Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección de Computación

Selección e integración dinámica de servicios Web orientada a aspectos

Tesis que presenta

Marisol Pérez Reséndiz

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Director de Tesis

Dr. José Oscar Olmedo Aguirre

México, D.F. **Octubre 2005**

A mis padres, quienes toda mi vida me han guiado
y de los cuales aprendo cada día.

A Karina, que siempre me cuestiona
pero confía en mi.

A Roberto, por la historia que inició
con una cajita de papel.

Agradecimientos

A Dios, siempre he creído que no importando el lugar donde me encuentre, él esta conmigo.

A mis padres porque siempre me han apoyado en todo lo que he emprendido en mi vida. Me han enseñado que siempre se pueden lograr las cosas por más difíciles que parezcan. Su confianza en mi siempre me ha dado fortaleza para continuar y no darme por vencida. Ustedes son las personas que además de ser mis padres han sido mis amigos. Gracias por todo, saben que los quiero y valoro mucho.

A mi hermana Karina que a pesar de no verla todos los días, sé que siempre se preocupa y está pendiente por mi. Su pasión por el arte es algo que aprendí de ella y que comparto. Gracias por confiar en tu enana.

A mi mejor amiga Nancy, que conozco desde hace muchos años y que considero mi segunda hermana. Aprendimos y vivimos muchas cosas juntas. Nunca nos dimos por vencidas, aunque el cansancio fue mucho, fueron más las recompensas que obtuvimos. Gracias por ser una verdadera amiga, sabes que te quiero mucho.

A mi chiquininito, quien es una persona muy importante en mi vida. Nunca imaginé conocer a alguien que me quisiera tanto y que me lo demostrara día a día. Se que tengo tu apoyo incondicional en todo momento. Gracias por todo, sabes que eres y siempre serás parte de mi vida.

Al Dr. Oscar Olmedo por ser la persona que dirigió este trabajo de tesis. Me ayudó siempre que lo necesité. Gracias por su apoyo y confianza en todo momento.

A todos mis amigos con los cuales tuve el gusto de estar estos dos años. Me quedo

con gratos recuerdos de cada uno de ustedes. Gracias por su amistad.

Al Dr. Gerardo de la Fraga y al Dr. Guillermo Morales por ser los sinodales de este trabajo de tesis. Gracias por sus consejos que me ayudaron a mejorar su contenido.

A Sofia Reza, quien siempre demostró ser una persona muy agradable en todos los aspectos. Gracias por todo tu apoyo y sobre todo por tu amistad.

Al IPN por ser la institución a la que pertenezco y en la cual me he desarrollado académicamente desde hace nueve años. En cada una de las escuelas donde estuve aprendí muchas cosas y de cada una de ellas me llevo momentos inolvidables.

Al CINVESTAV, ya que durante estos dos años de maestría fue la institución donde curse mis materias y pude desarrollar satisfactoriamente mi tesis.

A CONACYT, ya que gracias a su apoyo financiero durante estos dos años pude sustentar mis gastos.

A servicios escolares y a la biblioteca de Ingeniería Eléctrica por los servicios y ayuda que me brindaron cuando lo necesité.

Resumen

Con la aparición y adopción de nuevos estándares y tecnologías de servicios Web, desarrollar y mantener aplicaciones distribuidas se convierte en una tarea cada vez más compleja. Desafortunadamente, la mayoría de los enfoques tradicionales utilizados para desarrollar aplicaciones consideran los problemas asociados con la distribución física desde etapas tempranas de diseño.

Para reducir la complejidad en el desarrollo de aplicaciones basadas en la Web, en esta tesis proponemos un enfoque que provee transparencia de localidad para el diseño de las mismas, dejando los aspectos de la distribución física para las etapas de desarrollo de las aplicaciones. Para lograr la modularización de estas características, hemos adoptado la programación orientada a aspectos (POA) como modelo programación, ya que provee los medios para integrar los aspectos de distribución en una aplicación cuando sea necesario. Por lo tanto, este paradigma permitirá a los diseñadores obtener la versión distribuida de una aplicación al integrar la infraestructura de comunicación y coordinación correspondiente a los servicios Web.

La principal contribución de este trabajo consiste en simplificar el proceso de desarrollo de las aplicaciones Web, reduciendo su costo de producción y mantenimiento, obteniendo al mismo tiempo un incremento considerable en su flexibilidad y dinamismo.

Abstract

With the appearance and adoption of new Web services standards and technologies, developing and maintaining distributed applications are becoming complex tasks. Unfortunately most of the known approaches to develop applications embrace problems about physical distribution since the early phases of design.

In order to reduce the complexity in the development of Web-based applications, in this work we propose an approach that emphasizes locality transparency in the application design, leaving the physical distribution concerns to a later phase in the development process. In order to address this modularization concerns, we have adopted Aspect-Oriented Programming (AOP) as a programming model that provides the means to integrate the distribution aspects in an application whenever it is needed. The AOP paradigm allows the system designer to obtain a distributed version of the application by integrating a Web services communication and coordination infrastructure.

The main contribution of this work consists on simplifying the development process of Web-based applications, reducing their costs of production and maintenance, obtaining at the same time, a considerable increase in their flexibility and dynamism.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Este capítulo describe el panorama general en que se sitúa el problema que se aborda en esta tesis y para el cual se plantea una solución. Se presentan los objetivos que se alcanzaron durante el desarrollo de la misma, así como la metodología que se siguió. Finalmente, se mencionan las contribuciones del trabajo y la estructura del presente documento.

1.1. Dominio del problema

1.1.1. Aplicaciones distribuidas

Una aplicación distribuida es aquella que procesa peticiones de servicios y cuya respuesta puede ser proporcionada por diferentes procesos o componentes distribuidos (proveedores de servicios) localizados en diferentes computadoras.

En un inicio, las aplicaciones distribuidas se desarrollaban con el objetivo de proveer la funcionalidad requerida por la aplicación y no se consideraba la posibilidad de reutilizar cada uno de los componentes que la conformaban. Una vez que se consideró a cada uno de sus componentes como proveedores de servicios, los desarrolladores tuvieron la posibilidad de reutilizar dichos componentes como módulos que proveían cierta funcionalidad a las aplicaciones lógicas.

Cuando se diseñan aplicaciones distribuidas se tienen que tomar diversas decisiones para su desarrollo:

- Tipos de datos. Se tiene que considerar como manejar la incompatibilidad de

los diferentes tipos de datos entre diversas plataformas.

- Fallas del servidor. Debido a que los componentes de las aplicaciones son remotos, pueden existir mayores puntos de fallas. De esta manera, se tiene que determinar cómo manejar las fallas que se originen al perder respuesta del servidor.

- Fallas del cliente. Si el servidor monitorea el comportamiento de un cliente y este falla, se debe determinar cómo notificar al servidor.

- Seguridad. En las aplicaciones distribuidas, se debe considerar realizar autenticación y autorización, así como la manera de asegurar la comunicación entre un cliente y un servidor.

- Sincronización. Se tiene que asegurar la sincronización de los procesos entre clientes y servidores.

Con la adopción de nuevos protocolos, empezaron a desarrollarse diversas tecnologías (paso de mensajes, RPC, CORBA, DCOM, RMI) para la implementación de aplicaciones distribuidas. Estas tecnologías a pesar de proveer la funcionalidad requerida por las aplicaciones distribuidas tienen limitaciones de seguridad e interoperabilidad debido al uso de formatos propietarios de datos.

A mediados de los años 90's se empezó a desarrollar la infraestructura de protocolos que correspondían a simples protocolos basados en texto, desarrollados como medio de comunicación para las peticiones de servicios y envío de información en Internet. Fue así que la rápida adopción de dichos protocolos hizó ver a Internet como una plataforma viable para las aplicaciones distribuidas. Por lo tanto, en lugar de utilizar tecnologías propietarias, los estándares Web se consideraron como fundamento para las aplicaciones distribuidas basadas en la Web.

Por lo tanto, con el surgimiento y adopción de estándares Web como HTTP y XML, se buscaron nuevas alternativas para eliminar las limitaciones existentes, lo que condujo al surgimiento de los servicios Web.

1.1.2. El paradigma de los servicios Web

Los servicios Web son aplicaciones modulares descritas, publicadas, localizadas e invocadas en Internet [1][2] para realizar llamadas a procedimientos remotos y cuyas

características proporcionan un mejor funcionamiento sobre otras tecnologías existentes.

El objetivo de los servicios Web es permitir que las aplicaciones compartan información, así como invocar funciones de otras aplicaciones independientemente de cómo se hayan creado las aplicaciones, cuál sea el sistema operativo o la plataforma en que se ejecutan y cuáles sean los dispositivos utilizados para obtener acceso a ellas. A pesar de ser independientes entre sí, pueden vincularse y coordinarse para realizar en conjunto una tarea determinada.

De igual manera, los servicios Web se pueden implementar en cualquier lenguaje de programación por lo que no hay necesidad de aprender uno nuevo. Se invocan usando SOAP (Simple Object Access Protocol), el cual es un protocolo que provee una estructura simple para realizar llamadas a procedimientos remotos codificando las peticiones como documentos XML. Así, al comunicarse los servicios Web mediante HTTP y XML, cualquier nodo de la red que soporte dichas tecnologías podrá tener acceso a ellos de tal forma que la comunicación entre sistemas heterogeneos sea sencilla.

Así, desarrollar tecnologías que proporcionen nuevas alternativas para realizar la invocación de servicios remotos de una manera más simple e independiente del lenguaje y la plataforma es uno de los objetivos que se persigue en la implementación de aplicaciones.

1.1.3. El paradigma de la programación orientada a aspectos

La programación orientada a aspectos (AOP, Aspect Oriented Programming) es un enfoque de programación cuya área de investigación es muy reciente y a la cual se le está dando gran importancia para el desarrollo de aplicaciones debido a las ventajas que brinda.

Este enfoque surgió básicamente debido a la necesidad de modularizar el comportamiento de características o requerimientos que afectan múltiples módulos de una aplicación para detectar situaciones relevantes y reaccionar a ellas apropiadamente. Dicho enfoque engloba tres características principales:

1. Descomponer los requerimientos para identificar las relaciones entre cada uno de ellos.

2. Implementar cada uno de ellos individualmente.

3. Integrar el sistema final a través de la creación de módulos.

Dichas características son encapsuladas por la programación orientada a aspectos en módulos denominados aspectos[1], que se conforman básicamente de puntos de corte y modificadores de comportamiento y los cuales son integrados a una aplicación en tiempo de compilación, de carga o de ejecución de la misma.

1.2. Planteamiento del problema

Al desarrollar una aplicación, algunas características que se desean lograr son modularidad y flexibilidad para la misma. Cuando hablamos de modularidad, nos referimos al hecho de poder identificar y realizar modificaciones de una manera más rápida y precisa a cada uno de los módulos que conforman la aplicación. De esta manera, cuando se está en el proceso de desarrollo, ciertos módulos pueden ser probados con funciones, procedimientos o servicios locales. Podemos observar que dichos servicios pueden invocarse de manera remota para lograr mayor flexibilidad en la ejecución de servicios y para lograr esto, es necesario identificar los puntos específicos en que una aplicación debe modificar su comportamiento.

Desarrollar una aplicación centralizada que posteriormente requiera ser convertida a una aplicación distribuida presenta ciertas desventajas, debido a que tendrían que rediseñarse módulos de la misma. De esta manera, modificar código de la aplicación implicaría implementar múltiples requerimientos en un sólo módulo (*tangling code*) o en varios de ellos (*scattering code*).

Una limitante de los enfoques típicamente utilizados para integrar servicios Web en aplicaciones del lado del cliente, es que no permiten realizar cambios de los servicios dinámicamente o a la manera en que son utilizados. De igual manera, algunos problemas que se pueden presentar pueden originarse debido a condiciones no predecibles de la red o de las organizaciones que los implementan, lo cual ocasionaría que no estuvieran disponibles en el momento en que fueran invocados.

[1] Usaremos esta notación para distinguir entre el término que denota al concepto computacional de aquel que se usa como sinónimo de la palabra atributo.

Estos problemas tendrían como consecuencia tener que elegir un nuevo servicio Web y reescribir manualmente el código referente a su integración cada vez que se presentaran situaciones similares [3]. Como consecuencia se tendría que:

- Los desarrolladores invertirían mayor tiempo de desarrollo.
- La implementación podría no corresponder literalmente con el diseño de su arquitectura u organizacionalmente.
- El código se volvería cada vez menos comprensible.
- Los costos de producción incrementarían.
- El mantenimiento causaría cambios mayores a la aplicación y al comportamiento de la aplicación.

El problema que se desea resolver es realizar la selección e integración dinámica de servicios Web a una aplicación de tal forma que se evite modificar el código de la aplicación original agregando sólo aquellos módulos que provean la funcionalidad requerida.

1.3. Propuesta

Modificar o incrementar la funcionalidad de aplicaciones implica analizar la relación entre los módulos existentes y aquellos que se desean agregar. Como consecuencia, el código de la aplicación original debe modificarse. El problema que se aborda en esta tesis se enfoca principalmente al hecho de que a partir de una aplicación centralizada, se pueda obtener una aplicación distribuida (Figura 1.1) que provea mayor dinamismo y flexibilidad para su mantenimiento, y que dicho proceso se realice de manera sencilla y transparente a los desarrolladores. De esta manera, se identifican los métodos que serán sustituidos por servicios Web para que la infraestructura encargada de realizar la selección (WSSI) y el descubrimiento de servicios Web (Intermediario) genere un conjunto de aspectos, los cuales una vez integrados a la aplicación centralizada conformen una versión distribuida de dicha aplicación.

La solución que proponemos tiene como objetivo seleccionar e integrar dinámicamente servicios Web en las aplicaciones. La selección de los mismos se realiza mediante un servicio de intermediación y la integración mediante el uso de la programación

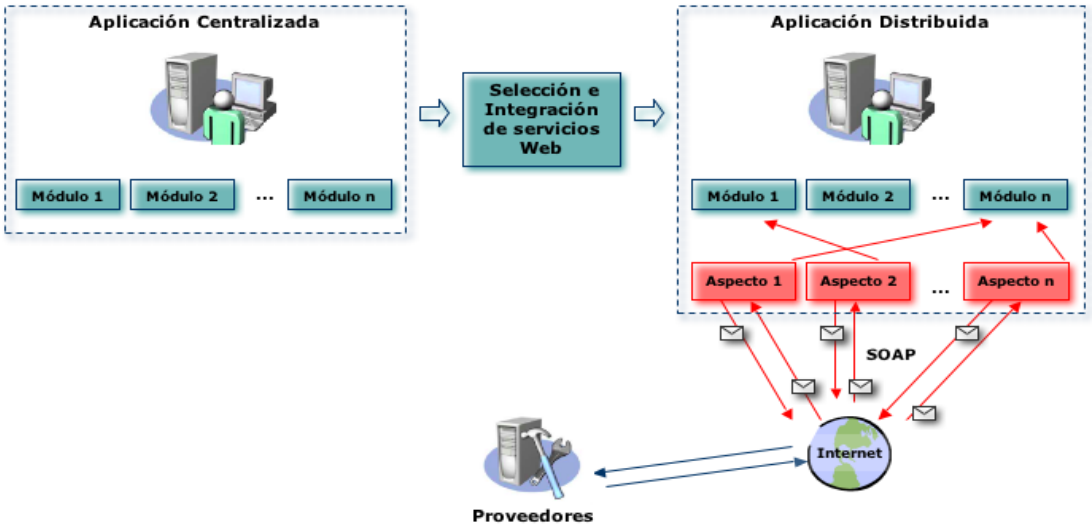

Figura 1.1: Diagrama general de la propuesta de tesis.

orientada a aspectos a través de la generación dinámica de código AspectJ correspondiente al aspecto que realizará la invocación de un servicio Web.

Debido a que la programación orientada a aspectos permite extender o definir un comportamiento alterno a puntos específicos de una aplicación, con esta propuesta se pretende identificar la ejecución de métodos locales y reemplazar su definición por la invocación de un servicio Web que provea la funcionalidad requerida mediante un aspecto que encapsule dicho comportamiento.

Es necesario, desarrollar una alternativa de implementación que permita generar un conjunto de aspectos de tal manera que se evite la invasividad de código y de esta manera contar con código más simple de diseñar y mantener.

Por lo tanto, se propone el desarrollo del servicio de intermediación que provea la funcionalidad básica de un nodo UDDI y de la infraestructura encargada de la definición y generación de aspectos.

1.4. Objetivo general

El objetivo general de este trabajo de tesis consiste en diseñar e implementar la arquitectura que permita realizar la selección e integración dinámica de servicios Web aplicando el enfoque de la programación orientada a aspectos. La arquitectura debe permitir que las aplicaciones satisfagan los requerimientos y funcionalidad deseados mediante la integración de un conjunto de aspectos y proporcionando mayor dinamismo en la selección de servicios Web de manera más simple y sin modificar el código original.

1.5. Objetivos particulares

Con el fin de diseñar e implementar la arquitectura propuesta, se han definido los siguientes objetivos particulares:

1. Diseñar el mecanismo para definir los elementos asociados a un aspecto.
2. Definir la estructura del lenguaje para identificar los elementos asociados a un aspecto.
3. Definir la infraestructura de coordinación y comunicación que dé soporte a los servicios Web.
4. Diseñar e implementar el servicio de intermediación.
5. Diseñar e implementar la infraestructura para la generación de aspectos.
6. Plantear un caso de estudio, así como su análisis y solución integrando los puntos anteriores.

1.6. Metodología

Para alcanzar los objetivos planteados en la sección anterior se siguieron los siguientes pasos:

1. Para la estructura del lenguaje que define elementos de un aspecto se realizó:
   a) La definición del esquema asociado a un documento XML.
   b) La definición del elemento correspondiente a un punto de corte.

   c) La definición del elemento correspondiente a un modificador de comportamiento.

2. Para el servicio de intermediación se definió el WSDL asociado a dicho servicio donde se especificaron las operaciones por la que se conforma y finalmente, se implementó la funcionalidad de cada una de ellas.

3. Para la infraestructura encargada de la generación de aspectos se definieron los módulos necesarios para el análisis de documentos que definen elementos de los mismos y la generación, ejecución, sincronización y serialización de los procesos que generan los archivos correspondientes.

4. Para el caso de estudio se implementó una aplicación centralizada, de tal manera que durante el proceso de selección e integración de servicios Web se analizaran diferentes casos para la generación de los aspectos correspondientes y finalmente, obtener una versión distribuida de la misma.

1.7. Contribuciones

Las principales contribuciones que este trabajo de tesis presenta son:

1. Obtención de una aplicación distribuida. A partir de una aplicación centralizada, obtenemos la versión distribuida de la misma.

2. Ocultamiento de los requerimientos de distribución. Todos los detalles relacionados con la distribución se ocultan en el conjunto de aspectos generados.

3. Creación de un directorio de servicios. Además de proveer las operaciones básicas de consulta y publicación como en un nodo UDDI, provee operaciones de suscripción. Por lo tanto, el directorio de servicios incluye ofertas y demandas que puede ser de gran utilidad en el problema de la cadena de suministro.

4. Diseño de una arquitectura que permita seleccionar e integrar servicios Web dinámicamente a una aplicación, de manera transparente a los desarrolladores.

1.8. Estructura del documento

La presente tesis ha sido estructurada de la siguiente manera. En el capítulo 2 presentamos con mayor detalle la tecnología relacionada con la propuesta de tesis,

así como los trabajos relacionados con la misma. En el capítulo 3 se plantea un caso de estudio para ejemplificar el funcionamiento general del sistema WSSI (Web Services Selection and Integration) y del Intermediario, de tal manera que se describen los casos que resuelven conjuntamente para generar el código asociado a cada aspecto y para descubrir dinámicamente los servicios Web que serán invocados respectivamente. Por otro lado, en el capítulo 4 se presenta el diseño y la perspectiva organizacional correspondiente a estos y la descripción del lenguaje WSSIAD (WSSI Aspect Definition), cuyo objetivo es especificar la información necesaria para la definición de los aspectos y el descubrimiento de servicios Web. En el capítulo 5, se describe la implementación realizada para WSSI y el Intermediario. Finalmente, en el capítulo 6 se presentan las conclusiones sobre el trabajo de tesis, los resultados obtenidos y el trabajo a futuro.

Capítulo 2

Antecedentes

En este capítulo presentamos los conceptos relacionados con la propuesta de este trabajo de tesis. Por tal razón, uno de los objetivos que persigue este capítulo es situar nuestra propuesta dentro de las áreas que abarca. De esta manera, describimos los conceptos básicos y la tecnología relacionada con los servicios Web y con el enfoque de la programación orientada a aspectos. Por otra parte, presentamos los trabajos relacionados con esta tesis, describimos el enfoque que presenta cada uno de ellos, así como sus ventajas, desventajas y diferencias con respecto a nuestra propuesta.

2.1. Servicios Web

Los servicios Web surgieron debido a la necesidad de desarrollar tecnologías que proporcionaran nuevas alternativas para realizar la invocación de servicios remotos de una manera más simple e independiente del lenguaje y la plataforma con respecto a la aplicación que realizará su invocación.

Un servicio Web es una aplicación que forma parte de la lógica de un negocio, se encuentra localizada en Internet y es accesible a través de estándares basados en Internet como HTTP o SMTP [4].

Los servicios Web son accedidos mediante el envío de mensajes que hacen uso de protocolos estándares codificando las peticiones como documentos XML [5] debido a las ventajas que ofrece para la representación de los datos y la comunicación entre aplicaciones, eliminando las limitaciones de cualquier otro protocolo (sistema operativo, plataforma, entre otros).

Aunque se ha desarrollado un gran número de tecnologías relacionadas con los servicios Web, son tres las principales tecnologías que se han adoptado como estándares, las cuales corresponden a SOAP, WSDL y UDDI.

La Figura 2.1 muestra por capas la infraestructura correspondiente a los servicios Web. La capa de transporte corresponde a los protocolos utilizados para el envío de la información, la capa de mensajes al protocolo SOAP, la capa de descripción a WSDL y la capa de procesos a las diversas tecnologías para la publicación, descubrimiento, composición, orquestación, desarrollo e integración de servicios Web.

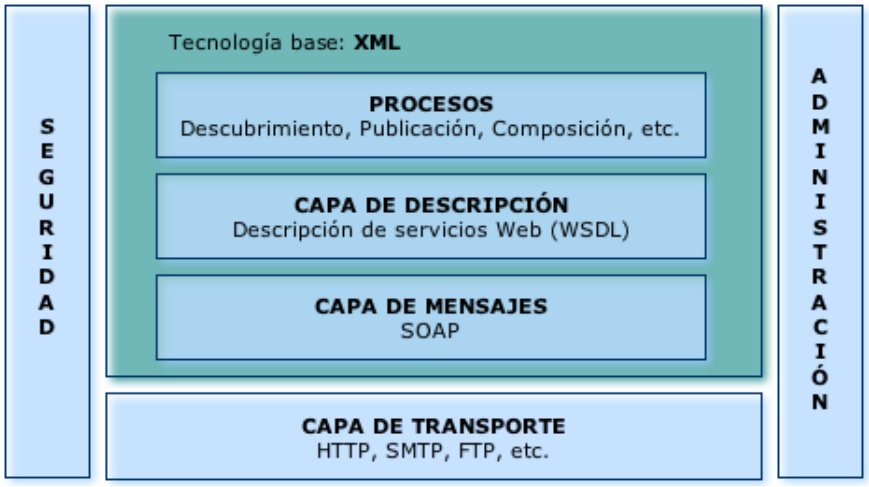

Figura 2.1: Infraestructura de los servicios Web.

La evolución de dichas tecnologías provee en conjunto un avance en la descripción o en el descubrimiento de los servicios Web. De cualquier manera, existe el reto de lograr su descubrimiento, integración e invocación sin la necesidad de intervención humana [4].

2.1.1. Arquitectura orientada a servicios

La Arquitectura Orientada a Servicios (SOA, Service-Oriented Architecture) representa una manera de utilizar y administrar un conjunto de servicios Web a través del uso de SOAP, WSDL y UDDI.

Los servicios son vistos como unidades lógicas, definidos en términos de la funcionalidad que proveen y descritos por documentos WSDL que sólo proveen los detalles necesarios para su invocación como los tipos de datos, las operaciones, protocolos y ubicación correspondiente a la implementación de los mismos [6]. Su invocación se realiza mediante el intercambio de mensajes de acuerdo a los estándares antes mencionados.

SOA está basada en las interacciones entre un proveedor, un registro o repositorio y un cliente (Figura 2.2). Básicamente la relación que existe entre ellos corresponde con publicar la descripción de un servicio, encontrar cuáles se encuentran disponibles y enlazar dichos servicios al cliente.

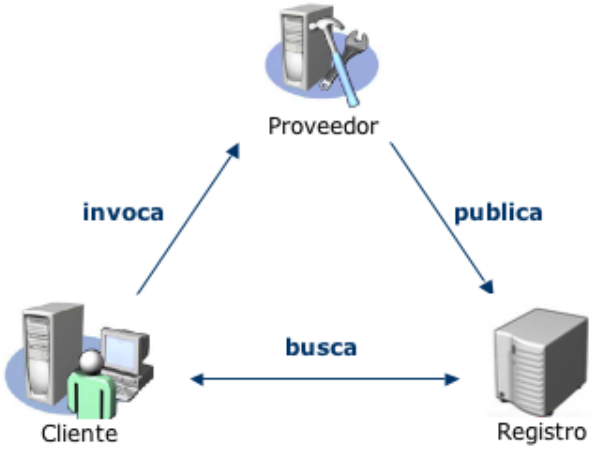

Figura 2.2: Relaciones entre los participantes de la Arquitectura Orientada a Servicios.

Un proveedor se considera como el prestador de un servicio y se encarga principalmente de publicar la descripción y administrar las solicitudes del mismo. El registro o repositorio administra la información de los proveedores (información del negocio) y los servicios que ofrecen (información para utilizar un servicio en específico). Finalmente un cliente es aquél que se encarga de descubrir un servicio a través del registro para posteriormente realizar su invocación.

2.1.2. SOAP, Simple Object Access Protocol

SOAP (Simple Object Access Protocol) [7] es una especificación que describe las convenciones del formato para encapsular datos, de tal manera que permita enviar y

recibir documentos XML independientemente del protocolo utilizado o de la estructura del documento que se envía [8].

SOAP puede ser utilizado para describir un documento XML de tal manera que siga la semántica de una llamada a un procedimiento remoto (Figura 2.3). Este tipo de mensaje describe entonces la llamada a un procedimiento y su respuesta mediante el nombre del procedimiento y el valor de sus parámetros.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV: Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">

  <SOAP-ENV: Body>
  <!--Inicio de la llamada al procedimiento del servicio Web -->
  <!--in0, in1, in2 son los parámetros de entrada del procedimiento -->
  <ns1:getDVDs xmlns:ns1="http://dvdonline-interface">
    <in0 xsi:type="xsd:string">Action/Adventure</in0>
    <in1 xsi:type="xsd:int">10</in1>
    <in2 xsi:type="xsd:string">rick@learningpatterns.com</in2>
  </ns1:getDVDs>
  <!-- Fin de la llamada al procedimiento del servicio Web -->
  </SOAP-ENV: Body>

</SOAP-ENV: Envelope>
```

Figura 2.3: Ejemplo de un mensaje SOAP de invocación.

La estructura de un mensaje SOAP se conforma de los siguientes elementos:

1. Un sobre que corresponde al elemento raíz <`envelope`>, el cual incluye el cuerpo del mensaje y opcionalmente un encabezado.

2. El encabezado <`header`> puede incluirse para describir metadatos referentes a la seguridad, transacciones, entre otros.

3. El cuerpo del mensaje corresponde al elemento <`body`> y contiene básicamente la información enviada o recibida de un servicio Web.

4. Los elementos denominados <faults> pueden ser utilizados para describir situaciones anormales que pueden ocurrir durante la lectura del mensaje.

De acuerdo a esta estructura, la Figura 2.3 corresponde al ejemplo de un mensaje SOAP de invocación de un servicio Web y la Figura 2.4 al ejemplo de un mensaje SOAP de respuesta.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV: Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <SOAP-ENV: Body>
  <!--Inicio del mensaje de respuesta-->
  <!--El valor de retorno es un arreglo de cadenas, donde su longitud
    en este caso es 1 -->
    <ns1:getDVDsResponse
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="http://dvdonline-interface">
      <getDVDsResult xsi:type="SOAP-ENC:Array"
       SOAP-ENC:arrayType="xsd:string1"
       xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
        <item xsi:type="xsd:string">Fight Club</item>
      </getDVDsResult>
    </ns1:getDVDsResponse>
  <!--Fin del mensaje de respuesta -->
  </SOAP-ENV: Body>

</SOAP-ENV: Envelope>
```

Figura 2.4: Ejemplo de un mensaje SOAP de respuesta.

El proceso general de invocación a un servicio Web mediante el protocolo SOAP se describe en la Figura 2.5. Como primer paso, el cliente consulta un nodo UDDI para obtener la información WSDL correspondiente a un servicio Web. Posteriormente, genera un mensaje SOAP correspondiente a la invocación del servicio. Este mensaje es analizado por el servicio Web de tal manera que recupera la información necesaria para ejecutar el proceso asociado a este. Finalmente, el servicio Web envía al cliente un mensaje SOAP con la respuesta de dicho proceso.

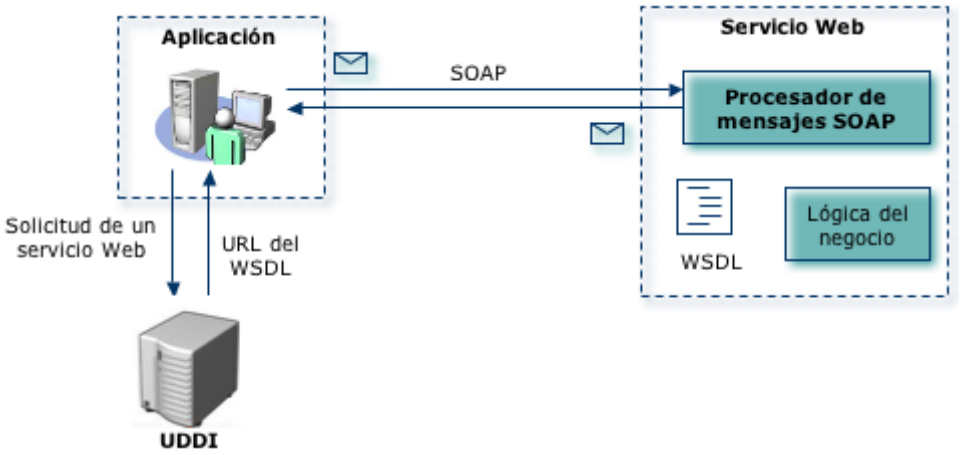

Figura 2.5: Proceso general de invocación a un servicio Web mediante SOAP.

2.1.3. Descripciones de servicios Web

La descripción de servicios Web corresponde a un lenguaje basado en XML denominado WSDL (Web Services Description Language) [9]. WSDL describe la sintaxis de las interfaces y ubicación asociadas a un servicio Web [10].

En la Figura 2.6 se muestra la estructura general de un documento WSDL. Esta estructura tiene el elemento <definitions> como raíz, el cual a su vez puede contener los elementos <types>, <message>, <portType>, <binding> y <service>.

La descripción correspondiente a cada elemento es la siguiente:

- El elemento <definitions> actúa como un contenedor para la descripción del servicio y permite realizar la declaración de los espacios de nombres asociados al resto del documento.
- El elemento <types> provee información de los tipos de datos complejos utilizados por el elemento <message>.
- El elemento <message> definen el formato de los datos asociados a los mensajes intercambiados entre un cliente y el servicio a través de los subelementos <part>.
- El elemento <portType> especifica un identificador único a un grupo de operaciones que pueden ejecutarse en uno o varios puntos de acceso.

```
< definitions ...>
    < types>
        <schema>...</schema>*
    < /types>
    < message name="...">*
        <part name="..." type="..."/>*
    < /message>
    < portType name="...">*
        <operation name="...">*
            <input message="..."/>
            <output message="..."/>
        </operation>
    < /portType>
    < binding name="..." type="...">*
        <operation name="...">*
            <input>...</input>
            <output>...</output>
        </operation>
    < /binding>
    < service name="...">*
        <port binding="..." name="...">...</port>
    < /service>
< /definitions>

Nota: Los elementos marcados con * pueden aparecer más de una vez.
```

Figura 2.6: Estructura general de un documento WSDL.

- El elemento <binding> describe cómo se puede invocar una operación, especificando el protocolo y el formato de los datos para cada una de las operaciones y sus correspondientes mensajes.
- El elemento <service> aparece generalmente al final del documento e identifica al servicio y define los subelementos <port> que identifican cada punto de acceso.

2.1.4. Repositorio de servicios Web UDDI

UDDI (Universal Description, Discovery and Integration) [11] provee métodos estandarizados para publicar y descubrir información asociada a los servicios Web. Así mismo, tiene el objetivo de crear una plataforma independiente para describir

servicios, descubrir negocios e integrar servicios de negocios.

Conceptualmente, un negocio puede registrar tres tipos de información en un nodo UDDI:

1. Páginas blancas (*white pages*). Corresponde a la información básica de una compañía incluyendo el nombre del negocio, dirección, información de contactos e identificadores únicos.

2. Páginas amarillas (*yellow pages*). Información que describe las categorías a las que pertenecen los servicios Web, basadas en ontologías establecidas.

3. Páginas verdes (*green pages*). Corresponde a la información técnica que describe el comportamiento y funciones soportadas por un servicio Web. Esta información incluye sus descripciones y ubicación.

UDDI tiene diferentes funciones dependiendo de la perspectiva de quien lo utilice, ya que puede verse como un proveedor de servicios para realizar la publicación de servicios Web, como un registro de servicios o como un servicio de descubrimiento para localizar y ligar un servicio Web con un cliente [8].

La especificación de UDDI provee APIs para realizar la consulta y publicación de información de servicios. La primera de ellas, ubica la información de un servicio y su descripción. La segunda, se utiliza para crear, almacenar y actualizar la información almacenada en un nodo UDDI. Algunas de estas operaciones son *find_business, find_service, get_businessDetail, get_tModelDetail, delete_business, delete_service, save_business* y *save_service*.

Actualmente, se han creado diversos nodos UDDI privados (disponibles en ibm.com, microsoft.com, uddi.org y xmethods.com), por lo que las compañías se enfocan al uso de los mismos, y a pesar de que UDDI cuenta con ciertas limitaciones, es un gran soporte para la industria ya que sigue siendo considerado como el estándar utilizado para registrar y descubrir servicios Web.

En las secciones anteriores se describieron los conceptos y tecnologías relacionadas con los servicios Web. En este trabajo proponemos ocultar los detalles de distribución, coordinación y comunicación correspondientes a los servicios Web a través de módulos que se entrelazarán con la aplicación original en tiempo de compilación. Por lo tanto, proponemos utilizar el enfoque de la programación orientada a aspectos.

2.2. Programación orientada a aspectos

En la actualidad, el paradigma de la programación orientada a objetos es el que predomina en el desarrollo e integración de sistemas, pero que a pesar de ser una idea clara, tiene ciertas limitantes como la dificultad de localizar características que engloben restricciones y comportamientos generales a nivel código [12] [13]. Por lo tanto, han surgido nuevos enfoques de programación que tienen como objetivo proporcionar mayor expresividad al paradigma de objetos y lograr una clara separación de las propiedades, requerimientos o áreas de interés (*concerns*) a nivel código.

La programación orientada a aspectos (POA) surgió recientemente como un nuevo enfoque de programación con el propósito de proveer las técnicas y herramientas que ayudarán a modularizar aquellos requerimientos que afectan múltiples módulos de un sistema [14].

Podemos visualizar un sistema de software complejo como la combinación de múltiples requerimientos o áreas de interés (Figura 2.7). Un sistema típico incluye la lógica de negocios, distribución, persistencia de datos, ingreso al sistema (*logging*), seguridad, manejo de excepciones y errores, entre otros. De igual manera, surgen nuevos requerimientos durante su etapa de desarrollo como la mantenibilidad, comprensibilidad y escalabilidad [15].

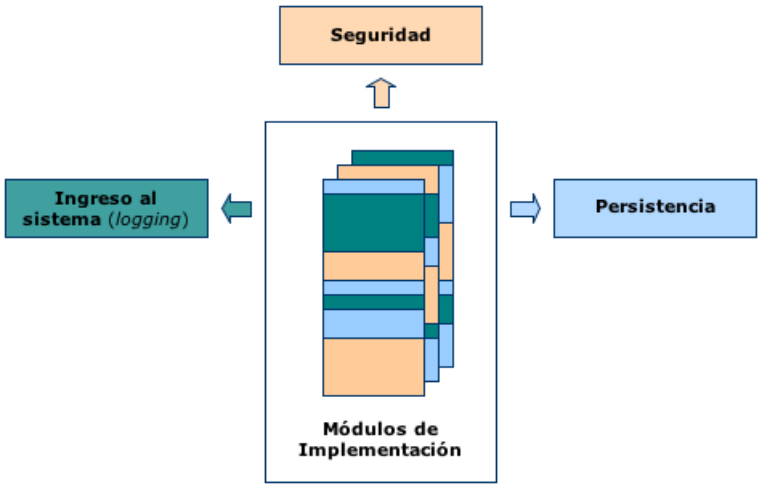

Figura 2.7: Módulos de implementación correspondientes a diversos requerimientos.

La programación orientada a aspectos se basa en la idea de que los sistemas se estructuran y programan eficientemente al separar y programar separadamente cada uno de sus correspondientes requerimientos y las descripciones de las relaciones existentes entre ellos, para integrarlos finalmente a un sistema. POA introduce el concepto de aspectos, los cuales encapsulan en módulos reutilizables aquellos comportamientos que afectan múltiples clases.

2.2.1. Conceptos básicos

Un aspecto se conforma esencialmente de puntos de unión, puntos de corte y modificadores de comportamiento. Los puntos de unión (*join points*) son puntos bien definidos en la ejecución de un programa que permiten identificar eventos que suceden sobre dichos puntos. Los puntos de corte (*pointcuts*) describen un conjunto de puntos de unión y, opcionalmente, los valores de los mismos. Los modificadores de comportamiento (*advices*) corresponden al código que se ejecuta antes, después o durante un punto de unión.

Debido a que un aspecto puede contribuir a la implementación de un número de procedimientos, módulos u objetos, hay que considerar la definición de puntos de unión y su interacción con el resto de la aplicación o sistema, si la implementación de los aspectos se realizará de manera estática o dinámica y en que momento se aplicará un aspecto a cierta parte de código.

Es así que existen tres modelos de integración de aspectos:

- En tiempo de compilación, lo cual corresponde al momento en que se compila el sistema a través de compiladores que proveen algunos de los lenguajes orientados a aspectos. De esta manera, los sistemas serán eficientes al cargar las clases así como en su ejecución.

- En tiempo de carga de clases a una máquina virtual, lo cual permite a los desarrolladores o administradores modificar la configuración y reiniciar el sistema.

- En tiempo de ejecución, lo cual permite integrar aspectos en cualquier momento durante la ejecución del sistema, sin necesidad de reiniciar el sistema o forzar la carga de las clases afectadas.

2.2.2. Lenguajes orientados a aspectos

Como cualquier otra metodología de programación, las implementaciones que se han desarrollado para la programación orientada a aspectos, consisten de dos partes: la especificación de un lenguaje y su respectiva implementación [15]. La primera describe la sintaxis del mismo y la segunda verifica la correctitud del código de acuerdo a la especificación del lenguaje para convertirlo en código ejecutable.

Un lenguaje orientado a aspectos define la sintaxis del código correspondiente a un aspecto y las reglas para realizar su integración a un sistema y obtener el sistema final. Como consecuencia, se han desarrollado varias herramientas que manejan los mecanismos del lenguaje correspondiente a la programación orientada a aspectos basadas en Java como AspectJ, AspectWerkz, JBoss AOP, Spring AOP, JAsCo, entre otras. Estas herramientas emplean un modelo para definir los puntos de unión así como los mecanismos para definir la estructura del programa para la implementación de los aspectos (ver Tabla 2.1).

<table>
<tr><th></th><th>Def. aspectos</th><th>Impl. <code>advice</code></th><th>Def. <code>pointcut</code></th><th>Configuración</th></tr>
<tr><td>AspectJ</td><td colspan="3">código (extensión al lenguaje)</td><td>archivo .lst</td></tr>
<tr><td>AspectWerkz</td><td rowspan="3">anotaciones o
XML</td><td rowspan="3">definición
método
Java</td><td rowspan="3">valor de
una
cadena</td><td>aop.xml</td></tr>
<tr><td>JBoss AOP</td><td>jboss-aop.xml</td></tr>
<tr><td>Spring AOP</td><td>springconfig.xml</td></tr>
<tr><td>JAsCo</td><td colspan="3">código (extensión al lenguaje)</td><td>conectores</td></tr>
</table>

Tabla 2.1: Comparación de sintaxis entre diferentes herramientas orientadas a aspectos.

AspectWerkz [16] provee dos maneras para realizar la declaración de los aspectos: anotaciones y definición de documentos XML. La más utlizada son las anotaciones, las cuales pueden realizarse al estilo Javadoc. La definición de aspectos a través de documentos XML se realiza al especificar etiquetas que definen aspectos, puntos de corte y `modificadores de comportamiento`, cuya implementación corresponde a la declaración de un método Java. Para la ejecución de un programa en AspectWerkz sólo es necesario incluir las librerías correspondientes al `CLASSPATH` y un archivo de configuración que indique la inclusión de los aspectos a una aplicación.

JBoss AOP [17] ofrece la posibilidad de realizar anotaciones al igual que AspectWerkz, pero también se puede realizar la declaración de un documento XML para la

definición de un aspecto. Al igual que AspectWerkz, la implementación de un modificador de comportamiento se realiza como un método Java. Dicho método es invocado a través del sistema de JBoss AOP, el cual provee mecanismos de reflexión para acceder objetos durante su ejecución. Así, el sistema de JBoss AOP determina durante la ejecución de una aplicación cuándo deben ser ejecutados dichos métodos.

Spring AOP [18] también se basa en la definición de documentos XML, con la diferencia de que se dichos documentos especifican mayor información para la definición y configuración de los aspectos. La implementación del modificador de comportamiento corresponde a un método Java con parámetros especiales, necesarios para la ejecución del mismo. Dicha ejecución es determinada por el sistema durante la ejecución de una aplicación y por lo tanto sólo es necesario realizar la configuración del sistema Spring.

JAsCo [19] se basa en AspectJ y componentes aspectuales. Introduce los conceptos de *aspect beans* y conectores. Los *aspect beans* describen el comportamiento que interfiere con la ejecución de un componente utilizando un tipo de clase interna llamada anzuelo y los conectores especifican cuando es necesario utilizarlos. Un anzuelo especifica al menos un constructor y un método de comportamiento (`before, replace, after`).

En este trabajo de tesis proponemos el uso de AspectJ [20] [21], el cual es una extensión al lenguaje Java para soportar el desarrollo de aplicaciones orientadas a aspectos. AspectJ provee un conjunto de directivas para trabajar con aspectos, los cuales se conforman por la declaración de puntos de corte y modificadores de comportamiento.

AspectJ cuenta con 17 primitivas que permiten realizar la definición de puntos de corte ya sea utilizándolas individualmente o a través de la combinación de las mismas. En esta propuesta utilizamos sólo las primitivas `execution` para identificar la ejecución de métodos y `args` para tener acceso a los valores de los argumentos de un punto de unión alcanzado. Así mismo, utilizamos los modificadores de comportamiento `before` y `around`, los cuales corresponderán al código que reemplazará la ejecución de un método local.

Proponemos utilizar AspectJ debido a que es la herramienta que provee una estructura sencilla y clara para la definición de los aspectos y que no involucra la definición de un documento XML para la inclusión y/o configuración de los mismos como en el caso de AspectWerkz, JBoss AOP y Spring AOP. Así mismo, ofrece la posibilidad de ejecutar el comportamiento original a través de `proceed()`, a diferencia de JAsCo.

La Figura 2.8 muestra el ejemplo de un aspecto que utiliza las primitivas `execution` y `around` para la definición de un punto de corte asociado a la transacción de la consulta de saldo de una cuenta. En este ejemplo se considera incorporar los requerimientos asociados a la validación del nip de un usuario y al registro de la transacción, antes y después de que se ejecute el proceso correspondiente a dicha transacción. De igual manera, se extiende su comportamiento al agregar el desplegado de información adicional.

```
1  aspect ejemploAspecto {
2
3    pointcut ejemplo(String s1, String s2) :
4      execution(Cuenta.saldo(String, String)) &&  args(s1, s2);
5
6    before (String s1, String s2) : ejemplo(s1, s2) {
7     validarNIP(s1, s2);
8   }
9
10  void  around (String s1, String s2) : ejemplo(s1, s2) {
11    System.out.println("Gracias por su preferencia.");
12     proceed(s1,s2);
13  }
14
15   after (String s1, String s2) : ejemplo(s1, s2) {
16    registrarTransaccion(s1, s2);
17  }
18 }
```

Figura 2.8: Ejemplo de un aspecto en AspectJ.

Construir un programa de AspectJ es similar a construir un programa de Java ya que su compilador genera código ejecutable para el código referente a los aspectos y el código fuente del sistema. Por lo tanto, realizar la ejecución de un programa con los aspectos ya integrados es igual que ejecutar un programa de Java.

2.3. Trabajo relacionado

Actualmente, el desarrollo de tecnologías para la implementación y mantenimiento de servicios Web sigue en continuo proceso. En contraste, el desarrollo de tecnologías

para dar soporte dinámico a las aplicaciones cliente que requieran integrar servicios Web es limitado. Aplicar la programación orientada a aspectos a los servicios Web es una idea que recientemente está tomando fuerza. Por lo tanto, podemos observar que no existen muchas propuestas relativas al área.

En esta sección presentamos los trabajos que consideramos tienen alguna relación con nuestra propuesta de tesis.

2.3.1. Modelo de servicios Web mediante mecanismos de reflexión

En [22] se propone un modelo de diseño de servicios Web denominado RAWS (Reflective and Adaptable Web Services), el cual utiliza mecanismos de reflexión para su implementación debido a las ventajas que presenta al permitir modificar una aplicación en tiempo de ejecución.

Las principales características que presenta este modelo son:

1. Presenta una arquitectura de dos capas donde se hace uso de la reflexión.
2. Permite realizar un diseño flexible y adaptable de los servicios Web.
3. Permite la modificación dinámica de la definición e implementación de servicios Web durante su ejecución.
4. Aplica la reflexión estructural para modificar la descripción de un servicio independientemente de su implementación. De igual manera, permite modificar la estructura del código correspondiente al mismo.
5. Aplica la reflexión de comportamiento para modificar el comportamiento de un servicio dinámicamente ya sea para extenderlo o modificarlo completamente.

Básicamente, la arquitectura que presenta este modelo se conforma de la capa asociada a la descripción del servicio Web (WSDL), así como su implementación y a la capa donde se define la metainformación asociada al mismo. Esta última es aquella que permite modificar la estructura y el comportamiento de un servicio Web de manera dinámica.

Un servicio Web se comunica entonces con la capa de su correspondiente metainformación de tal manera que cuando se requiera realizar un cambio en su estructura

o en su implementación se realice a través de dicha metainformación y se modifique la estructura básica del servicio.

Al igual que nuestra propuesta, se identifican los módulos que pueden ser modificados y se evita modificar el código de la implementación original al realizar cambios en el comportamiento de la misma. A diferencia de nuestra propuesta, donde consideramos cualquier aplicación este artículo sólo se enfoca a la implementación de los servicios Web, lo cual representa una limitación en los módulos que pueden ser modificados. Así mismo, los enfoques utilizados para lograr los objetivos presentados son diferentes, aunque coinciden con el objetivo de sustituir el comportamiento de métodos de la implementación.

2.3.2. Composición de servicios Web con AO4BPEL

En el artículo [23] se propone una extensión de BPEL4WS (Bussiness Process Execution Language for Web Services) enfocada a la programación orientada a aspectos para la composición de servicios Web. Con el modelo de dicha extensión, se pretende mejorar la modularidad de procesos e incrementar la flexibilidad y adaptabilidad para la composición de servicios Web.

El modelo que presenta AO4BPEL no es específicamente para BPEL, por lo que puede ser aplicado a cualquier lenguaje de composición de servicios que soporte la ejecución de procesos de negocios. En este caso, BPEL fue seleccionado debido a que es un lenguaje que está convirtiéndose en el estándar para la composición de servicios Web.

Los aspectos y procesos están especificados en XML y, así como AspectJ, soporta los modificadores de comportamiento `before`, `after` y `around`, los cuales corresponden a actividades especificadas en BPEL que deben ser ejecutada antes, después o en lugar de otra actividad.

Con los aspectos se captura la composición de los servicios en una forma modular, de tal forma que se convierta en una composición más dinámica cuando se requieran cambios. Por tal motivo, se pretende que las limitaciones de los lenguajes de composición, tales como la modularización y la adaptación dinámica de composición de servicios Web sean eliminadas mediante el uso de este enfoque de programación.

Al proponer la extensión de un lenguaje composicional como BPEL, se define la

estructura de los aspectos mediante etiquetas XML de tal manera que se identifiquen puntos de corte y modificadores de comportamiento a través de la definición de nuevas etiquetas.

La idea de modularizar un conjunto de actividades mediante la definición de un aspecto es una buena alternativa para la definición de un proceso que requiere separar requerimientos que afecten varias actividades del proceso. Esta idea es similar a la que presentamos en nuestra propuesta, con la diferencia que nosotros no partimos de un lenguaje composicional, sino de una aplicación escrita en Java.

Una de las aportaciones de este trabajo al nuestro es la idea de definir información asociada a los puntos de corte y modificadores de comportamiento a través de etiquetas XML. En nuestro caso, la definición de dicha información corresponderá a un documento XML independiente de la aplicación.

2.3.3. Capa de administración de servicios Web

En [24] se propone una capa de administración localizada entre la aplicación y los servicios Web denominada WSML (Web Services Management Layer). Dicha capa permite la selección e integración dinámica de servicios Web, así como la administración de los mismos del lado del cliente y el soporte a reglas de selección, integración y composición.

WSML contempla el uso de aspectos, por lo que para la integración de estos propone el uso de un lenguaje dinámico de programación llamado JAsCO, el cual introduce dos nuevos conceptos: *aspect beans* y conectores. Los *aspect beans* corresponden a extensiones de Java Beans que especifican cuándo la ejecución de un método debe ser interceptada y qué comportamiento extra debe ejecutarse. Los conectores indican dónde es necesario usar *aspect beans* y declarar cómo deben colaborar.

Las peticiones de las aplicaciones se realizan de manera abstracta mediante la definición de interfaces de servicios, y la redirección de aspectos define la lógica para interceptar las peticiones de servicios reemplazándolos por una invocación concreta de otro servicio Web. Cada uno de los aspectos definidos son genéricos, por lo que no se refieren a un servicio Web en específico. Por lo tanto, la especificación de cada uno de ellos se realiza a través de los conectores.

Al controlar la integración y configuración de los servicios Web en el lado del

cliente, las aplicaciones se vuleven más robustas pues manejan dinámicamente las fallas de los servicios para reemplazarlos por aquellos que provean la funcionalidad equivalente y lograr mayor flexibilidad en la integración de estos.

WSML es uno de los trabajos donde se maneja la integración de los servicios Web mediante aspectos, ya que se pretende administrarlos eficientemente para realizar la sustitución de estos cuando sea requerido.

Las ideas que presenta este trabajo nos ayudan a analizar las diferentes alternativas con las que se implementan los aspectos y la importancia de realizar su integración a una aplicación dinámicamente.

A diferencia de WSML que se enfoca a sustituir servicios Web previamente integrados a una aplicación por otros, nosotros nos enfocamos en aplicaciones centralizadas y proponemos una arquitectura más simple que selecciona e integra dinámicamente servicios Web para convertirla en una aplicación distribuida mediante la generación de un conjunto de aspectos.

Por otra parte, nuestra propuesta difiere con la de este trabajo en el lenguaje de programación utilizado y en la manera de definir los aspectos. Nosotros proponemos utilizar el lenguaje AspectJ en lugar del lenguaje JAsCo debido a que el modificador de comportamiento `around` que provee AspectJ se considera más robusto al ofrecer la capacidad de ejecutar el comportamiento original en caso de ser requerido, lo cual no soporta el método `replace` de JAsCo.

WSML propone realizar la selección de servicios en base a reglas, lo cual no contemplamos en la presente propuesta. No obstante, nosotros proponemos una extensión al directorio de servicios al implementar un mecanismo de suscripción de servicios Web.

2.3.4. Integración de WSLAs a los servicios Web

En [25] se propone utilizar los conceptos de la programación orientada a aspectos para integrar WSLAs (Web Service Level Agreement) a un servicio Web, las cuales tienen el objetivo de especificar restricciones para la ejecución de un servicio Web. En este artículo se propone el uso de varias herramientas para lograr dicho objetivo, las cuales corresponden a Spi, XIP, ISG.

Básicamente, Spi es un lenguaje que captura el diseño de la aplicación describiendo cada paso como un poceso (*pipe*) que tiene sus respectivas entradas, salidas y funcionalidad. Una vez que se realiza la descripción en Spi, se compila a través de XIP (XML for Infopipes) para obtener estructuras XML y de esta manera servir como entrada a ISG (Infopipe Stub Generator), el cual genera las interfaces (*stubs*) necesarias para manejar las conexiones, la serialización de datos, etc. ISG está conformado de varios módulos entre los cuales se encuentra AXpect, cuya funcionalidad principal es entrelazar los aspectos con el código original.

Este artículo describe la necesidad de especificar cada aspecto a nivel de XIP, ya que simplemente se especifican mediante etiquetas XML el nombre e información adicional de los aspectos que serán integrados. AXpect se encarga entonces de interpretar las etiquetas donde se especifica cada aspecto que se utilizará, cargarlo desde disco y generar el documento XSLT correspondiente al código entrelazado con los aspectos, y finalmente, generar el código C del servicio Web con las restricciones ya integradas.

Al igual que algunos de los trabajos descritos anteriormente, se identifica la utilidad de la programación orientada a aspectos y en este caso se aplica para integrar restricciones a las implementaciones de servicios Web (WSLAs). Como ya se ha mencionado, nosotros proponemos integrar servicios Web a una aplicación de una manera más simple evitando modificar el código de la aplicación original y sin la necesidad de conocer un lenguaje como Spi para realizar la definición de los procesos involucrados con una aplicación. A diferencia de este trabajo nosotros proponemos la generación dinámica de aspectos que al ser integrados no afectan el código original.

A pesar de presentar un enfoque diferente al de nuestra propuesta, podemos observar la necesidad común de realizar una especificación de los módulos que serán afectados por un conjunto de aspectos y entrelazarlos.

2.4. Sumario

En este capítulo presentamos los conceptos correspondientes a los servicios Web y al enfoque de la programación orientada a aspectos. Así mismo, presentamos el trabajo relacionado con nuestra propuesta de tesis.

En el caso de los servicios Web, presentamos la tecnología con la cual se relacionan. SOAP, el cual es utilizado como protocolo de comunicación ya que provee una

estructura simple para enviar mensajes y realizar llamadas a procedimientos remotos; WSDL, el cual corresponde al estándar para la descripción de servicios; y UDDI, el cual corresponde al repositorio de descripciones de servicios Web.

En el caso de la programación orientada a aspectos, presentamos los conceptos que introduce dicho enfoque y las características de los lenguajes que se han desarrollado para su implementación. Podemos observar que los beneficios de este enfoque de programación son principalmente modularizar la implementación asociada a los requerimientos entrelazados, facilitar la modificación de sistemas, permitir reutilizar código y permitir integrar nuevos requerimientos como nuevos aspectos.

Para el trabajo relacionado que presentamos en la sección 2.3 describimos la propuesta de cada uno de ellos, así como las similitudes y diferencias con nuestra propuesta. Es claro que el campo de estudio relativo a los servicios Web y a la programación orientada a aspectos empieza a experimentar mayor auge y que por lo tanto, nuevas propuestas empiezan a desarrollarse. Algunas de ellas se enfocan al desarrollo de servicios Web orquestados y otras, al igual que en nuestra propuesta, a la selección e invocación dinámica de servicios.

Capítulo 3

Descripción del Sistema WSSI

El descubrimiento dinámico de servicios Web, así como su integración dinámica, son los objetivos fundamentales de WSSI (Web Services Selection and Integration) y del Intermediario (Broker). En este capítulo se presenta la descripción general del funcionamiento de WSSI, el cual realiza una serie de consultas al Intermediario y genera los archivos asociados a cada aspecto, y posteriormente un caso de estudio que describe los problemas que resuelve conjuntamente con el Intermediario. El caso de estudio ejemplifica cómo a partir de una aplicación centralizada se obtiene la versión distribuida de la misma.

3.1. Descripción general

Típicamente, la integración de servicios Web a una aplicación se considera desde el diseño de la misma, ya que se tienen que establecer los protocolos y las herramientas que se utilizarán para su integración. Como consecuencia se debe conocer la localización de todos los servicios Web que se invocarán. Por lo tanto, este proceso de desarrollo presenta restricciones de implementación y mantenimiento debido a que pueden resultar una tarea compleja.

En esta sección describimos el proceso que se lleva a cabo para obtener la versión distribuida de una aplicación centralizada al realizar la integración de servicios Web mediante el uso de la programación orientada a aspectos y se describen los casos que se contemplan para la generación de los aspectos:

1. Consulta a un documento WSDL.

2. Descubrimiento de servicios Web, los cuales pueden o no estar publicados.
   Para el caso en que no se encuentran publicados los servicios Web, se describe:

   a) La generación de suscripciones.

   b) La serialización y deserialización de suscripciones por las que se esperaba respuesta.

   c) La notificación de respuesta a una suscripción.

   d) La dependencia de servicios Web no implementados.

3.1.1. Identificación de métodos

La identificación de métodos es lo primero que se debe realizar para poder iniciar el proceso de selección e integración de servicios Web.

En una aplicación centralizada, se tienen que identificar las operaciones correspondientes a métodos locales pertenecientes a una clase, los cuales deben ser sustituidos por invocaciones a servicios Web.

Para que WSSI y el Intermediario realicen el proceso de selección e integración de servicios Web, hay que considerar la existencia de dos casos fundamentales:

1. Se tiene conocimiento de la ubicación del WSDL correspondiente a un servicio Web.

2. No se tiene conocimiento de la ubicación de un servicio Web.

Según sea el caso al que corresponda, a partir de la definición de un documento WSSIAD (WSSI Aspect Definition)[1], el cual corresponde a una descripción XML donde se especifica la información de cada uno de los métodos identificados, WSSI iniciará la generación de aspectos, los cuales serán los encargados de realizar las invocaciones a los servicios Web integrados a la aplicación.

3.1.2. Generación de aspectos: consulta a documentos WSDL

Cuando se tiene conocimiento de la ubicación del WSDL correspondiente a un servicio Web que se desea integrar, la definición del documento WSSIAD tiene que especificar el nombre del método y la clase a la que pertenece. De igual manera, se

[1] La descripción detallada del lenguaje WSSIAD se encuentra en el Capítulo 4, sección 4.2.

especifica el URL correspondiente al documento WSDL del servicio Web.

El proceso que WSSI ejecuta corresponde a realizar un análisis del documento WSDL con el objetivo de obtener el nombre de los datos asociados a los mensajes SOAP de invocación y respuesta correspondientes a la operación definida en el documento.

Así, con la información obtenida durante el proceso, se genera entonces el archivo correspondiente a un aspecto AspectJ que define el punto de corte asociado al método especificado y a los modificadores de comportamiento before y around.

El modificador de comportamiento before especifica el URL del WSDL y el nombre de la operación del servicio Web, así como un objeto con la instancia de los datos necesarios para la invocación del servicio Web. De igual manera, se inicializan los elementos necesarios para la generación del mensaje SOAP de invocación (espacio de nombres, puerto, tipo de codificación de los datos, entre otros). Por otro lado, el modificador de comportamiento around realiza la invocación del servicio Web para analizar la respuesta de este según corresponda.

Observamos que el proceso de generación de aspectos que realiza consultas a documentos WSDL es el caso más simple, ya que WSSI sólo tiene que realizar el análisis de dichos documentos.

3.1.3. Generación de aspectos: descubrimiento de servicios Web

Consideremos ahora el segundo caso, en donde no se tiene conocimiento de la ubicación de los servicios Web que se desean integrar. En este caso, la definición del documento WSSIAD tiene que incluir el URL del Intermediario en lugar del URL de un documento WSDL y la descripción de los datos asociados a los mensajes SOAP de invocación y respuesta asociadas a la operación del servicio Web.

WSSI crea un conjunto de procesos encargados de generar los archivos correspondientes a los aspectos. Cada proceso realiza una consulta al Intermediario de acuerdo a la descripción de la operación especificada por el documento.

En caso de que el Intermediario (Broker) tenga publicada una descripción que

corresponda con la solicitada, este regresa como respuesta el URL del WSDL y se genera el aspecto correspondiente. En caso contrario se genera una suscripción que obligará a WSSI esperar por una respuesta. Así, una vez que un proveedor publique alguna de las descripciones solicitadas, el Intermediario notificará la respuesta a la suscripción correspondiente por la cual WSSI esperaba.

3.1.4. Serialización de suscripciones

Aquellas suscripciones por las que WSSI espera una respuesta y que corresponden a suscripciones generadas por el Intermediario, pueden ser serializadas cuando sea necesario detener la ejecución de WSSI.

El proceso de serialización comienza entonces cuando todos los procesos hayan terminado su ejecución o se encuentren en espera de una respuesta. Básicamente se serializa la información correspondiente al proceso: identificador del proceso, UUID asociado a la suscripción y toda la información recabada durante su ejecución hasta ese punto.

3.1.5. Deserialización de suscripciones

La información de los procesos correspondientes a suscripciones es deserializada al ejecutarse nuevamente WSSI ya que primero se verifica si el archivo contiene información serializada. En caso de ser así, se deserializa la información de cada uno de ellos para generar un nuevo proceso que reanude la espera por la suscripción correspondiente.

3.1.6. Notificación de respuesta a una suscripción

Como se describió anteriormente, cuando se realiza al Intermediario la consulta de una descripción asociada a la operación de un servicio Web que no tenga publicada, se genera una suscripción de la misma. En ese momento WSSI espera por una respuesta hasta que un proveedor publique el servicio correspondiente a través del *script* que WSSI proporciona para la publicación de servicios.

Cuando un proveedor publica el URL del documento WSDL correspondiente a un servicio, el Intermediario realiza el análisis del documento y obtiene la descripción de la operación para agregarla a su repositorio. Posteriormente, identifica qué suscripciones pueden ser contestadas. En ese momento se notifica la respuesta a cada una

de ellas. Por su parte, WSSI recibe la respuesta y genera el archivo correspondiente a un aspecto, así como los archivos de compilación para la aplicación.

3.1.7. Dependencia de servicios Web no implementados

Cuando un proveedor desea implementar un servicio Web que necesita a su vez de otros servicios Web puede hacer uso de WSSI para descubrir dinámicamente los servicios necesarios. Esta dependencia de servicios corresponde al problema de la cadena de suministro, donde el caso más simple corresponde a una cadena de suministro de dos niveles. Así, el proceso completo para la generación de los aspectos necesarios, integra en el segundo nivel de la cadena un servicio Web a la implementación de otro servicio Web, y en el primer nivel un servicio a una aplicación cliente.

Tanto para el primer y segundo nivel de la cadena, el proceso que se realiza es el mismo pero entre diferentes participantes. El primer nivel correspondería a las interacciones entre WSSI-cliente, Intermediario y proveedor 1. Para el segundo nivel serían las interacciones entre WSSI-proveedor 1, Intermediario y proveedor 2.

Una vez que el proveedor 2 publica la descripción requerida por el proveedor 1, el Intermediario notifica una respuesta a su correspondiente suscripción. Como consecuencia, se genera el aspecto que se integrará a la implementación del servicio Web implementado por el proveedor 1. De esta manera, el proveedor 1 puede publicar el servicio Web requerido por el cliente, lo cual implica que el Intermediario notifique la respuesta a la suscripción por la que este esperaba y por lo tanto se genere el aspecto correspondiente.

Es entonces que el aprovisionamiento de los servicios Web finaliza satisfactoriamente para ambos niveles de la cadena de suministro.

3.2. Caso de estudio

Supongamos un escenario en el que se tiene una aplicación centralizada (Figura 3.1) que realiza las operaciones básicas de consulta, venta y registro de libros.

La aplicación Book Store funciona correctamente de manera local, pero se requiere convertirla a una aplicación distribuida haciendo uso de servicios Web. Los problemas que presenta realizar esta conversión son esencialmente:

- Identificar los módulos de la aplicación que deben ser modificados.
- Definir los módulos que deben ser agregados a la aplicación.
- Determinar el protocolo y herramientas para realizar invocaciones a servicios Web.
- Rediseñar algunos módulos de la aplicación o rediseñar completamente la misma.

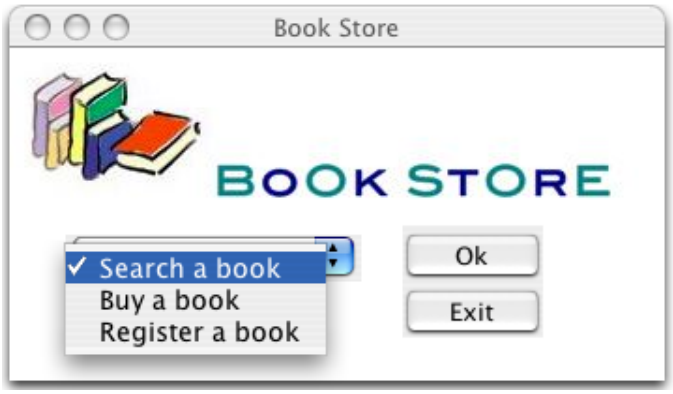

Figura 3.1: Caso de estudio: Tienda de libros Book Store.

Es así, que presentamos como WSSI y el Intermediario permiten realizar la selección e integración de los servicios Web a esta aplicación de una manera más simple y transparente a los desarrolladores sin tener que modificar el código de la aplicación original. Se describen los diferentes casos que se pueden presentar y cómo cada uno de ellos es resuelto por los mismos.

3.2.1. Identificación de métodos

En la Tabla 3.1 se encuentra la descripción asociada a cada una de las operaciones que forma parte de la aplicación correspondiente a la tienda de libros Book Store. Dichas operaciones corresponden a los métodos de la clase `Store` que deben ser sustituidos por invocaciones a servicios Web.

Operación	Descripción
Búsqueda de libros	`String searchBook(String)`
Obtener cotización	`float getQuote(String,String)`
Compra de libros	`String buyBook(String,int,float,String,String)`
Registro de libros	`String registerBook(String,String,String, float,int)`

Tabla 3.1: Descripción de las operaciones asociadas a una tienda de libros.

3.2.2. Generación de aspectos: consulta a documentos WSDL

Consideremos que se tiene conocimiento de un servicio Web que provee la funcionalidad de registrar libros de acuerdo al id del libro proporcionado. La definición del documento WSSIAD que necesita WSSI para iniciar el proceso de generación de aspectos sería el que se muestra en la Figura 3.2.

```
1  <?xml version="1.0"?>
2  <aspect xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xsi:noNamespaceSchemaLocation="aspects.xsd">
4
5  <pointcut name="registerBook" operation="registerBook" class="Store"/>
6
7  <advice bind-to="registerBook"
8   url="http://store:6080/register/services/registerPort?wsdl"/>
9
10 </aspect>
```

Figura 3.2: Documento wssiadStore.xml

Este documento especifica el nombre del método y la clase a la que pertenece mediante el elemento <pointcut>. De igual manera, se especifica el URL correspondiente al documento WSDL del servicio Web mediante el elemento <advice>.

WSSI realiza un análisis del documento WSDL con el objetivo de obtener la descripción de los datos asociados a los mensajes SOAP de invocación y respuesta correspondientes a la operación `registerBook`.

Con la información obtenida, se genera el archivo `registerBookAspect.java` (Figura 3.3), el cual corresponde a un aspecto AspectJ que define el punto de corte correspondiente al método `registerBook` y los modificadores de comportamiento before y around (líneas 8, 13 y 32 respectivamente).

El modificador de comportamiento before especifica el URL del WSDL y el nombre de la operación del servicio Web y un objeto con la instancia de los datos necesarios para la invocación del servicio Web. El modificador de comportamiento around realiza la invocación del servicio Web y analiza la respuesta correspondiente a dicha invocación.

Con la generación de este aspecto, la aplicación Book Store puede ser considerada como una primera versión distribuida al realizar la compilación de la aplicación en conjunto con dicho aspecto.

3.2.3. Generación de aspectos: descubrimiento de servicios Web

Consideremos que para el resto de los métodos de la clase `Store` anteriormente mencionados no se conoce la ubicación de los servicios Web que provean la funcionalidad requerida.

La definición del documento WSSIAD (Figura 3.4) tiene que incluir el URL del Intermediario en lugar del URL de un documento WSDL y la descripción de los datos (nombre y tipo) asociados a los mensajes SOAP de invocación y respuesta asociadas a la operación del servicio Web.

En este caso, WSSI inicia el proceso de generación de aspectos identificando cada elemento <`pointcut`> con su respectivo elemento <`advice`>. Cada proceso realiza una consulta al Intermediario de acuerdo a la descripción de la operación especificada por el documento.

```
1  import java.util.Vector;
2  import org.w3c.dom.Document;
3
4   aspect registerBookAspect {
5
6    wssi.InvokeService service = null;
7
8     pointcut registerBook( String id,String title,String author,
9      float price,int available ) : execution( String
10     Store.registerBook(String,String,String,float,int) ) &&
11      args( id,title,author,price,available );
12
13    before( String id,String title,String author,float price,int
14     available ) : registerBook( id,title,author,price,available ){
15     try {
16       String wsdl = "http://store:6080/register/services/registerPort?wsdl";
17       String operation = "registerBook";
18
19       Vector dataPart = new Vector();
20       dataPart.add(String.valueOf(id));
21       dataPart.add(String.valueOf(title));
22       dataPart.add(String.valueOf(author));
23       dataPart.add(String.valueOf(price));
24       dataPart.add(String.valueOf(available));
25
26       service = new wssi.InvokeService();
27       service.setElementsRequest(wsdl,operation,dataPart);
28     }
29     catch(Exception e) { System.out.println("service not available"); }
30   }
31
32   String around( String id,String title,String author,float price,
33     int available ) : registerBook( id,title,author,price,available ){
34     String reply = null;
35     try {
36       Document document = service.invoke();
37       service.getSOAPreply(document.getDocumentElement());
38       String SOAPreply = service.getReply();
39       reply = SOAPreply;
40     }
41     catch(Exception e) {
42       System.out.println("local method is going to be executed ...");
43       return proceed(id,title,author,price,available);
44     }
45   }
46 }
```

Figura 3.3: Aspecto registerBookAspect.java

```
<?xml version="1.0"?>
<aspect xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="aspects.xsd">

<pointcut name="searchBook" operation="searchBook" class="Store"/>
<pointcut name="getQuote" operation="getQuote" class="Store"/>
<pointcut name="buyBook" operation="buyBook" class="Store"/>

<advice bind-to="searchBook"
 url="http://broker:6080/broker/services/brokerPort?wsdl">
  <message name="searchBookRequest">
    <part name="title" type="string"/>
  </message>
  <message name="searchBookResponse">
    <part name="data" type="string"/>
  </message>
</advice>

<advice bind-to="buyBook"
 url="http://broker:6080/broker/services/brokerPort?wsdl">
  <message name="buyBookRequest">
    <part name="id" type="string"/>
    <part name="quote" type="int"/>
    <part name="purchase" type="float"/>
    <part name="card" type="string"/>
    <part name="account" type="string"/>
  </message>
  <message name="buyBookResponse">
    <part name="confirmation" type="string"/>
  </message>
</advice>

<advice bind-to="getQuote"
 url="http://broker:6080/broker/services/brokerPort?wsdl">
  <message name="getQuoteRequest">
    <part name="id" type="string"/>
    <part name="quote" type="string"/>
  </message>
  <message name="getQuoteResponse">
    <part name="purchase" type="float"/>
  </message>
</advice>

</aspect>
```

Figura 3.4: Documento wssiadStore2.xml

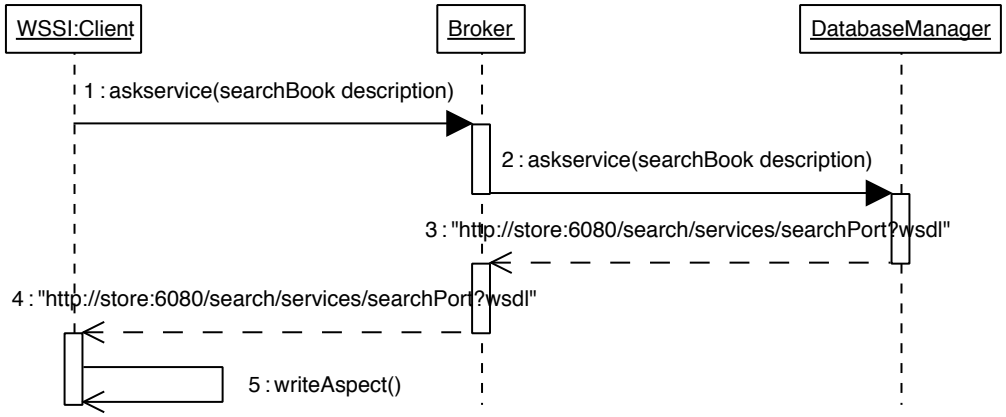

Figura 3.5: Consulta al Intermediario: servicio Web publicado.

Servicios Web publicados

Supongamos que el Intermediario tiene publicada la descripción correspondiente a la operación `searchBook`.

La Figura 3.5 muestra cómo se realiza la consulta de la descripción de la operación `searchBook` al Intermediario, el cual a su vez consulta el directorio de servicios para obtener el WSDL correspondiente al servicio Web `search`. Dicha respuesta es la que se envía al cliente para finalmente generar el `aspecto` asociado al archivo searchBookAspect.java[2], el cual se encargará de realizar la invocación del servicio Web correspondiente.

Servicios Web no publicados

Por otra parte, supongamos ahora que para las operaciones `buyBook` y `getQuote`, el Intermediario no tiene publicadas las descripciones correspondientes a las mismas. En este caso, se realiza el proceso de suscripción donde:

1. El Intermediario genera un UUID que identifica la suscripción para cada descripción.

2. WSSI recibe como respuesta el UUID generado para cada una de ellas y espera por la respuesta de ambas suscripciones, debido a que el tipo de invocación es

[2]Código correspondiente al archivo se encuentra en el Apéndice A, página 93.

síncrona.

3. El Intermediario bloquea el proceso de suscripción asociado a cada UUID hasta que sea publicada por parte de un proveedor, la descripción de las operaciones solicitadas.

Para el presente caso de estudio, la salida correspondiente al proceso de suscripción que genera las suscripciones y sus respectivos UUIDs es:

```
...
waitForService
answer of subscribe: 79f5387-105a3ec40b3-ae3eecdb179509917cc9343226a
f5bb7
waiting for susbcription: 79f5387-105a3ec40b3-ae3eecdb179509917cc934
3226af5bb7
...
waitForService
answer of subscribe: 79f5387-105a3ec58e0-ae3eecdb179509917cc9343226a
f5bb7
waiting for susbcription: 79f5387-105a3ec58e0-ae3eecdb179509917cc934
3226af5bb7
```

3.2.4. Serialización de suscripciones

Supongamos que es necesario reiniciar o detener la máquina donde se estaba ejecutando WSSI. De esta manera, las suscripciones generadas para el caso de estudio tienen que ser serializadas cuando todos los procesos hayan terminado su ejecución o se encuentren en espera de una respuesta con el fin de deserializarlas y reanudarlas después.

El proceso de serialización realizado por WSSI para las dos suscripciones que esperan respuesta genera la salida:

```
...
beginning serialization process
serializing threads
...
serializing thread Thread[Thread-2,5,main]
serialization of 79f5387-105a3ec58e0-ae3eecdb179509917cc9343226af5bb7
confirmation of serialization: ok
...
serializing thread Thread[Thread-3,5,main]
serialization of 79f5387-105a3ec40b3-ae3eecdb179509917cc9343226af5bb7
```

```
...
stopping process buyBook
stopping process getQuote
```

3.2.5. Deserialización de suscripciones

Una vez reiniciada la máquina, puede ejecutarse nuevamente WSSI. Para este caso, WSSI verifica si existe información serializada y de ser así, deserializa la información de cada uno de ellos para generar un nuevo proceso que reanude la espera por cada una de las suscripciones. La salida que genera WSSI para las suscripciones correspondientes al caso de estudio es entonces:

```
...
2 subscriptions deserialized
...
creating thread for the subscription 79f5387-105a3ec40b3-ae3eecdb17
9509917cc9343226af5bb7 that was serialized

creating thread for the subscription 79f5387-105a3ec58e0-ae3eecdb17
9509917cc9343226af5bb7 that was serialized
...
waitForService
waiting for susbcription: 79f5387-105a3ec58e0-ae3eecdb179509917cc93
43226af5bb7
...
waitForService
waiting for susbcription: 79f5387-105a3ec40b3-ae3eecdb179509917cc93
43226af5bb7
...
```

Como puede observarse los UUIDs corresponden con aquellas mostradas en la sección 3.2.3.

3.2.6. Notificación de respuesta a una suscripción

Como hemos descrito, las dos suscripciones correspondientes a las operaciones `getQuote` y `buyBook` se encuentran en espera por una respuesta, la cual se recibirá cuando un proveedor publique el servicio que provea la operación solicitada.

La Figura 3.6 muestra el proceso realizado desde la consulta al Intermediario (1, 2, 3, 4), la generación de la suscripción correspondiente (5, 6) y espera por la misma (7). Para el presente caso de estudio, supongamos que el proveedor X desea publicar el

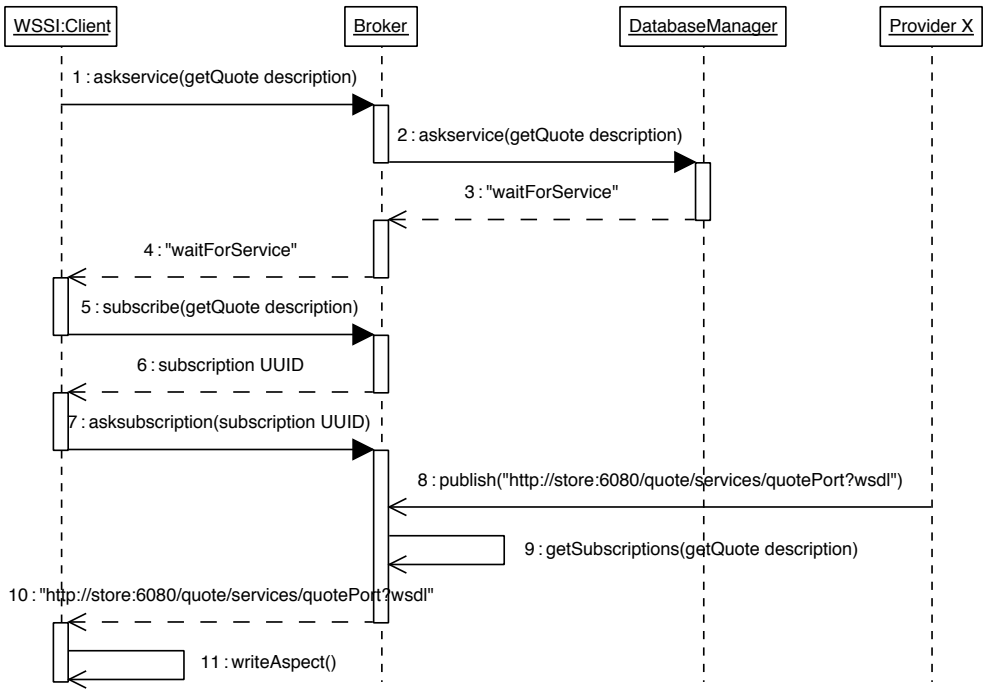

Figura 3.6: Proceso de suscripción y publicación de un servicio Web.

servicio correspondiente a `getQuote`. Por lo tanto, mediante el *script* que WSSI proporciona para la publicación de servicios el proveedor publica el URL del documento WSDL correspondiente al mismo (8).

El Intermediario realiza el análisis del documento y obtiene la descripción de la operación `getQuote` para agregarla a su repositorio. Posteriormente, identifica que la suscripción `79f5387-105a3ec58e0-ae3eecdb179509917cc9343226af5bb7` puede ser contestada (9). En ese momento se notifica la respuesta a dicha suscripción. Por su parte, WSSI recibe la respuesta a esta última (10) y genera el archivo asociado al aspecto `getQuoteAspect.java` [3] (11), así como los archivos de compilación para la aplicación (appfiles.lst y compile.bat).

La salida generada por WSSI es:

```
...
```

[3]Código correspondiente al archivo se encuentra en el Apéndice A, página 94.

```
answer of subscription: http://store:6080/quote/services/quotePort?
wsdl

writting getQuoteAspect.java
writting appfiles.lst
writting compile.bat
...
```

3.2.7. Dependencia de servicios Web no implementados

Retomando el caso de estudio, la suscripción asociada a la operación buyBook sigue esperando por una respuesta. Supongamos entonces que el proveedor Y necesita un servicio Web cuya descripción corresponda a `boolean fundsFee(String,String,float)`. Por lo tanto, el proceso que se realiza para el descubrimiento dinámico del servicio Web correspondiente es el mismo que describimos en la subsección 3.1.3.

Para el presente caso de estudio, la cadena de suministro es de dos niveles tal y como muestra la Figura 3.7.

El primer nivel de la cadena corresponde al proceso de consulta al Intermediario (1, 2, 3, 4) y a la generación de la suscripción asociada con la descripción de la operación buyBook (5, 6), así como la espera por una respuesta (7). El segundo nivel corresponde al mismo proceso para el primer nivel, pero para la operación `fundsFee`. En la Figura 3.7 del paso 1 al paso 7 corresponden al paso 8 hasta el paso 14 respectivamente.

Consideremos que durante la espera por una respuesta el proveedor Y publica el servicio Web requerido por el provedor X (15). El Intermediario contesta la suscripción correspondiente (16, 17) por lo que se genera el archivo fundsFeeAspect.java[4] y los archivos para la compilación de la aplicación (18). Así, la compilación de la aplicación integra dicho aspecto al servicio Web y de esta manera su implementación puede considerarse finalizada. Como consecuencia, el proveedor Y publica el servicio Web (19) y por lo tanto la suscripción `79f5387-105a3ec40b3-ae3eecdb179509917cc9343226af5bb7` asociada a la operación buyBook es contestada (20, 21).

WSSI genera entonces el aspecto `buyBookAspect.java`[5] al recibir la respuesta por

[4]Código correspondiente al archivo se encuentra en el Apéndice A, página 96.
[5]Código correspondiente al archivo se encuentra en el Apéndice A, página 95.

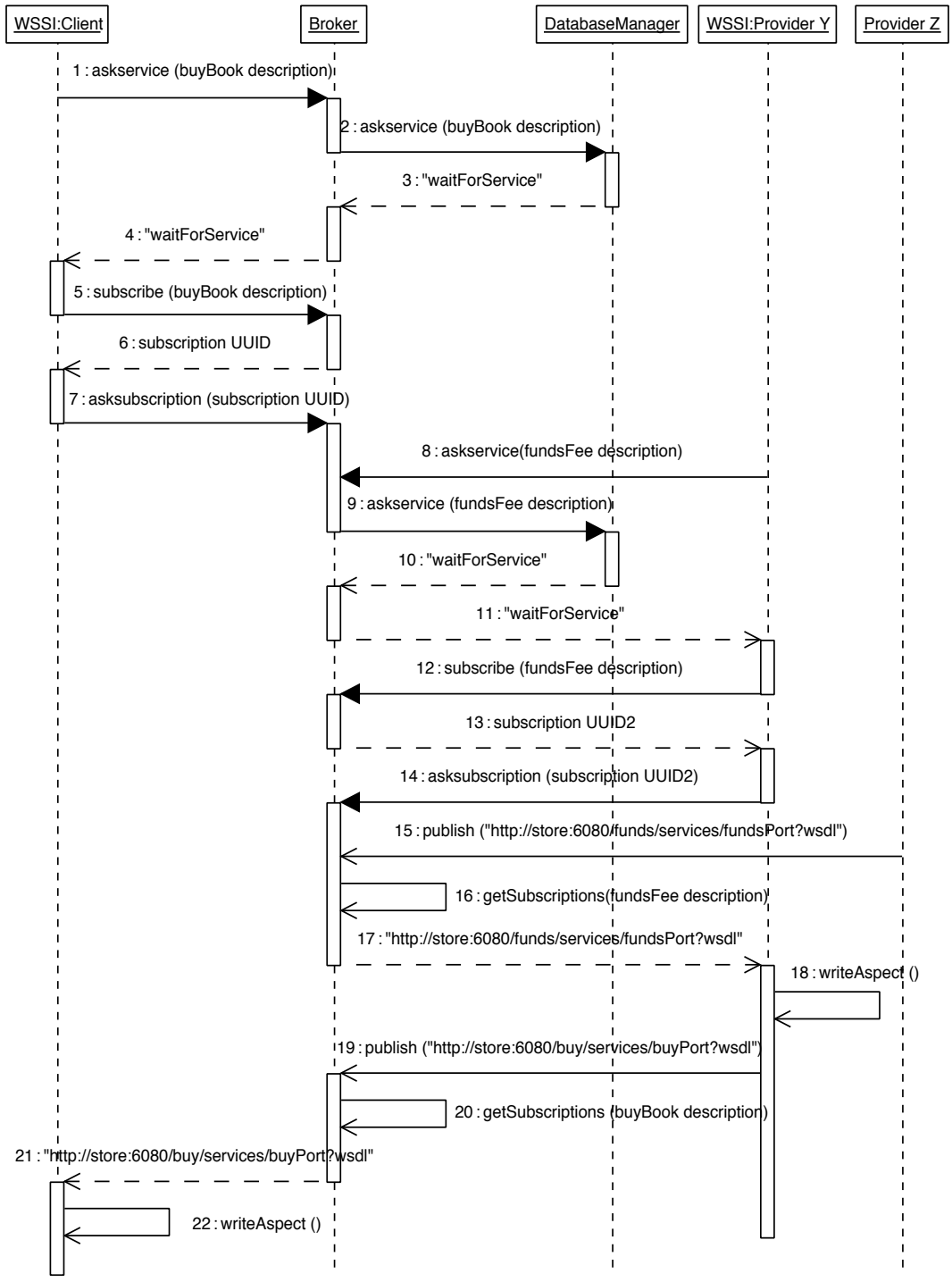

Figura 3.7: Dependencia de servicios Web no implementados.

parte del Intermediario (22). De esta manera, el proceso de generación de aspectos finaliza correctamente.

3.3. Sumario

El caso de estudio que se presentó a lo largo de este capítulo corresponde a una aplicación centralizada que durante el proceso de descubrimiento dinámico de servicios Web llevado a cabo por WSSI y el Intermediario presenta los siguientes casos:

1. Consulta a documentos WSDL asociado a un servicio Web del que se tiene conocimiento. Proceso simple que realiza WSSI para obtener información referente a una operación en específico.

2. Descubrimiento dinámico de servicios Web a partir de la descripción correspondiente a la operación asociada a un servicio Web. Proceso realizado por WSSI mediante una serie de consultas realizadas al Intermediario, cuya respuesta puede ser:

    a. URL del WSDL asociado a un servicio Web que provea la operación solicitada.

    b. Notificación del UUID correspondiente a la suscripción generada.

3. Dependencia de servicios Web no implementados. Cuando un servicio Web que requiere ser implementado necesita a su vez de otro(s) servicio(s) Web.

Una vez que se generaron todos los aspectos, se realiza la integración de los mismos mediante la compilación de la aplicación. De esta manera, se obtiene la versión distribuida de la aplicación Book Store debido a que realiza invocaciones a servicios Web en vez de ejecutar métodos locales.

Describimos como en un escenario típico, para una cadena de suministro todos los participantes son previamente establecidos. Por lo tanto, una de las ventajas de aplicar este enfoque a este problema es que dichos participantes pueden ser cambiados de acuerdo a las necesidades de la misma.

Con este capítulo describimos el proceso de integración de servicios Web, el cual se realizó de una manera más simple y transparente al usuario de tal manera que no se modificará el código de la aplicación original.

Capítulo 4

Diseño conceptual

En este capítulo presentamos la perspectiva organizacional y funcional de los módulos que intervienen en el proceso de selección e integración dinámica de servicios Web, los cuales corresponden a WSSI e Intermediario. Así mismo explicamos cuáles son y cómo ocurren las interacciones entre cada participante durante todo el proceso de tal manera que a partir de una aplicación centralizada se integre un conjunto de aspectos a la aplicación para obtener una versión distribuida de la misma. El diseño que presentamos en este capítulo incluye la estructura de documentos WSSIAD para definir los puntos de corte asociados a cada aspecto, el diseño del Intermediario y diseño de WSSI, así como la estructura de los aspectos generados de acuerdo a la sintaxis del lenguaje AspectJ.

4.1. Perspectiva organizacional

El proceso de selección e integración dinámica de servicios Web es el objetivo principal de WSSI (Web Services Selection and Integration) y del Intermediario. Desde una perspectiva organizacional, los participantes involucrados en este proceso son básicamente los clientes, WSSI, el Intermediario y los proveedores de servicios Web.

En la Figura 4.1 se puede apreciar de una manera muy general cada uno de los participantes y las relaciones que existen entre ellos. Los clientes son entidades que inician el proceso mediante la definición de un documento WSSIAD. WSSI se encarga de analizar dicho documento y realizar la generación de cada archivo asociado a un aspecto, de acuerdo a una serie de consultas realizadas al Intermediario o a un documento WSDL. Los proveedores por su parte, son los que publican las descripciones

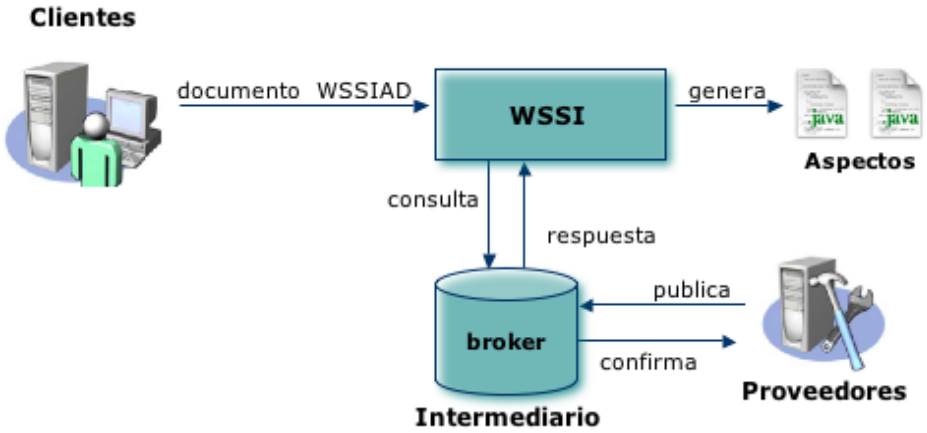

Figura 4.1: Participantes en el proceso de selección e integración de servicios Web.

de los servicios que proveen.

4.2. Definición de documentos WSSIAD

WSSIAD (WSSI Aspect Definition) es la descripción XML que permite realizar la definición de los puntos de corte asociados a la ejecución de un programa, así como especificar información necesaria que determine el proceso que se seguirá para la generación de código AspectJ asociado a cada punto de corte, y el cual corresponderá a un aspecto.

Los elementos que define el esquema asociado a WSSIAD (Figura 4.2) son esencialmente <pointcut> y <advice>. Cada uno de estos elementos define a su vez una serie de atributos y/o subelementos que pueden o deben ser especificados.

4.2.1. Definición de un aspecto

El enfoque de la programación orientada a aspectos se basa en la definición de módulos denominados aspectos, la cual dependerá del lenguaje que sea utilizado. El objetivo de estos es principalmente modificar y/o extender el comportamiento de puntos específicos durante la ejecución de un programa evitando así la invasividad de código.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="aspect" type="aspectType"/>

<xs:complexType name="aspectType">
  <xs:sequence>
    <xs:element name="pointcut">
      <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="operation" type="xs:string" use="required"/>
        <xs:attribute name="class" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="advice" type="adviceType"/>
  </xs:sequence>
  <xs:attribute name="package" type="xs:string"/>
</xs:complexType>

<xs:complexType name="adviceType">
  <xs:sequence>
    <xs:element name="message" type="messageType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="bind-to" type="xs:string" use="required"/>
  <xs:attribute name="url" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="messageType">
  <xs:sequence>
    <xs:element name="part">
      <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="type" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

</xs:schema>
```

Figura 4.2: Esquema aspects.xsd correspondiente a documentos WSSIAD.

Básicamente, un documento WSSIAD tiene el objetivo de especificar aquellos métodos que serán reemplazados por invocaciones a servicios Web y cuya localización puede o no ser especificada. La definición de dichos documentos proveerá entonces cierta información que se utilizará durante la generación del código correspondiente a un aspecto.

De acuerdo al esquema aspects.xsd, la estructura general de un documento WSSIAD se presenta en la Figura 4.3. Para definir un aspecto se define el elemento <aspect> que debe especificar el espacio de nombres `http://www.w3.org/2001/XMLSchema-instance`, ya que tiene el propósito de identificar al documento como una nueva instancia del esquema definido por aspects.xsd. De igual manera, el atributo `package` puede ser definido para especificar el paquete al que pertenece la aplicación.

```
1  < !- Definición de un aspecto -->
2  <aspect xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:noNamespaceSchemaLocation="aspects.xsd" package="...">
4
5     < !- Definición de un elemento pointcut -->
6     <pointcut name="..." operation="..." class="..."/>
7
8     < !- Definición de un elemento advice como etiqueta vacía -- >
9     <advice bind-to="..." url="..."/>
10
11    < !- Definición de un elemento advice con subelementos message -->
12    <advice bind-to="..." url="...">
13       <message name="Request">
14          <part name="..." type="..."/>
15       </message>
16       <message name="Response">
17          <part name="..." type="..."/>
18       </message>
19    </advice>
20
21 </aspect>
```

Figura 4.3: Estructura general de un documento WSSIAD.

Puntos de corte

El elemento <pointcut> define información para realizar la definición de un punto de corte a través de los atributos:

- `name` [obligatorio]: Nombre del punto de corte.
- `operation` [obligatorio]: Nombre de la operación que será reemplazada por la invocación de un servicio Web.
- `class` [obligatorio]: Clase a la cual pertenece el método especificado por `operation`.

Con la información de cada uno de los atributos y la información que se obtiene durante la etapa de generación de aspectos, el código que se generará para definir cada punto de corte corresponde a la sintaxis de AspectJ que se muestra a continuación:

```
pointcut <nombre> (<parámetros>) : execution(<definición-método>) &&
                                   args(<nombre-parámetros>);
```

La primitiva `pointcut` se utiliza para definir el punto de corte, el cual debe ser seguido por el nombre con el cual será identificado. La primitiva `execution` identifica los puntos de unión asociados a la ejecución de un método y cuya definición corresponde al prototipo `retorno clase.método(tipos-parámetros)`. La primitiva `args` permite conocer el valor de los parámetros asociados a cada punto de unión, los cuales se enlazan a los modificadores de comportamiento definidos en el aspecto.

Descripciones de servicios Web

El elemento <advice> asocia a cada uno de los elementos <pointcut> con la descripción de un servicio Web. Los atributos que debe especificar son:

- `bind-to` [obligatorio]: Nombre del punto de corte al cual está asociado dicho elemento.
- `url` [obligatorio]: URL del documento WSDL correspondiente a un servicio Web o URL del Intermediario.

Un elemento <advice> se define como etiqueta vacía siempre y cuando el atributo url especifique el URL de un documento WSDL. En caso de que este atributo especifique el URL del Intermediario, es necesario definir subelementos <message>, los cuales deben corresponder a la descripción asociada a los mensajes SOAP de invocación y respuesta del servicio Web. Por tal razón, podemos observar que los elementos <message> tienen una estructura similar a la que se define en un documento WSDL ya que de la misma manera, especifica subelementos <part>.

Los elementos <part> especifican los atributos:

- name [obligatorio]: Nombre asociado a un tipo de dato que forma parte de un mensaje SOAP.
- type [obligatorio]: Tipo de dato asociado a un dato que forma parte de un mensaje SOAP.

La definición de un documento WSSIAD es el punto de partida para que WSSI interprete la información especificada por el mismo y como resultado obtenga un conjunto de aspectos que serán integrados a una aplicación.

4.3. Diseño del intermediario

El intermediario (Broker) es un servicio de intermediación que tiene como objetivo proveer la funcionalidad de un nodo UDDI básico. El conjunto de operaciones que provee están asociadas a la consulta, publicación y suscripción de servicios Web.

4.3.1. Consulta de servicios Web

La operación de consulta askservice que provee el Intermediario corresponde a la consulta de la descripción asociada a una operación de un servicio Web.

La Figura 4.4 muestra cómo el Intermediario recibe la descripción del servicio conformada por el nombre de la operación y los datos asociados a los mensajes SOAP de invocación y respuesta. El Intermediario consulta dicha descripción en su repositorio y obtiene una respuesta, la cual puede ser el URL del WSDL asociado a un servicio Web o simplemente un mensaje que indique que no se ha publicado ningún servicio

que corresponda a la descripción especificada, pero que puede realizar la suscripción de la misma.

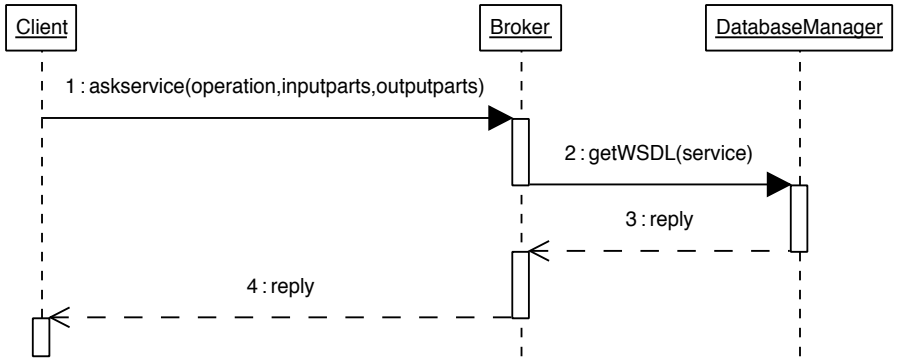

Figura 4.4: Consulta de un servicio Web.

4.3.2. Publicación de servicios Web

La Figura 4.5 muestra el proceso de publicación de un servicio Web correspondiente a la operación `publish`. Este proceso inicia cuando un proveedor desea publicar la descripción de un servicio Web, la cual corresponde a su respectivo WSDL (1). Primeramente, el intermediario analiza el documento asociado al WSDL (2) para obtener las operaciones especificadas por el mismo (3, 4), y por cada una de ellas obtiene los datos asociados a los mensajes SOAP de invocación (5, 6) y respuesta (7, 8) correspondientes para formar la descripción de cada operación. De esta manera, el Intermediario almacena en su repositorio el registro de cada una de las descripciones (9). Cuando se almacenan dichos registros se verifica si algunas de las suscripciones que siguen pendientes pueden ser contestadas (10, 11), lo cual causaría continuar con su proceso (12). Finalmente, el Intermediario confirma al proveedor que el servicio Web ha sido publicado (13).

Cuando un proveedor desea eliminar el registro de un servicio Web (Figura 4.6), a través de la operación `unpublish` simplemente especifica al Intermediario el WSDL correspondiente al mismo. El Intermediario elimina los registros que coincidan con dicho WSDL y notifica al proveedor que el servicio ha sido eliminado satisfactoriamente.

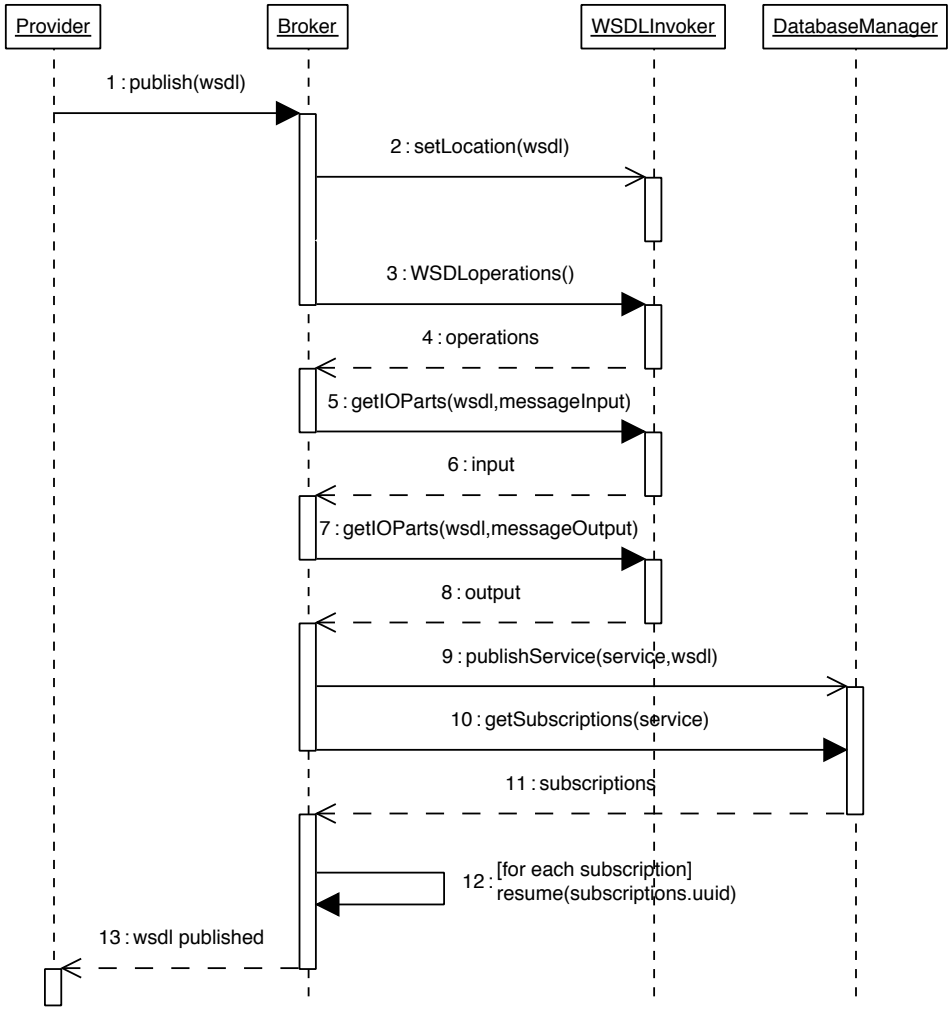

Figura 4.5: Publicación de un servicio Web.

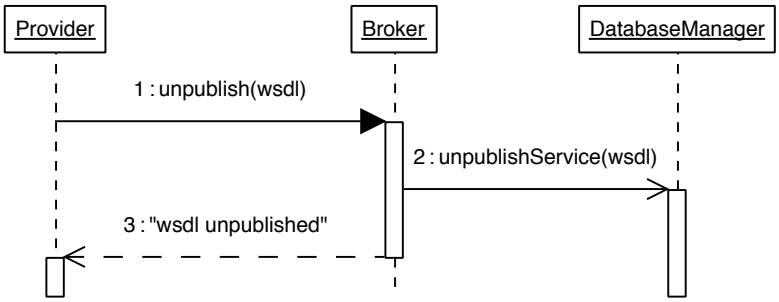

Figura 4.6: Eliminación del registro de un servicio Web.

4.3.3. Suscripción de servicios Web

Consultar la descripción correspondiente a la operación de un servicio Web que el Intermediario no tenga registrado implica realizar la suscripción de la misma a través de la operación `subscribe` (Figura 4.7). La descripción que será suscrita debe estar conformada por el nombre de la operación y los datos asociados (nombre y tipo) a los mensajes SOAP de invocación y respuesta como una cadena de la forma `operación(nombre`$_1$`:tipo`$_1$`, ..., nombre`$_n$`:tipo`$_n$`):retorno`. El Intermediario generará un identificador único universal (UUID) para la suscripción, la cual será almacenada en el repositorio del mismo. Dicho identificador será la respuesta que recibirá el cliente y mediante el cual estará en posibilidad de esperar por una respuesta a la suscripción realizada.

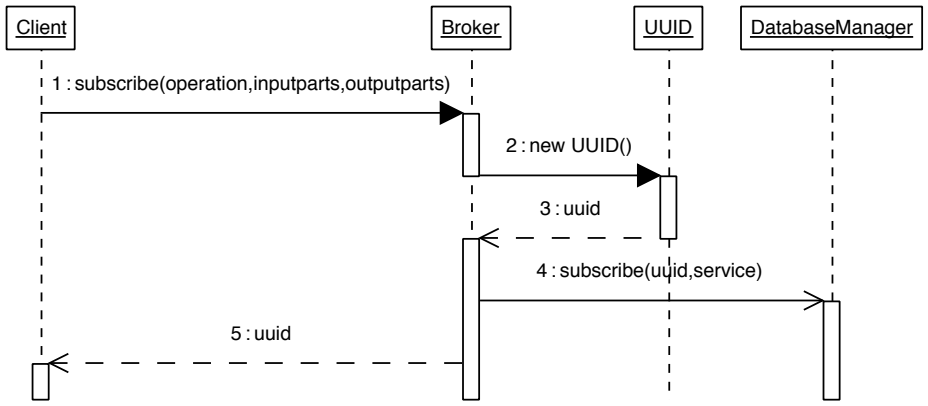

Figura 4.7: Suscripción de la descripción asociada a una operación de un servicio.

La operación `unsubscribe` tiene el objetivo de cancelar una suscripción (Figura 4.8). El Intermediario tiene que recibir el identificador asociado a dicha suscripción para poder desbloquear y finalizar su proceso, y finalmente eliminar el registro del repositorio. La respuesta correspondiente a este proceso es la notificación de cancelación por parte del Intermediario.

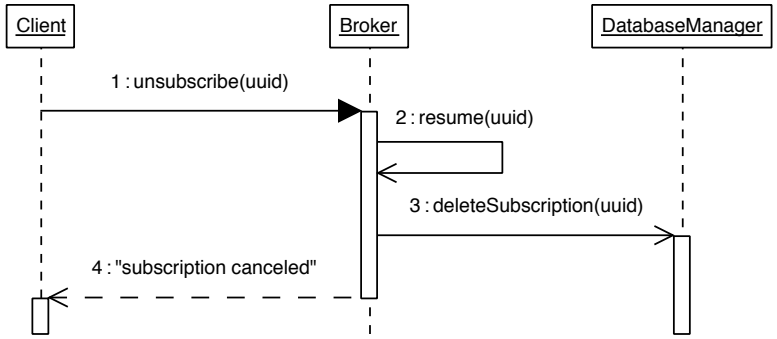

Figura 4.8: Cancelación de una suscripción.

Existe de igual manera la posibilidad de serializar la información de una suscripción como se muestra en la Figura 4.9 que corresponde a la operación `notifyserialization`. Este proceso de serialización no es realizado por el Intemediario, por lo que este último sólo tiene que ser notificado del proceso. Por lo tanto, el registro correspondiente a la suscripción es actualizado de tal manera que se identifique como un proceso serializado. Como consecuencia, el proceso de suscripción es desbloqueado para terminar su ejecución y finalmente confirmar al cliente que el proceso de notificación finalizó.

La Figura 4.10 muestra el diagrama de secuencia asociado al proceso de espera por la respuesta a una suscripción (operación `asksubscription`). Una vez que se ha realizado una suscripción, el proceso continúa al notificar que se esperará por su respuesta (1). En este caso, el Intermediario tiene que recibir el identificador de la misma para obtener la descripción de la operación solicitada (2, 3) y verificar si se trata de una suscripción previamente serializada (4, 5). Posteriormente, el proceso se suspende (6) hasta que un proveedor publique la descripción que corresponda con la suscripción, sea cancelada o se notifique su serialización. En ese momento, se desbloquea el proceso y se realiza una consulta al repositorio del Intermediario (7,

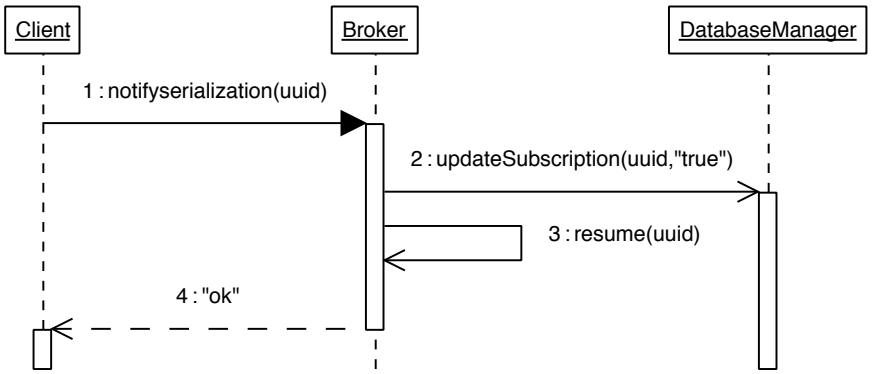

Figura 4.9: Notificación de serialización de una suscripción.

8). La respuesta que se obtenga de dicha consulta corresponderá a alguno de los tres casos mencionados anteriormente y será la que se regrese como respuesta (9).

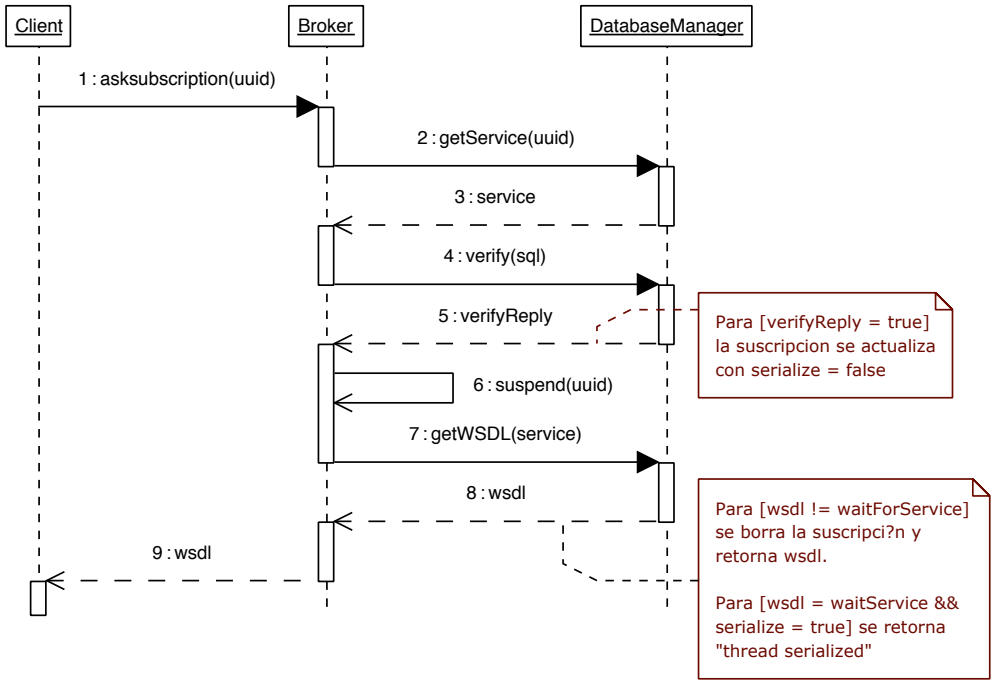

Figura 4.10: Proceso de espera por la respuesta de una suscripción.

Algunos de los procesos generados por WSSI tienen el objetivo de realizar una serie de consultas al Intermediario, de tal manera que este seleccione el servicio Web que cumpla con la descripción especificada por cada proceso y en caso de no tener ningún registro que cumpla con la descripción, realizar la suscripción de la misma.

4.4. Diseño de WSSI

Cuando se diseña una aplicación distribuida, todos las características relacionadas a la misma se establecen desde la etapa de diseño. En caso de que se considere integrar servicios Web a la aplicación, es necesario conocer la localización de cada uno de los servicios Web y establecer los protocolos y herramientas que se utilizarán para realizar su invocación. Como consecuencia, el desarrollo y mantenimiento de dichas aplicaciones se convierte en una tarea más compleja.

WSSI se encarga de seleccionar e integrar un conjunto de servicios Web a través del uso de la programación orientada a aspectos de tal manera que no se consideren características de distribución desde las etapas de diseño. Por lo tanto, el proceso de desarrollo y mantenimiento de aplicaciones se vuelve una tarea más simple para los desarrolladores.

4.4.1. Generación de aspectos

Una vez que se ha definido un documento WSSIAD, WSSI es el encargado de iniciar el proceso de selección e integración dinámica de un conjunto de servicios Web a una aplicación. En la Figura 5.1 se muestra el proceso que se lleva a cabo para la generación de los aspectos y sus respectivos archivos de compilación.

Inicialmente, el documento WSSIAD es analizado por WSSI para obtener el valor de los atributos de cada elemento <advice>, así como el valor de los atributos del elemento <pointcut> asociados a este.

Cada proceso generado es inicializado con la información obtenida durante el análisis y agregado a un grupo de procesos para su ejecución posterior. De acuerdo a la información especificada por cada elemento <advice> se pueden generar dos tipos de procesos:

1. Proceso encargado de analizar el documento WSDL especificado por el atributo url.

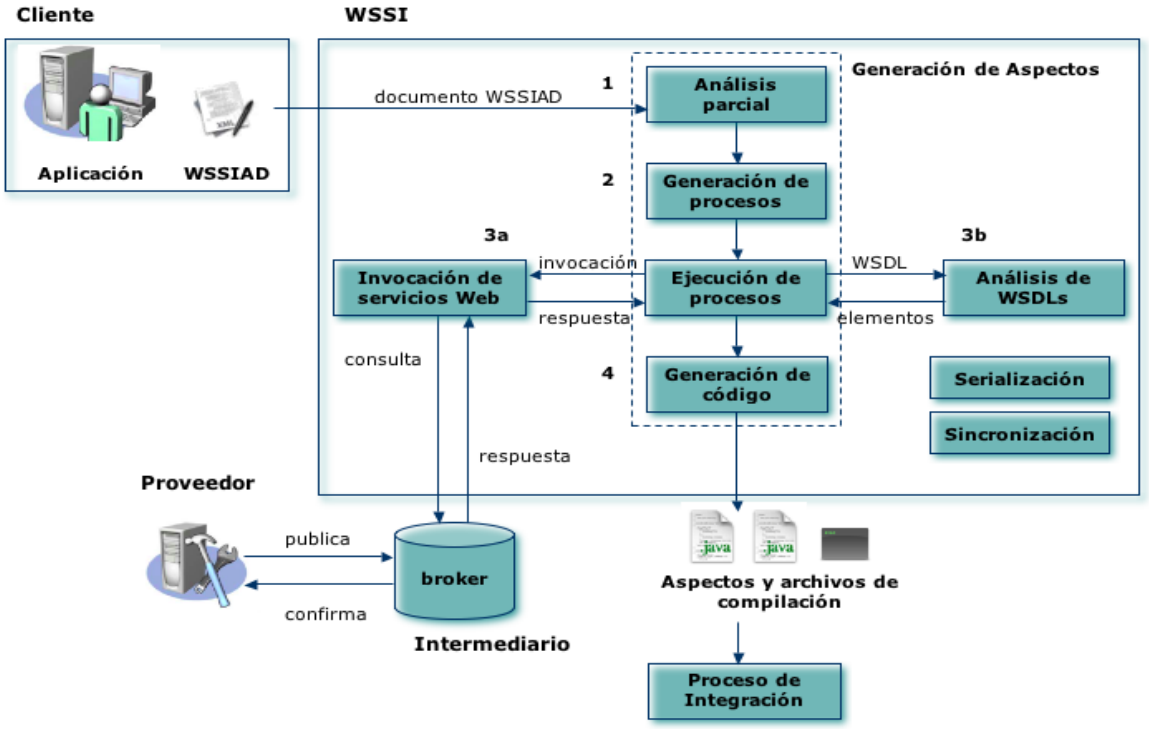

Figura 4.11: Etapa de generación de aspectos.

2. Proceso encargado de descubrir un servicio Web que provea la operación especificada por el valor del atributo `operation` y que tenga asociados los datos especificados por cada subelemento <`part`> para los mensajes SOAP de invocación y de respuesta.

El proceso encargado de realizar el análisis de un documento WSDL es muy simple ya que su objetivo es obtener los datos correspondientes a los mensajes de invocación y respuesta asociados a una operación ofrecida por el servicio Web.

Descubrimiento dinámico de servicios Web

El proceso encargado de descubrir un servicio Web inicia cuando se consulta al Intermediario la descripción de una operación. Si el Intermediario encuentra un servicio que provea dicha operación regresa como resultado el URL del documento WSDL asociado al servicio. En caso contrario, se lleva a cabo el proceso de suscripción descrito en la Figura 4.12.

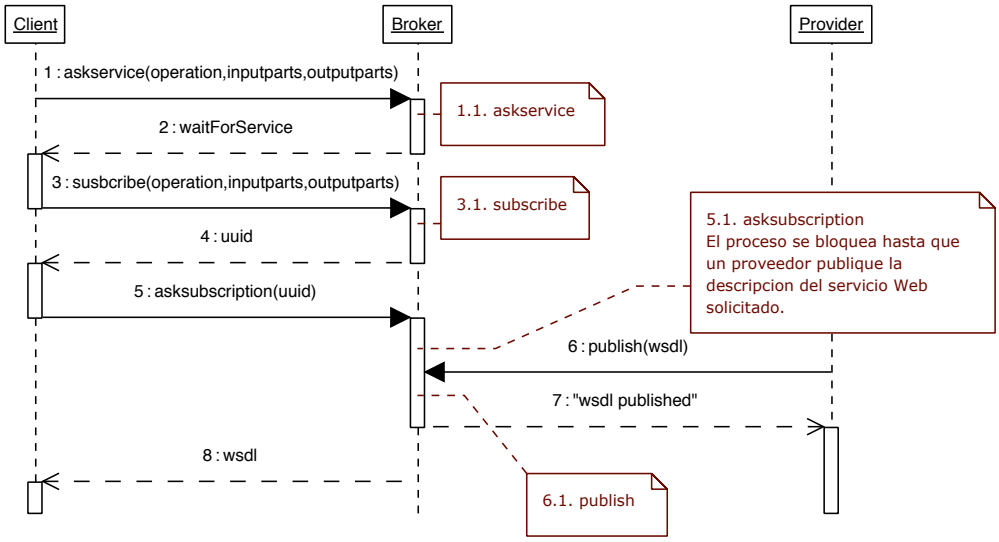

Figura 4.12: Proceso de suscripción.

El proceso de suscripción genera el registro correspondiente a la descripción de la operación solicitada, la cual será identificada a través de un UUID generado por el Intermediario. Cuando se tiene conocimiento del identificador de la suscripción, WSSI notifica al Intermediario que esperará por la respuesta. Así, cuando un proveedor publica un servicio Web tal que alguna de las descripciones asociadas a sus operaciones coincida con la descripción de la operación solicitada, el Intermediario regresa inmediatamente una respuesta a WSSI. Esta respuesta corresponde al WSDL asociado al servicio Web.

Finalmente, con la información obtenida durante el análisis del documento WSSIAD y durante cada proceso se genera el código del aspecto correspondiente al mismo según la sintaxis del lenguaje AspectJ. Cada aspecto realizará la invocación a un servicio Web y estará estructurado como se muestra en la Figura 4.13.

El código asociado al modificador de comportamiento before se encarga de inicializar todos los elementos necesarios para realizar la invocación a un servicio Web a través del protocolo SOAP. Por su parte, el código asociado al modificador de comportamiento around se encarga de realizar la invocación al servicio Web y de analizar su respuesta. En el caso de no tener una respuesta del servicio Web, entonces se especifica que el método local será el que se ejecutará.

```
1  package <paquete>;
2
3  aspect <nombre-aspecto>Aspect
4  {
5     pointcut <nombre> (<parámetros>):
6        execution(<definición-método>) && args(<nombre-parámetros>);
7
8     before(<parámetros>) : <nombre>(<nombre-parámetros>)
9     {
10      /**
11       *   Inicialización de los elementos necesarios para
12       *   la creación del mensaje SOAP de invocación.
13       */
14    }
15
16    <retorno> around(<parámetros>) : <nombre>(<nombre-parámetros>)
17    {
18      /**
19       *   Creación del mensaje SOAP, invocación y obtención
20       *   de la respuesta correspondiente al servicio Web.
21       */
22    }
23 }
```

Figura 4.13: Estructura general de un aspecto generado.

Proceso de serialización

Durante la ejecución del conjunto de procesos generados WSSI puede detener su ejecución y por lo tanto serializar la información de aquellos procesos que correspondan con una suscripción. En este caso, el estado de cada proceso se verifica para determinar si el proceso de serialización puede llevarse a cabo.

Debido a que cada proceso corresponde a un *thread*, puede estar en uno de cuatro estados:

1. New. El proceso ha sido creado pero no ha iniciado su ejecución.
2. Runnable. El proceso se encuentra en ejecución.
3. Dead. La ejecución del proceso finalizó.

4. Blocked. La ejecución del proceso se detiene debido a que realizó una suscripción al Intermediario. Su ejecución se reanuda cuando el Intermediario notifica una respuesta para dicha suscripción.

El estado en que los procesos deben encontrarse dependerá del tipo al que correspondan. Los procesos que realizan consultas a documentos WSDL deben finalizar su ejecución, por lo que su estado deberá ser dead. Los procesos que realicen el descubrimiento de un servicio Web a través del Intermediario deberán haber finalizado su ejecución (dead) o encontrarse en el estado blocked.

Cuando los procesos se encuentran en dichos estados, la serialización de la información correspondiente a los procesos bloqueados es agregada a un objeto que se almacenará como una secuencia de bytes en el archivo threads.ser. Esta información podrá ser recuperada en una etapa posterior de tal manera que al iniciar nuevamente WSSI, se verifique el archivo y se generen nuevos procesos para continuar con su ejecución.

4.4.2. Integración de aspectos

La etapa de generación de aspectos finaliza al generar un conjunto de archivos correspondientes a los mismos, así como los archivos necesarios para su compilación e integración.

El objetivo fundamental de cada aspecto es realizar la invocación de un servicio Web, ya que a través de estos se evita modificar el código de la aplicación original y simplemente se realiza su integración en tiempo de compilación (Figura 4.14), debido a que AspectJ realiza el entrelazado de código durante la compilación de los archivos correspondientes a la aplicación y a los aspectos generados.

El archivo de compilación generado por WSSI para cada aplicación cliente establece los valores de las variables de entorno con el propósito de especificar los archivos .jar necesarios para que el compilador ajc que proporciona AspectJ compile los archivos especificados por appfiles.lst (archivos de la aplicación cliente y aspectos generados por WSSI).

Una vez que se realizó la compilación, la aplicación estará lista para ejecutarse y realizar las invocaciones a los servicios Web integrados.

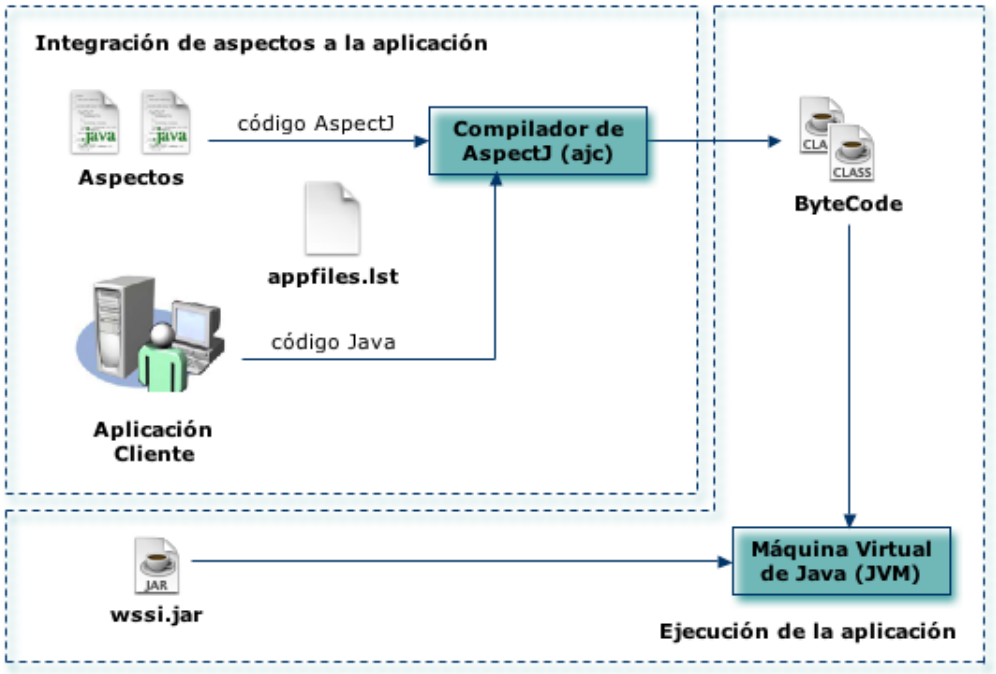

Figura 4.14: Integración de aspectos a una aplicación cliente.

4.5. Sumario

En este capítulo se describió la manera de realizar la selección e integración dinámica de servicios Web a una aplicación, donde para lograr los objetivos presentados por el diseño se presentó como la programación orientada a aspectos fue utilizada.

De esta manera, discutimos la estructura y objetivo de los documentos WSSIAD para la definición de puntos de corte e información necesaria para realizar el descubrimiento de los servicios Web y generar los aspectos correspondientes.

Una vez descritos los casos que se presentan para la generación de aspectos, presentamos la estructura generada de acuerdo a la sintaxis del lenguaje AspectJ y finalmente, presentamos cómo se realiza el entrelazado (*weaving*) con la aplicación original, lo que evita rediseñar y modificar la misma.

La programación orientada a aspectos ha influenciado el diseño de WSSI para la generación de los aspectos. En el siguiente capítulo se presenta la implementación de WSSI y del Intermediario, la cual sigue los principios presentados en este capítulo.

Capítulo 5

Implementación

El desarrollo de servicios Web se encuentra en constante evolución y por lo tanto el desarrollo de tecnologías asociadas a los mismos en un crecimiento constante. Actualmente existen muchas herramientas que facilitan el desarrollo de servicios Web como WSDK (WebSphere SDK for Web Services) de IBM y Java WSDP (Java Web Services Developer Pack), pero son muy pocas las que se enfocan a la selección e integración de los mismos. En este capítulo presentamos el conjunto de clases correspondientes a WSSI (Web Services Selection and Integration) y al Intermediario, las cuales corresponden a una implementación asociada al diseño presentado en el capítulo anterior.

5.1. Descripción general

En la Figura 5.1 se puede apreciar el flujo general correspondiente a la etapa de generación de código de un conjunto de aspectos especificados por un documento WSSIAD. El bloque de análisis parcial realiza la lectura de este documento para determinar los elementos <advice> que correspondan con cada elemento <pointcut>. El número de procesos que se generan corresponde al bloque generación de procesos, el cual inicializa cada uno de ellos y los agrega a un grupo de procesos para su ejecución posterior. El bloque ejecución de procesos se encarga de ejecutar paralelamente el conjunto de procesos generados, los cuales determinan si se debe realizar la consulta a un documento WSDL o al Intermediario para descubrir dinámicamente un servicio Web. Ambos tipos de procesos obtienen la información necesaria para que el bloque generación de código genere los archivos correspondientes a cada aspecto, así como los archivos de compilación necesarios para su integración.

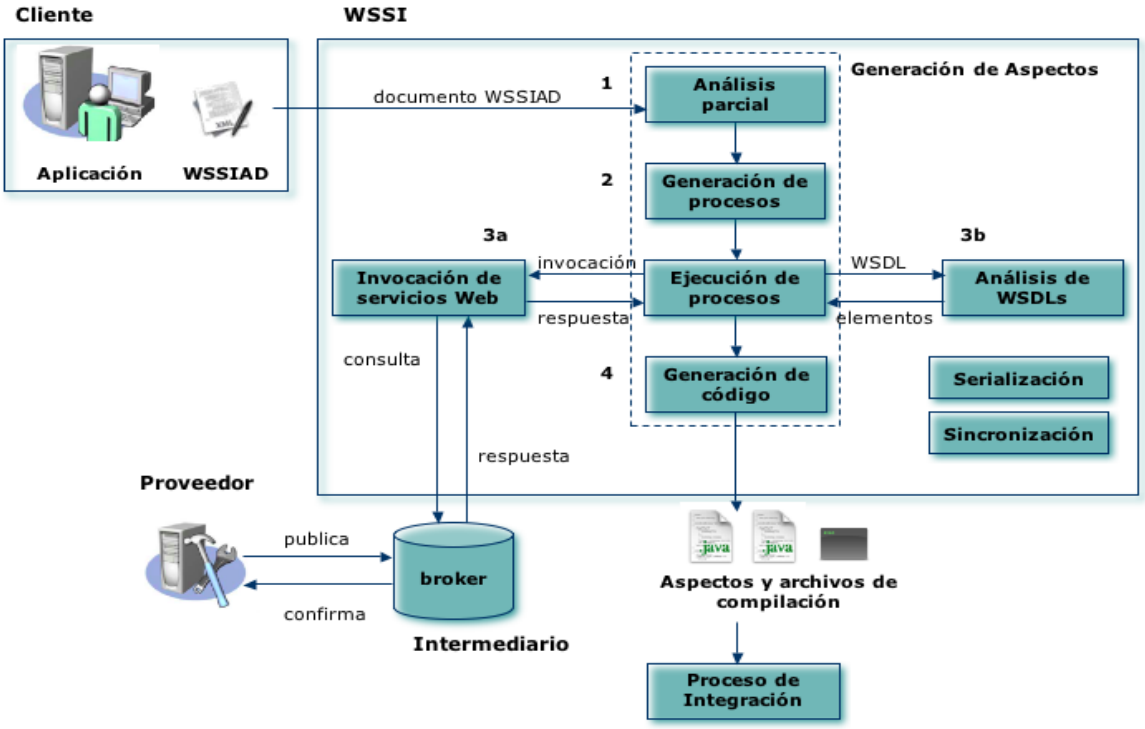

Figura 5.1: Etapa de generación de aspectos.

Cada uno de estos bloques tiene asociado un conjunto de clases, tal y como se muestra en la Tabla 5.1, en el cual se especifican además los bloques de inicialización y ejecución de WSSI. Así mismo, las relaciones existentes entre cada una de ellas se pueden apreciar en el diagrama de clases correspondiente a la Figura 5.2.

5.2. WSSI: Selección e Integración de Servicios Web

WSSI está conformado por un conjunto de clases desarrolladas en Java. Básicamente se implementaron clases para la inicialización y ejecución del proceso principal de WSSI. El proceso principal de WSSI hace uso de clases para la generación de código, ejecución paralela, sincronización, serialización y administración de procesos de tipo `RequestThread`. La especificación de las clases que conforman WSSI se encuentran descritas en el Apéndice B.

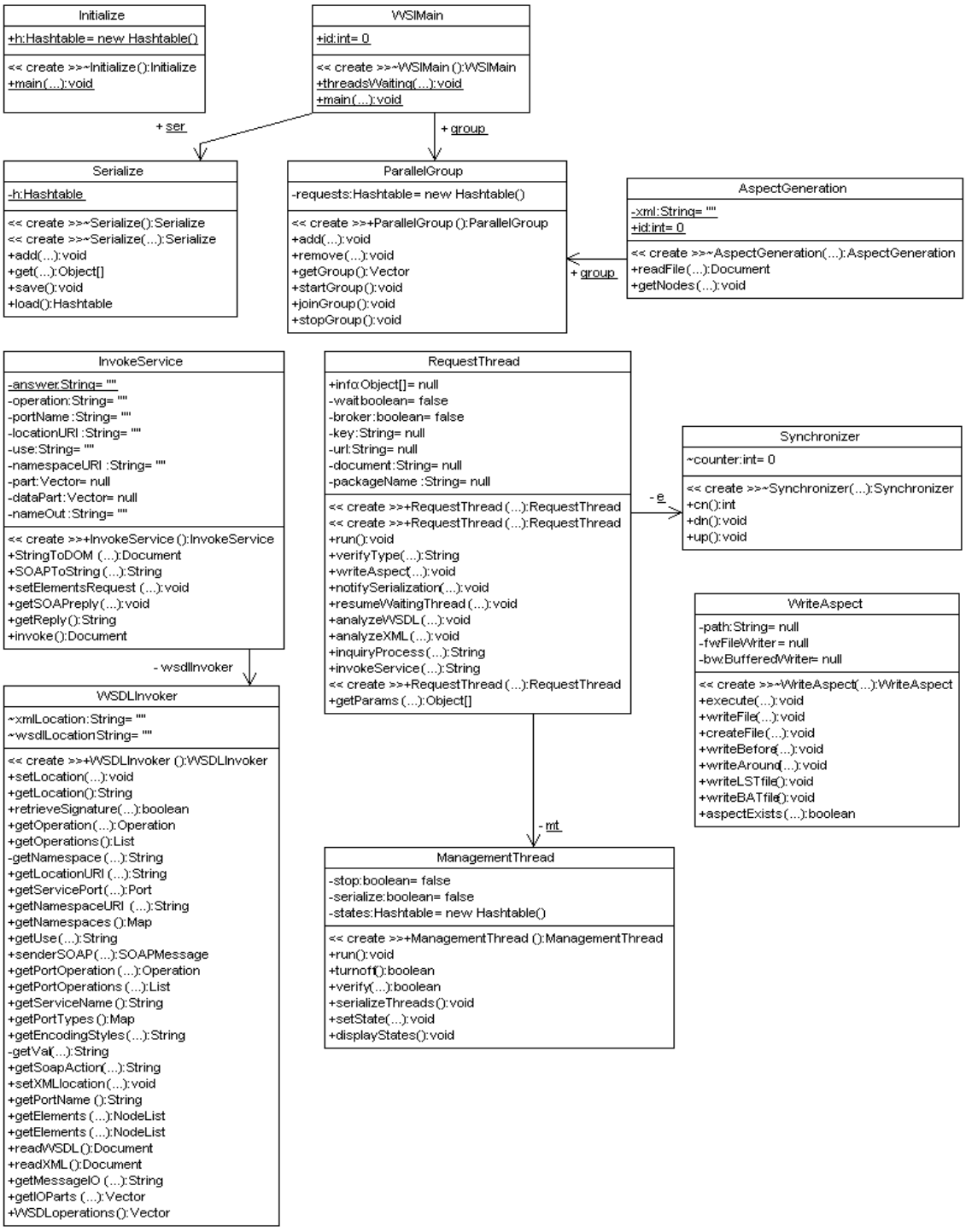

Figura 5.2: Diagrama de clases de WSSI.

Bloque	Clases asociadas
Inicialización de WSSI	`Initialize`
Ejecución de WSSI	`WSSIMain`
Análisis parcial	`AspectGeneration`
Generación de procesos	`AspectGeneration`

Ejecución de procesos
- Sincronización
- Serialización
	`ParallelGroup, RequestThread`
	`Synchronizer`
	`ManagementThread, Serialize`

Generación de código	`WriteAspect`
Análisis de WSDLs	`WSDLInvoker`
Invocación de servicios Web	`InvokeService`

Tabla 5.1: Clases que conforman WSSI.

5.2.1. Inicialización de WSSI

La clase que corresponde a la inicialización de WSSI es `Initialize` y básicamente su propósito es crear e inicializar mediante el método `init()` el archivo threads.ser. En caso de que se decida detener la ejecución de WSSI, este archivo almacena la información serializada correspondiente a cada proceso de generación de código que se encuentre en espera de la respuesta a una suscripción.

5.2.2. Ejecución de WSSI

La clase que inicia todo el proceso de generación de código es `WSSIMain`. En primera instancia, simplemente verifica si existen procesos de suscripción pendientes para reanudar su ejecución. Posteriormente, genera un objeto `AspectGeneration` con el objetivo de realizar un análisis parcial del documento XML que se recibe como

parámetro, iniciar la ejecución del grupo de procesos `RequestThread` generados y esperar por el término de los mismos para que finalmente se actualice la información del archivo threads.ser en caso de que el proceso de serialización se haya llevado a cabo.

5.2.3. Análisis parcial de documentos WSSIAD y generación de procesos

La etapa de generación de aspectos tiene como entrada la definición de un documento WSSIAD, el cual especifica elementos necesarios para realizar la definición de puntos de corte y sus respectivos modificadores de comportamiento. Esta información se utiliza para la generación de procesos correspondientes a objetos `RequestThread`.

Clase AspectGeneration

Esta clase es la que inicia el proceso de generación de código y tiene como objetivo realizar un primer análisis del documento XML especificado por el atributo `xml`, de tal manera que se genere un objeto `RequestThread` para cada uno de los métodos de la aplicación descritos por cada elemento <`pointcut`> especificados en el documento WSSIAD.

Cuando se genera un objeto de esta clase se realiza la lectura del documento WSSIAD que se recibe como parámetro mediante el método `readFile()` que utiliza métodos del API JAXP para obtener la estructura del documento. Posteriormente, el método `getNodes()` se encarga de asociar cada uno de los elementos <`advice`> con su correspondiente elemento <`pointcut`>. Con la información especificada por los atributos de ambos elementos se crea e incializa un objeto `RequestThread` que es identificado por el atributo `id` y agregado a un grupo de *threads* para su posterior ejecución.

5.2.4. Ejecución de procesos

Cada uno de los procesos generados durante el análisis de un documento WSSIAD tiene el objetivo de obtener toda la información para generar el código correspondiente a cada aspecto durante su ejecución.

Clase RequestThread

Esta clase extiende a la clase Thread y su principal objetivo es determinar el tipo de proceso que se realizará para obtener la información necesaria y poder generar los archivos correspondientes a cada uno de los aspectos que serán integrados a una aplicación cliente. Son tres los casos que se contemplan:

1. Cuando se ha especificado el URL correspondiente al WSDL del servicio Web.

2. Cuando no se sabe el URL correspondiente al WSDL del servicio Web, pero si la descripción de los datos asociados a los mensajes SOAP de invocación y respuesta correspondientes al servicio.

3. Cuando se requiere reanudar un procedo que se encontraba en espera de la respuesta a una suscripción.

En primera instancia, el tipo de los parámetros que se reciben en el constructor determinan si se trata de cualquiera de los dos primeros casos o en particular del tercero. La distinción entre los dos primeros casos se establece al determinar si el elemento <advice> del documento WSSIAD es vacío o no.

Para el primer caso, el método `analyzeWSDL()` recupera el documento WSDL especificado por el atributo `url` para analizarlo y obtener la descripción de los datos asociados a sus correspondientes mensajes SOAP de invocación y respuesta. Una vez que se obtiene toda la información necesaria del WSDL el proceso finaliza con la generación del archivo asociado a la invocación de un servicio Web.

Para el segundo caso, el método `analyzeXML()` se encarga de analizar los elementos <message> descritos en el documento WSSIAD obteniendo la descripción de los datos asociados a los mensajes SOAP de invocación y respuesta que se desea satisfaga el servicio Web requerido. Una vez que se obtiene dicha información, se realiza un proceso de consulta al Intermediario mediante el método `inquiryProcess()` para obtener el WSDL del servicio solicitado o para generar una suscripción del mismo y esperar por su respuesta. Dicho proceso de consulta realiza una serie de invocaciones al servicio correspondiente al Intermediario (cuyo WSDL es especificado en este caso por el atributo `url`) haciendo uso del método `invokeService()` y así obtener su respuesta. Los procesos de este tipo que generen suscripciones pueden ser forzados a terminar su ejecución, no sin antes serializar la información referente a los mismos y notificar el proceso de serialización al Intermediario mediante el método `notifySe-`

`rialization()`.

Finalmente, el tercer caso corresponde cuando al iniciar el proceso de WSSI se recupera del archivo threads.ser la información asociada a cada *thread* que se encontraba en espera por la respuesta de una suscripción. Dicha información se asocia a un nuevo objeto `RequestThread` que determinará mediante el método `resumeWaitingThread()` si el *thread* en espera debe reanudar o finalizar su ejecución ya que la descripción del servicio Web requerido pudo haber sido publicada en el Intermediario por un proveedor.

La ejecución de cada objeto `RequestThread` finaliza con la generación del archivo correspondiente a un aspecto. El método que realiza esta tarea es el método `writeAspect()`, el cual utiliza la clase `WriteAspect` siempre y cuando no haya ocurrido un error, no se haya cancelado una suscripción o no se haya serializado la información de la misma. Para poder generar correctamente el archivo de cada aspecto hay que considerar la manera en que se especifican los tipos de datos en la descripción de un servicio Web. El tipo correspondiente a cadenas de caracteres se especifica como `string` en las descripciones de los servicios Web (WSDL) a diferencia de Java que lo especifica como `String` por lo que se debe realizar esta consideración y realizar la conversión del mismo. Así, para estos fines se cuenta con el método `verifyType()`.

Clase ParallelGroup

Como ya se ha descrito, cada uno de los objetos `RequestThread` tiene como propósito obtener la información necesaria para generar el código del aspecto que realizará la invocación de cada servicio Web. Cada uno de estos objetos se ejecuta como un proceso independiente y en paralelo con el resto de los objetos generados.

La clase `AspectGeneration` genera un objeto de la clase `ParallelGroup`, la cual representa a un grupo de objetos `RequestThread` (*threads*) y tiene como objetivo brindar métodos que permitan la manipulación de los mismos.

Las operaciones básicas para la manipulación de *threads* son `start(), join()` y `stop()`. Cada una de ellas permite iniciar, esperar a que termine y finalizar la ejecución de un *thread* respectivamente. En nuestro caso, la clase `ParallelGroup` implementa los métodos `startGroup(), joinGroup()` y `stopGroup()` que permiten iniciar, esperar a que termine y finalizar la ejecución paralela de todo un grupo de

threads, respectivamente. Para agregar y eliminar un objeto al grupo se cuenta con los métodos `add()` y `remove()`. De igual manera, se cuenta con el método `getGroup()` que retorna todos los elementos del grupo.

5.2.5. Sincronización entre procesos

Debido a que la ejecución del proceso correspondiente a cada uno de los objetos `RequestThread` se realiza en paralelo se implementó la clase `Synchronizer` para asegurar la sincronización entre los procesos.

Clase Synchronizer

Existen diferentes mecanismos para garantizar la sincronización entre procesos. Uno de estos mecanismos corresponde a los semáforos, los cuales tienen el objetivo de asegurar que se encuentre un sólo un proceso a la vez dentro de una región crítica. Para nuestro caso, se requiere garantizar la sincronización de procesos correspondientes a *threads*. Por lo tanto, la clase `Synchronizer` crea e implementa las operaciones asociadas a un semáforo.

Los métodos que pertenecen a esta clase son `dn()` y `up()`. Dichos métodos tienen el propósito de decrementar e incrementar el contador correspondiente a un semáforo y de verificar cuando se debe realizar el bloqueo y desbloqueo de procesos que desean acceder a una misma región crítica. Un proceso se bloquea cuando el contador toma un valor negativo o cero y se desbloquea cuando dicho contador toma un valor positivo. Ambos métodos se implementan mediante el uso de los métodos `wait()` y `notify()` definidos en el API de Java.

5.2.6. Serialización de objetos

Durante la ejecución de los procesos encargados de generar el código correspondiente a cada aspecto, puede llevarse a cabo el proceso de serialización de la información correspondiente a algunos de dichos procesos. Las clases encargadas de determinar y realizar el proceso son `ManagementThread` y `Serialize` respectivamente.

Clase ManagementThread

`ManagementThread` extiende a la clase Thread y los objetivos que persigue son establecer el estado de ejecución en el que se encuentran los objetos `RequestThread` y realizar el proceso de serialización de la información correspondiente a los procesos que se encuentren en estado de espera.

Desde que se inicia el proceso de generación de objetos `RequestThread`, la clase `ManagementThread` se encarga de actualizar el estado correspondiente a cada objeto `RequestThread` mediante el método `setState()`. El estado new se establece cuando se crea el objeto, el estado runnable cuando se inicia la ejecución del proceso correspondiente a cada objeto, el estado blocked cuando se encuentra en espera por la respuesta a una suscripción y el estado dead cuando finalizó la ejecución del proceso correctamente o porque se serializó la información del mismo y se forzó a terminar su ejecución.

Cuando se actualiza el estado de cada objeto, `displayStates()` despliega en pantalla la información referente a cada proceso con el fin de que el cliente tenga la posibilidad de dar seguimiento al proceso de generación de código de cada aspecto.

Para determinar y llevar a cabo el proceso de serialización antes mecionado esta clase cuenta con el método `turnoff()` que tiene como propósito determinar si el archivo turnoff.txt existe, lo cual indica que el proceso de serialización tendrá que llevarse a cabo cuando se determine que el estado de los objetos `RequestThread` corresponda al estado blocked o al estado dead. En caso de ser así, el método `serializeThreads()` es el que se encarga de almacenar la información referente a los objetos y notificar a cada uno de ellos que su información se serializará.

Clase Serialize

Cuando se decide finalizar el proceso de WSSI sin que la ejecución de los procesos haya finalizado se lleva a cabo el proceso de serialización, por lo que `Serialize` es la encargada de generar un objeto para almacenar la información de cada uno de ellos y posteriormente actualizar el archivo threads.ser. El proceso de serialización es realizado por `ManagementThread` y el proceso de deserialización por `WSSIMain` a través de los métodos que proporciona esta clase.

Básicamente, `Serialize` crea un objeto Hashtable, al cual se pueden agregar u

obtener elementos correspondientes a la información de objetos `RequestThread` mediante los métodos `add()` y `get()`. El proceso de serialización simplemente actualiza el archivo threads.ser al escribir el objeto Hashtable como una secuencia de bytes a través del método `save()`. El proceso inverso simplemente deserializa el objeto Hashtable del archivo threads.ser a través del método `load()` para que se pueda obtener la información referente a cada objeto.

5.2.7. Generación de código

Una vez que se cuenta con toda la información necesaria para la generación del código correspondiente a un aspecto, se genera el archivo del mismo y los archivos para realizar la compilación de la aplicación con el objetivo de integrar las invocaciones a servicios Web.

Clase WriteAspect

Los métodos que proporciona esta clase tienen el propósito de generar el archivo correspondiente a un aspecto así como los archivos necesarios para su compilación e integración a una aplicación cliente.

Cada proceso `RequestThread` crea un objeto de esta clase para poder invocar el método `writeFile()`, el cual recibe como parámetros toda la información obtenida durante el proceso de análisis como el URL del WSDL, el nombre de la operación y la descripción de los datos asociados a los mensajes SOAP de un servicio Web, entre otros. Primeramente, crea el archivo xAspect.java tal que el prefijo `x` corresponda al nombre de la operación mediante `createFile()`. Posteriormente, escribe en el archivo la definición del punto de corte y el código de los modificadores de comportamiento `before` y `around` mediante `writeBefore()` y `writeAspect()`. Finalmente, genera los archivos appfiles.lst y compile.bat. El primero de ellos incluye una lista de todos los archivos fuente de la aplicación cliente así como el del aspecto recién generado. El segundo de ellos realiza la compilación de todos los archivos especificados por appfiles.lst mediante el compilador de AspectJ con el objetivo de integrar los aspectos a la aplicación cliente.

Esta clase también cuenta con el método `execute()`, el cual simplemente ejecuta comandos del sistema y por lo tanto es el que se encarga de mover cada uno de los archivos generados a la ruta especificada por el atributo `path`.

5.3. Análisis de WSDLs e invocación de servicios Web

Para el análisis de documentos WSDL y la invocación de servicios Web se implementaron las clases `WSDLInvoker` e `InvokeService` respectivamente.

WSSI utiliza ambas clases por medio de la clase `RequestThread` debido a que se analizan documentos WSDL y se realizan invocaciones al Intermediario. El Intermediario por su parte utiliza la clase `WSDLInvoker` para realizar el análisis de documentos WSDL de servicios Web que los proveedores publican.

Debido a que las aplicaciones que integran los `aspectos` generados por WSSI requieren ambas clases para su compilación y ejecución, se creó el archivo **wssi.jar** que contiene dichas clases.

Clase WSDLInvoker

El análisis de documentos WSDL se realiza a través de la clase `WSDLInvoker`, la cual utiliza WSIF (Web Services Invocation Framework) [26] para invocar servicios Web sin importar cómo o dónde los servicios son provistos, ya que se basa en la descripción de un servicio Web (WSDL) para poder invocar los servicios. De igual manera, WSIF contiene utilerías para la obtención de objetos WSDL4J (Web Services Description Language for Java), la cual permite mapear información de un documento WSDL a objetos Java para poder manipularlos de una manera simple.

Algunos de los métodos de esta clase fueron implementados en [27]. Por lo tanto, la clase sirvió como base para agregar métodos necesarios para nuestra implementación.

Esta clase provee métodos para realizar el análisis de un documento WSDL, los cuales son representados en primera instancia como objetos `Definition` para poder realizar su análisis con las utilerías que proporciona WSIF. Básicamente cada uno de estos métodos tiene el objetivo de obtener información necesaria de un documento WSDL. Los principales métodos que utilizan las clases `RequestThread` e `InvokeService` son descritos en las Tablas 5.2 y 5.3.

Método	Descripción
`getLocationURI()`	Obtiene el URL del servicio Web.
`getNamespaceURI()`	Obtiene los espacios de nombres asociados al servicio Web.
`getServicePort()`	Obtiene el subelemento <`port`> del elemento <`service`>.
`getUse()`	Obtiene el atributo *use* de la operación que se especifique y que corresponda con el elemento <`binding`> del servicio Web.
`senderSOAP()`	Genera el mensaje SOAP que realizará la invocación del servicio Web.
`setLocation()`	Localiza un documento WSDL para realizar su conversión a un objeto `Definition` de Java.

Tabla 5.2: Métodos de la clase `WSDLInvoker` [27].

Clase InvokeService

Esta clase está conformada por los métodos necesarios para realizar la invocación a los servicios Web y analizar la respuesta de los mismos.

Cuando se requiere realizar la invocación a un servicio Web primero se deben obtener todos los elementos necesarios del documento WSDL necesarios para formar el mensaje SOAP. Una vez que se cuenta con dichos elementos se puede realizar la invocación del servicio mediante el método `senderSOAP()` de la clase `WSDLInvoker` y así obtener el mensaje SOAP de respuesta. Este mensaje se convierte a una cadena y posteriormente a un objeto `Document`, lo cual facilita su análisis para obtener finalmente el valor del elemento correspondiente a la respuesta del servicio Web.

Método	Descripción
`getElements()`	Método al que se especifica el nombre de los elementos o subelementos que se desean obtener de un documento.
`getIOParts()`	Obtiene los subelementos <`part`> del elemento <`message`> que se especifique.
`getMessageIO()`	Obtiene el nombre de los mensajes de entrada o salida asociados a una operación en específico.
`getPortName()`	Obtiene el atributo *name* del subelemento <`port`>.
`readWSDL()`	Realiza la conversión de un documento WSDL a un objeto `Document`.
`readXML()`	Realiza la lectura de un documento XML para obtener un objeto `Document`.
`setXMLlocation()`	Establece la ruta donde se localiza un documento XML.
`WSDLOperations()`	Obtiene las operaciones asociadas a un documento WSDL de un servicio Web.

Tabla 5.3: Métodos agregados a la clase `WSDLInvoker` [27].

5.4. Intermediario

El Intermediario (Broker) corresponde a un servicio Web desarrollado e implementado con la herramienta `WSDL2WebService` que ofrece WSDK [28]. El propósito general del Intermediario es proporcionar las operaciones básicas de consulta, publicación y suscripción al igual que un nodo UDDI.

Los archivos generados por `WSDL2WebService` corresponden a `Broker, BrokerBindingImpl, BrokerBindingStub, BrokerLocator` y `BrokerPortType`. Cada uno de ellos corresponde a las interfaces y clases necesarias para la implementación del servicio Intermediario (Figura 5.3). La única clase que se terminó de implementar fue `BrokerBindingImpl` debido a que esta corresponde a la implementación de cada una de las operaciones que conforman el servicio Intermediario. La especificación de las clases que conforman al Intermediario se encuentran descritas en el Apéndice C.

5.4.1. Descripción

La clase `BrokerBindingImpl` es aquella que implementa cada una de las operaciones que conforman al Intermediario. Dichas operaciones pueden ser divididas en los módulos que corresponden a la consulta de servicios, publicación de servicios, suscripción de servicios y notificación de serialización de suscripciones.

Consulta de servicios

El Intermediario cuenta con una operación básica de consulta `askservice` para recuperar información referente a los registros de los servicios Web. Dicha operación realiza la búsqueda de un servicio Web que cumpla con las características especificadas, las cuales corresponden al nombre de la operación y a la descripción de los datos (nombre y tipo) asociados a los mensajes de petición y repuesta de un servicio Web.

Publicación de servicios

La operación `publish` realiza la publicación de la descripción de un servicio Web (WSDL). El proceso de publicación inicia con el análisis del documento WSDL asociado a un servicio Web para obtener una descripción de cada una de las operaciones que lo conforman. Posteriormente, se verifica si alguna de las descripciones obtenidas corresponde a descripciones de operaciones asociadas a servicios Web suscritas previamente por WSSI. En caso de ser así, se obtienen todos los identificadores de dichas suscripciones para desbloquear el proceso `asksubscription` asociado a cada una de ellas, regresar la respuesta correspondiente y finalizar su proceso de suscripción.

La operación `unpublish` se encarga de eliminar todos los registros que coincidan con el URL del documento WSDL especificado. Una vez eliminados los registros se notifica al proveedor que los registros se han eliminado satisfactoriamente.

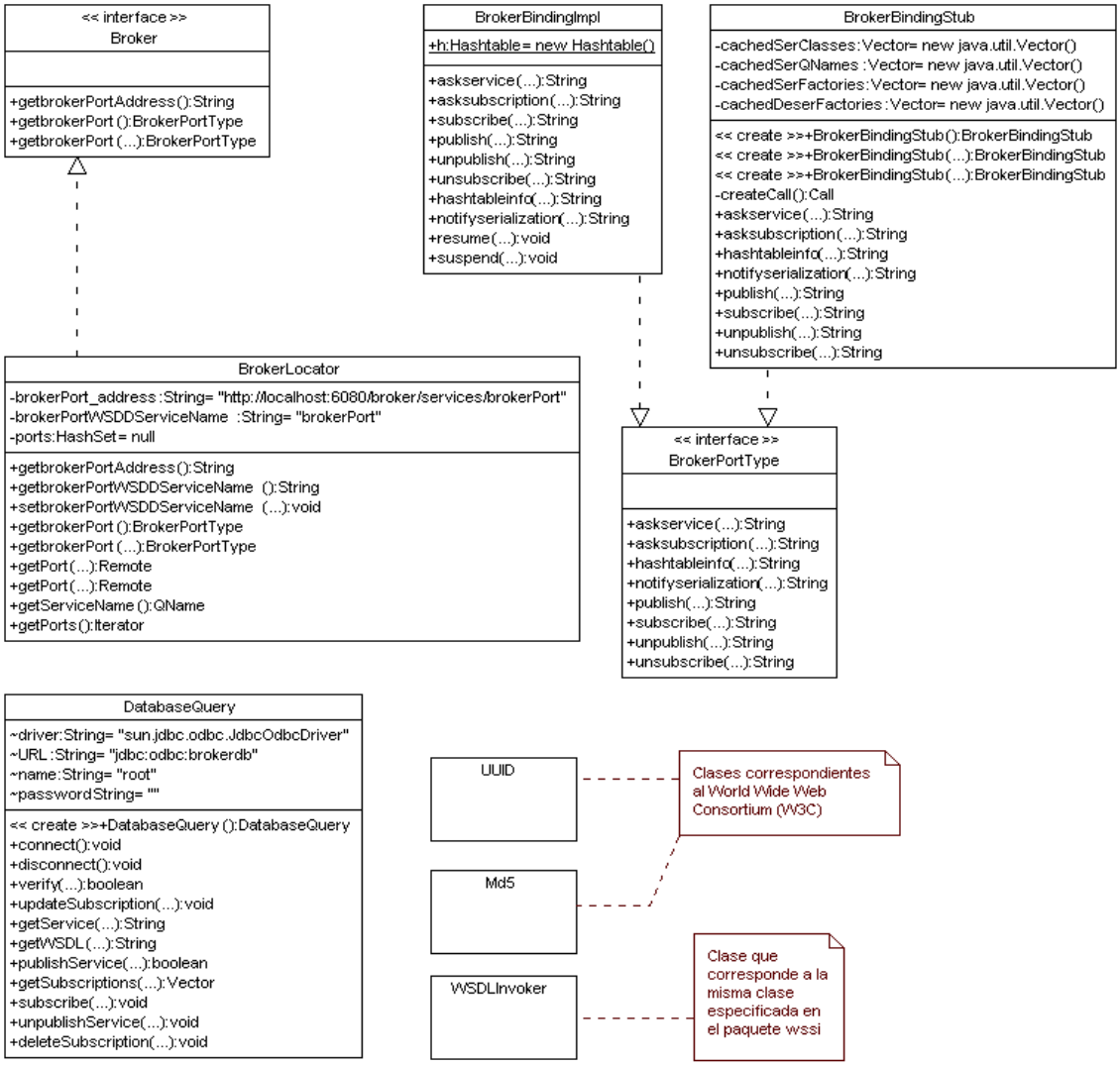

Figura 5.3: Diagrama de clases del Intermediario.

Suscripción de servicios

Cuando se realiza la consulta de un servicio Web, puede ocurrir que dicho servicio no se encuentre publicado. En este caso, se realiza la suscripción de la descripción correspondiente a una operación asociada a un servicio Web.

El proceso de suscripción es realizado mediante la operación `subscribe`, la cual almacena el registro correspondiente a la descripción de una operación asociada a un servicio Web que se requiere invocar y el identificador que se genera para dicha suscripción.

Una vez que una suscripción se ha registrado, la operación `asksubscription` obliga a esperar por la respuesta de la suscripción y como consecuencia, el proceso se bloquea. Los casos en los que el proceso puede llegar a desbloquearse son los siguientes:

1. Cancelación de la suscripción.
2. Publicación de la descripción de una operación asociada a un servicio Web que corresponda con la descripción suscrita.
3. Suspensión de la ejecución de WSSI, lo cual indica que toda la información de las suscripciones asociadas a él serán serializadas.

Cuando se tiene conocimiento del identificador generado para una suscripción en específico, la operación `unsubscribe` puede ser invocada para realizar su cancelación. El proceso de cancelación simplemente requiere del identificador correspondiente a la suscripción para que primeramente se desbloquee el proceso `asksubscription` asociado a la misma de tal manera que se elimine su registro y regrese como respuesta la notificación de que la suscripción ha sido cancelada.

La operación `notifyserialization` tiene como propósito notificar al Intermediario que la información referente a una suscripción va a ser serializada por WSSI, por lo cual el proceso `asksubscription` se desbloquea para continuar con su ejecución y regresar una respuesta de confirmación indicando que el registro correspondiente a la suscripción ha sido actualizado como un proceso serializado.

5.4.2. Consulta al directorio de servicios Web

La clase `DatabaseQuery` tiene el objetivo de realizar la conexión al repositorio de descripciones de servicios Web que forma parte del Intermediario. Esta conexión se

realiza a través del driver `sun.jdbc.odbc.JdbcObdcDriver` que el API de Java ofrece.

Cada una de las operaciones del intermediario utiliza los métodos que implementa esta clase, los cuales ejecutan consultas SQL para crear un nuevo registro, consultar o actualizar la información referente al mismo y/o eliminarlo.

5.4.3. Generación de UUIDs

La clase que las operaciones del Intermediario utilizan para la generación de identificadores únicos y universales corresponde a `UUID` [29], la cual a su vez utiliza la clase `Md5` [30]. El objetivo de la clase `UUID` es generar un identificador de 128 bits para cada una de las suscripciones generadas por el Intermediario y de esta manera identificarlas de manera única. La clase `Md5` simplemente es una clase auxiliar para la generación de cada identificador. Ambas clases fueron obtenidas del World Wide Web Consortium (W3C) [31].

5.5. Sumario

En el presente capítulo describimos la implementación que se realizó para WSSI y el Intermediario. Para el primero de ellos se describieron las clases escritas en el lenguaje Java, por lo que esta parte de la implementación puede ser considerada portable a cualquier plataforma siempre y cuando se cuente con una Máquina Virtual de Java (JVM). Por otra parte, se describe cómo el Intermediario fue implementado como un servicio Web a través del uso de las herramientas que provee WSDK.

Se ha observado que la implementación descrita en este capítulo tiene un buen tiempo de respuesta para cada una de las invocaciones realizadas al Intermediario y la cual corresponderá de acuerdo al procesador de la máquina donde se ejecute. De igual manera, hay que considerar que debido a que las operaciones del Intermediario son síncronas, cada uno de los procesos generados del lado del cliente, tiene que esperar hasta recibir una respuesta. Por tal motivo, hay que considerar que el número de procesos (*threads*) soportados dependerá igualmente de la máquina donde se realice su ejecución.

Capítulo 6

Conclusiones y trabajo a futuro

Un sistema contempla como primera etapa el diseño de su arquitectura, la cual corresponde a la identificación de subsistemas de manera independiente, el modelo de control que especifique las relaciones existentes entre cada uno de ellos y la descomposición modular de cada subsistema. Este diseño afecta la ejecución, robustez, seguridad, disponibilidad y mantenibilidad del mismo.

El diseño de sistemas distribuidos implica de igual manera diseñar su arquitectura considerando características como compartición de recursos de hardware y/o software, concurrencia, escalabilidad, tolerancia a fallas y transparencia. Típicamente, dichas características tienen que ser contempladas desde un inicio, lo cual implica realizar varias consideraciones para cada una de ellas. Como consecuencia, los sistemas distribuidos tienden a ser más complejos que los sistemas centralizados, la seguridad es díficil de manejar, el mantenimiento requiere un mayor esfuerzo y el tiempo de respuesta puede variar de acuerdo a las condiciones de la red.

Todas estas características que se contemplan durante la etapa de diseño se simplifican considerablemente para un sistema centralizado. De esta manera, la presente tesis se enfoca a la idea de partir de una aplicación centralizada para obtener una versión distribuida de la misma sin necesidad de realizar cambios a su diseño. Como consecuencia, los desarrolladores no necesitan invertir tiempo para agregar o realizar cambios a los módulos, debido a que se reduce el proceso de desarrollo.

A lo largo de los capítulos anteriores hemos presentado el diseño e implementación correspondiente a una solución que permite identificar los puntos específicos en que una aplicación sustituye el comportamiento de métodos locales por invocaciones a

servicios Web. Así, proponemos utilizar el enfoque de la programación orientada a aspectos para evitar la invasividad de código y proporcionar mayor dinamismo en la selección de servicios Web.

6.1. Contribuciones

El trabajo de tesis presenta contribuciones al proceso de integración de servicios Web a una aplicación a través de la programación orientada a aspectos.

Obtención de aplicaciones distribuidas

El proceso de generación de aspectos tiene el objetivo de que una vez finalizada su generación, sean integrados a una aplicación centralizada para sustituir las invocaciones de métodos locales por invocaciones a servicios Web. Cada uno de los aspectos generados realiza la invocación de un servicio Web en específico, por lo que gracias a las ventajas que presenta el paradigma de la programación orientada a aspectos, la aplicación original no se modifica en código ni en diseño.

Las decisiones correspondientes a las características de distribución que realizamos para la implementación de cada uno de los aspectos encargados de realizar la invocación de un servicio Web fueron principalmente:

- El uso del lenguaje orientado a aspectos AspectJ para la implementación e integración de los servicios Web.
- El uso de WSDL4J y WSIF para el análisis de documentos WSDL e invocación de servicios Web respectivamente.
- Las consideraciones sobre la manera en que se definen los tipos de datos en Java y en las descripciones de servicios Web (WSDLs).
- En cuanto a la tolerancia a fallas por parte de los servicios, consideramos que si durante la ejecución de los aspectos ocurre una excepción, el aspecto indicará y ejecutará el comportamiento original correspondiente al método local.
- En cuanto a la seguridad y sincronización se refiere, contamos con las características que provee la infraestructura de servicios Web para proveer la seguridad y sincronización entre clientes y servidores.

En general, la versión distribuida que se obtiene a partir de una aplicación centralizada tendrá los problemas inherentes a las aplicaciones distribuidas. En nuestro caso, los problemas correspondientes a fallas de los servidores, serán detectados por el aspecto (a través de su manejo de excepciones) de tal manera que la aplicación no presente problemas durante su ejecución. La sincronización y seguridad son problemas que en este trabajo de tesis no fueron considerados, por lo que representan trabajo a futuro.

Por lo tanto, el proceso de desarrollo de una aplicación se simplifica considerablemente ya que la integración se realiza dinámicamente.

Ocultamiento de los requerimientos de distribución

Como ya se ha descrito, cuando se diseña una aplicación distribuida se tienen que realizar muchas consideraciones para su implementación. Con el enfoque que proponemos no es necesario realizar consideraciones de distribución. De esta manera, no se tiene que lidiar con decisiones asociadas a los protocolos, las herramientas y los mismos servicios Web que se utilizarán.

Así, se logran modularizar los requerimientos correspondientes a la distribución física de una manera simple y transparente a los programadores, ya que sólo hay que identificar los puntos específicos en que se requiere modificar el comportamiento de la aplicación.

La descripción de los servicios Web de los cuales no se tiene conocimiento corresponde a los tipos de datos asociados a sus mensajes de invocación y respuesta, y las cuales se realizan de acuerdo a la definición de los métodos locales correspondientes a la aplicación centralizada.

Creación de un directorio de servicios

El estándar que actualmente se utiliza para publicar y descubrir servicios Web es la especificación correspondiente a UDDI. En esta tesis proponemos la creación de un directorio de servicios básico, así como un servicio de intermediación (Intermediario) para realizar la consulta y publicación de descripciones de servicios Web.

De igual manera, proponemos las operaciones correspondientes al proceso de suscripción de aquellas descripciones no publicadas en el directorio, lo cual permite que

el directorio de servicios incluya además de ofertas, las demandas de las descripciones de servicios Web requeridas.

El objetivo principal del Intermediario es entonces, asegurar el aprovisionamiento de los servicios Web en el momento en que un proveedor publique un servicio que corresponda con alguna de las suscripciones por las que se espera por una respuesta.

Diseño de una arquitectura

La arquitectura diseñada para la selección e integración de servicios Web a aplicaciones escritas en el lenguaje Java corresponde a una arquitectura por capas (Figura 6.1), las cuales corresponden a la capa de transporte (TCP/IP, HTTP), la capa de mensajes, la capa asociada a los procesos involucrados con los servicios Web (AspectJ, JVM) y la capa correspondiente a las aplicaciones.

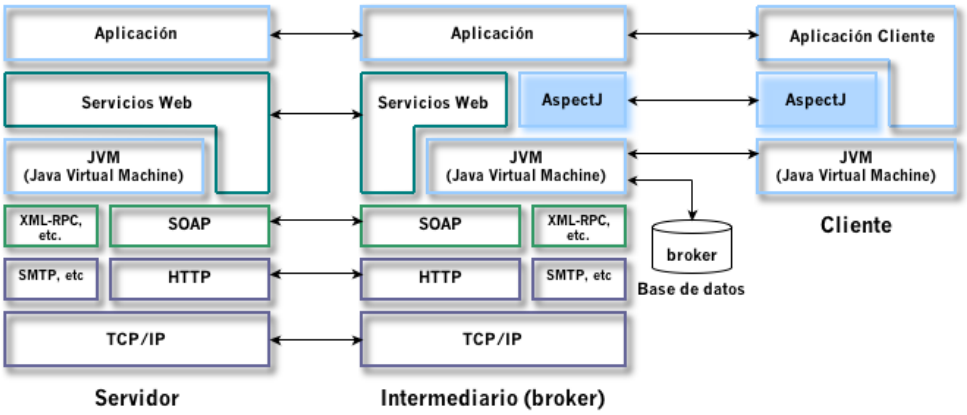

Figura 6.1: Arquitectura propuesta.

Dicha arquitectura ofrece una nueva alternativa para el desarrollo de aplicaciones centrándose en el desarrollo de la infraestructura necesaria para el descubrimiento dinámico de servicios Web, así como la generación de los archivos correspondientes a cada aspecto, su implementación y su integración a una aplicación.

La funcionalidad de WSSI y del Intermediario han sido probados con casos correspondientes a diferentes aplicaciones, lo cual ha permitido observar su utilidad, ya que a pesar de ser un enfoque sencillo permite manejar problemas complejos. El caso de estudio que presentamos en el capítulo 3 contempla la descripción de los casos que

pueden surgir durante el descubrimiento de los servicios y la generación dinámica del conjunto de aspectos, y la manera en que WSSI y el Intermediario interactúan para lograr dichos objetivos.

6.2. Trabajo a futuro

Los resultados obtenidos durante esta tesis han sido satisfactorios y presentados en el artículo [32]. Como se ha mencionado no se han desarrollado muchas propuestas que contemplen el uso de la programación orientada a aspectos para realizar la integración de servicios Web. No obstante, se pueden agregar elementos que incrementen la funcionalidad que proporciona WSSI y el Intermediario.

La propuesta considera descripciones de operaciones asociadas a un servicio Web, las cuales están conformadas por el nombre y tipo de datos asociados a los mensajes de invocación y respuesta de un servicio Web. En este caso, se podrían integrar diferentes reglas de selección de servicios Web (costo, SLA's, restricciones de acceso, restricciones basadas en la calidad de servicio, entre otras), ya que típicamente se realiza la selección de los mismos basándose en la funcionalidad que ofrecen. Por lo tanto, con la integración de nuevas reglas la selección de los servicios se realizaría de acuerdo a necesidades específicas que satisfagan de una mejor forma los requerimientos de una aplicación.

Al proponer un diseño básico, es claro pensar en extender la funcionalidad del directorio de servicios y del Intermediario. De igual forma, el hecho de considerar nuevas reglas de selección de servicios Web implica realizar modificaciones a ambos, por lo que se tendrían que agregar operaciones para la consulta, publicación y suscripción de servicios.

Debido a la naturaleza de AspectJ, el proceso de integración de los servicios Web se realiza de manera estática (en tiempo de compilación). Por lo tanto, sería una alternativa estudiar otro lenguaje o herramienta que permita integrarlos en tiempo de ejecución sin la necesidad de realizar la compilación y reinicialización de las aplicaciones.

Así mismo, se ha considerado realizar la conjunción de este trabajo con otro trabajo de investigación [33] [34] que tiene el objetivo de orquestar servicios Web a través de la definición de un lenguaje de coordinación sencillo denominado BPCL (Busi-

ness Process Coordination Language). La principal relación con esta tesis consiste en que se podrá realizar la selección e integración dinámica de servicios Web que sean coordinados a través de BPCL.

Apéndice A

Aspectos correspondientes al caso de estudio

En este apéndice se incluyen los archivos de los aspectos generados para la aplicación Book Store correspondiente al caso de estudio, los cuales son:

- registerBookAspect.java
- searchBookAspect.java
- getQuoteAspect.java
- buyBookAspect.java

De igual manera, se incluye el código del aspecto generado para el servicio `buyBook`, el cual corresponde al archivo fundsFeeAspect.java.

A.1. Aspecto registerBookAspect

```
1  import java.util.Vector;
2  import org.w3c.dom.Document;
3
4   aspect registerBookAspect {
5
6    wssi.InvokeService service = null;
7
8     pointcut registerBook( String id,String title,String author,
9       float price,int available ) : execution( String
10      Store.registerBook(String,String,String,float,int) ) &&
```

```
11      args( id,title,author,price,available );
12
13    before( String id,String title,String author,float price,int
14      available ) : registerBook( id,title,author,price,available ){
15      try {
16        String wsdl = "http://store:6080/register/services/registerPort?wsdl";
17        String operation = "registerBook";
18
19        Vector dataPart = new Vector();
20        dataPart.add(String.valueOf(id));
21        dataPart.add(String.valueOf(title));
22        dataPart.add(String.valueOf(author));
23        dataPart.add(String.valueOf(price));
24        dataPart.add(String.valueOf(available));
25
26        service = new wssi.InvokeService();
27        service.setElementsRequest(wsdl,operation,dataPart);
28      }
29      catch(Exception e) { System.out.println("service not available"); }
30    }
31
32    String around( String id,String title,String author,float price,
33      int available ) : registerBook( id,title,author,price,available ){
34      String reply = null;
35      try {
36        Document document = service.invoke();
37        service.getSOAPreply(document.getDocumentElement());
38        String SOAPreply = service.getReply();
39        reply = SOAPreply;
40      }
41      catch(Exception e) {
42        System.out.println("local method is going to be executed ...");
43        return proceed(id,title,author,price,available);
44      }
45    }
46  }
```

A.2. Aspecto searchBookAspect

```
1  import java.util.Vector;
2  import org.w3c.dom.Document;
3
```

```
aspect searchBookAspect {

  wssi.InvokeService service = null;

   pointcut searchBook( String title):
     execution( String Store.searchBook(String)) && args( title );

   before( String title ) : searchBook( title ){
    try {
      String wsdl = "http://store:6080/search/services/searchPort?wsdl";
      String operation = "searchBook";

      Vector dataPart = new Vector();
      dataPart.add(String.valueOf(title));

      service = new wssi.InvokeService();
      service.setElementsRequest(wsdl,operation,dataPart);
    }
    catch(Exception e) { System.out.println("service not available"); }
  }

  String around( String title ) : searchBook( title ){
    String reply = null;
    try {
      Document document = service.invoke();
      service.getSOAPreply(document.getDocumentElement());
      String SOAPreply = service.getReply();
      reply = SOAPreply;
    }
    catch(Exception e) {
      System.out.println("local method is going to be executed ...");
      return proceed(title);
    }
  }
}
```

A.3. Aspecto getQuoteAspect

```
import java.util.Vector;
import org.w3c.dom.Document;

aspect getQuoteAspect {
```

```
5
6   wssi.InvokeService service = null;
7
8    pointcut getQuote( String id,String quote ) :
9      execution( float Store.getQuote(String,String) ) && args( id,quote );
10
11   before( String id,String quote ) : getQuote( id,quote ){
12    try {
13      String wsdl = "http://store:6080/quote/services/quotePort?wsdl";
14      String operation = "getQuote";
15
16      Vector dataPart = new Vector();
17      dataPart.add(String.valueOf(id));
18      dataPart.add(String.valueOf(quote));
19
20      service = new wssi.InvokeService();
21      service.setElementsRequest(wsdl,operation,dataPart);
22    }
23    catch(Exception e) { System.out.println("service not available"); }
24  }
25
26  float  around( String id,String quote ) : getQuote( id,quote ){
27    float reply = 0;
28    try {
29      Document document = service.invoke();
30      service.getSOAPreply(document.getDocumentElement());
31      String SOAPreply = service.getReply();
32      reply = Float.parseFloat(SOAPreply);
33    }
34    catch(Exception e) {
35      System.out.println("local method is going to be executed ...");
36      return proceed(id,quote);
37    }
38  }
39 }
```

A.4. Aspecto buyBookAspect

```
1 import java.util.Vector;
2 import org.w3c.dom.Document;
3
4  aspect buyBookAspect {
```

```
5
6   wssi.InvokeService service = null;
7
8    pointcut buyBook( String id,int quote,float purchase,String card,
9     String account ) : execution( String Store.buyBook(String,int,float,
10    String,String) ) && args( id,quote,purchase,card,account );
11
12   before( String id,int quote,float purchase,String card,String account
13     ) : buyBook( id,quote,purchase,card,account ){
14    try {
15      String wsdl = "http://store:6080/buy/services/buyPort?wsdl";
16      String operation = "buyBook";
17
18      Vector dataPart = new Vector();
19      dataPart.add(String.valueOf(id));
20      dataPart.add(String.valueOf(quote));
21      dataPart.add(String.valueOf(purchase));
22      dataPart.add(String.valueOf(card));
23      dataPart.add(String.valueOf(account));
24
25      service = new wssi.InvokeService();
26      service.setElementsRequest(wsdl,operation,dataPart);
27    }
28    catch(Exception e) { System.out.println("service not available"); }
29  }
30
31  String around( String id,int quote,float purchase,String card,String
32    account ) : buyBook( id,quote,purchase,card,account ){
33    String reply = null;
34    try {
35      Document document = service.invoke();
36      service.getSOAPreply(document.getDocumentElement());
37      String SOAPreply = service.getReply();
38      reply = SOAPreply;
39    }
40    catch(Exception e) {
41      System.out.println("local method is going to be executed ...");
42      return proceed(id,quote,purchase,card,account);
43    }
44  }
45 }
```

A.5. Aspecto fundsFeeAspect

```
1  import java.util.Vector;
2  import org.w3c.dom.Document;
3
4   aspect fundsFeeAspect {
5
6    wssi.InvokeService service = null;
7
8     pointcut fundsFee( String bankdb,String account,float purchase ):
9       execution( boolean BuyBookRMIImpl.fundsFee(String,String,float) )
10      && args( bankdb,account,purchase );
11
12    before( String bankdb,String account,float purchase ) :
13      fundsFee( bankdb,account,purchase ){
14     try {
15       String wsdl = "http://store:6080/funds/services/fundsPort?wsdl";
16       String operation = "fundsFee";
17
18       Vector dataPart = new Vector();
19       dataPart.add(String.valueOf(bankdb));
20       dataPart.add(String.valueOf(account));
21       dataPart.add(String.valueOf(purchase));
22
23       service = new wssi.InvokeService();
24       service.setElementsRequest(wsdl,operation,dataPart);
25     }
26     catch(Exception e) { System.out.println("service not available"); }
27   }
28
29   boolean around( String bankdb,String account,float purchase ) :
30      fundsFee( bankdb,account,purchase ){
31     boolean reply = false;
32     try {
33       Document document = service.invoke();
34       service.getSOAPreply(document.getDocumentElement());
35       String SOAPreply = service.getReply();
36       reply = ((SOAPreply.compareTo("1") == 0) ? true:false);
37     }
38     catch(Exception e) {
39       System.out.println("local method is going to be executed ...");
40       return proceed(bankdb,account,purchase); }
41   }
42 }
```

Apéndice B

Paquete WSSI

Class Summary	
AspectGeneration	Clase que realiza la lectura y análisis de un documento XML para generar objetos RequestThread que realizarán el proceso de generación de código de cada aspecto correspondiente a cada elemento <pointcut>.
Initialize	Clase que inicializa WSSI.
InvokeService	Clase que provee los métodos necesarios para realizar la invocación a servicios Web y analizar la respuesta de los mismos.
ManagementThread	Clase encargada de establecer el estado en que se encuentra un objeto RequestThread y detectar el momento en que se realizará la serialización de la información correspondiente a dichos objetos.
ParallelGroup	Clase encargada de ejecutar en paralelo los procesos correspondientes a objetos RequestThread.
RequestThread	Clase que determina la manera en que se realizará el proceso de generación de código correspondiente a cada aspecto.
Serialize	Clase que serializa la información de los procesos encargados de realizar la generación de código correspondiente a los aspectos.

Synchronizer	Clase que tiene como objetivo garantizar la sincronización de los procesos tal que sólo un proceso se encuentre a la vez en una región crítica.
WSDLInvoker	Clase encargada de realizar el análisis de documentos WSDL correspondientes a servicios Web, así como la invocación de los mismos a través de los métodos que proporciona WSIF.
WSSIMain	Clase principal de WSSI que a través de la cual se accesa al sistema y se inicia el proceso de selección e integración de servicios Web.
WriteAspect	Clase que tiene como objetivo crear el archivo de cada aspecto, así como todo el código referente al mismo.

Class AspectGeneration

wssi.AspectGeneration

public class **AspectGeneration**

Clase que realiza la lectura y análisis de un documento XML para generar objetos RequestThread que realizarán el proceso de generación de código de cada aspecto correspondiente a cada elemento <pointcut>.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
public static int	**id** Identificador de cada objeto RequestThread.
private static String	**xml** Ruta donde se localiza el documento XML.

Association Summary	
public ParallelGroup	**group** Grupo que representa un conjunto de objetos RequestThread.

Constructor Summary	
	AspectGeneration(String xml, ParallelGroup group, int id) Constructor que crea un objeto AspectGeneration.

Method Summary	
public void	**getNodes**(Document document, Node node) Analiza cada uno de los elementos del documento XML descrito por document para obtener la información necesaria y poder generar el objeto RequestThread correspondiente.
public Document	**readFile**(String fileName) Realiza la lectura del documento XML especificado y genera un objeto Document que representa la estructura del documento.

Class Initialize

wssi. Initialize

public class **Initialize**

Clase que inicializa WSSI.

Authors: Marisol Pérez Reséndiz.

Attribute Summary

public static Hashtable	**h** Objeto que almacena la información correspondiente a objetos RequestThread.

Constructor Summary	
	Initialize() Constructor que crea un objeto Initialize.

Method Summary	
public static void	**main**(String[] args) Serializa h y lo almacena en el archivo *threads.ser*.

Class InvokeService

wssi. InvokeService

public class **InvokeService**

Clase que provee los métodos necesarios para realizar la invocación a servicios Web y analizar la respuesta de los mismos.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
private static String	**answer** String que representa la respuesta del servicio Web.
private Vector	**dataPart** Valor de los elementos de una operación correspondiente a un servicio Web.
private String	**locationURI** Nombre del puerto correspondiente a un servicio Web.

private String	**nameOut** String que representa el nombre del elemento que contiene el valor de la respuesta del servicio Web.
private String	**namespaceURI** Espacio de nombres correspondiente a un servicio Web.
private String	**operation** Nombre de una operación perteneciente a un servicio Web.
private Vector	**part** Nombre de los elementos de una operación correspondiente a un servicio Web.
private String	**portName** Nombre del puerto correspondiente a un servicio Web.
private String	**use** Tipo de codificación de los datos de una operación correspondiente a un servicio Web.

Association Summary	
private WSDLInvoker	**wsdlInvoker** Objeto WSDLInvoker.

Constructor Summary	
public	**InvokeService**() Constructor que crea un objeto InvokeService.

Method Summary	
public String	**SOAPToString**(SOAPMessage msg) Convierte el valor de un objeto SOAPMessage a un objeto String.
public Document	**StringToDOM**(String messageSOAPstring) Convierte el valor de un objeto String a un objeto Document.
public String	**getReply**() Retorna el valor de reply.

public void	**getSOAPreply**(Node node) Procesa recursivamente los nodos hijos del nodo especificado por node hasta encontrar el elemento que corresponda con el valor de SOAPreply y obtener su valor para almacenarlo en reply.
public Document	**invoke**() Realiza la invocación del método senderSOAP de la clase WSDLInvoker y obtiene la respuesta de dicha invocación para convertirla a un objeto String y posteriormente a un objeto Document.
public void	**setElementsRequest**(String wsdl, String operation, Vector dataPart) Obtiene e inicializa todos los elementos necesarios para realizar la invocación de un servicio Web a través de métodos pertenecientes a la clase WSDLInvoker.

Class ManagementThread

wssi. ManagementThread

public class **ManagementThread**

Clase encargada de establecer el estado en que se encuentra un objeto RequestThread y detectar el momento en que se realizará la serialización de la información correspondiente a dichos objetos.

Authors: Marisol Pérez Reséndiz.

See also: Thread

Attribute Summary	
private boolean	**serialize** Indica si el proceso de serialización debe llevarse a cabo.

private Hashtable	**states** Hashtable que almacena el *hash code* y un arreglo de objetos (estado e información) correspondiente a un objeto RequestThread.
private boolean	**stop** Indica si la ejecución del método run debe continuar o terminar.

Association Summary	
public RequestThread	**requestThread**

Constructor Summary	
public	**ManagementThread**() Constructor que crea un objeto ManagementThread e inicia la ejecución del método run.

Method Summary	
public void	**displayStates**() Despliega el estado de cada uno de los objetos RequestThread.
public void	**run**() Método encargado de verificar si el valor de serialize es true así como el el valor de retorno del método verify(1), lo cual determina que el proceso de serialización debe realizarse.
public void	**serializeThreads**() Se agrega a un objeto Hashtable la información de cada uno de los objetos RequestThread siempre y cuando su estado sea *blocked* y el valor de RequestThread.broker sea true.
public void	**setState**(String key, Object[] obj) Actualiza el estado y la información de un objeto RequestThread.
public boolean	**turnoff**() Verifica la existencia del archivo *threads.ser*.
public boolean	**verify**(int type) Verifica el estado de los objetos RequestThread.

Class ParallelGroup

wssi. ParallelGroup

public class **ParallelGroup**

Clase encargada de ejecutar en paralelo los procesos correspondientes a objetos RequestThread.

Authors: Marisol Pérez Reséndiz.
See also: RequestThread

Attribute Summary	
private Hashtable	**requests** Hashtable que almacena objetos RequestThread.

Association Summary	
public AspectGeneration	**aspectGeneration**
public WSSIMain	**wSSIMain**

Constructor Summary	
public	**ParallelGroup**() Constructor que crea un objeto ParallelGroup.

Method Summary	
public void	**add**(String key, RequestThread thread) Agrega a group un objeto RequestThread.
public Vector	**getGroup**() Regresa un Vector que contiene todos los elementos pertenecientes a group.

public void	**joinGroup**() Espera a que termine la ejecución de cada objeto perteneciente a group.
public void	**remove**(String key) Elimina de group el objeto RequestThread según el identificador especificado.
public void	**startGroup**() Inicia la ejecución de cada objeto perteneciente a group.
public void	**stopGroup**() Método encargado de forzar que cada objeto perteneciente a group termine su ejecución.

Class RequestThread

wssi. RequestThread

public class **RequestThread**

Clase que determina la manera en que se realizará el proceso de generación de código correspondiente a cada aspecto.

Authors: Marisol Pérez Reséndiz.

See also: Thread

Attribute Summary	
private boolean	**broker** Indica si el proceso de cada objeto realiza consultas al servicio intermediario (broker).
private String	**document** String que especifica la ruta de un documento XML.

public Object[]	**info** Arreglo de objetos que almacena información de cada objeto.
private String	**key** Identificador único de cada objeto.
private String	**packageName** String que especifica el paquete al que pertenece una aplicación.
private String	**url** String que especifica el URI de un servicio Web o del servicio intermediario (broker) según sea el caso.
private boolean	**wait** Indica si el proceso de cada objeto se encuentra en espera de respuesta.

Association Summary	
private Synchronizer	**e** Objeto Synchronizer.
private Node	**messageElements** Objeto que almacena los elementos <message> correspondientes a un documento XML.
private ManagementThread	**mt** Objeto ManagementThread.
private NamedNodeMap	**pointCut** Objeto que almacena los atributos de un elemento <pointcut> correspondiente a un documento XML.

Constructor Summary	
public	**RequestThread**(String url, NamedNodeMap pointCut, String packageName, String document) Constructor que crea un objeto RequestThread.
public	**RequestThread**(Object[] info) Constructor que crea un objeto RequestThread.

public	**RequestThread**(String url, NamedNodeMap pointCut, String packageName, String document, Node messageElements) Constructor que crea un objeto RequestThread.

Method Summary	
public void	**analyzeWSDL**(String operation, WriteAspect wa) Analiza el documento WSDL especificado por url para poder generar el código correspondiente al aspecto que será integrado a una aplicación.
public void	**analyzeXML**(String operation, WriteAspect wa) Analiza el documento XML especificado por document e invoca el método inquiryProcess para realizar consultas al servicio intermediario (broker).
public Object[]	**getParams**(Vector partsName, Vector partsType) Mapea la descripción de los subelementos <part> asociados al elemento <input> correspondiente a una operación de un servicio Web a objetos String.
public String	**inquiryProcess**(String service, String operation, Object[] params, Vector partsName, String returnType) Proceso de consulta al servicio intermediario (broker) que tiene como propósito descubrir el servicio Web correspondiente con las especificaciones establecidas por los parámetros.
public String	**invokeService**(String operation, Vector data, String serviceBroker) Proceso de consulta al servicio intermediario (broker) que tiene como propósito
public void	**notifySerialization**(String uukey) Notifica al servicio intemediario (broker) que la información de un proceso ha sido serializada y por lo tanto su ejecución debe finalizar.
public void	**resumeWaitingThread**(String operation, WriteAspect wa) Ejecuta el método inquiryProcess de procesos que habían sido finalizados para serializar su información.
public void	**run**() Método encargado de determinar el tipo de proceso de generación de código que se llevará a cabo.

public String	**verifyType**(String type) Realiza la conversión del tipo de dato string definido para servicios Web al tipo de dato String definido en Java.
public void	**writeAspect**(WriteAspect wa, String wsdl, Object[] params, Vector partsName, String returnType) Determina si el código correspondiente a un aspecto debe ser generado.

Class Serialize

wssi. Serialize

public class **Serialize**

Clase que serializa la información de los procesos encargados de realizar la generación de código correspondiente a los aspectos.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
private static Hashtable	**h** Hashtable que almacena arreglos de objetos.

Association Summary	
public WSSIMain	**wSSIMain**

Constructor Summary	
	Serialize() Constructor que crea un objeto Serialize.
	Serialize(Hashtable h) Constructor que crea un objeto Serialize e inicializa h.

Method Summary	
public void	**add**(String key, Object[] obj) Agrega a h un arreglo de objetos.
public Object[]	**get**(Object key) Regresa y posteriormente elimina de h el objeto que corresponda con el identificador especificado.
public Hashtable	**load**() Recupera del archivo *threads.ser* el contenido de un objeto Hashtable previamente serializado
public void	**save**() Serializa el objeto Hashtable en el archivo *threads.ser*.

Class Synchronizer

wssi. Synchronizer

public class **Synchronizer**

Clase que tiene como objetivo garantizar la sincronización de los procesos tal que sólo un proceso se encuentre a la vez en una región crítica.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
int	**counter** Variable correspondiente al valor del semáforo.

Association Summary	
public RequestThread	**requestThread**

Constructor Summary	
	Synchronizer(int c) Constructor que crea un objeto Synchronizer e inicializa el valor del semáforo.

Method Summary	
public int	**cn**() Regresa el valor del semáforo.
public void	**dn**() Método que decrementa el valor del semáforo.
public void	**up**() Método que incrementa el valor del semáforo.

Class WriteAspect

wssi. WriteAspect

public class **WriteAspect**

Clase que tiene como objetivo crear el archivo de cada aspecto, así como todo el código referente al mismo.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
private BufferedWriter	**bw** Objeto BufferedWriter.
private FileWriter	**fw** Objeto FileWriter.
private String	**path** Ruta donde se localiza la aplicación.

Constructor Summary	
	WriteAspect(String path) Constructor que crea un objeto WriteAspect estableciendo el valor de path.

Method Summary	
public boolean	**aspectExists**(String file) Verifica si el archivo correspondiente a un aspecto ha sido creado.
public void	**createFile**(Stringñame, String package_val) Crea el archivo correspondiente a un aspecto.
public void	**execute**(String command) Ejecuta el comando especificado como un proceso independiente.
public void	**writeAround**(String returnType) Genera el código correspondiente al modificador de comportamiento *around*.
public void	**writeBATfile**() Crea el archivo *compile.bat* para la compilación de la aplicación.
public void	**writeBefore**(String operation, String wsdl, Vector partsName) Genera el código correspondiente al modificador de comportamiento *before*.
public void	**writeFile**(NamedNodeMap attr, String package_val, String wsdl, Object[] parts, Vector partsName, String returnType) Crea y genera el código de un aspecto que formará parte de la aplicación.
public void	**writeLSTfile**() Crea el archivo *appfiles.lst* que contiene el nombre de los archivos fuente de la aplicación.

Class WSDLInvoker

wssi. WSDLInvoker

public class **WSDLInvoker**

Clase encargada de realizar el análisis de documentos WSDL correspondientes a servicios Web, así como la invocación de los mismos a través de los métodos que proporciona WSIF.

Authors: César Sandoval Hernández, Marisol Pérez Reséndiz.

Attribute Summary	
String	**wsdlLocation** Ruta donde se localiza un documento WSDL.
String	**xmlLocation** Ruta donde se localiza un documento XML.

Association Summary	
Definition	**def** Objeto Definition.
public InvokeService	**invokeService**

Constructor Summary	
public	**WSDLInvoker**() Constructor que crea un objeto WSDLInvoker.

Method Summary	
public Vector	**WSDLoperations**() Obtiene la lista de todas las operaciones de un servicio Web
public NodeList	**getElements**(Document d, String tag) Obtiene los elementos descritos por tag de un documento WSDL.
public NodeList	**getElements**(Element e, String tag) Obtiene los subelementos descritos por tag de un elemento correspondiente a un documento WSDL.

public String	**getEncodingStyles**(Binding binding, String operationName) Obtiene la manera en que se encuentran codificados los datos de un mensaje SOAP asociados a una operación de un servicio Web.
public Vector	**getIOParts**(String typeDoc, String message) Obtiene el nombre de cada subelemento <part> asociados al elemento <input> o al elemento <output> asociados a una operación de un servicio Web.
public String	**getLocation**() Regrea la ruta de un documento WSDL.
public String	**getLocationURI**(String port) Obtiene el URI de un servicio Web.
public String	**getMessageIO**(String operation, String type) Obtiene el mensaje de entrada o salida asociado a una operación de un servicio Web.
private String	**getNamespace**(String opName, Map ns) Obtiene el espacio de nombres correspondiente a una operación de un servicio Web.
public String	**getNamespaceURI**(Binding binding, String operationName) Obtiene los espacios de nombres asociados a una operación de un servicio Web.
public Map	**getNamespaces**() Obtiene los espacios de nombres asociados a un servicio Web.
public Operation	**getOperation**(String name) Obtiene un objeto Operation que representa el subelemento <operation> que corresponda a name.
public List	**getOperations**() Obtiene la lista de todas las operaciones de un servicio Web
public String	**getPortName**() Obtiene el atributo name del elemento <port> de un documento WSDL asociado a una operación.
public Operation	**getPortOperation**(String operation, String portName) Obtiene un objeto Operation que representa el elemento <operation> de un documento WSDL.

public List	**getPortOperations**(String portName) Obtiene una lista de objetos que representan todos los elementos <operation> asociados al portName.
public Map	**getPortTypes**() Obtiene todos los elementos <portType> de un documento WSDL.
public String	**getServiceName**() Obtiene el nombre de un servicio Web.
public Port	**getServicePort**(String portName) Obtiene un objeto Port que representa el subelemento <port> de un documento WSDL.
public String	**getSoapAction**(Binding binding, String operationName) Obtiene el elemento <soap> de un documento WSDL.
public String	**getUse**(Binding binding, String operationName) Obtiene la manera en que se encuentran codificados los datos de entrada asociados a una operación de un servicio Web.
private String	**getVal**(String str, String val) Obtiene de un elemento <soap> representado como String el valor de su atributo style.
public Document	**readWSDL**() Realiza la lectura del documento WSDL especificado y genera un objeto Document que representa la estructura del documento.
public Document	**readXML**() Realiza la lectura del documento XML especificado y genera un objeto Document que representa la estructura del documento.
public boolean	**retrieveSignature**(String partType) Verifica si un tipo de dato es primitivo.
public SOAPMessage	**senderSOAP**(String url, String nameinputop, String namespaceinput, Vector nameparts, Vector datos, boolean isComplex, String complexName, String use, String soapAction) Forma el mensaje SOAP para realizar la invocación a una operación de un servicio Web.
public void	**setLocation**(String wsdlLocation) Mapea un documento WSDL a un objeto Definition.
public void	**setXMLlocation**(String xmlLocation) Establece la ruta donde se localiza un documento XML.

Class WSSIMain

wssi. WSSIMain

public class **WSSIMain**

Clase principal de WSSI que a través de la cual se accesa al sistema y se inicia el proceso de selección e integración de servicios Web.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
public static int	**id** Identificador de cada objeto RequestThread.

Association Summary	
public ParallelGroup	**group** Objeto ParallelGroup.
public Serialize	**ser** Objeto Serialize.

Constructor Summary	
	WSSIMain() Constructor que crea un objeto WSIMain.

Method Summary	
public static void	**main**(String[] args) Función principal de WSSI.
public static void	**threadsWaiting**(Hashtable h) Cada elemento perteneciente a h se obtiene para crear un nuevo objeto RequestThread y agregarlo a group.

Apéndice C

Paquete Intermediario (broker)

Class Summary	
BrokerBindingImpl	Clase que implementa la funcionalidad de cada operación del servicio **broker**.
BrokerBindingStub	BrokerBindingStub.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.
BrokerLocator	BrokerLocator.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.
DatabaseQuery	Clase que implementa funciones para la consulta a la base de datos **brokerdb**.
Md5	Clase auxiliar para la generación de un identificador universal único.
UUID	Clase que genera un identificador universal único.
WSDLInvoker	Clase encargada de realizar el análisis de documentos WSDL correspondientes a servicios Web, así como la invocación de los mismos a través de los métodos que proporciona WSIF.

Interface Summary	
Broker	Broker.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.
BrokerPortType	BrokerPortType.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.

Interface Broker

public interface **Broker**

Broker.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.

Method Summary	
public broker.BrokerPortType	**getbrokerPort**()
public broker.BrokerPortType	**getbrokerPort**(java.net.URL portAddress)
public String	**getbrokerPortAddress**()

Class BrokerBindingImpl

broker. BrokerBindingImpl

All known implemented Interfaces: BrokerPortType

public class **BrokerBindingImpl**

Clase que implementa la funcionalidad de cada operación del servicio **broker**. Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache

Axis WSDL2Java.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
public static Hashtable	**h** Hashtable que almacena los objetos que bloquean la ejecución de un proceso.

Method Summary	
public String	**askservice**(String operation, String inputparts, String outputparts) Operación encargada de consultar si existe el registro de un servicio Web solicitado.
public String	**asksubscription**(String uukey) Operación encargada de bloquear su ejecución hasta que un proveedor publique un servicio Web solicitado.
public String	**hashtableinfo**(String typeinfo) Operación que borra o proporciona información referente a las suscripciones realizadas al **broker**.
public String	**notifyserialization**(String uukey) Operación que notifica la serialización de una suscripción.
public String	**publish**(String wsdl) Operación que publica el registro de un servicio Web.
public void	**resume**(String uukey) Método auxiliar encargado de resumir un proceso bloqueado.
public String	**subscribe**(String operation, String inputparts, String outputparts) Operación encargada de generar una suscripción de un servicio Web solicitado.
public void	**suspend**(String uukey) Método auxiliar encargado de bloquear un proceso.
public String	**unpublish**(String wsdl) Operación que borra el registro de un servicio Web.
public String	**unsubscribe**(String uukey) Operación que borra el registro de una suscripción.

Class BrokerBindingStub

broker. BrokerBindingStub

All known implemented Interfaces: BrokerPortType

public class **BrokerBindingStub**

BrokerBindingStub.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.

Attribute Summary	
private java.util.Vector	**cachedDeserFactories**
private java.util.Vector	**cachedSerClasses**
private java.util.Vector	**cachedSerFactories**
private java.util.Vector	**cachedSerQNames**

Constructor Summary	
public	**BrokerBindingStub**()
public	**BrokerBindingStub**(java.net.URL endpointURL, javax.xml.rpc.Service service)
public	**BrokerBindingStub**(javax.xml.rpc.Service service)

Method Summary	
public String	**askservice**(String operation, String inputparts, String outputparts)

public String	**asksubscription**(String uukey)
private org.apache.axis.client.Call	**createCall**()
public String	**hashtableinfo**(String typeinfo)
public String	**notifyserialization**(String uukey)
public String	**publish**(String wsdl)
public String	**subscribe**(String operation, String inputparts, String outputparts)
public String	**unpublish**(String wsdl)
public String	**unsubscribe**(String uukey)

Class BrokerLocator

broker. BrokerLocator

All known implemented Interfaces: Broker

public class **BrokerLocator**

BrokerLocator.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.

Attribute Summary	
private String	**brokerPortWSDDServiceName**
private String	**brokerPort_address**

private java.util.HashSet	**ports**

Method Summary	
public java.rmi.Remote	**getPort**(Class serviceEndpointInterface)
public java.rmi.Remote	**getPort**(javax.xml.namespace.QName portName, Class serviceEndpointInterface)
public java.util.Iterator	**getPorts**()
public javax.xml.namespace.QName	**getServiceName**()
public broker.BrokerPortType	**getbrokerPort**()
public broker.BrokerPortType	**getbrokerPort**(java.net.URL portAddress)
public String	**getbrokerPortAddress**()
public String	**getbrokerPortWSDDServiceName**()
public void	**setbrokerPortWSDDServiceName** (String name)

Interface BrokerPortType

broker. BrokerPortType

public interface **BrokerPortType**

BrokerPortType.java Este archivo se ha generado automáticamente a partir de WSDL por el emisor de Apache Axis WSDL2Java.

Method Summary	
public String	**askservice**(String operation, String inputparts, String outputparts)
public String	**asksubscription**(String uukey)
public String	**hashtableinfo**(String typeinfo)
public String	**notifyserialization**(String uukey)
public String	**publish**(String wsdl)
public String	**subscribe**(String operation, String inputparts, String outputparts)
public String	**unpublish**(String wsdl)
public String	**unsubscribe**(String uukey)

Class DatabaseQuery

broker. DatabaseQuery

public class **DatabaseQuery**

Clase que implementa funciones para la consulta a la base de datos **brokerdb** correspondiente al directorio de servicios.

Authors: Marisol Pérez Reséndiz.

Attribute Summary	
String	**URL** Establece el URL de la base de datos.
String	**driver** Establece el tipo de conexión con la base de datos.

String	**name** Establece el usuario asociado a la base de datos.
String	**password** Establece el password asociado al usuario de la base de datos.

Association Summary	
public Connection	**con** Objeto Connection que establece una conexión con la base de datos.

Constructor Summary	
public	**DatabaseQuery**() Constructor que crea un objeto DatabaseQuery.

Method Summary	
public void	**connect**() Establece una conexión con la base de datos especificada por URL.
public void	**deleteSubscription**(String uukey) Borra de la base de datos el registro correspondiente a una suscripción asociada a un servicio Web.
public void	**disconnect**() Cierra una conexión con la base de datos previamente establecida.
public String	**getService**(String uukey) Obtiene la descripción de un servicio asociada al identificador de una suscripción.
public Vector	**getSubscriptions**(String service) Obtiene todas las suscripciones que correspondan con la descripción del servicio Web especificado.
public String	**getWSDL**(String service) Consulta si existe algún registro de un servicio Web que corresponda con la descripción especificada.

public boolean	**publishService**(String service, String wsdl) Agrega un registro a la base de datos correspondiente a la publicación de un servicio Web.
public void	**subscribe**(String uukey, String service) Agrega un registro de una nueva suscripción a la base de datos.
public void	**unpublishService**(String wsdl) Borra de la base de datos el registro correspondiente a un servicio Web.
public void	**updateSubscription**(String uukey, String serialize) Actualiza el campo serialize de un registro asociado a una suscripción con el valor de serialize.
public boolean	**verify**(String sql) Realiza una consulta a la base de datos.

Class WSDLInvoker

broker. WSDLInvoker

public class **WSDLInvoker**

Clase encargada de realizar el análisis de documentos WSDL correspondientes a servicios Web, así como la invocación de los mismos a través de los métodos que proporciona WSIF.

Authors: César Sandoval Hernández, Marisol Pérez Reséndiz.

Nota: La descripción de esta clase corresponde a la clase descrita en el Apéndice B.

Apéndice D

Glosario

1. SOAP. Simple Object Access Protocol es un protocolo que provee los mecanismos para la ejecución de llamadas a procedimientos remotos entre programas, de tal manera que se puedan establecer de manera eficiente las comunicaciones [7].
2. SQL. Structure Query Language es un estándar en el lenguaje de acceso a bases de datos que actualmente está adoptado por ISO.
3. UDDI. Universal Description, Discovery and Integration tiene el objetivo de definir un registro y protocolos asociados para la búsqueda y localización de servicios Web. Básicamente, provee servicios de consulta y publicación. El primero de ellos permite realizar búsquedas sobre negocios y servicios publicados. El segundo tiene el propósito de registrar, modificar y eliminar dichos negocios y servicios [11].
4. UUID. Universal Unique Identifier es un valor de 128 bits utilizado para la identificación, por lo que ningún otro objeto, tipo o interfaz puede hacer uso de un UUID que ya ha sido asignado [29].
5. W3C. World Wide Web Consortium es la organización apadrinada por el MIT y el CERN, entre otros, cuyo cometido es el establecimiento de los estándares relacionados con WWW [31].
6. WSIF. Web Services Invocation Framework es una API de Java que proporciona la funcionalidad para invocar servicios Web no importando cómo o dónde se encuentran los mismos [26].

7. WSDL. Web Services Description Language es una plantilla o interfaz que permite a las aplicaciones describir a otras aplicaciones las reglas para interactuar entre sí [9].

8. WSDL4J. Es una implementación de IBM para el lenguaje Java, que facilita la creación, representación y manipulación de documentos WSDL.

9. WSDK. IBM WebSphere SDK for Web Services provee una plataforma y herramientas de soporte para crear de manera simple y directa sistemas basados en servicios Web [28].

10. XML. Extensible Markup Language es un meta-lenguaje creado por el W3C, el cual permite definir lenguajes de marcado adecuados a usos determinados. Corresponde a un estándar que permite a diferentes aplicaciones interactuar con facilidad a través de la red.

11. AspectJ. Extensión al lenguaje Java para el desarrollo de aplicaciones orientadas a aspectos. Este lenguaje define unidades modulares de asuntos llamadas aspectos. Establece palabras reservadas y construcciones específicas para definir aspectos, puntos de corte y modificadores de comportamiento [21].

12. Aspecto. Módulo reutilizable que define uno o varios puntos de corte y las acciones que deben ejecutarse cuando en la ejecución de un programa se alcance cualquiera de ellos [12] [13].

13. Llamada a procedimiento remoto (RPC). Es un protocolo que se utiliza para solicitar un servicio localizado en una computadora remota a través de la red. RPC utiliza el modelo cliente-servidor, donde el cliente solicita el servicio que un servidor provee.

14. Modificador de comportamiento (advice). Código que se ejecuta antes, durante o después un punto de unión [3] [14].

15. Programación Orientada a Aspectos. Enfoque de programación que se basa en la idea de estructurar y programar los sistemas eficientemente, con el propósito de modularizar aquellos requerimientos que afectan múltiples módulos de un sistema [3] [14].

16. Protocolo. Corresponde a un conjunto de normas y/o procedimientos para la transmisión de datos que tiene que ser observado por los dos extremos de un proceso de comunicación entre emisor y receptor.

17. Punto de corte (pointcut). Declaración de un conjunto de puntos de unión [3] [14].

18. Punto de unión (join point). Puntos bien definidos en la ejecución de un programa que identifican eventos que suceden sobre dichos puntos [3] [14].

Bibliografía

[1] V. B. and C. M. A., "AOP for Dynamic Configuration and Management of Web Services," *International Journal on Web Services Research (JWSR)*, vol. 1, no. 3, 2004.

[2] C. M. A. and V. B., "Modularizing Web Services Management with AOP," System and Software Engineering Lab, Vrije Universiteit Brussel, Tech. Rep., 2003.

[3] R. Bodkin and R. Laddad, "Zen and the art of aspect-oriented programming," *Linux Magazine*, 2004.

[4] D. Chappell and T. Jewell, *Java Web Services*, 1st ed. O'Reilly, 2002.

[5] F. Curbera, M. Duftler, R. Khalaf, W.Ñagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to soap, wsdl, and uddi," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86–93, 2002.

[6] D. Booth, H. Haas, F. McCabe, and et. al. (2004) Web services architecture. W3C Working Group. [Online]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[7] M. Gudgin, M. Hadley, N. Mendelsohn, and et. al. (2003) Soap version 1.2. W3C Working Group. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[8] DeveloperWorks, "Introduction to Web services and the WSDK," IBM, Tech. Rep.

[9] R. Chinnini and et. al. (2004) Web Services Description Language (WSDL) Specification. W3C Working Group. [Online]. Available: http://www.w3.org/TR/wsdl20/

[10] O. W. Services and E. Support, "Web Services Overview," Oracle Corporation, 2002.

[11] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, and et. al., "UDDI Version 3.0 Published Specification," Tech. Rep., 2002.

[12] T. Elrad, R. E. Filman, and A. B. (editores), "Aspect-Oriented Programming," *Special Section Of Communications of the ACM*, vol. 44, no. 10, pp. 29–32, 2001.

[13] T. Elrad, M. Aksit, G. Kiczales, K. Lieberher, and H. Ossher, "Disscussing Aspects of AOP," *Special Section Of Communications of the ACM*, vol. 44, no. 10, pp. 33–38, 2001.

[14] F. Beltagui, "Features and Aspects: Exploring feature-oriented and aspect-oriented programming interactions," Computing Department, Lancaster University, Lancaster, Tech. Rep., 2003.

[15] R. Laddad, "Separate software concerns with aspect-oriented programming," 2002.

[16] J. Bonér and A. Vasseur. AspectWerkz. [Online]. Available: http://aspectwerkz.codehaus.org/index.html

[17] B. Burke. JBoss Aspect Oriented Programming. JBoss. [Online]. Available: http://www.jboss.org/products/aop

[18] R. Johnson and et. al. Spring - Java/J2EEE Application Framework. [Online]. Available: http://www.springframework.org/docs/reference/index.html

[19] D. Suvee, W. Vanderperren, and V. Jonckers, "JAsCo: An Aspect-Oriented approach tailored for Component Based Software Development," in *Proceedings of the 2nd. International Conference on AOSD*, 2003.

[20] A. Colyer. AspectJ project. Eclipse. [Online]. Available: http://eclipse.org/aspectj/

[21] R. Laddad, *AspectJ in Action*. Manning Publications Co., 2003.

[22] J. Parra, S. Sánchez, O. Sanjuán, and L. Joyanes, "RAWS: Reflective Engineering for Web Services," in *Proceedings of the IEEE International Conference on Web Services*, Madrid, Spain, 2004.

[23] A. Charfi and M. Mezini, "Aspect-Oriented Web Service Composition with AO4BPEL," in *Proceedings of European Conference on Web Services ECOWS 2004*, ser. LNCS, vol. 3250, 2004.

[24] B. Verheecke and M. A. Cibrán, "AOP for Dynamic Configuration and Management of Web Services," *International Journal on Web Services Research (JWSR)*, vol. 1, no. 3, 2004.

[25] G. S. Swint and C. Pu, "Code Generation for WSLAs using AXpect," in *Proceedings of the IEEE International Conference on Web Services (ICWS)*, 2004.

[26] M. J. Duftler, N. K. Mukhi, A. Slominski, and S. Weerawarana, "Web Services Invocation Framework (WSIF)," IBM T.J. Watson Research Center, Tech. Rep., 2001.

[27] C. S. Hernández, "Generación dinámica de workflows organizacionales escritos en BPEL4WS," Tesis de Maestría, CINVESTAV-IPN, noviembre 2004.

[28] IBM. "IBM WebSphere SDK for Web Services (WSDK) Version 5.0.1". [Online]. Available: http://www-106.ibm.com/developerworks/webservices/wsdk/

[29] B. Mahé. (2000) UUID.java. [Online]. Available: http://dev.w3.org/cvsweb/java/classes/org/w3c/util/UUID.java

[30] W3C. (1996) Md5.java. [Online]. Available: http://dev.w3.org/cvsweb/java/classes/w3c/tools/crypt/Attic/Md5.java

[31] World Wide Web Consortium. [Online]. Available: http://www.w3.org/

[32] M. Pérez and O. Olmedo, "Dynamic invocation of Web services by using aspect-oriented programming," in *Proceedings of the International Conference on Electrical and Electronics Engineering and Conference on Electrical Engineering (ICEEE-CIE)*, september 2005.

[33] N. Cova and O. Olmedo, "Aspect oriented Web services orchestration," in *Proceedings of the International Conference on Electrical and Electronics Engineering and Conference on Electrical Engineering (ICEEE-CIE)*, september 2005.

[34] N. C. Suazo, "Orquestación de servicios Web orientada a aspectos," Tesis de Maestría, CINVESTAV-IPN, octubre 2005.

Cada final es un nuevo comienzo...