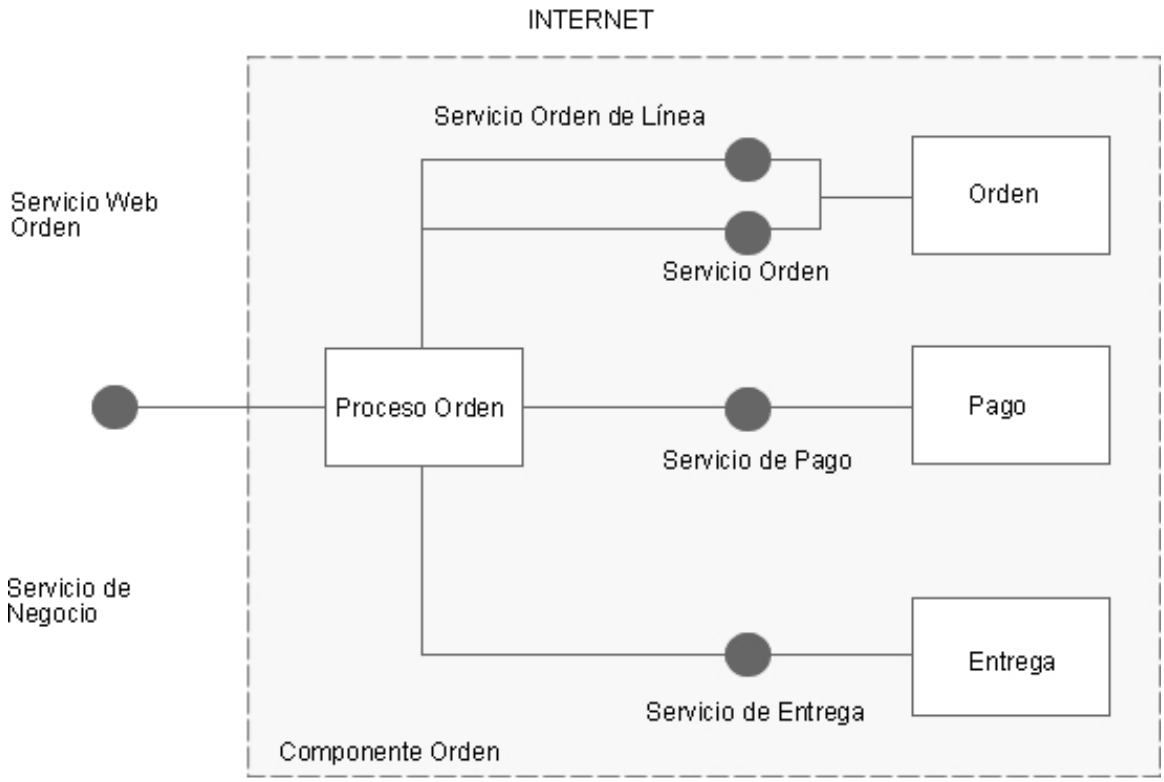


Figura 2.1. Componente que expone un servicio de negocio abstracto.

El propósito de este principio es:

1. El consumidor del servicio no tiene que conocer como usar todas las implementaciones individuales en las que están basados los servicios internamente para pedir una orden.
2. Desde una perspectiva ágil, la forma en la que el proveedor de servicios elige configurar sus componentes internos puede cambiar con el tiempo sin afectar el consumidor del servicio.

3. Los servicios internos siguen estando disponibles a otros desarrolladores en el proyecto que pueden usarlos para componer otros servicios de negocio.

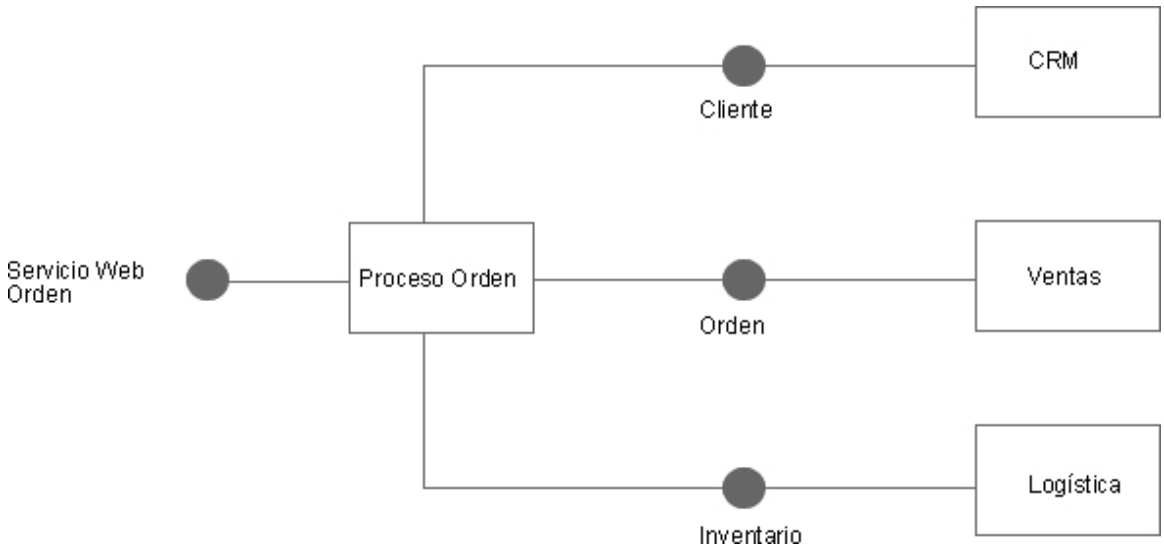


Figura 2.2 Un componente de procesamiento provee una fachada sobre implementaciones existentes.

Con más frecuencia el proceso de negocio es implementado por un número de aplicaciones existentes. Como se muestra en la figura 2.2 el mismo patrón es aplicable, un nuevo proceso basado en componentes abstrae el servicio de negocio de la implementación de los servicios internos. Nuevamente esos servicios internos siguen publicados, de modo que pueden ser utilizados para componer otros servicios con otros requerimientos de negocio.

2. **Generalización**: El principio de la generalización se aplica a la creación de un servicio de modo que este pueda ser usado en un número grande de escenarios, incluyendo escenarios inesperados, eliminando la necesidad de construir servicios específicos para cada nuevo requerimiento.

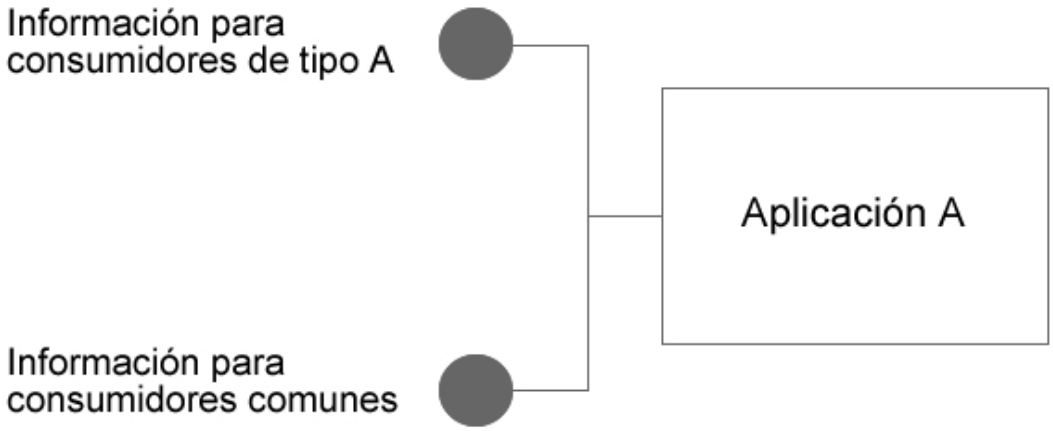


Figura 2.3 Un servicio generalizado separa lo común de lo específico.

Entre los objetivos para la generalización de servicios están:

- Separación de los datos y comportamientos comunes de los específicos, de modo que las partes comunes den respuesta a un conjunto amplio de requerimientos y puedan ser usados para componer otros servicios, como muestra la figura 2.3.
- Incluir un rango amplio de datos y comportamientos en el servicio que consumidores individuales del servicio pueden requerir, de modo que satisfaga las necesidades de un rango amplio de consumidores sin tener que implementar un servicio específico para satisfacer sus necesidades.

Estos aspectos pueden parecer contradictorios. Sin embargo, reflejan las dos etapas del proceso para entregar sistemas ágiles. Primero, descomponer el servicio para encontrar las partes comunes y las especializadas y luego componerlas para formar un servicio de granularidad gruesa.

Al usar la generalización para usar un servicio de granularidad gruesa tratamos de reducir el numero de servicios que necesitan ser expuestos y mantenidos. Por ejemplo puede reducir el mantenimiento que es necesario cuando el consumidor del servicio desea utilizar el servicio en un nuevo contexto, o cuando se da cuenta que necesita utilizar cierta información del servicio que antes no necesitaba. El servicio generalizado puede ayudarnos a prevenir cambios en la interfaz del servicio.

3. **Cumplimiento de estándares**: El cumplimiento de estándares no debe ser visto como un problema de principios, sino de conveniencia. Una razón clave es que los estándares ayudan a los sistemas a ser más adaptables pues tienen una inercia considerable y demoran mucho tiempo en cambiar. Desde el punto de vista de las arquitecturas orientadas a servicios los estándares hacen las cosas más cómodas lo cual puede ser atractivo para los consumidores de servicios pues esto los hace más ágiles al darles la posibilidad de seleccionar proveedores que cumplan con el mismo estándar. De manera similar, esto puede habilitar la entrada de un proveedor de servicios en un determinado sector de la industria simplemente cumpliendo los estándares existentes, sin necesidad de crear nuevos tipos de servicios.

   El cumplimiento de estándares debe implicar también un compromiso para mantener un servicio en línea con la evolución de los mismos.
   Las ventajas del cumplimiento de estándares incluyen:
   - Amplia compatibilidad entre proveedores y consumidores de servicios.
   - Los servicios basados en estándares existentes funcionarán para nuevos consumidores de servicios que cumplan con los estándares.
   - Los proveedores de servicios alternativos basados en estándares pueden ser utilizados con un impacto mínimo para las aplicaciones consumidoras, posibilitando en cambio dinámico de proveedores de servicios.

4. **Granularidad:** La granularidad se refiere al alcance de una funcionalidad proporcionada por un servicio Web. Se ha hecho la mejor práctica recomendar que los servicios Web sean de grano grueso. Esto es en parte una reflexión del hecho que la percepción inicial sobre los servicios Web era de un recurso que sería desplegado a través del Internet con una conexión lenta y no fiable. Entonces, usando un pequeño número de mensajes de grano grueso reduciría el número de transmisiones y aumentaría la probabilidad de llegada de mensajes y completamiento de transacciones mientras estuviese disponible la conexión. Aunque la calidad de la conexión vaya mejorando con el tiempo, es una buena práctica hacer esto para servicios Web externos a través del Internet. Un aspecto importante relacionado con la granularidad es que varía a través de las capas de la aplicación.

Los servicios Web pueden ser expuestos en cualquier capa de la aplicación, o sea, desde cualquier base de datos, objeto o componente. En cada una de las capas siguientes ilustradas en la Figura 2.4 habrá diferentes niveles de granularidad.

Capas como los objetos de negocio, la base de datos, componentes de negocio o componentes de proceso de negocio son ejemplos de capas con granularidad distinta, en general, a medida que los servicios se acercan a la aplicación servidora, son de granularidad más gruesa.

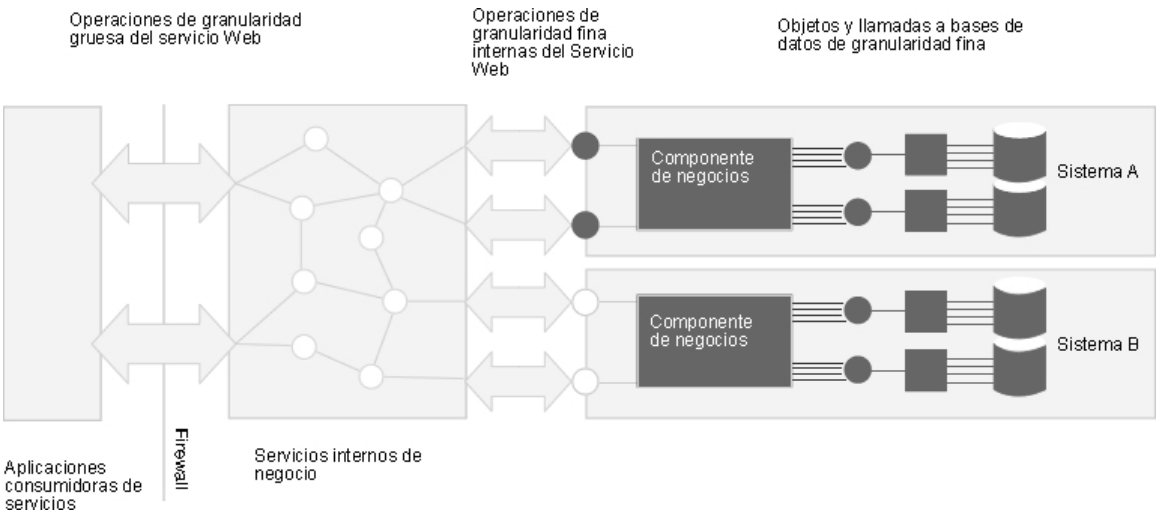


Figura 2.4 Aumento de la granularidad en el sistema.

Aunque las organizaciones no quieren una proliferación innecesaria de servicios Web debido a la sobrecarga de mantenimiento y desarrollo, no hay razón para fijar el acceso a un servicio a un nivel de granularidad determinado. Por ejemplo, un servicio de negocio puede ser expuesto como un servicio de granularidad gruesa para ser usado por clientes externos, y además un conjunto de servicios de granularidad fina para uso interno o socios de negocio, este modo tomamos lo mejor de ambas soluciones. Hay 2 formas de hacer esto, brindando un servicio paralelo o reusando servicios Web existentes y agregándolos en uno de granularidad más gruesa.

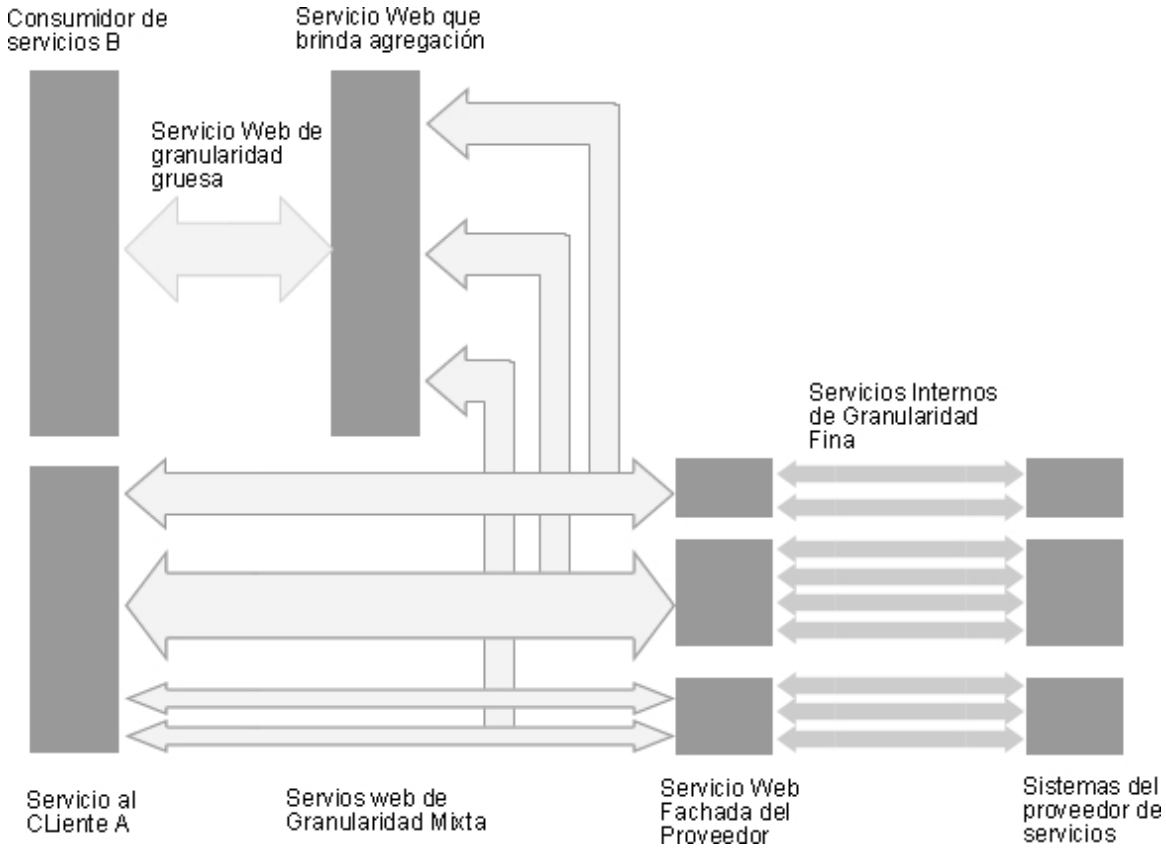


Figura 2.5 Agregación de servicios.

La figura 2.5 ilustra esto los servicios pueden ser entregados con granularidades diferentes para satisfacer exigencias diferentes. El consumidor A puede estar dentro de la misma organización y tener un transporte de alta velocidad disponible. Mientras, el cliente de servicios B tiene acceso a los sistemas a través de Internet.

5. **Puntos de articulación**

En SOA podemos introducir un numero que llamamos puntos de articulación alrededor del cual podemos introducir flexibilidad para el consumidor y el proveedor.

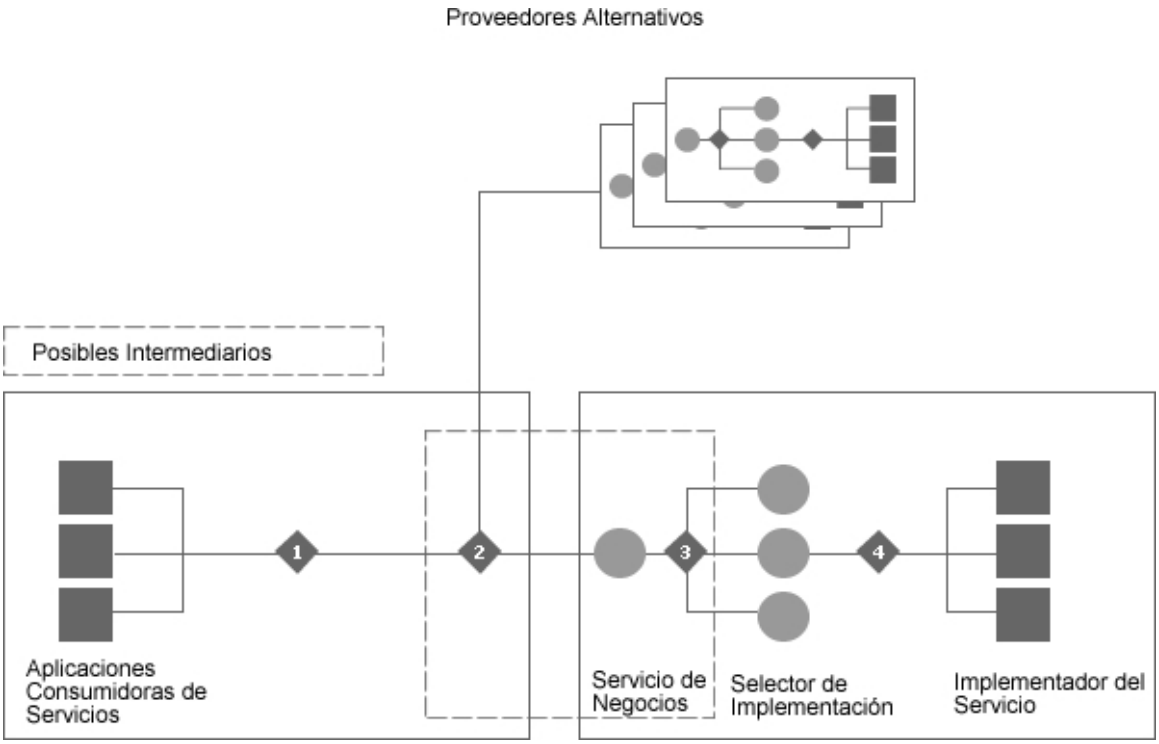


Figura 2.6 Puntos de Articulación

La figura 2.6 enumera algunos puntos de articulación en los cuales las siguientes decisiones pueden ser tomadas.

1. ¿Qué servicio deberá la aplicación consumidora utilizar?
2. ¿A qué proveedor de servicios se le deberá enviar el pedido?
3. ¿Qué implementación del servicio deberá recibir el pedido.
4. ¿Qué implementación deberá procesar el pedido?

A continuación se presenta un modelo que tiene en cuenta estos principios y que puede ser aplicado a problemas con características similares.

2.3 Un modelo de intercambio de datos basado en XML

Una de las cuestiones más difíciles de responder durante la implementación de una arquitectura orientada a servicios es como definir apropiadamente los servicios. Una empresa normalmente tiene un conjunto de aplicaciones que realizan la mayoría de las funcionalidades de negocio. Estas aplicaciones pueden estar implementadas usando una variedad de plataformas de hardware, sistemas operativos y lenguajes de programación. Algunas son aplicaciones "standalone" y otras pueden estar integradas usando técnicas para ese efecto. Algunas aplicaciones están implementadas por la empresa y otras por sus socios de negocio.

Una arquitectura orientada a servicios requiere la definición de servicios que realicen las funcionalidades y procesos de negocio requeridos y cumpla con requisitos técnicos como escalabilidad, agilidad, eficiencia entre otros.

El modelo que se propone plantea que los servicios deben definirse basados en el modelo de negocios de la empresa y luego recomponerlos de modo que cumplan con los requisitos técnicos o arquitectónicos de la empresa [26.].

Básicamente consta de 4 pasos:

1. Descomposición jerárquica basada en el modelo de negocios de la empresa de modo que se asegure que los servicios resultantes estén alineados con las funcionalidades de negocio de la empresa.
2. Especificar las interfaces de intercambio de mensajes y asegurar la interoperabilidad de los servicios resultantes.
3. Recomponer los servicios resultantes para asegurar el cumplimiento de los requisitos técnicos o arquitectónicos, como agilidad, eficiencia, escalabilidad entre otros.
4. Utilizar patrones de diseño en la jerarquía de servicios.

Si al implementar los servicios aparecen problemas de latencia, autonomía, disponibilidad, granularidad o visibilidad es necesario regresar al paso 3 del modelo hasta tanto se satisfagan estas restricciones.

2.3.1 Descomposición jerárquica basada en el modelo de negocio.

Desafortunadamente todas las empresas no tienen un modelo de negocios. Realmente muchos modelos de negocios están pobremente definidos, desactualizados o ni siquiera existen realmente, sin embargo no es necesaria una guía exhaustiva del negocio para definir los servicios, generalmente los modelos de alto nivel son suficientes para direccionar, particionar y establecer una taxonomía de servicios, el modelo de negocios puede ir evolucionando en paralelo con la definición de los servicios mientras la empresa continúa funcionando y evolucionando [27].

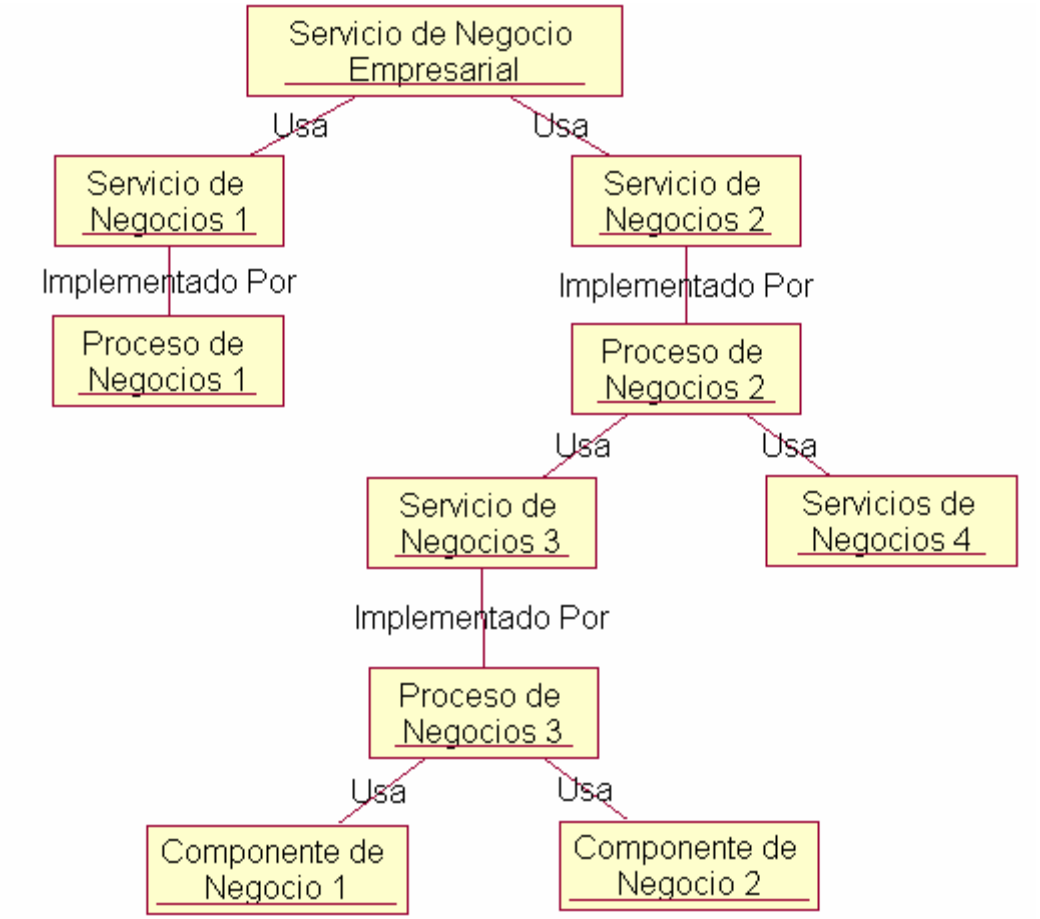


Figura 2.7 Procesos, servicios y componentes de negocio.

El particionamiento del servicio guiado por el modelo de negocios de la empresa corresponde a un diseño top-down. En el nivel más elevado el sistema empresarial puede ser representado como un proceso de negocios que orquesta muchos servicios. Cada uno de estos servicios puede ser descompuesto en subprocesos que orquestan servicios de nivel inferior.

Finalmente en el nivel más bajo los servicios son implementados como procesos de negocios internos que orquestan componentes de negocio. Estos componentes de negocio pueden ser parte de la implementación del servicio o servicios fachadas de aplicaciones existentes.

Muy unido a la descomposición jerárquica está la definición de las interfaces de los servicios identificados de modo que quede claro cuales son los mensajes usados para invocar servicios y entregar el resultado de la ejecución de un grupo de tareas de vuelta a los consumidores del servicio.

2.3.2 Definición de la interfaz de los servicios

La especificación de la interfaz de los servicios debe hacerse de modo que minimice el impacto de un posible cambio de la misma en el futuro.
Las interfaces en las Arquitecturas Orientadas a Servicios ocultan su implementación interna detrás de interfaces públicas. Una vez construidas las interfaces tienden a permanecer bastante estáticas con respecto a la configuración del servicio. Por tanto la interfaz de un servicio que hace varias cosas (soporta varios métodos) está sujeta a cambiar siempre y cuando algunos de sus métodos cambie.

De modo que debe descomponerse el problema garantizando que los servicios resultantes tengan interfaces simples.

Una vez hecho el diseño de los servicios y especificada su interfaz basados en la lógica de negocio de la empresa es necesario considerar los aspectos técnicos o arquitectónicos de la solución propuesta.

2.3.3 Recomposición teniendo en cuenta los aspectos técnicos.

Una solución basada en el modelo de negocios de la empresa brinda guías para definir los servicios Web de la empresa, pero no es suficiente en el contexto de una arquitectura orientada a servicios, los siguientes aspectos relacionados con los servicios Web deben ser tenidos en cuenta.

Latencia: En las aplicaciones SOA los servicios interactúan remotamente por lo que la latencia es elevada. Una descomposición inicial puede definir 3 servicios, A, B y C. Pero

si la interacción entre B y C tiene restricciones adicionales en cuanto a la latencia, entonces puede ser conveniente recomponer el diseño en 2 servicios A y BC.

Autonomía: Los servicios son autónomos. Una descomposición inicial de 3 servicios puede hacer dependiente el servicio A del B y el B del C, puede ser conveniente entonces recomponer estos servicios en uno solo ABC.

Disponibilidad: La invocación de un servicio está sujeta a fallas en la red o el servicio, por ejemplo si un servicio A muy importante, depende de los datos entregados por un servicio B y B no puede proveer los datos a A en un rango de tiempo tolerable entonces quizás sea conveniente implementar un solo servicio AB.

Granularidad: Los servicios pertenecientes a niveles altos en el proceso de negocios deberán tener una granularidad más gruesa que los de un nivel más bajo. Esto tiene sentido pues los servicios de alto nivel generalmente utilizan los servicios de bajo nivel para implementar sus propias funcionalidades.

Visibilidad: A medida que aumenta el nivel del servicio deberá aumentar su visibilidad. Por ejemplo un proceso negocio de nivel empresarial deberá ser visible por toda la empresa, otro de bajo nivel deberá ser visible solo en determinada área de la empresa, fuera de ese ambiente deberá ser visto indirectamente a través de otro servicio de un nivel superior.

2.3.4 Utilización de patrones de diseño en la jerarquía de servicios.

La mejor solución para la construcción de sistemas complejos siempre ha sido construir pequeños mecanismos que solucionan problemas bien específicos. A lo largo de la construcción del sistema estos mecanismos se combinan entre sí para construir el sistema completo.

El conocimiento de estos pequeños mecanismos no es ni mucho menos trivial, muchas veces viene de la experiencia de muchos desarrolladores que se han enfrentado muchas veces a los mismos problemas. En años recientes se dio a conocer al mundo una nueva forma de atacar los modelos complejos y que extiende las potencialidades del diseño orientado a objetos: el diseño basado en patrones [29].

Un patrón describe un problema recurrente que ocurre en un contexto determinado y recomienda una solución que puede ser la colaboración entre dos o más objetos, servicios, etc.

Los patrones proveen un mecanismo efectivo para el desarrollo de soluciones probadas que permiten a los desarrolladores nutrirse de la experiencia de otros. El diseño basado en patrones no resultaría demasiado útil si los patrones no describieran relaciones entre ellos. La unidad elemental del diseño orientado a objetos es la clase, sin embargo una clase tiene su máximo nivel de funcionalidad a partir de las relaciones que esta clase puede tener con otras clases y entre las instancias de ambas. Un patrón está compuesto por un conjunto de clases y no elimina la relación de estas con otras clases presentes en otros patrones. Luego en un diseño donde aparezcan miles de clases estas pueden ser representadas en patrones que son mucho más manejables [30].

En general, un patrón tiene cuatro elementos esenciales [29]:

Nombre: El nombre del patrón es un identificador que puede ser utilizado para describir un problema de diseño, su solución y consecuencias en una o dos palabras. Al nombrar un patrón inmediatamente se incrementa el vocabulario de diseño, lo cual permite diseñar a un nivel de abstracción más alto. Al tener un vocabulario para los patrones podemos hablar de estos con nuestros colegas, comprender mejor la documentación de un sistema en particular e incluirlos en nuestros propios documentos.

Problema: El problema describe cuando aplicar el patrón, explica el problema y su contexto. A veces el problema puede incluir una lista de condiciones que deben ser satisfechas antes de que se aplique el patrón.

Solución: La solución describe los elementos que forman el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe una implementación o diseño concreto porque los patrones son como plantillas que pueden ser aplicadas en diversas situaciones. En vez de eso los patrones proveen una descripción abstracta de un problema de diseño y un grupo general de elementos que lo resuelve.

Consecuencias: Las consecuencias explican las ventajas y desventajas de la aplicación del patrón. Son importantes para evaluar las alternativas de diseño y para comprender los costos y los beneficios de la aplicación del patrón. Generalmente las consecuencias de un patrón analizan el impacto del patrón en la flexibilidad, extensibilidad o portabilidad de un sistema.

La utilización de patrones de diseño es más beneficiosa cuando se combinan varios patrones dentro de diferentes capas de la aplicación. La figura 2.8 muestra la aplicación de este principio en el modelo propuesto.

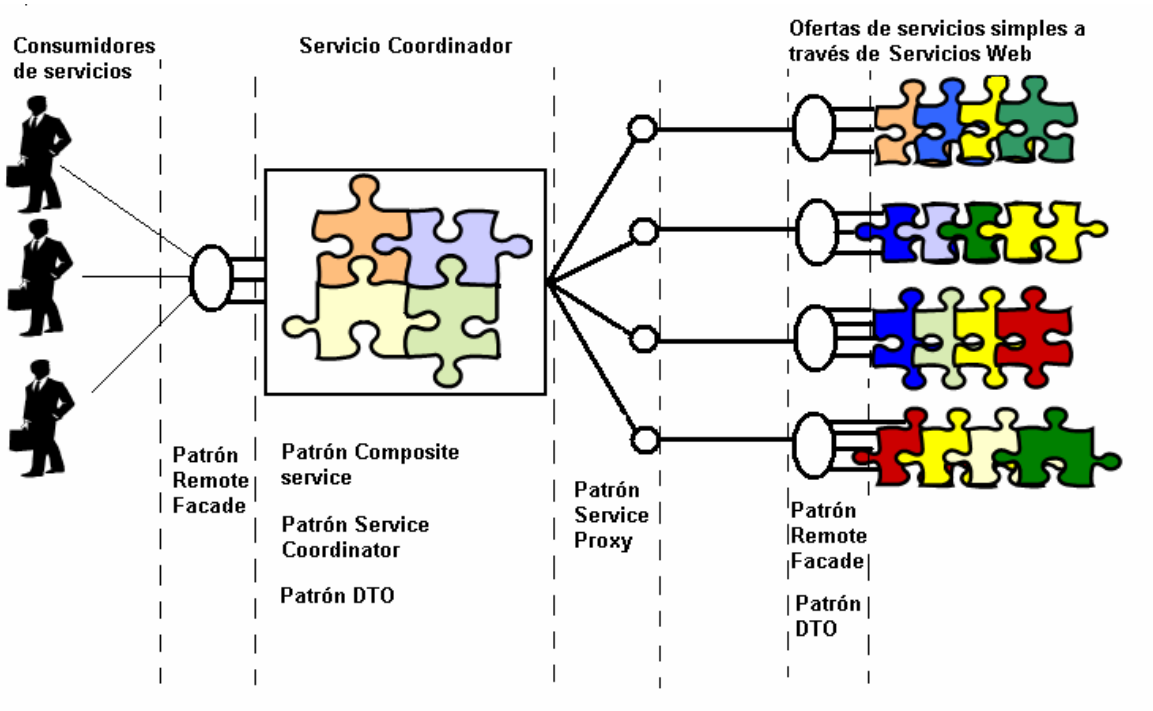


Figura 2.8 Patrones en la jerarquía de servicios.

A continuación se exponen los patrones propuestos, se separó el contexto del problema del propio problema para una mejor comprensión.

Patrón Remote Facade [31]

Contexto: Un cliente de un servicio Web necesita acceder remotamente a un modelo de objetos de granularidad fina. Estos objetos no tienen interfaz externa y constan generalmente de un número no pequeño de métodos.

Problema: En un modelo orientado a objetos, se trabaja mejor con objetos sencillos que tienen métodos sencillos. Lo cual da muchas oportunidades para controlar y sustituir comportamientos. Dentro de un único espacio de direcciones interacciones de grano fino son adecuadas pero cuando se hacen llamadas entre procesos esto cambia. Las llamadas remotas son más costosas pues tienen más cosas que hacer: Los datos tienen que ser empaquetados, la seguridad debe ser chequeada y los paquetes deben ser enrutados a través de varios dispositivos. En resumen, cualquier llamada inter-proceso es mucho más cara que otra dentro del mismo proceso, incluso, si ambos procesos están en la misma máquina.

Solución: Cualquier objeto que necesite usar otro objeto remoto necesita una interfaz de granularidad gruesa que minimice el número de llamadas necesarias para llevar a cabo una tarea. De esta forma se afecta los objetos que se devuelven, por ejemplo, en vez de solicitar la información de las órdenes de compra y luego solicitar los productos comprados individualmente, se solicita toda esta información de una vez y se actualizan ambos objetos.

Una fachada remota es una fachada de granularidad gruesa [29] sobre un grupo de objetos de granularidad fina. Ninguno de los objetos de granularidad fina tiene una interfaz remota, y la fachada remota no contiene lógica de dominio. Todo lo que la fachada remota hace es convertir los métodos de granularidad gruesa en métodos de granularidad fina.

Consecuencias: La fachada no solo encapsula la implementación y brinda una vista de caja negra, sino que puede ser usada para abstraer el servicio Web de la implementación, la fachada no contiene lógica de negocios. La fachada puede ser usada para obtener la granularidad correcta de un servicio Web desde el punto de vista del consumidor.

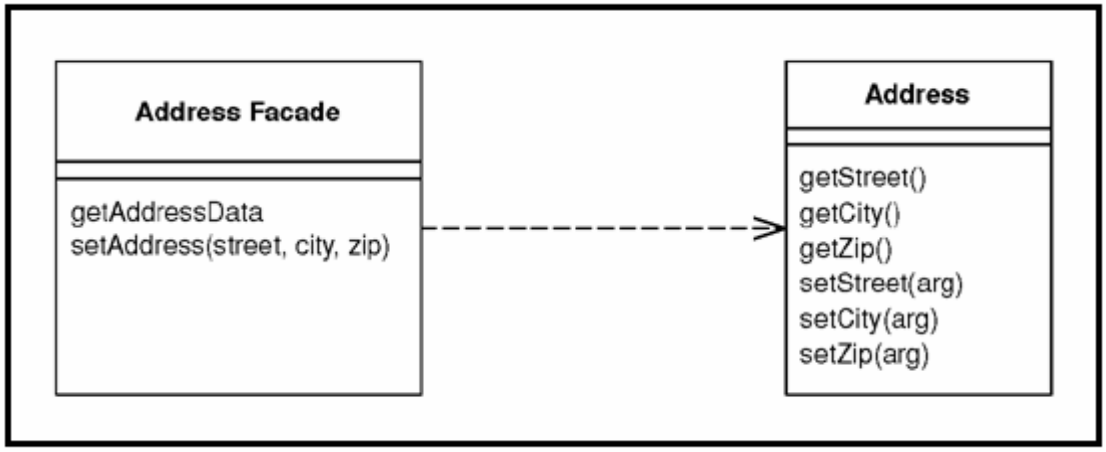


Figura 2.9 Patrón Remote Facade, ejemplo de fachada sobre un objeto Address (Dirección)

Patrón Data Transfer Object (DTO) [31]

Contexto: Al trabajar con interfaces remotas en los servicios Web, como la Fachada Remota, cada llamada es muy costosa, por lo que es necesario reducir el número de interacciones entre el cliente del servicio Web y el servidor, lo cual significa que hay que transferir muchos datos por cada llamada entre aplicaciones que corren en procesos distintos.

Problema: La utilización de muchos parámetros en la función remota que se llama no es conveniente pues esto es difícil de programar, de hecho hay lenguajes como Java que retornan un solo valor. La interpretación del XML resultante de la comunicación es difícil de manejar directamente.

Solución: Crear un objeto de transferencia de datos (Data Transfer Object) que puede contener todos los datos de la llamada. Este objeto necesita ser serializable para ir a través de la conexión. Normalmente, en el servidor, otro objeto es el encargado de transferir datos entre el DTO y los objetos del dominio. Un DTO solo contendrá datos y métodos para acceder y modificar sus datos. Normalmente otro objeto llamado Assembler es el encargado de mover la información entre el DTO y los objetos de negocio.

Consecuencias: Se crea un intermediario entre la fachada remota y los objetos entidades de negocio que desconecta la fachada del servicio de su implementación interna. Alivia el trabajo con estructuras XML pues no hay que manipular el modelo de objetos de XML para acceder a los valores de los nodos de los mensajes.

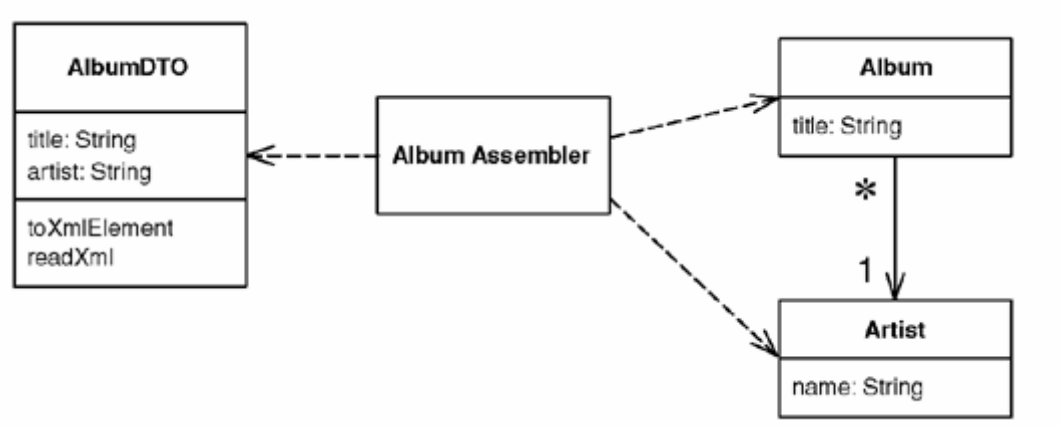


Figura 2.10 Patrón DTO. El objeto AlbumDTO contiene los datos que serán transferidos por los objetos del dominio Álbum y Artist.

Patrón Service Proxy , (derivado de Proxy[29])

Contexto: Un cliente necesita hacer una llamada a un servicio Web externo.

Problema: La llamada debe hacerse de modo que exista el menor acople posible entre el cliente y el servicio, debe ser fácil de rehusar el mismo servicio en otras partes de la aplicación cliente del servicio Web.

Solución: Crear una clase que encapsule las funcionalidades del servicio y que maneje el paso de parámetros, la llamada al servicio y la lectura de resultados del mismo, el cliente se comunicará con el servicio Web a través del objeto Proxy, el cual, para sus ojos es el Servicio Web.

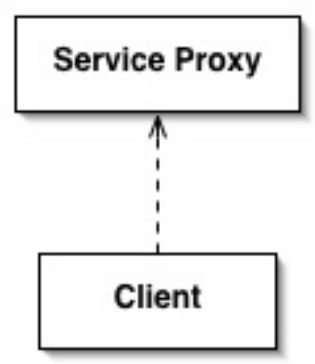


Figura 2.11 Patrón Proxy. El objeto cliente llama las funciones remotas a través del objeto Proxy.

Consecuencias: Se encapsula el servicio Web en un objeto local del cliente que oculta que el servicio reside en un espacio de direcciones diferente. El Proxy puede encargarse de realizar tareas extras cuando es accedido. El Proxy puede ser utilizado sin dificultad desde diversas áreas del servicio cliente.

Patrón Service Coordinator (derivado de Strategy[29])

Contexto: En una arquitectura basada en servicios los servicios Web realizan interacciones basadas en determinado proceso de negocio.

Problema: Cuando existe un grupo grande de servicios puede llegar a existir una explosión de dependencias entre los diferentes servicios que puede hacer difícil de mantener la solución basada en servicios Web.

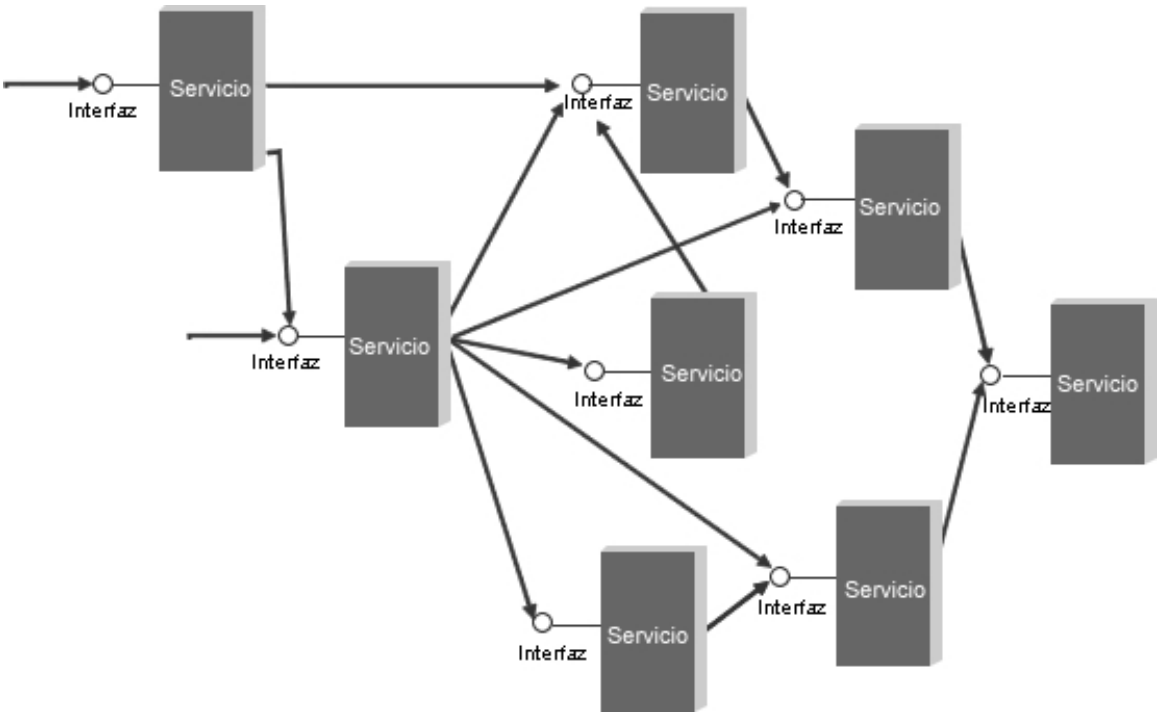


Figura 2.12 Servicios con un número elevado de dependencias.

Solución: Crear un nuevo servicio coordinador que se encarga de manejar la lógica de los procesos de negocio y coordinar las interacciones entre todos los servicios.

Consecuencias: El problema de la orquestación de los servicios se hace más manejable al organizar las interacciones entre los mismos.
Un servicio independiente es el encargado de coordinar la las interacciones entre los servicios dependientes para dar respuesta a peticiones de servicios clientes.

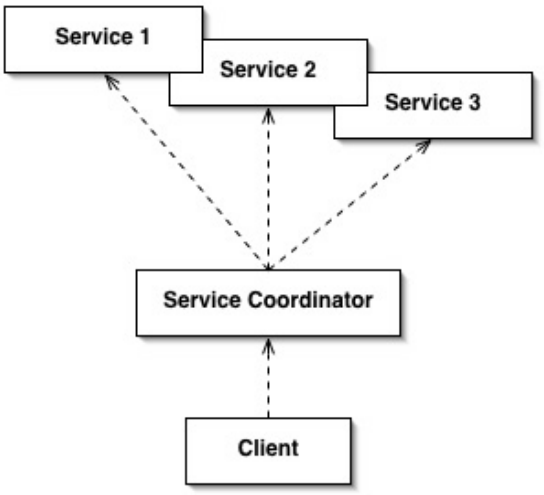


Figura 2.13 Patrón Service Coordinator

Patrón Composite Service, derivado de Composite[29]

Contexto: Una aplicación de negocios provee varios servicios Web internos que exponen sus funcionalidades e implementan la mayoría de los elementos funcionales del proceso de la empresa. Cada servicio individual es importante para resolver problemas específicos, pero a menudo la solución determinados problemas implica combinar la funcionalidad de varios servicios existentes. La funcionalidad combinada puede de hecho ser recursivamente compuesta con otros servicios en una solución a un nivel más alto y así sucesivamente.

Problema: Como combinar la funcionalidad de múltiples servicios y hacerlos disponibles a los consumidores interesados en los servicios como un todo y no como implementaciones individuales.

Solución: Exponer los servicios involucrados en satisfacer los requerimientos funcionales y su coordinación como un servicio aparte. Este servicio compuesto brinda un único punto de acceso a la funcionalidad deseada. La combinación de los servicios participantes en un servicio nuevo y simple es una solución con valor agregado para el consumidor.

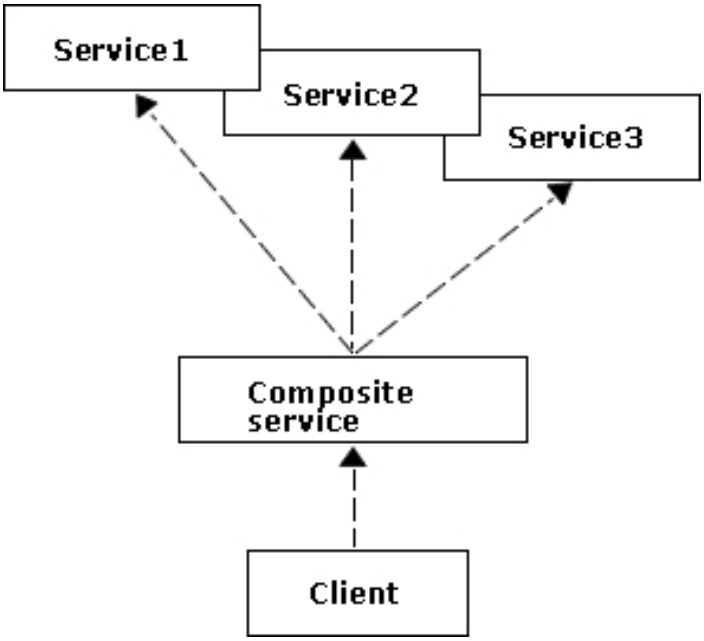


Figura 2.14 Patrón Composite service. Las funcionalidades de varios servicios pueden ser accedidas por los clientes a través de uno solo.

Consecuencias: Define una jerarquía consistente en servicios primitivos y servicios compuestos. Hace que los clientes del servicio sean más simples pues estos tratan con los objetos compuestos como si fueran objetos simples. No debe confundirse este servicio con el servicio fachada remota. La fachada remota provee una interfaz de granularidad gruesa para servicios internos de granularidad fina, involucrando generalmente varios objetos de negocio.

Conclusiones parciales

1. El modelo propuesto sirve de guía para la implementación de arquitecturas orientadas a servicios Web XML en empresas que quieran lograr soluciones duraderas y adaptables al entorno de Internet.
2. Con el uso de los patrones de diseño orientados a servicios Web, se brindan herramientas a los desarrolladores que pueden servir de guía en la resolución de problemas similares.
3. El bajo acople, agilidad, cumplimiento de estándares y abstracción del modelo destacan su carácter práctico, permitiendo su aplicación tanto en servicios intra-empresariales como entre servicios inter-empresariales.
4. El modelo en su conjunto constituye un aporte a la ingeniería del software en la esfera de las arquitecturas orientadas a servicios Web basados en XML pues agrupa un conjunto de técnicas y patrones y los aplica de forma novedosa.
5. La aplicación del modelo desarrollado implica cambios en las formas de pensar y actuar en las organizaciones respecto a la manera de implementar las aplicaciones, la orientación a proyectos inhabilita el uso de componentes y servicios de software, por lo que juega un papel primordial la capacitación en función del cambio de la cultura organizacional, de modo que el pensamiento a corto plazo, los secretos de negocio y las habilidades para abstraer no afecten la agilidad del negocio.

CAPÍTULO III

GUÍAS PARA EL USO DEL MODELO PROPUESTO

Las empresas que quieren aumentar sus ventas necesitan cada vez más presencia en Internet. Con el aumento de clientes y proveedores aumenta también la necesidad de intercambio de datos entre diversas partes de la empresa y con otras entidades externas a la misma que brindan servicios. Este proceso debe ser bien pensado pues las partes que requieren intercambiar datos remotamente pueden estar en diversas plataformas y subredes distintas.

En este capítulo se aplica el modelo propuesto en el capítulo II para lograr una adecuada arquitectura orientada a servicios SOA, en particular servicios Web en una empresa. Concretamente se toma como caso de estudio la Agencia de viajes Universitur donde el modelo propuesto contribuye a la solución de sus necesidades de intercambio de datos.

3.1 Descripción del problema

El Estado Cubano ha dedicado mucho esfuerzo a elevar sus capacidades de brindar servicios turísticos, para ello ha creado organizaciones llamadas Agencias de Viajes que tienen la misión de confeccionar y vender productos a través de turoperadores o clientes independientes, así como coordinar la gestión de los proveedores para brindar dichos productos. También tienen la tarea de cobrar a sus clientes y pagar a sus proveedores los servicios que presta. Generalmente un producto o paquete turístico está formado por un conjunto de servicios de alojamiento, transporte terrestre, transporte aéreo, cabaret, excursiones o eventos, estos servicios son contratados a entidades externas a la agencia de viajes que son los proveedores de este negocio.

Los paquetes turísticos son vendidos luego a turoperadores, que a su vez los comercializan en sus propios países o a clientes individuales que se interesan en Cuba por un paquete determinado.

Para realizar esta gestión es preciso que estas agencias sean muy ágiles por lo cual es necesario utilizar soluciones informáticas que den velocidad a la misma con vistas a brindar un producto con valor añadido orientado a la calidad.

La Agencia de Viajes Universitur es una de las empresas cubanas que se dedican a este tipo de negocio, con la particularidad que sus paquetes turísticos generalmente están relacionados con el turismo universitario, y son comprados por personas interesadas en asistir a eventos y convenciones que se realizan en Cuba, y que durante su estancia en nuestro país necesitan hospedarse, ser transportados y que normalmente visitan centros culturales y recreativos. De todas las agencias de viaje cubanas esta probablemente sea la que más sucursales tenga a nivel de país pues cuenta con representación en muchas universidades dado el segmento de mercado al que está dirigida. La casa matriz de la empresa radica en Ciudad Habana y la gestión de los procesos se realiza a través del sistema SIAV Suite que es utilizado igualmente en las sucursales.

3.1.1 Modelo de ejecución de los procesos del SIAV Suite.

En la figura 3.1 se muestra el modelo de ejecución de los procesos para la Agencia de Viajes. Esta consta de cinco departamentos, Contratación, Marketing, Ventas, Operaciones y Contabilidad y Finanzas de ellos los cuatro primeros obtienen colaboración para su gestión en el SIAV Suite

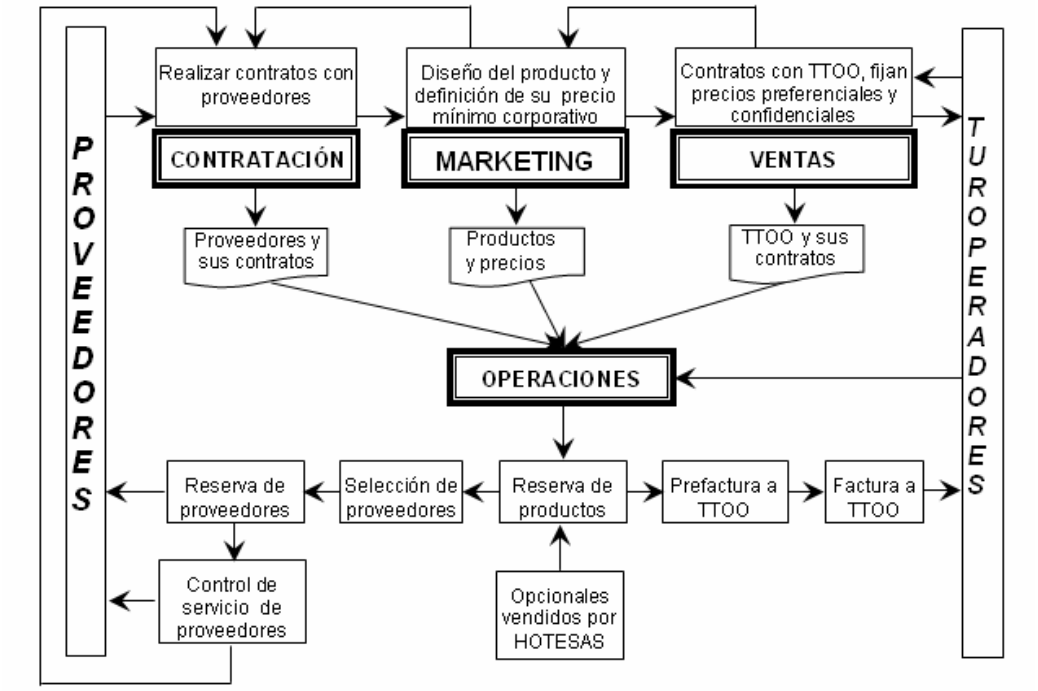


Figura 3.1 Modelo de ejecución de los procesos de la agencia gestionado por SIAV SUITE.

Contratación realiza la gestión con los proveedores, los localiza, los evalúa, efectúa la negociación con ellos y los contrata por un año, introduce en el SIAV Suite los datos de identificación, comunicación y contratación con los proveedores.

Marketing diseña los productos y calcula sus costos, utiliza la información introducida por contratación, con el fin de diseñar los nuevos productos y de actualizar los productos que aún se mantiene activos ya que tiene los proveedores necesarios para ejecutarlos.

Este departamento retro alimenta al Departamento de Contratación la información de sus necesidades de proveedores para la confección de nuevos productos.

Ventas realiza la negociación con los turoperadores, utiliza la información que brinda Marketing y negocia con los turoperadores los productos para que sean comercializados, auxiliado por el SIAV Suite fija los precios preferenciales y confidenciales, este departamento retro alimenta al Departamento de Marketing el resultado de su gestión para que sea evaluada por este y permite mejorar los productos y futuros planes de la organización.

Operaciones consulta a través del SIAV Suite el resultado de la gestión de los tres departamentos anteriores, observando proveedores registrados y los contratos firmados con ellos por la organización, los productos activos, su descripción y costo y los turoperadores con los cuales se ha realizado alguna negociación y en qué términos se han firmado sus contratos.

Operaciones recibe de los turoperadores la solicitud de reserva, registra la reserva de los productos solicitados en un estado sin confirmar y solicita a los proveedores la confirmación de los servicios previamente contratados, esto se hace desde SIAV SUITE a través de correo electrónico o por teléfono, si el proveedor confirma la disponibilidad del servicio la reserva se pasa manualmente al estado confirmada, en caso contrario es necesario buscar un servicio del mismo tipo proporcionado por un proveedor alternativo.

La agencia de viajes quiere implementar una solución que le permita al módulo de operaciones confirmar los servicios de los productos de los proveedores no solo a través del correo electrónico, sino consumiendo servicios Web donde los proveedores exponen estos exponen sus funcionalidades. El departamento de operaciones necesita también un informe de las ventas de productos que se han realizado en cada una de las sucursales y consolidarlos para hallar las ventas totales.

3.2 Modelación de la arquitectura del sistema de intercambio de datos

La solución que la empresa requiere debe ser independiente de la plataforma, pues necesita intercambiar datos con proveedores que no pertenecen a su organización, debe estar basada en estándares pues es esto es un requisito indispensable para lograr la interoperabilidad entre aplicaciones basadas en servicios, debe tener también un bajo nivel de acople entre sus partes para posibilitar agilidad y adaptabilidad a los cambios constantes del mercado, así como debe permitir componer servicios de mayor granularidad a partir de otros de menor granularidad.

Una arquitectura basada en Servicios Web sería una solución adecuada a este problema, pero esto requiere un diseño correcto de los servicios Web y sus relaciones con otros servicios Web internos o externos.

3.3 Descomposición jerárquica basada en el modelo de negocios de la empresa

Nuestra arquitectura debe tener en cuenta que el modelo de negocios de la empresa, una vista general de este modelo, particularizando en los requisitos que se quieren implementar es la siguiente:

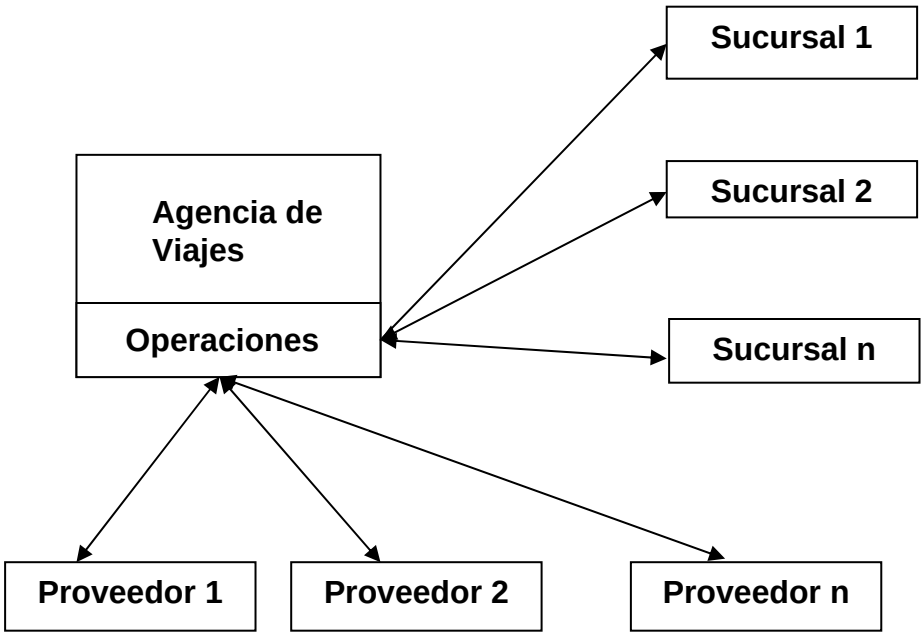
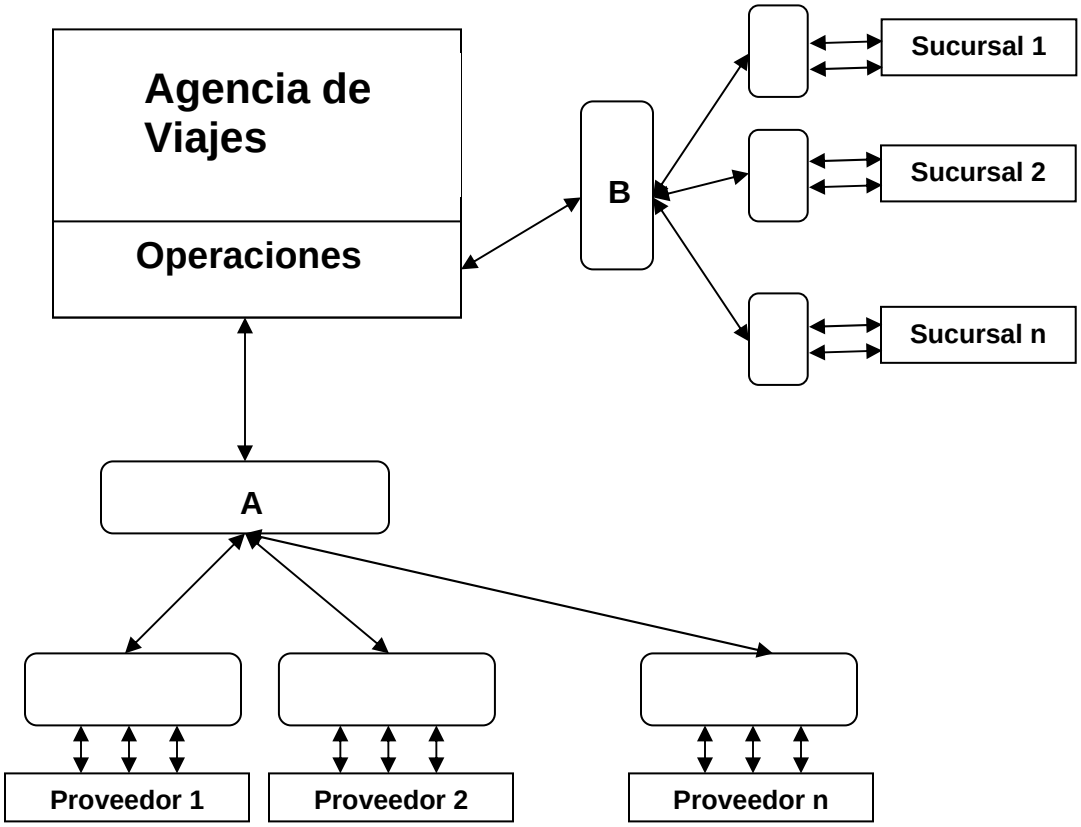


Figura 3.2 Vista del modelo de negocios.

Aplicando descomposición jerárquica basada en el proceso de negocio quedaría el siguiente diagrama:



Leyenda: Rectángulo redondeado=Servicio Web.

Figura 3.3 Definición en servicios Web aplicando descomposición Jerárquica.

Nótese que se ha definido un servicio Web para cada proveedor, uno entre la agencia y las sucursales llamado B y otro entre la agencia y los proveedores llamado A. Los servicios Web de los proveedores ya existían con anterioridad.

3.4 Especificar las interfaces de intercambio de mensajes.

Para especificar las interfaces de los servicios resultantes hay que comprender cuales son los datos que se quieren intercambiar.

El módulo de operaciones está interesado en confirmar la información de sus servicios, por tanto envían los datos de cada servicio a su proveedor y reciben de este una respuesta positiva o negativa, pero en arquitecturas orientadas a servicios es importante enviar mensajes con diferentes granularidades.

Concretamente Operaciones enviaría un mensaje al servicio **A** con granularidad gruesa, es decir, con los datos de todos los proveedores de servicio de un paquete en particular y este servicio se encargaría de contactar a cada proveedor individual con sus datos específicos, es decir, enviaría un mensaje de granularidad fina al servicio Web de cada proveedor. Luego los proveedores envían la respuesta al servicio **A** y este la compone en un solo mensaje con todas las respuestas y lo devuelve al departamento de operaciones. La tabla 3.1 muestra este proceso y lo que debería contener cada mensaje.

Emisor	**Receptor**	**Mensaje**	**Granularidad**
Operaciones	Servicio Web A	Lista de servicios a confirmar y sus respectivos proveedores	Gruesa
Servicio Web A	Servicios Web de los proveedores	Servicio turístico específico a confirmar	Fina
Servicios Web de los proveedores	Servicio Web A	Confirmación o no del servicio turístico	Fina
Servicio Web A	Operaciones	Lista de servicios confirmados y no confirmados	Gruesa

Tabla 3.1 Mensajes que se intercambian entre los diferentes servicios y el módulo de operaciones en el proceso de confirmación de los servicios turísticos.

Operaciones también solicita un reporte de las ventas de cada sucursal, por lo que envía un mensaje con toda esa información al servicio Web **B** y este lo descompone en otros de granularidad fina solicitando a cada sucursal su informe de ventas particular. Finalmente el servicio intermedio integra la información que devuelve cada sucursal y envía un mensaje de granularidad gruesa de vuelta a operaciones. La tabla 3.2 muestra este proceso y lo que debería contener cada mensaje.

Emisor	Receptor	Mensaje	Granularidad
Operaciones	Servicio Web B	Lista de sucursales que deben enviar informe de ventas	Gruesa
Servicio Web B	Servicios Web de las sucursales	Solicitud de informe de ventas.	Fina
Servicios Web de las sucursales	Servicio Web B	Informe de ventas	Fina
Servicio Web B	Operaciones	Informe de ventas total	Gruesa

Tabla 3.2 Mensajes que se intercambian entre los diferentes servicios y el módulo de operaciones en el proceso de obtención de los informes de ventas.

3.5 Recomponer los servicios resultantes para asegurar el cumplimiento de los requisitos técnicos o arquitectónicos.

El planteamiento del problema no adiciona restricciones especiales, por lo que no es necesario variar el diseño que se tiene hasta el momento.

3.6 Utilizar patrones de diseño en la jerarquía de servicios.

Patrones a utilizar en los servicios Web de las sucursales:

1. Remote Facade (fachada remota), es utilizado para encapsular la implementación de la lógica de negocio de las sucursales y brindar una vista de caja negra, la fachada no contiene la lógica de negocios de las sucursales.
2. En conjunto con la fachada remota aplicamos el patrón Data Transfer Object (DTO), de modo que desconectamos la interfaz del servicio Web de las entidades de negocio de las sucursales. Cuando la fachada recibe datos remotos, estos son transferidos a través de un DTO a los objetos de negocio de las sucursales.

Patrones a utilizar en el servicio Web B:

1. Service proxy: Encapsula los servicios Web remotos en objetos locales que ocultan que los servicios de las sucursales residen en un espacio de direcciones diferente.
2. Remote Facade (fachada remota), es utilizado para encapsular la implementación del proceso de negocio de calculo de las ventas. Es una fachada remota para el cliente de operaciones.
3. En conjunto con Remote Facade aplicamos el patrón Data Transfer Object (DTO), de modo que desconectamos la interfaz del servicio Web de las entidades de negocio, que en este caso podría ser una lista de objetos sucursales.
4. Service Coordinator: Se encarga de manejar la lógica de los procesos de negocio y coordinar las interacciones entre todos los servicios que brindan las sucursales.

Patrones a utilizar en el servicio A:

1. Service proxy: Encapsula los servicios Web remotos en objetos locales que ocultan que los proveedores de servicios residen en un espacio de direcciones diferente.
2. Remote Facade (fachada remota), es utilizado para encapsular la implementación del proceso de negocio de confirmación de la reserva.
3. En conjunto con Remote Facade aplicamos aquí también el patrón Data Transfer Object (DTO), de modo que desconectamos la interfaz del servicio Web de las entidades de negocio, que en este caso podría ser una lista de los proveedores de servicios.
4. Service Coordinator: Se encarga de manejar la lógica de los procesos de negocio y coordinar las interacciones entre todos los servicios que brindan los proveedores de servicios remotos.

Conclusiones parciales

1. La implementación de una solución basada en el sistema de intercambio de datos aporta los siguientes beneficios a la Agencia de Viajes Universitur:
   - El departamento de Operaciones puede confirmar los servicios de los productos de los proveedores no solo a través del correo electrónico, sino consumiendo servicios Web donde los proveedores exponen sus funcionalidades. El departamento de operaciones puede obtener informes relativos a las ventas de sus productos en cada una de las sucursales, a través de servicios Web.
   - Existe Independencia de la plataforma, se hace uso de estándares, así como se posibilita analizar información con diferentes niveles de granularidad.
   - La solución es interoperable con proveedores externos, los cuales poseen plataformas, tecnologías y redes de diversa índole.
   - La solución se ha basado en la composición de servicios, lo cual permite soluciones con bajo acople, cuestión de suma importancia para la mantenibilidad de las aplicaciones que se ven frecuentemente necesitadas de cambios producto de la naturaleza cambiante de los negocios hoy en día.

2. La aplicación del modelo permitió constatar su adaptabilidad, pertinencia y consistencia, fundamentalmente para intercambiar datos entre distintas plataformas y subredes de manera desacoplada.

CONCLUSIONES

A partir de la caracterización del estado del arte en el empleo de las tecnologías existentes para intercambio de datos entre aplicaciones, se determinó la necesidad de establecer guías que permitan a los desarrolladores usar dichas tecnologías para construir aplicaciones que demanden servicios de interconexión de datos.

Se propuso un modelo de intercambio de datos para aplicaciones en diferentes plataformas, subredes y dispositivos.

Se mostró el desarrollo de mecanismos obtenidos por el autor empleando el modelo anterior mediante el diseño de una arquitectura orientada a servicios en la Agencia de Viajes Universitur.

El modelo de intercambio de datos desarrollado ayudará a las empresas a diseñar arquitecturas de este tipo pues las guiará en la definición de los servicios Web que necesita y las interacciones de estos con otros servicios Web.

El modelo aporta una serie de resultados novedosos en este campo, entre los más notables se encuentran:

a. Se obtiene un modelo capaz de guiar el desarrollo de aplicaciones que demandan el intercambio de información consecuente con las necesidades actuales.
b. Se adaptaron patrones de diseño para su aplicación a la implementación de arquitecturas orientadas a servicios Web.
c. A fin de lograr agilidad en la interacción entre servicios Web, se integraron los conceptos de: abstracción, generalización, cumplimento de estándares y granularidad.

El bajo acople, agilidad, cumplimiento de estándares y abstracción del modelo destacan su carácter práctico, permitiendo su aplicación tanto en servicios propios de una empresa como entre servicios inter-empresariales.

RECOMENDACIONES

A partir de los resultados obtenidos durante el desarrollo de esta investigación proponemos las siguientes recomendaciones:

1 Enriquecer el modelo con la inclusión de patrones de diseño relacionados con motores de orquestación.

2 Incorporar herramientas visuales y asistentes que faciliten a los desarrolladores la aplicación del modelo.

REFERENCIAS BIBLIOGRÁFICAS

1. CORBA: Common Object Request Broker Architecture, http://www.omg.org.
2. Java: Remote Method Invocation;http://java.sun.com/j2se/1.4.2/docs/guide/rmi/index.html.
3. The Open Group, Enterprise Architecture Standards, http://www.opengroup.org/
4. A. Birrell and B. J. Nelson, Implementing Remote Procedure Calls, ACM Transactions on Computer Systems, Vol. 2, No. 1, Feb 1984, pp.39-59.
5. The Component Object Model Specification, http://www.microsoft.com/oledev/olecom/title.htm.
6. DCE 1.1: Remote Procedure Call Specification, The Open Group, http://www.rdg.opengroup.org/public/pubs/catalog/c706.htm.
7. D. Rogerson, Inside COM, Redmond, Washington: Microsoft Press, 1996.
8. The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995, http://www.omg.org/corba/corbiiop.htm.
9. Java Remote Method Invocation, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/index.html
10. .NET Remoting http://www.microsoft.com/net/remoting.aspx
11. .NET Remoting Versus Web Services, http://www.developer.com/net/net/article.php/2201701
12. XML-RPC, http:// www.xml-rpc.com
13. W3C Organizations Web Services definitions, http://www.w3.org/2002/ws/
14. SOAP specification, http://www.w3.org/TR/SOAP/
15. WSDL specification, http://www.w3.org/TR/wsdl
16. UDDI specification, http://www.w3.org/TR/UDDI
17. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.
18. http://www.omg.org/technology/documents/spec_catalog.htm.
19. Christoph Bussler, Alexander Maedche, Dieter Fensel, Web Services: Quo Vadis? IEEE Intelligent Systems, January/February (2003)80-82.
20. V. Richard Benjamins, Web Services Solve Problems, and Problem-Solving Methods Provide Services, IEEE Intelligent Systems, January/February (2003) 76-77.

21. Elspeth Wales, Web Services Security, computer fraud & security,1, 2003, 15-17.
22. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
23. Satish Thatte, XLANG-Web Services for Business Process Design, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
24. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
25. Maarten Mullender and Mike Burneo. Application Architecture:Conceptual View. Microsoft Corporation: July 2002 http://www.msdn.microsoft.com/architecture/application/default.aspx?pull=/library/en-us/dnea/html/eaappconland.asp
26. David L. Parnas. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), December 1972
27. Boris Lublinsky. SOA design: Meet in the middle. Java Pro, August 2004.
28. Boris Lublinsky. Unifying data, documents and processes. Enterprise Architect, 2(2):6–11, Summer 2004.
29. Erich Gamma, Richard Helm, Ralph Jonson, John Vlissides. Design Patterns,.Addison-Wesley, 1995.
30. David Trowbridge, Dave Manzini, Dave Quick, Gregor Hohpe, James Newkirk, David Lavigne. Microsoft Patterns Enterprise Solution Patterns Using Microsoft.Net Microsoft Corporation.April 003 http://www.msdn.microsoft.com/architecture/patterns/default.aspx
31. Addison-Wesley, Patterns Of Enterprise Application Architecture

BIBLIOGRAFÍA

1. CORBA: Common Object Request Broker Architecture, http://www.omg.org.
2. Java: Remote Method Invocation;http://java.sun.com/j2se/1.4.2/docs/guide/rmi/index.html.
3. The Open Group, Enterprise Architecture Standars, http://www.opengroup.org/
4. A. Birrell and B. J. Nelson, Implementing Remote Procedure Calls, ACM Transactions on Computer Systems, Vol. 2, No. 1, Feb 1984, pp.39-59.
5. The Component Object Model Specification, http://www.microsoft.com/oledev/olecom/title.htm.
6. DCE 1.1: Remote Procedure Call Specification, The Open Group, http://www.rdg.opengroup.org/public/pubs/catalog/c706.htm.
7. D. Rogerson, Inside COM, Redmond, Washington: Microsoft Press, 1996.
8. The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995, http://www.omg.org/corba/corbiiop.htm.
9. Java Remote Method Invocation, http://java.sun.com/j2se/1.4.2/docs/guide/rmi/index.html
10. .NET Remoting http://www.microsoft.com/net/remoting.aspx
11. .NET Remoting Versus Web Services, http://www.developer.com/net/net/article.php/2201701
12. XML-RPC, http:// www.xml-rpc.com
13. W3C Organizations Web Services definitions, http://www.w3.org/2002/ws/
14. SOAP specification, http://www.w3.org/TR/SOAP/
15. WSDL specification, http://www.w3.org/TR/wsdl
16. UDDI specification, http://www.w3.org/TR/UDDI
17. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.
18. http://www.omg.org/technology/documents/spec_catalog.htm.
19. Christoph Bussler, Alexander Maedche, Dieter Fensel, Web Services: Quo Vadis? IEEE Intelligent Systems, January/February (2003)80-82.

20. V. Richard Benjamins, Web Services Solve Problems, and Problem-Solving Methods Provide Services, IEEE Intelligent Systems, January/February (2003) 76-77.
21. Elspeth Wales, Web Services Security, computer fraud & security,1, 2003, 15-17.
22. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
23. Satish Thatte, XLANG-Web Services for Business Process Design, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.
24. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
25. Maarten Mullender and Mike Burneo. Application Architecture:Conceptual View. Microsoft Corporation: July 2002 http://www.msdn.microsoft.com/architecture/application/default.aspx?pull=/library/en-us/dnea/html/eaappconland.asp
26. David L. Parnas. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), December 1972
27. Boris Lublinsky. SOA design: Meet in the middle. Java Pro, August 2004.
28. Boris Lublinsky. Unifying data, documents and processes. Enterprise Architect, 2(2):6–11, Summer 2004.
29. Erich Gamma, Richard Helm, Ralph Jonson, John Vlissides. Design Patterns,.Addison-Wesley, 1995.
30. David Trowbridge, Dave Manzini, Dave Quick, Gregor Hohpe, James Newkirk, David Lavigne. Microsoft Patterns Enterprise Solution Patterns Using Microsoft.Net Microsoft Corporation.April 003 http://www.msdn.microsoft.com/architecture/patterns/default.aspx
31. Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford, Addison-Wesley, Patterns Of Enterprise Application Architecture, 2002
32. Jim Rumbaugh, Grady Booch, Ivar Jacobson. .El proceso unificado de desarrollo de software, Addison-Wesley , 2004.
33. Craig Larman, UML y Patrones, 2000.
34. Brian Benz. XML Programming Bible. Jhon Wiley & Sons, September 2003
35. Lázlo Roth, George L. Wybenga. The Packaging Designer's Book of Patterns. 2nd Edition. Jhon Wiley & Sons, January 15, 2000.
36. Max Jacobson, Murray Silverstein, Barbara Winslow, Patterns of Home: The Ten Essentials of Enduring Design. Taunton Pr, August 27, 2002.

37. Stan Ward, Per Kroll. Building Web Solutions with theRational Unified Process:Unifying the Creative Design Process and the Software Engineering Process. Context Integration, 2000.
38. Ivar Jacobson, Grady Booch, James Rumbaugh. El proceso de unificado de software. Addisoy Wesley, 2000.
39. Jim Conallen. Modeling Web Application Architectures with UML. Rational Software White Paper, 1999.
40. Christian Thilmany. .Net Patterns : Architecture, Design, and Process. 1st edition. Addison Wesley Professional. January, 2003.

ANEXOS

A continuación se muestran algunas clases utilizadas el los servicios Web de las sucursales, en todos los segmentos de código se utilizó el lenguaje C#.

```
//namespace del dominio de negocio, debe estar en un componente aparte.
namespace DomainModel
{
    //Clase sucursal
    public class Branch
    {
        public Branch()
        {

        }
        public string getBranchName()
        {
            //Se retorna el nombre de la sucursal
            return "Universitur Cienguegos";
        }
        public decimal getTodaySales()
        {
            //Se retorna la venta del día.
            return SalesProcess.SalesManager.getTodaySales();
        }


    }
 }
//namespace donde se implementa el patron DataTransferObject
 namespace DataTransferObjects
  {
   using DomainModel;
     //DataTransfer Object de las ventas de la sucursal
     //Solo tiene 2 campos y propiedades para accederlos,
     //no realiza ningún cálculo adicional.
     //Es el objeto que se serializa a XML y viaja por la red
     public class SalesDTO
     {
         public SalesDTO()
         {
             //
             // TODO: Add constructor logic here
             //
         }
         private decimal todaySales;
         private string branchName;


         public decimal TodaySales
         {
```

```
            get
            {
                return todaySales;
            }
            set
            {
                todaySales = value;
            }
        }
        public string BranchName
        {
            get
            {
                return branchName;
            }
            set
            {
                branchName = value;
            }
        }

    }
    //Assembler del DataTransferObject SalesDTO
    //Recibe una sucursal y esribe su nombre y la venta del día en el
    //DTO de ventas, luego lo devuelve.
    public class SalesAssembler
    {
        public SalesDTO WriteDTO(Branch subject)
        {
            SalesDTO result= new SalesDTO();

            result.BranchName = subject.getBranchName();
            result.TodaySales = subject.getTodaySales();

            return result;

        }
    }

}
```

A continuación se muestra un segmento de código de la función que el servicio Web de una sucursal expone como fachada remota, nótese que recibe la llamada y le pasa el procesamiento a un objeto Branch que es localizado por BranchFinder, luego el Assembler llena el DTO que es devuelto por la red serializado a XML.

```
    [WebMethod]
    public SalesDTO getTodaySales() {

      Branch result = BranchFinder.GetBranch();

      return new SalesAssembler().WriteDTO(result);
    }
```

A continuación un segmento de código XML donde se ve un SalesDTO Serializado

```
<SalesDTO>
<TodaySales>1000.99</TodaySales>
<BranchName>Universitur Cienguegos</BranchName>
</SalesDTO>
```

A continuación el archivo WSDL del servicio Web de una sucursal.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://tempuri.org/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
      <s:element name="getTodaySales">
        <s:complexType />
      </s:element>
      <s:element name="getTodaySalesResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="getTodaySalesResult" type="tns:SalesDTO" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="SalesDTO">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="TodaySales"
type="s:decimal" />
          <s:element minOccurs="0" maxOccurs="1" name="BranchName"
type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="getTodaySalesSoapIn">
    <wsdl:part name="parameters" element="tns:getTodaySales" />
  </wsdl:message>
  <wsdl:message name="getTodaySalesSoapOut">
    <wsdl:part name="parameters" element="tns:getTodaySalesResponse" />
  </wsdl:message>
  <wsdl:portType name="BranchWebServiceSoap">
    <wsdl:operation name="getTodaySales">
      <wsdl:input message="tns:getTodaySalesSoapIn" />
      <wsdl:output message="tns:getTodaySalesSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
```

```
  <wsdl:binding name="BranchWebServiceSoap"
type="tns:BranchWebServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getTodaySales">
      <soap:operation soapAction="http://tempuri.org/getTodaySales"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="BranchWebServiceSoap12"
type="tns:BranchWebServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getTodaySales">
      <soap12:operation soapAction="http://tempuri.org/getTodaySales"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="BranchWebService">
    <wsdl:port name="BranchWebServiceSoap"
binding="tns:BranchWebServiceSoap">
      <soap:address
location="http://localhost:1979/Cienfuegos%20Web%20Site/Service.asmx"
/>
    </wsdl:port>
    <wsdl:port name="BranchWebServiceSoap12"
binding="tns:BranchWebServiceSoap12">
      <soap12:address
location="http://localhost:1979/Cienfuegos%20Web%20Site/Service.asmx"
/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

A continuación un segmento de código donde una función de un servicio Web, invoca a servicios Web de 2 sucursales remotas y combina el resultado en un DTO que retorna (`BranchesSalesDTO`).

```
public BranchesSalesDTO getSalesfromBranches()
  {
    //VillaClaraWS y CienfuegosWS son proxies para la función
    invocadora. En Visual estudio .Net se pueden crear proxies
    automáticamente si se adicionan Web References.
    VillaClaraBranch.BranchWebService VillaClaraWS = new
    VillaClaraBranch.BranchWebService();
   CienguegosBranch.BranchWebService CienfuegosWS = new
```

```
 CienguegosBranch.BranchWebService();

 //Se combinan las ventas de las sucursales en un solo
 //BranchesSalesDTO que se llena con el objeto
 //branchSalesAssembler
  branchSalesAssembler.AddSales(VillaClaraWS.getTodaySales());
  branchSalesAssembler.AddSales(CienfuegosWS.getTodaySales());

  return branchSalesAssembler.BranchesSalesDTO();
}
```