UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

ARQUITECTURA WEB ORIENTADA A SERVICIOS: INTEROPERABILIDAD CON UN MÓDULO DE EVALUACIÓN PARA UN SISTEMA DE ENSEÑANZA EN LÍNEA

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

JESÚS SAIDEL LÓPEZ HERNÁNDEZ

JORGE LUIS ORTEGA ARJONA

2008

Autorizo a la Dirección General de Bibliotecas de la UNAM a difundir en formato electrónico e impreso el contenido de mi trabajo recepcional.

NOMBRE: JESÚS SAIDEL LÓPEZ HERNÁNDEZ

FECHA: 3 DE NOVIEMBRE DE 2008

FIRMA: Saidel López H

Datos del Jurado

1. Datos del alumno	
Apellido Paterno	López
Apellido Materno	Hernández
Nombre(s)	Jesús Saidel
Teléfono	55 67 06 41
Universidad	Universidad Nacional Autónoma de México
Escuela o Facultad	Facultad de Ciencias
Carrera	Ciencias de la Computación
Número de cuenta	098325755
2. Datos de tutor	
Grado	Dr
Nombre(s)	Jorge Luis
Apellido Paterno	Ortega
Apellido Materno	Arjona
3. Datos de sinodal 1	
Grado	Dr.
Nombre(s)	Sergio
Apellido Paterno	Rajsbaum
Apellido Materno	Godoresky
3. Datos de sinodal 2	
Grado	Dr.
Nombre(s)	Héctor
Apellido Paterno	Benítez
Apellido Materno	Pérez
3. Datos de sinodal 3	
Grado	Mat.
Nombre(s)	Salvador
Apellido Paterno	López
Apellido Materno	Mendoza
3. Datos de sinodal 4	
Grado	M. en C.
Nombre(s)	Gustavo Arturo
Apellido Paterno	Márquez
Apellido Materno	Flores
3. Datos del trabajo escrito	
Título	Arquitectura Web Orientada a Servicios
Subtítulo	Interoperabilidad con un módulo de enseñanza en línea
Número de páginas	145 p.
Año	2008

Arquitectura Web Orientada a Servicios

Interoperabilidad con un Módulo de Evaluación para un Sistema de Enseñanza en Línea

Jesús Saidel López Hernández
Facultad de Ciencias, UNAM

12 de octubre de 2008

Índice general

Índice de figuras

Capítulo 1

Introducción

Una Arquitectura Web Orientada a Servicios (*Service Oriented Web Architecture* o simplemente SOWA) es una descripción para la creación de sistemas de *software* en Internet, que sigue la filosofía de satisfacer las necesidades de programación mediante el uso, publicación y búsqueda de servicios remotos. De tal manera, un programador únicamente requiere buscar el servicio remoto que cumpla con sus necesidades para poder usarlo para el desarrollo de su sistema de *software*.

Utilizando una SOWA, el programador no tiene que entender toda la documentación y estructura de los servicios remotos. El programador requiere entender la funcionalidad de un servicio remoto para poder hacer uso del mismo, implementando un medio de comunicación entre su sistema y el servicio, dando lugar a una aplicación distribuida. El sistema de *software* se constituye entonces como la suma de funciones locales y los servicios remotos que lo conforman. Los servicios remotos pueden ser implementados en cualquier lenguaje de programación y ubicarse en cualquier parte de la red. La idea es la construcción de aplicaciones a partir de servicios remotos previamente desarrollados, modificándolos y adaptándolos a las necesidades locales, y canalizando los esfuerzos del programador en conocer de qué manera puede manejar los servicios remotos disponibles, así como de la comunicación entre estos servicios y su sistema.

La presente tesis desarrolla un caso de estudio de una SOWA para la evaluación del desempeño de los estudiantes en un centro educativo. Se trata de un sistema de software independiente dedicado a la evaluación, y al mismo tiempo, se trata de un proyecto Web orientado a servicios. Sin embargo,

nótese que el término "evaluación" puede extenderse más allá de un proyecto educativo. También puede integrarse al nivel que considere el programador y poder usarlo para calificar cualquier otro tipo de entidad, como productos en una tienda virtual, calidad de atención en un establecimiento, etc.

1.1. Contexto

El entorno Web existe para atender la necesidad de comunicación por medio de Internet. La infraestructura de hardware detrás de dicha comunicación mejora a cada momento, permitiendo la existencia de sistemas Web que explotan esas capacidades. Mejoras tecnológicas en los cables de fibra óptica, módems de mayor capacidad, incremento en número de torres de transmisión y repetidoras, así como el intercambio de datos por medios inalámbricos, son ejemplos de este progreso.

La creación de software en Web está restringida por la tecnología de *hardware* existente y el *software* nativo asociado a ese *hardware* [12]. Es por ello que se deben explotar todas estas limitaciones con el objetivo de desarrollar sistemas Web a la altura del hardware. En la actualidad los sistemas Web muestran contenido cada vez más complejo. Comparándolos con los contenidos disponibles hace diez años, los sistemas Web actuales tienen mejoras en la muestra de videos, animaciones interactivas, multimedia e intercambio de archivos desde cualquier punto en la red. Esto representa mejoras en la experiencia multimedia y en nuevas formas de interacción con las computadoras.

Los sistemas Web pueden estar enfocados a ciertos propósitos específicos. Sistemas de hardware, como *notebooks, teléfonos celulares, sidekicks, localizadores GPS o sistemas de cámaras en circuito cerrado*, usan aplicaciones a través de la red con fines específicos. El número de actividades cotidianas llevadas al entorno Web se incrementa con el tiempo, por lo que la fusión de *hardware* y *software* provee las acciones necesarias para conseguirlo.

Las acciones dadas por *hardware* y *software* son servicios útiles para realizar las actividades diarias. Cuando estos servicios son modificados, se da lugar a la creación de nuevos servicios: dependiendo de una necesidad en específico, los servicios pueden cambiarse para responder a esta necesidad. Cuando los sistemas Web son vistos como un conjunto de servicios, las posibles configuraciones, dadas por la modificación de cada uno de los servicios,

se convierten en un conjunto de soluciones a necesidades aún no previstas. Si bien se trata de un mismo sistema de software, su comportamiento se adapta a necesidades previstas.

Finalmente, los servicios pueden comunicarse con otros sistemas Web usando Internet como medio. Con esto, se puede integrar la funcionalidad del servicio al sistema Web que lo requiera.

1.2. Problema

Cuando la comunicación de dos sistemas de *software* ocurre por medio de la red, existe la llamada interoperabilidad entre sistemas. Interoperabilidad se define como "*la capacidad de intercomunicar, ejecutar programas o transferir datos entre varias unidades funcionales de manera que el usuario requiere poco o ningún conocimiento de las características únicas de estas unidades*" [23]. Así pues, los sistemas construídos usando SOWA y los servicios involucrados son las unidades funcionales; el conocimiento mínimo está dado por las descripción de los servicios remotos presentes en la red; finalmente, con lo anterior, es posible intercomunicar, ejecutar servicios remotos y intercambiar datos por medio de la red.

La interoperabilidad se ve afectada por las siguientes variables:

- **Serialización de datos:** Es una forma de transmitir datos por medio de la red. La acción de **serializar** se trata de un proceso de empaquetar una referencia al dato a transmitir. Una vez trasmitido este paquete, el receptor desempaqueta la referencia y reconstruye el dato en un proceso de **deserialización** [12]. Los diferentes lenguajes de programación tienen su propia forma de entender la serialización y deserialización, por ejemplo, transmitir un objeto `String` de Java debería ser entendido en el lenguaje C como un arreglo de tipo `char`. Sin embargo, cuando se intenta serializar otras estructuras como `Map` de Java, el proceso de deserialización podría no tener su similar en otros lenguajes [26]. La interoperabilidad se afecta cuando los datos intercambiados no son comprendidos por los receptores.
- **Sincronización de mensajes:** Es la coordinación de eventos que ocurren entre dos elementos con respecto al tiempo [16]. En el caso de

la interoperabilidad entre computadoras, el envío y recepción de datos debe seguir un orden. Si una computadora envía un dato y espera hasta recibir respuesta del receptor, se le llama *comunicación con mensajes síncronos* [31]. En contraste, los *mensajes asíncronos* permiten que las computadoras envíen sus mensajes sin necesidad de esperar respuesta del receptor [31]. Si las computadoras involucradas no tienen la capacidad de soportar el intercambio de mensajes síncronos o asíncronos, según sea el caso, la interoperabilidad no se da [31].

1.3. Hipótesis

La presente tesis pretende dar respuesta a la siguiente pregunta:

> *¿Cuáles son las condiciones de un entorno de programación Web bajo las cuales se obtiene una interoperabilidad entre aplicaciones, siendo estas descritas en términos de una Arquitectura Web Orientada a Servicios (SOWA)?*

A pesar de que la combinación de SOWA y WS promete un intercambio de datos entre aplicaciones implementados en diferentes lenguajes de programación, debe analizarse las razones por las que esto ocurre. Si se pueden abstraer características en común de algunos lenguajes que sean indispensables para conseguir una interoperabilidad, es posible generalizar las condiciones bajo las cuales dos implementaciones puedan interoperar en la Web.

1.4. Aproximación: SOWA y *Web Services*

La formalización del concepto de "servicio" como unidad funcional en una arquitectura para construir sistemas en Web pretende dar solucion a los problemas de interoperabilidad. De esta manera se comprende a un sistema Web como la suma de acciones de hardware y software que lo conforman como "servicios locales". Así pues, la extensión de esta idea es la exposición de los servicios locales a través de la red como "servicios remotos".

La SOWA es ejemplo de esta descripción, y se caracteriza por comunicar a un servidor (o proveedor del servicio remoto) con un cliente (o consumidor de servicios remotos). Además existe una fase de exposición y descubrimiento de servicios remotos para su fácil localización y búsqueda.

El uso de *Web Services* (servicios Web o símplemente WS) definen el uso de la SOWA, proponiendo mecanismos específicos para implementar las fases de la arquitectura. Con WS, se estandariza la comunicación entre sistemas por medio de un lenguaje en común: XML [33]. Así pues, se ataca el problema de serialización pues cada lenguaje de programación está encargado de comprender un mensaje en XML y no las estructuras complejas que podrían enviarse durante la comunicación. De esta manera, los sistemas Web del cliente y servidor pueden estar implementados en cualquier lenguaje de programación que comprenda y manipule mensajes en formato XML. Además, WS propone el envío de mensajes a través del protocolo HTTP, que atraviesa de manera segura los mecanismos mínimos de seguridad de las computadoras en red [33]. Con esto, se asegura que las computadoras tengan la capacidad de mandar y recibir mensajes síncronos [31] atacando el problema de sincronización de mensajes.

A pesar de que los WS permiten el intercambio de datos usando cualquier protocolo además de HTTP, la propuesta en esta tesis es la combinación de soluciones que trate de minimizar los problemas de interoperabilidad de sistemas Web y servicios remotos.

1.5. Contribuciones

Toda arquitectura consta de tres características. La primera delimita un sistema con sus componentes y conectores, así como el contexto en el cual residen. La segunda establece un objetivo, o una meta alcanzable con ese sistema. La tercera abstrae características funcionales o estructurales que permitan alcanzar esa meta para sistemas similares al que se estudia.

En el caso de la SOWA, se delimita el sistema con componentes como cliente, servidor y el contexto de la Web. El objetivo de esta tesis, así como de la arquitectura orientada a servicios, es la interoperabilidad entre sistemas usando diferentes características de la implementación. Finalmente, las características funcionales abstraídas son la descripción necesaria para conseguir interoperabilidad entre sistemas.

La mayor contribución de esta tesis es proveer una experiencia documentada tras probar una arquitectura Web orientada a servicios, mostrando ventajas y desventajas de la implementación de un servicio remoto y su consumo a partir de diferentes lenguajes de programación. Los resultados

de estos experimentos representan un conjunto de alternativas a problemas comunes que promueven el desarrollo de sistemas Web basados en servicios remotos, con el fin de hacer interoperables aplicaciones Web.

1.6. Organización de la tesis

El **capítulo 2** presenta los antecedentes de la SOWA y los WS. Además de los conceptos de servicio remoto y de la orientación a servicios, se muestra una visión de algunos patrones de *software* utilizados en esta tesis: *Model-View-Controller, Façade, Decorator, Proxy y Dependency Injection*[1].

El **capítulo 3** describe trabajos relacionados con la SOWA. Se presenta la forma en la que empresas de *software* ven a la orientación a servicios. En particular, se analizan la *Service Oriented Network Architecture* de Oracle y la *Service Oriented Modeling and Architecture* de IBM.

El **capítulo 4** desarrolla la vista general de un módulo de *software* aplicado a un caso de estudio. A partir de ciertos modelos gráficos se analizan la construcción y el comportamiento de este módulo, desde el punto de vista de la SOWA y de la exposición y consumo de servicios.

En el **capítulo 5** se realizan pruebas de interoperabilidad consumiendo y exponiendo servicios remotos en el caso de estudio. Estas acciones ocurren indistintamente utilizando diferentes implementaciones como AJAX, Java, .NET y PHP.

El **capítulo 6** presenta las conclusiones finales de este trabajo. Se analizan los resultados de las pruebas de interoperabilidad entre sistemas Web, y se comparan con los otros métodos y arquitecturas descritos en el capítulo 3 de trabajos relacionados.

Finalmente se incluye un **Apéndice**, donde se muestra una breve visión de las herramientas de software usadas en el desarrollo del caso de estudio que van desde diagramadores para ayudar en el diseño, hasta generadores de código y archivos de configuración para ayudar en la implementación.

[1] También llamado *Inversion of Control* o IoC por tecnologías como Spring.

Capítulo 2

Antecedentes

En este capítulo se establecen los antecedentes de la SOWA y los WS. En la primer sección, se muestran arquitecturas de sistemas como Cliente-Servidor y el Cómputo Distribuído y su aplicación en Web. De esta manera se llega a los conceptos de SOWA y WS en la segunda sección. Finalmente, se mencionan algunos patrones de software usados en la construcción del caso de estudio. Se presentan los patrones de *Model View Controller, Façade, Decorator, Dependency Injection* y *Proxy*.

2.1. Antecedentes de la SOWA

"*Internet* es una colección de redes" [25].

Las **redes de computadoras** sirven para intercambiar datos entre computadoras. Esta comunicación ha incrementado sus capacidades a lo largo de los años. A partir de redes locales donde se compartían archivos o acceso a dispositivos de *hardware*, las redes se extendieron hasta formar redes globales, uniendo organizaciones completas hasta finalmente extenderse al mundo entero [25]. **Internet** es, entonces, una colección de redes locales y de amplia cobertura, con el objetivo principal de compartir información y acceso a recursos computacionales. Para lograr esto, es necesaria una regulación, una forma en la cual las computadoras sepan dónde encontrar los recursos y cómo pueden llegar a ellos. Descripciones acerca de cómo ocurre esto son variadas, siendo la más común el esquema de *Cliente-Servidor*.

2.1.1. Arquitectura de Internet: Cliente-Servidor

Un **cliente** es una computadora que necesita de algun recurso, se encuentra conectada a una red de computadoras y puede establecer comunicación con ciertas computadoras en esa red[1]. Esta comunicación es, en esencia, una **petición electrónica** del recurso, que usualmente va acompañada de datos de entrada o archivos a ser procesados.

Un **servidor** es, por otro lado, una computadora a cargo de recursos, que cuenta con dos características principales:

- Tiene la capacidad de administrar las peticiones de los clientes, decidiendo quién tiene acceso al recurso en cuestión, así como el orden en el que lo obtendría. Debe ser capaz de esperar por un tiempo indefinido por peticiones de los clientes, y debe soportar el acceso a cierto número de clientes esperando a resolver su petición.

- Toma una segunda decisión: quién procesará la información. Una vez que una petición es hecha, el servidor puede tomar los datos de entrada y realizar los procesos pertinentes. Un ejemplo es el acceso a una impresora: los datos son mandados desde el cliente, y el servidor procesa esos datos para imprimirlos. Por otro lado, el servidor puede tomar los datos y devolver código ejecutable al cliente, para que éste realice los procesos pertinentes. Ejemplos de esta acción está en el cómputo distribuído.

Debe considerarse al servidor como un acceso a recursos de *software*, *hardware* o datos, que permiten satisfacer una necesidad, así como el encargado de la decisión de quién procesará los datos del cliente. La Figura 2.1 muestra la interacción entre cliente y servidor.

2.1.2. Arquitectura de Internet: Cómputo Distribuído

El **cómputo distribuído** es una forma de describir la comunicación de computadoras a través de redes [12]. La extensión de esta arquitectura al entorno de Internet es un paso lógico. La ejecución de programas con el cómputo distribuído es dependiente de las redes de computadoras, pero independiente del ambiente de Internet. Cabe mencionar que las ideas del cómputo distribuído existen desde mucho tiempo antes de la existencia de

[1]Sólo con aquellas computadoras con quien esté autorizado comunicarse, y que tengan acceso al recurso en cuestión.

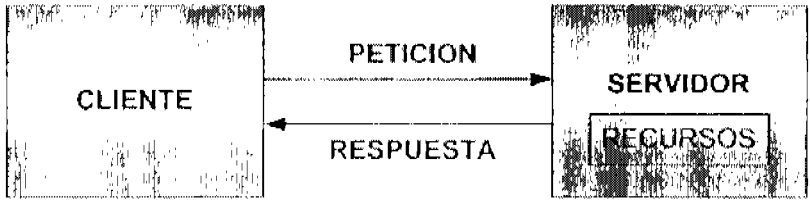

Figura 2.1: Diagrama de bloques de una arquitectura Cliente-Servidor

Internet, pero sólo en la actualidad puede ser explotado desde el enfoque de Internet para la creación y estudio de nuevas arquitecturas.

Desde el principio de la computación se tenía en mente la colaboración entre sistemas de cómputo para la resolución de problemas muy complicados al ser atendidos por una sola computadora. La idea parece simple: si un elemento puede hacer un trabajo en un tiempo definido, entonces incrementando el número de elementos es posible hacer el mismo trabajo en menos tiempo. Para ello es necesario tener en cuenta algunas consideraciones. En principio se deben conocer de antemano las labores que las computadoras pueden hacer y sobre todo si son mejores en realizar esa labor. Esto se logra siguiendo ciertos estándares predefinidos [12]. Además se debe conocer el orden en que las acciones se llevan a cabo para mejorar en calidad o cantidad de trabajo con ayuda de un planificador conocido también como **scheduler** [12].

Finalmente debe existir una computadora encargada de que las labores se lleven a cabo y determinar que estén divididas de forma justa entre las computadoras. Esta afirmación no implica que la división de las labores sea equitativa. Existen computadoras que pueden hacer más labores, por lo que deben ser consideradas en función de su desempeño. Esta computadora hace la misma función del servidor de la arquitectura Cliente-Servidor.

Dado un problema, computacionalmente hablando, es posible dividirlo en varios subproblemas lo más independientes uno del otro como sea posible, con el objetivo de ser procesados por sistemas de manera simultánea y disminuir los tiempos de proceso. Los procesadores de cada computadora

tienen a su cargo labores al nivel de sus capacidades. Esto es importante, en el sentido de que muchas computadoras de poca capacidad de proceso pueden hacer las mismas labores que hace una sola computadora con capacidades superiores.

El cómputo distribuido también está conformado por los siguientes elementos [12]:

- Los **procesos** son programas de computación en ejecución. Un sistema operativo puede tener varios procesos en ejecución al mismo tiempo. Los procesos sirven para describir una serie de pasos en un lenguaje de programación definido y que pueden ser ejecutados por el sistema operativo. Cuando esto ocurre, los procesos usan ciertos recursos de la computadora, como tiempo del procesador o dispositivos de entrada y salida de datos.

- Los **hilos de ejecución** (*threads*) están asignados a cada proceso en un sistema operativo. Los procesos pueden tener más de un hilo de ejecución para controlarlos, y son independientes uno de otro.

- El **planificador** (*scheduler*) controla estas acciones porque algunos hilos están a cargo de dispositivos de entrada y salida de datos, mientras que otros hilos están a cargo de establecer tiempos para ser procesados en el sistema. De esta manera, cualquier proceso que requiera de varios recursos del sistema se le asigna el número de hilos de ejecución necesario para mantenerlo independiente de los demás procesos. Así, se da la impresión de que todos los procesos son ejecutados al mismo tiempo, o al menos, con la idea de hacer eficiente a la computadora con respecto al tiempo usado y los resultados esperados. En un ambiente distribuido que incluya más de una computadora debe tenerse pleno control de tales cosas. Es decir, dado un proceso, es importante saber dividir los subprocesos y mantener la comunicación persistente entre ellos. Solo por mencionar un ejemplo, los hilos de ejecución que controlan la salida de datos deben estar sincronizados con la entrada de datos en el otro sistema.

La Figura 2.2 muestra la estructura general de este modelo de cómputo distribuído.

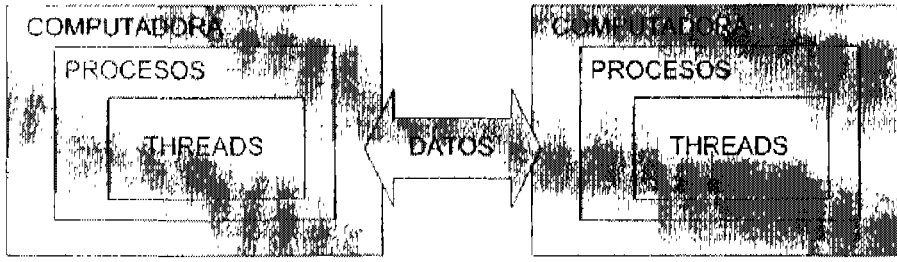

Figura 2.2: Diagrama de bloques de la estructura del cómputo distribuido

Ventajas del cómputo distribuído [12]

1. Costo económico. Con el presupuesto necesario para comprar una computadora de alto desempeño, es posible crear una red de computadoras más económicas que hagan la misma labor en el mismo tiempo.

2. Desempeño. Grandes cantidades de información y datos pueden ser controlados o administrados. Es decir, cuando un proceso necesita de ciertos datos de entrada, es necesario que los pida al servidor, sin que los otros procesos se afecten. La independencia de procesos implica una independencia de los datos que están procesando.

3. Tolerancia a fallos. Si una computadora falla (mientras no sea el servidor) entonces el trabajo aun puede terminarse. Dado que el servidor funge como un supervisor, entonces nota que una computadora no está mandando datos de salida por lo que se hace una reasignación de labores entre las computadoras.

Desventajas del cómputo distribuído [12]

1. Si falla el servidor se puede perder toda la información procesada y el avance conseguido por las demás computadoras.

2. La asignación de tareas en un ambiente distribuido es una tarea complicada. Es necesario saber qué computadoras del sistema distribuído son capaces de realizar qué labores.

3. Debe hacerse un análisis del problema a resolver. Si la solución no necesita de un cómputo distribuido, entonces su aplicación puede ser contraproducente, pues se pueden saturar los medios de comunicación, retrasando los tiempos de respuesta. La heurística para considerar su uso, debe ser la velocidad del procesador en la computadora en que se ejecuta la aplicación. Si ésta cumple con los requerimientos de hardware para realizar el proceso por si mismo, en un tiempo dado, entonces no se debe usar el cómputo distribuido. Por otro lado, si los resultados de ejecutar una aplicación no satisfacen en términos de tiempo, entonces se busca un medio de comunicación confiable, que mantenga la información y los datos persistentes en su viaje de una computadora a otra. Además, el uso de los procesadores involucrados deben brindar una mejora con respecto a la primera opción, entonces se puede implementar un ambiente distribuido. Una mala elección para usar el cómputo distribuído, dado un problema, puede significar mayores pérdidas de tiempo.

4. Un sistema distribuido debe garantizar que no hay pérdida de datos y mantener canales seguros para su transmisión. Si existe un problema de seguridad, es posible que existan datos de salida incoherentes o que no permitan el libre proceso de las demás computadoras. Esto mismo ocurre si el medio físico de comunicación, como los cables de transmisión de datos, son poco confiables, provocando retardos en el paso de la información.

5. Existe un problema general con respecto a procesos concurrentes. Cuando los procesos necesitan de un dato en específico, lo buscan del servidor. Aqui pueden ocurrir dos cosas: la primera es que obtenga el dato pero modificado anteriormente por otro proceso. La segunda es que no haya sido cambiado. Como el planificador controla el orden en que ocurren los procesos, entonces no se puede conocer el estado de esos datos en un tiempo definido. Por ello es que otras áreas de la computación como bases de datos y sistemas operativos tienen problemas de concurrencia. Sólo con una buena planeación y análisis del problema pueden minimizar la situación, más no erradicarla totalmente.

2.1.3. Web, Aplicaciones Web y Sistemas Web

Web es la forma de designar a una red de computadoras que intercambian datos entre sí por medio de Internet usando la arquitectura cliente-servidor. El servidor tiene a su control ciertos recursos que quieren usar

los clientes. Por tanto, necesita haber un medio de comunicación capaz de permitir las peticiones y una forma de controlarlas. Los recursos a cargo del servidor pueden ser *hardware, software*, datos o información, que sirvan al usuario de la computadora cliente. Cuando se trata de información en la Web, es necesario que el cliente interprete todos los datos que ofrece el servidor para que sean mostrados de una manera que el usuario de la computadora cliente pueda comprender. Para ello existen programas de software llamados navegadores (**browsers**) capaces de realizar esta interpretación. La información llega en forma de un *lenguaje común de marcas* como *Hipertext Markup Language* (HTML) o *Extended Markup Language* (XML). Los navegadores toman la información, e interpretan las marcas en el lenguaje para organizar la información contenida entre las mismas. El resultado es la vista de la información como en las páginas de Internet comunes en estos días.

Una **aplicación Web** es un programa de software capaz de funcionar en el entorno de la Web. Un **sistema Web** es la recopilación de una o varias aplicaciones Web con la finalidad de satisfacer una necesidad empresarial, educativa o de entretenimiento por medio de Internet. Los sistemas Web pueden ser vistos en cualquier momento al acceder a un sitio de Internet. Toda la funcionalidad que provee el sitio Web se trata de un conjunto de aplicaciones Web, ocurriendo y esperando por los datos de entrada necesarios para ejecutarse. Cuando las páginas visitadas son *estáticas*, la información contenida en ellas no cambia, por lo que los procesos involucrados en la aplicación Web sólo son para mostrar el contenido de el archivo a ser interpretado por el navegador. Si las páginas son *dinámicas*, la información de las mismas es generado dinámicamente a partir de datos provistos por el usuario o acciones predefinidas por la aplicación Web.

Como todo esto ocurre sobre la Web, entonces es posible utilizar las arquitecturas de Internet vistas anteriormente, extendiendo sus funciones, en comparación con programas de *software* que ocurren en ausencia de un ambiente de red. Por ejemplo, un sistema Web destinado a recibir y guardar información dado por un usuario sigue la arquitectura Cliente-Servidor, pues el usuario usa su computadora como cliente al acceder a un sitio en Internet. Los mensajes y toda la información que el cliente puede ver es provista por el servidor. Además, el servidor brinda los mecanismos para que el usuario pueda mandar la información a guardar. Por otro lado, también puede ser visto como un sistema distribuído. Desde el punto de vista técnico, la ejecución de la aplicación Web depende solamente de los datos de entrada que ocurren en la computadora del usuario. Por tanto, la ejecución de la aplicación Web

ocurre tanto en el cliente como en el servidor, por lo que se considera un ambiente distribuído y, por lo mismo, cómputo distribuído. Existen autores como Farley [12] que no consideran la estructura de Cliente-Servidor como una particularización del cómputo distribuído, pues consideran que el cliente sólo sirve como control de flujo de ejecución de los programas del sistema Web, es decir, el cliente, a pesar de interactuar con la ejecución de la aplicación Web, no procesa datos como en la definición anteriormente dada de cómputo distribuído. Esto ocurre cuando se trata de una página estática: el usuario sólo pide ver cierta información, y la página solo se presenta con la información previamente solicitada. Sin embargo, el servidor tiene la labor de repartir de trabajo. Las páginas dinámicas tienen este comportamiento. El cliente ahora procesa información por petición del servidor. Tanto el cliente y el servidor hacen una sola labor, previamente planeada por medio de subtareas, compartiendo los datos de salida entre sí y manteniendo la mayor independencia posible entre esas subtareas.

2.1.4. Servicios Remotos

Definición de Servicios Remotos

> "Un servicio es toda aquella funcionalidad provista por algo para satisfacer cierta necesidad colectiva" [16].

Analizando la definición anterior, para los fines del presente trabajo, la funcionalidad debe ser una cuestión de recursos computacionales. Ese "algo" es un módulo o aplicación de software. El término "necesidad colectiva" prevee que la funcionalidad sirva para un conjunto de entidades, sean instituciones, personas o procesos computacionales. Cuando esta funcionalidad se da por medio de Internet, es para satisfacer la necesidad colectiva de usuarios de un sistema Web.

La funcionalidad brindada por *hardware* y *software* resulta ser un servicio. Cuando las acciones ocurren localmente (en una computadora sin conexión a la red) son **servicios locales**. Ejemplos de este tipo de servicio son programas de edición de texto, calculadora, etc. Cuando los servicios ocurren a través de una red de computadoras, se considera como *hardware* a todas las computadoras que forman la red, y como *software* a todos los procesos que estén disponibles dentro de esa red. A pesar del nivel de complejidad, las acciones provistas de esta red son vistos como **servicios remotos**.

> "...entonces, los *servicios remotos* son *representantes* de módulos de software disponibles en la red, con el objetivo de realizar funciones específicas para sistemas de software" [33].

En la Orientación a Objetos (OO), los objetos son ejemplares de una clase. En la Orientación a Servicios, los servicios son los ejemplares de un módulo de software a través de Internet. Con esta idea, los diferentes paradigmas de programación aprovechan este concepto de diferentes maneras. Por ejemplo, la forma de expresar los servicios en un paradigma Orientado a Objetos es por medio de las **clases abstractas** nativas del lenguaje.

Con lo anterior, es posible definir más formalmente a un **servicio remoto** como:

> "Conjunto de acciones (función de negocio) expuesto en la red, autocontenido, con una interfaz bien definida y estable que recibe requerimientos de sus clientes. El servicio no depende del contexto de sus clientes y puede ser consumido por varios sistemas sin ser modificados... Los servicios permanecen disponibles sin consumir recursos hasta que son invocados" [33].

Un servicio está formado por acciones, es decir, por un algoritmo o pasos a seguir. La función de un servicio está ligado por el número de acciones que lo representen. El servicio está disponible en una red para ser consumido por otros sistemas. Está autocontenido, pues es independiente de cualquier otro servicio. Debe de existir una forma de usar este servicio y de encontralo en la red. El servicio no se modifica para adaptarse al consumidor y, sobre todo, debe mantenerse disponible sin gastar recursos computacionales por ninguna de las partes involucradas.

Existen tecnologías y métodos para conseguir la comunicación para acceder a un servicio remoto, como los métodos de *Hessian, Burlap, RMI, HTTP invoker* o *Servicios Web*, siendo esta última la más compleja pero ampliamente usada en la actualidad, pues se rompe la barrera del lenguaje en el que el servicio remoto está implementado [32].

Los servicios remotos deben ser expuestos por un "proveedor de servicios" e integrados, o usados por un "consumidor de servicios" que lo use. Por ejemplo, en Java, es posible consumir un servicio por los métodos de Hessian, Burlap o HTTP invoker, gracias a la tecnología de *Spring* [2].

2.2. Concepto de SOWA

Una SOWA es la descripción de un sistema Web y de todos los servicios remotos que lo conforman. Esta formada por un consumidor, un proveedor, un medio de comunicación que conduce cada mensaje que ocurre entre ellos y las convenciones usadas para creación y envío de mensajes. El protocolo de comunicación puede ser cualquiera conocido para comunicar procesos en la red, como CORBA, RMI o Web Services [12]. Los servicios deben estar disponibles por algún medio que permita su publicación y explotación, como UDDI o ebXML [12].

2.2.1. Elementos de una SOWA

El **Proveedor de Servicios** o **Servidor** es un sistema de hardware y software en Internet que se encarga de poner a disposición los servicios. Se trata de un mecanismo para dar a conocer los servicios a todos aquellos que requieran integrarlo a su proyecto Web. Cuando el programador quiera usar el servicio se conecta de manera remota a este sistema, mandando los datos de entrada pertinentes. Se le conoce como Servidor pues cumple con las mismas características del servidor de la arquitectura Cliente-Servidor de Internet.

El **Consumidor de Servicios** o **Cliente** es un sistema Web que usa los servicios remotos que encuentre disponibles y que sean útiles para cumplir su función. Debe conocer la dirección en Internet (URL) donde se encuentra el servicio remoto y programar un medio de comunicación con el proveedor de servicio. También se le llama Cliente pues concuerda con ese elemento de la descripción de Internet Cliente-Servidor.

El **Descubrimiento** es la forma en que el servidor expone en Internet sus servicios remotos para su búsqueda. Por medio del Descubrimiento es posible encontrar nuevos servicios remotos por palabras claves de su descripción, protocolos de comunicación e incluso información de la compañia que

[2] www.springframework.org

creó el servicio remoto. El Descubrimiento en SOWA es una etapa necesaria si se quiere dar a conocer un servicio remoto a la comunidad de Internet. La fase de **Encuentro** y **Publicación** son interacciones del cliente y servidor para explotar y exponer los servicios remotos.

Para la fase de **Interacción** entre el cliente y el servidor en la SOWA, debe existir cierto entendimiento y ejecutar la comunicación. Para ello se define el ente más impórtante para la SOWA: el Contrato.

El **Contrato** es la descripción del servicio en un lenguaje común. Este lenguaje debe ser comprendido por el proveedor y consumidor del servicio remoto. Se describe el nombre del servicio, ubicación en la red, nombre de métodos, tipos de datos usados en la entrada y salida de cada método, etc.

La Figura 2.3 muestra la forma en la que está conformada una SOWA.

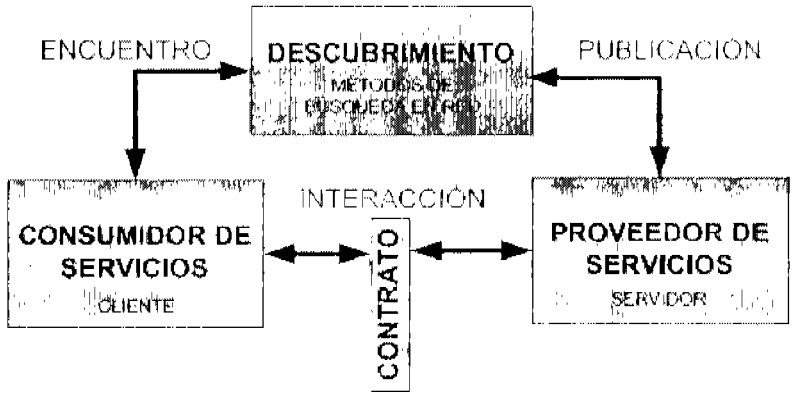

Figura 2.3: Diagrama de bloques para la estructura de una SOWA

2.2.2. Funcionalidad de los elementos de una SOWA

El Servidor pone a disposición sus servicios remotos en la red con la etapa de Publicación. El Cliente busca y encuentra los servicios remotos en la red por medio de la fase de Descubrimiento. Todos los servicios remotos establecidos en él siguen algún Contrato. Se inicia la etapa de Interacción. Usando las convenciones dadas en el contrato, se programa un Medio de Comunicación para mandar y recibir mensajes en un lenguaje convenido, por

ejemplo XML, texto plano o un formato original. El Cliente manda datos al Servidor que procesa y regresa nuevos datos mismo medio. Esto implica que la ejecución del Cliente se lleve a cabo distribuidamente en al menos dos computadoras.

2.2.3. Servicios Web

> "*Servicios Web* (Web Services o simplemente WS) son aplicaciones Web que pueden ser *descubiertas, descritas* y *accedidas* basados en XML y protocolos Web estándar sobre Internet" [9].

Los WS son un medio de exponer y usar un servicio remoto. Por medio de herramientas de software es posible descubrirlos en la red para ser usados cuando sea posible. Cuando los servicios remotos son implementados se pueden publicar, es decir, mostrar por medio de la red a cualquier persona o sistema que lo necesite mientras son descritos para dar un mejor entendimiento de su funcionalidad a quien los busque. Tras la implementación de un medio de comunicación es posible que el servicio sea accedido. El uso de XML es para formar una convención en la comunicación y los protocolos Web estándar son la forma en que la información viaja entre computadoras, por ejemplo HTTP, SMTP o FTP. Estos mensajes son importantes porque los datos de entrada y salida son enviados por este medio así como llamadas a acciones en particular de un servicio. El *Web Services Architecture Working Group* [3] define la estructura de los WS describiéndolos como la conformación de una **Lenguaje para Definir un Web Service** (WSLD), un **Protocolo de Acceso a Objetos Simples** (SOAP) y una **Descripción Universal de Descubrimiento e Integración** (UDDI) [32]. La Figura 2.4 muestra la estructura jerárquica de un WS.

El **Lenguaje para Definir un Web Service** o WSDL es la forma en que se describe un servicio remoto usando el lenguaje XML [9]. La información contenida en cada descripción incluye los siguientes elementos [35]:

- **Port Type:** Es la definición de los métodos que son expuestos en la red. Por ejemplo si el servicio cuenta con utilidades para cadenas de caracteres y números, se puede hacer un port type para cadenas y otro para números dentro del mismo servicio.

[3] Grupo formado por la World Wide Web Consortium (W3C) para definir informes y documentos descriptivos de los servicios remotos.

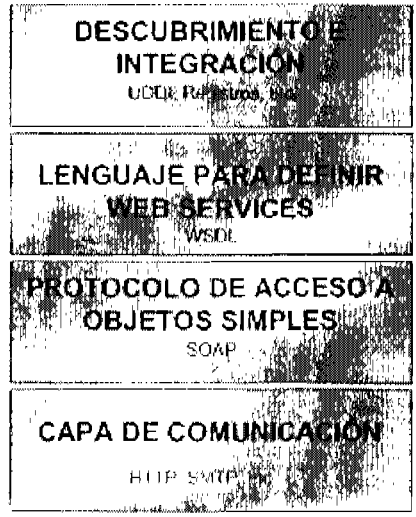

Figura 2.4: Diagrama de bloques de la estructura de capas de los WS

- **Binding:** Define el formato de los mensajes y el protocolo de comunicación. Por ejemplo, la combinación de mensajes en texto plano con el protocolo de SMTP (e-mail), o mensajes en XML por medio del protocolo de HTTP (como en el caso de estudio).
- **Port:** Combina un Port Type y un Binding. Si alguien quiere acceder al servicio con un Port Type y Binding específicos, entonces se les asigna un Port para conseguir la comunicación.
- **Target Namespace:** Encapsula los nombres con los que son llamados cada uno de los métodos y variables dentro de un servicio.

Como ejemplo, supóngase que se tiene la definición de un servicio que suma números enteros. La estructura del documento debe mencionar cada uno de los elementos anteriores. Así se vería el documento WSDL referente a este servicio de suma de números.

```
<?xml version="1.0" encoding="UTF-8"?>
<service name=''ServicioDeSuma''>
...
<!-- Se establecen las estructuras de -->
<!-- datos usados en el servicio -->
<!-- Los elementos se guardan -->
<!-- en la variable espacioDeNombres -->
<schema targetNamespace=''foo'' ...>
<!-- los elementos se definen gracias a los -->
<!-- elementos de XML Schema -->
<!-- los numeros enteros son llamados int de XML -->
<element name=''arg1'' type=''xsd:int'' />
<element name=''arg2'' type=''xsd:int'' />
<!-- se definen los argumentos de la suma -->
<element name=''sumaInput'' />
   <complexType>
     <sequence>
        <element name=''sumando1'' type=''foo:arg1''>
        <element name=''sumando2'' type=''foo:arg2''>
     </sequence>
   </complexType>
<element>
<!-- Se define la salida del metodo -->
<element name=''sumaOutput'' type=''xsd:int'' />
...
<!-- Se mencionan entradas y salidas de los metodos -->
<wsdl:portType ... >
   <operation name=''suma''>
     <input name=''sumaI'' type=''foo:sumaInput'' />
     <output name=''sumaO'' type=''foo:sumaOutput'' />
   </operation>
</wsdl:portType>
...
<!-- Se mencionan el tipo de protocolo y mesajes por SOAP>
<wsdl:binding transport="http://schemas.xmlsoap.org/soap/http>
...
<operation name=''foo:suma'' />
...
</wsdl:binding>
</service>
```

Una alternativa a explicar un documento XML lleva a mostrar el WSDL como un diagrama de clases [2]. Este diagrama es una extensión de UML que usa estereotipos. Así pues, las banderas usadas en un XML son estereotipos. Las reglas usadas en la contrucción de un diagrama de clases se mantienen. Es decir, en la exposición de un servicio, los elementos *binding* y *portype* pueden existir uno o más en el WSDL. Estos elementos están agregados en la estructura general del servicio. En la Figura 2.5 se muestra el diagrama de clases respectivo al WSDL de suma de números enteros.

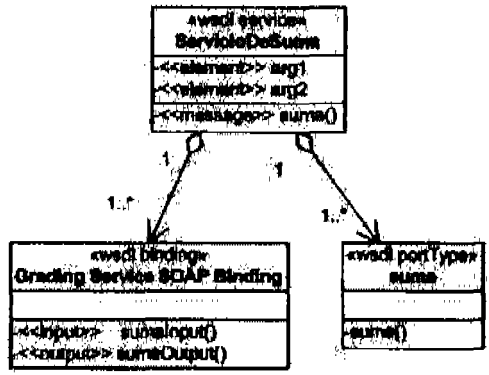

Figura 2.5: Diagrama de clases de un servicio de suma

El **protocolo de acceso a objetos simples** o SOAP[4] es el mensaje entre consumidor y proveedor del servicio remoto. Es una especificación de paso de mensajes para describir el codificado y las reglas de empaquetado para comunicaciones basadas en XML [14]. Todos los mensajes que ocurran deben seguir esta convención y serán llamados mensajes SOAP.

SOAP está formado por los siguiente componentes [14] y [9]:

- **SOAP Envelope:** Describe cómo es un mensaje y quién debe consumirlo. Aqui se escriben los HTTP Binding y las Reglas de Codificación.

[4]SOAP ha cambiado de significado con el tiempo. Dependiendo del autor también puede significar "Protocolo de Arquitectura Orientada a Servicios" [11]. Aunque en la actualidad SOAP no tiene traducción directa [32].

- **SOAP Header:** En esta parte se escriben partes de configuraciones de SOAP Header o escribirse datos a ser enviados. Passwords pueden ser mandados entre estos datos[5] haciendo a los WS seguros.
- **Reglas de codificación:** Define cómo deben enviarse los datos y las estructuras de datos usados para su serialización y deserialización.
- **Soporte RPC:** Define las llamadas a métodos en particular de un servicio remoto. A nivel de comunicaciones son llamadas a procedimientos desde otro sistema de manera remota (Remote Procedure Calls o RPC).
- **HTTP Binding:** Son extensiones al protocolo de comunicación HTTP para definir tamaños de paquetes de envío. Esto es, cómo los datos son enviados a nivel de bytes.
- **SOAP Body:** Es el cuerpo del mensaje. Todas aquellas funciones a llamarse se invocan en esta sección.

A continuación se presenta un ejemplo de mensaje SOAP de envío mostrando sus componentes [14]:

```
<!-- mensaje de envio SOAP  -->
<!-- Este es el SOAP Envelope. Se evita el uso de SOAP Header-->
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/

<!-- la siguiente linea muestra las reglas de codificacion -->
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<!-- el cuerpo del soap. Se llaman operaciones desde aqui -->
<SOAP-ENV:Body>
<!-- se manda a llamar un metodo llamado getProductPrice con RPC-->
<m:GetProductPrice xmlns:m="MyCompany-URI">
<productId>123456</productId>
</m:GetProductPrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

[5]Los mensajes SOAP no se mandan codificados por lo que si se requiere mandar mensajes seguros se debe realizar un preproceso para mandarse una llave pública en el mensaje, por ejemplo.

Finalmente la **Descripción Universal de Descubrimiento e Integración** o UDDI se usa para describir la ubicación del servicio remoto en la red, mediante el cual se estandariza el descubrimiento de WS en la red [33]. Es un conjunto de descripciones WSDL disponible en línea [32]. Por otro lado UDDI permite la clasificación de los servicios disponibles por clasificación geográfica o códigos industriales. La información provista a un registro UDDI consiste de tres componentes: las **páginas blancas** para contactar a la compañia que implementó el servicio remoto; las **páginas amarillas** mencionan que clase de información provee el servicio; finalmente están las **páginas verdes** con información técnica y documentación del servicio [14]. Toda la información se presenta como una página Web que funcione como directorio[6].

Ventajas de los WS

- Los estándares de transporte usados por los WS como HTTP y XML son soportados por la mayoría de los sistemas operativos además de ser ampliamente aceptados y estudiados [33].
- Los WS son independientes del lenguaje de programación de su implementación ya que la forma de describirlos es por medio de XML [3].
- Los protocolos de comunicación son soportados por los *firewalls* corporativos por lo que sobrepasan la seguridad de un sistema de cómputo sin ser invasivo [33].

Desventajas de los WS

- Cuando el servicio no está disponible en la red, todos los sistemas Web que lo consumen pierden esa funcionalidad. En algunos casos, los sistemas Web dependen totalmente de los servicios por lo que un error en el servidor de WS afecta una serie de sistemas [31].
- Cuando los servicios son muy generales en su funcionalidad, los sistemas que requieren del servicio exigen que tenga funcionalidades particulares a su necesidad. Est puede llevar a problemas de programación o incluso a problemas de diseño del servicio [31].
- El uso del protocolo HTTP no garantiza envío o recepción de mensajes entre cliente y servidor. Existen protocolos más confiables al respecto

[6] Ejemplo www.webmethods.com

como JMS pero que limita el número de sistemas con los que se puede intercomunicar [31].

- Los servicios mas frecuentemente usados pertenecen a compañías vendedoras de software y no asi a implementaciones de software libre [37].

Relación entre WS y SOWA

WS se considera una especificación de SOWA. Mientras SOWA es la descripción de la exposición y consumo de servicios remotos, WS establece medios más especificos para conseguirlo. La siguiente tabla describe tal relación.

Tabla relacional entre WS y SOWA

SOWA	WS
Servidor SOWA	Servidor WS
Consumidor SOWA	Consumidor WS
Contrato	WSDL
Descubrimiento	UDDI
Interacción	SOAP, XML, HTTP
Lenguaje de implementación del servicio es estático	Cualquier lenguaje de programación

La implementación de un WS ofrece ventajas, como la estandarización de protocolos utilizados y la interoperabilidad natural obtenida por la comunicación por HTTP y XML [33]. La mayor desventaja es el tiempo de proceso de datos que se incrementa con la serialización y deserialización de datos. El transporte de datos por XML se traduce en que los mensajes son entendidos desde cualquier lenguaje de programación, pero es lento si es comparado con métodos que reconozcan el paso de mansajes entre sistemas usando el mismo lenguaje de programación. Las herramientas de desarrollo que esperan apresurar esta comunicación apenas están siendo creadas, por lo que existe cierta espectativa para la adopción oficial de esta unión [33].

2.2.4. Definiciones y Contratos

Las **Definiciones** son las "reglas establecidas" para la creación de contratos [34]. Un contrato describe un servicio remoto de la manera que quiera el programador. Esto podría crear incertidumbre, pues dos servicios remotos, representando una misma funcionalidad, pueden ser expuestos con dos

diferentes contratos y, por tanto, dos formas diferentes de consumir una misma funcionalidad. De tal modo, las definiciones sirven para estandarizar la forma de implementar un módulo de software para su posterior exposición como servicios remotos. Asi pues, los servicios que cumplen con una funcionalidad específica son similares en la forma en la que son consumidos. La Figura 2.6 muestra gráficamente las Definiciones y Contratos al consumir un servicio remoto .

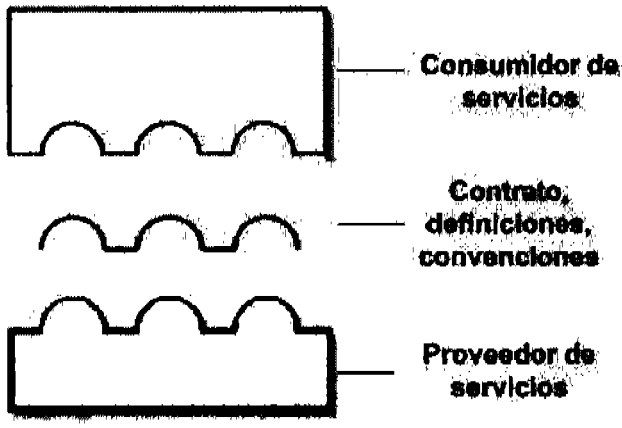

Figura 2.6: Diagrama de bloques de una definición en una SOWA

Grupos de desarrolladores crean las definiciones. La más popular de las definiciones es la **Open Service Gateway Initiative** (OSGI). Son definiciones para el desarrollo de cualquier sistema Web, como telefonía móvil, así como de negocios en Internet. Las definiciones usadas en el caso de estudio es **Open Services Interface Definition** (OSID), que forma parte de una iniciativa del Tecnológico de Massachusetts llamado *Open Knowledge Initiative* (OKI), dedicado particularmente a los sistemas de enseñanza en línea.

Open Knowledge Initiative

Con OSID se establecen convenciones para desarrollar servicios Web para manejar cursos en línea, el manejo de la calificaciones, la búsqueda de recur-

sos, etcétera. Se trata de interfaces para lenguajes de programación Orientados a Objetos como Java y PHP. Consta de once conjuntos de definiciones de "servicios comunes", además de cuatro que se consideran necesarias para la creación de un sistema educativo [34]. A continuación se presenta un pequeño compendio de estos últimos.

- **Course Management:** Son un conjunto de interfaces para manejar los cursos del sistema educativo. A partir de ellas es posible el manejo, autenticación y creación de vínculos entre los cursos y los usuarios: estudiantes e instructores.

- **Repository:** Con estas interfaces se almacenan y buscan recursos entre sistemas en la red. Con su ayuda se pueden guardar y acceder archivos de video, imágenes, cuestionarios o cualquier cosa que el creador del sistema de software considere de carácter pedagógico.

- **Assesment:** Son interfaces para definir una asignatura, curso o tutoría. Existen mecanismos a partir de los cuales es posible tener los datos que identifiquen a una asignatura, sus dependencias con recursos del sistema, dueños y contenidos.

- **Grading:** Es la forma en la cual se pueden calificar todos los recursos disponibles en un sistema Web. Cuando una asignatura es creada es posible que existan recursos que puedan ser calificados. Supóngase que tras crear una asignatura sea posible la creación de un exámen o evaluación. Los alumnos entonces podrían resolverlo y ser calificados posteriormente por los instructores. Se provee de formas a partir de las cuales crear escalas de calificación, tomar calificaciones por letras, números, etc.

Existen otras definiciones consideradas "compartidas". Los alumnos y profesores pueden ser implementados a partir de la definición *Agent*, asi como describir sus propiedades e identificadores con *Id, Type* y *Properties*.

El uso de OSID tiene las siguientes ventajas [34]:

- **Encapsula tecnología, protocolos e implementaciones:** Esto es cierto porque se oculta toda información acerca de la implementación.

- **Modelo de creación común:** Esta característica se traduce como reuso de código. Es posible cambiar completamente el comportamiento de una aplicación con sólo con cambiar los servicios que lo conforman.

Como son definiciones, entonces los métodos y nombres solo deben ser reescritos o sobrecargados para cambiar la funcionalidad.

- **Libertad para desarrolladores:** Tomando OSID se puede llegar a la implementación de un sistema ajeno a lo educativo. La libertad del programador está entonces en la cantidad de combinaciones posible con las interfaces y las diferentes funcionalidades que se le puedan lograr.

Para el servidor [34]:

- **Ayuda a que los servicios generados sean descubiertos de manera más sencilla**, pues todos los mecanismos de búsqueda de las universidades y asociaciones que implementan OSID buscan servicios similares a los suyos. Por medios como UDDI es posible hacer público el servicio y permite que los buscadores encuentren el servicio cumpliendo el contrato de OKI cada vez más usado entre proveedores y consumidores de servicios [34].

- **Aumenta la tecnología existente**, ya que la creación de sistemas a partir de servicios deja mayor tiempo a la investigación y generación de tecnologías capaces de elevar el nivel de tecnología a ritmos mayores que antes [34].

- **No existen dependencias**. Tanto el proveedor como el consumidor del servicio se mantienen aislados todo el tiempo, permitiendo que el consumidor use otra implementación en otro lugar remoto, sin que el proveedor original o el nuevo proveedor estén conscientes del hecho. La idea es que el cambio de implementación usado sea en "tiempo real", esto es, al momento que el cliente mande ciertos datos de entrada, el sistema busque, encuentre y use el servicio que más convenga en ese caso. La realidad puede ser más demandante en el sentido de que no es posible la integración de un sistema sin que el consumidor de servicios tome ciertas decisiones, aunque las condiciones ya existen para que esto sea posible [3].

- **Bajos costos de integración**. Los costos existen con respecto a tiempo de desarrollo de un sistema y en el grado de capacitación de un programador, al solo buscar el servicio e integrarlo al proyecto. Los sistemas basados en servicios Web son representativos de bajos costos en al menos estos dos aspectos [9].

Para el cliente [34]:

- **Integración tipo** *Plug-in.* Los servicios Web están aislados del consumidor, por lo cual se puede cambiar de un servicio a otro sin afectar la ejecución del sistema.

- **Se reduce la complejidad del código**, porque sólo es necesario saber lo estipulado en el contrato, es decir, las definiciones de OKI para mandar a llamar métodos y funcionalidades del módulo. No es necesario hacer búsquedas ni lidiar con diferentes nombres de las clases.

- **Toda la aplicación no debe cambiar para usar un nuevo servicio remoto.** Si el consumidor de servicios desea integrar uno nuevo, es posible que sólo programe el acceso al servicio.

- **El uso de servicios podría hacerse en tiempo de ejecución,** en el cual el usuario, desde el cliente, manda información al servidor (en este caso, el consumidor de servicios).

Particularmente, la definición Grading de OSID, tiene importancia para los alcances de esta tesis. Se analiza una implementación de Grading como caso de estudio para su exposición y consumo como servicio remoto.

Vista General de Grading

El OSID Grading consta principalmente de tres interfaces que abstraen la funcionalidad de guardar, modificar y borra evaluaciones. La Figura 2.7 muestra una vista general de Grading y a continuación una explicación de sus elementos.

- **Gradable Object:** Es la forma de definir a un "objeto calificable". Es decir, un exámen, tarea, cuestionario, etcétera. Por medio del mismo es posible poner un nombre, forma en la que se evalúa y la escala de calificación.

- **Grade Record:** Se trata de una representación de las calificaciones en un registro. Está conformado por tres partes escenciales. La persona a quién se va a calificar, el tipo de objeto que se califica y el valor de la evaluación. Por ejemplo, *La Persona X, obtiene Y de calificación en el Exámen Z* muestra la forma en la que se deben interpretarse los datos en un Grade Record. La idea original de esta definición es proveer sólo los identificadores de cada una de estas partes. Así pues,

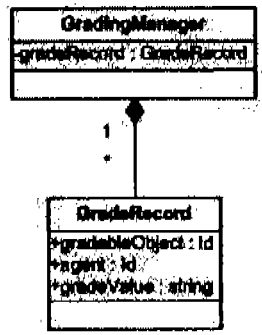

Figura 2.7: Diagrama de clases de Grading

en un manejador de bases de datos, es posible hacer llamados a los verdaderos datos a partir de estos identificadores.

- **Grading Manager:** Se trata de la definición que controla todo el módulo. Gracias a ésta, es posible crear, modificar y eliminar a los Gradable Objects y Grade Records. Es obligación del programador definir el comportamiento del mismo. Por ejemplo, las anteriores acciones podrían existir para manipulación de entidades a nivel de memoria de la computadora o extenderse a un manejador de bases de datos y mantener estos datos persistentes.

Viendo más a fondo la estructura de un Gradable Object, se encuentran otras entidades. Están contenidas en una definición OKI llamada *Type*. Por si misma no forma parte de Grading, pero es compartida por las definiciones de OSID. Dependiendo de su implementación puede dar cierta información que el programador considere necesaria. En la Figura 2.8 se muestra la estructura de Gradable Object y posteriormente la explicación de sus componentes a partir de la definición Type.

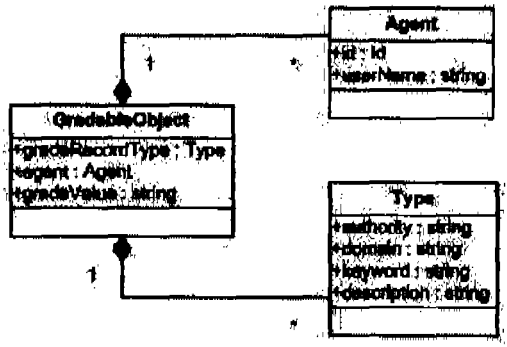

Figura 2.8: Diagrama de clases de Gradable Object

- **Grade Record Type:** Es la forma de asignar el tipo de calificación que usa el Gradable Object. Esta puede ser numérica, por letras, o cualquier otro tipo que permita el programador. Por ejemplo, ser calificado por medio de frases que describan una calificación, como "Pasa" y "No Pasa".

- **Grade Scale:** Es la escala en la cual se limita la calificación. Con esta, es posible definir la frontera de un Grade Record Type. Por ejemplo, si el tipo de calificar es numérica, entonces con el Grade Scale se define si es de cero a diez, del diez al cien o del uno al quince.

- **Scoring Definition:** Define el "significado" de la calificación. Se interpreta el valor de la calificación en base a lo mencionado por este. Por ejemplo, no es lo mismo la escala de calificaciones de la A a la F que de la F a la A. Gracias a Scoring Definition se dá el sentido de cual es el mayor, el menor y el orden en la escala dada.

Las tres definiciones anteriores implementan la interfaz *Type*. Está formada por cuatro cadenas de palabras. En **authority** se menciona la autoridad involucrada en el objeto a calificar, por ejemplo "unam.ciencias". En **domain** el tipo de objeto a calificar, por ejemplo "examen.extraordinario". Con **keyword** se menciona el tipo de calificar como se mencionó en Grade Record Type, por ejemplo "calificación.numerica". Y **description** para mostrar una ligera descripción del tipo, por ejemplo "Formado por un número y un signo + o -". Formar los tipos de Grade Record Type, Grade Scale y Scoring Definition depende solamente de la implementación que se quiera, por ejemplo,

si se quiere que el domain represente la escala al implementar al Scoring Definition, no existe alguna razón de peso que afecte el comportamiento del sistema. La Figura 2.9 muestra las implementaciones de Type en Grading.

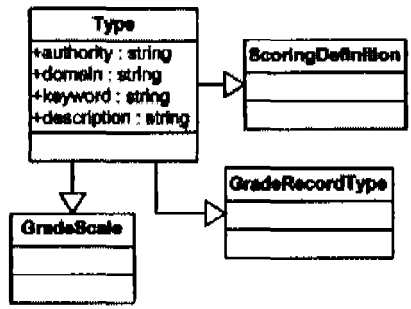

Figura 2.9: Diagrama de clases de Type y sus implementaciones

Por otro lado, Grade Record está conformado por los siguiente elementos, según la Figura 2.10:

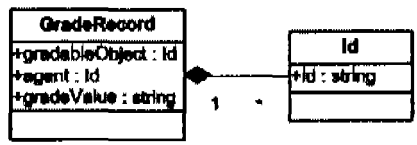

Figura 2.10: Diagrama de clases de Grade Record

- **Gradable Object:** Es el identificador de la definición previa de un objeto a calificar con las características dadas.
- **Agent:** Es el identificador a una interfaz ajena a Grading pero integrante de las definiciones compartidas de OKI. Es la forma de identificar a un personaje dentro de las implementaciones de los módulos. Los agentes pueden ser desde alumnos o profesores hasta cualquier otra definición de usuarios del sistema.
- **Grade Value:** Finalmente se trata del valor numérico o literal asignado al agente con respecto al Gradable Object. La correlación exis-

tente entre estas tres es escencial para mantener datos coherentes en el módulo.

Los valores en Grade Record están dados por los identificadores. Por medio de la definición *Id* de OKI es posible la construcción de identificadores tan extensos o complejos como se desee.

Además de todas las definiciones anteriores, existen otras para el manejo de los objetos en el módulo. Estos son iteradores de Gradable Object, Grade Record y Type. Con ellas se obtienen conjuntos de objetos que cumplen ciertas características. Los nombres se mantienen referentes al tipo de dato y estos son *Gradable Object Iterator*, *Grade Record Iterator* y *Type Iterator*. También existe la forma de preveer errores en la ejecución del sistema con *Grading Exception*.

Para comprender el comportamiento del módulo de Grading, considérese la siguiente frase:

> El alumno X obtuvo una calificación de Y en una escala literal de A a B, donde A es el mayor y B el menor, en el exámen Z.

El "alumno X" es un *Agent*; la "calificacion Y" es el *Grade Value*; la "escala de A a B" es el *Grade Scale*; el hecho de ser "literal" es el *Grade Record Type*. La mención de "A mayor que B" está dado por *Scoring Definition* y el nombre de "exámen Z" es el *Gradable Object*. De hecho toda la información correspondiente al exámen Z está dado en el Gradable Object y el registro de la evaluación pertenece al Grade Record dado por *Id*'s. Todo esto puede ser creado, alterado o borrado del módulo por medio del *Grading Manager* mientras cualquier error del sistema reportado por *Grading Exception*.

2.3. Patrones de software

> "Los patrones de software son descripciones para comunicar objetos y clases que han sido modificadas para resolver un problema general de diseño dado un contexto" [15].

> "Los patrones identifican y especifican abstracciones más allá del nivel de simples clases e instancias o de sus componentes" [7].

> "Los patrones de diseño se enfoncan más en el reuso de temas de diseño arquitectónico recurrente, mientras los frameworks se enfocan en el diseño detallado e implementación" [7].

En la sección anterior (2.2), se reconocen los componentes de una SOWA. Particularmente, se menciona una definición, como OKI Grading, para tener una idea más específica de objetos y clases en una implementación que puede seguir la SOWA. Los patrones de software sirven para describir la comunicación entre los componentes de software dada una serie de condiciones.

La primer definición hace mención a las condiciones para aplicar un patrón de software. Si el problema radica en la implementación de un sistema a partir de definiciones previamente dadas, entonces se puede hacer un análisis de la situación e identificar un contexto necesario, es decir, encontrar la situación adecuada para usar un patrón que nos brinde una solución general mientras las consecuencias del mismo no afecten la integridad o funcionamiento del sistema.

La segunda definición reforza la idea de la abstracción de un patrón. No se trata de una solución específica sino de un consejo de implementación. Dadas ciertas circunstancias, se propone una descripción de la solución, aunque no indica estructuras de datos o implementación de módulos.

El tercer concepto hace una comparación de los patrones de software con los *frameworks* para mostrar la abstracción de los patrones y la forma en la cual se pueden reusar dependiendo de la situación. Los frameworks son fragmentos de programas que sirven para resolver una serie de problemas específicos. En contraste, los patrones tratan de encontrar el comportamiento de un problema de diseño para encontrar una solución general y aproximada cuando ocurren todas las condiciones para ser aplicados. Los frameworks son usados en problemas específicos, dado un lenguaje de programación y sirven para ahorrar tiempo y esfuerzo en programar la solución mas no necesariamente implica ahorros en desempeño.

2.3.1. Model-View-Controller

Problema

Las especificaciones que llevan a la creación de un sistema de software suelen cambiar con el tiempo. En algunos casos, estos cambios pueden afectar

toda la funcionalidad del sistema. Por consecuencia, la programación detrás del sistema también se ve afectada con las dependencias existentes entre los elementos involucrados. Esto se traduce en tiempo perdido en nuevos ciclos de desarrollo de software.

Solución

Model-View-Controller es un patrón de software proveniente del diseño de sistemas en Web. Se divide el problema de diseño en tres partes lógicas [7]: un Modelo de Datos con partes computacionales del programa, la Vista para la interfaz de usuario y el Controlador para intermediar entre el modelo y la vista. De esta manera, los cambios en las especificaciones pueden clasificarse en alguna de estas partes lógicas, haciendo los cambios pertinentes en alguna de esas partes sin afectar a las demás.

El **Modelo** es la representación de los datos de la aplicación, así como las **reglas de negocios**. Estas últimas son las reglas, a partir de las cuales, se pueden cambiar los datos de la aplicación.

La **Vista** es lo que el usuario final puede ver del sistema. Es la forma en la cual los datos del modelo se muestran. La Vista jamás debe modificar directamente al Modelo o modificar sus reglas de negocios. Dependiendo del tipo de usuario del sistema, es posible que existan diferentes tipos de Vistas. Por ejemplo, el administrador del sistema debe tener los mecanismos para hacer modificaciones en los datos del sistema y tener control de cosas que cualquier otro usuario del sistema no puede hacer. La Vista en ambos casos debe ser diferente.

El **Controlador** funge como mediador entre el Modelo y la Vista. Define el flujo de información en el sistema. Interpreta eventos del usuario y los transforma en mensajes para Modelo y Vista. El Controlador o los Controladores hacen las interpretaciones necesarias a las diferentes Vistas para mantener el flujo de datos.

La Figura 2.11 ilustra el modelo MVC. En la implementación de un sistema adoptando este patrón, se puede evitar la frontera entre las partes. Se recomienda mantenerlas en la total independencia aunque no existen consecuencias directas si no se siguen las indicaciones. Existe una versión de MVC llamada Modelo 2 donde se exige la independencia de los elementos de MVC. El caso de estudio sigue el Modelo 2.

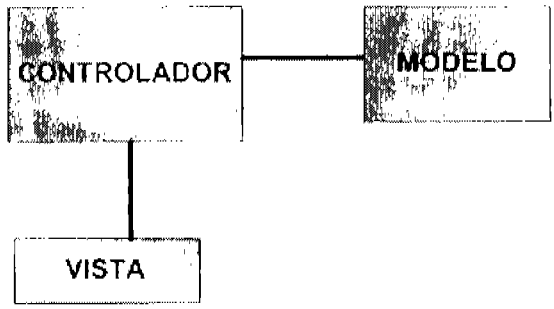

Figura 2.11: Diagrama de bloques de Model-View-Controller

Estas son la ventajas de usar MVC [7]:

1. Si esta división lógica no favorece el desarrollo de un sistema, puede ser contraproducente y provocar más dependencias de las que se tratan de evitar.

2. Es más útil sólo en implementaciones Web.

2.3.2. Façade

Problema

Se tiene un conjunto de clases con un mismo comportamiento. Esto crea una serie de dependencias que siguen una misma estructura, generándose exceso de código que dificulta el mantenimiento del sistema. Las dependencias entre clases se puede incrementar pues la comunicación de clases no está controlada. La Figura 2.12 muestra el problema de dependencias. Suponiendo que los bloques exteriores (Clase 1, 2 y 3) dependen de las clases consideradas "comunes" (Subsis 1, 2 y 3), se unen por flechas para simbolizar esta dependencia.

Solución

El patrón llamado **Façade**[7] es una forma en la cual se intenta unificar una serie de interfaces con un comportamiento similar. La similitud en el

[7]Prononciese "Fasád". Palabra en francés que significa "Fachada"

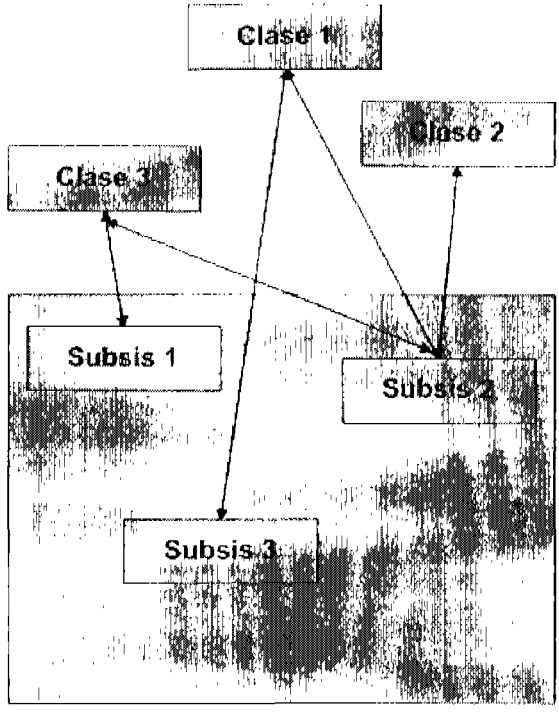

Figura 2.12: Diagrama de bloques de dependencias de subsistemas

comportamiento queda en la apreciación del programador. Estructurar un sistema en subsistemas ayuda en disminuir la complejidad [15]. Usualmente se implementa este patrón cuando se trata de minimizar las comunicaciones y dependencias entre los subsistemas. La Figura 2.13 muestra cómo la interfaz eleva el grado de abstracción, haciendo de la *Façade* el único intermediario entre las clases y los subsistemas.

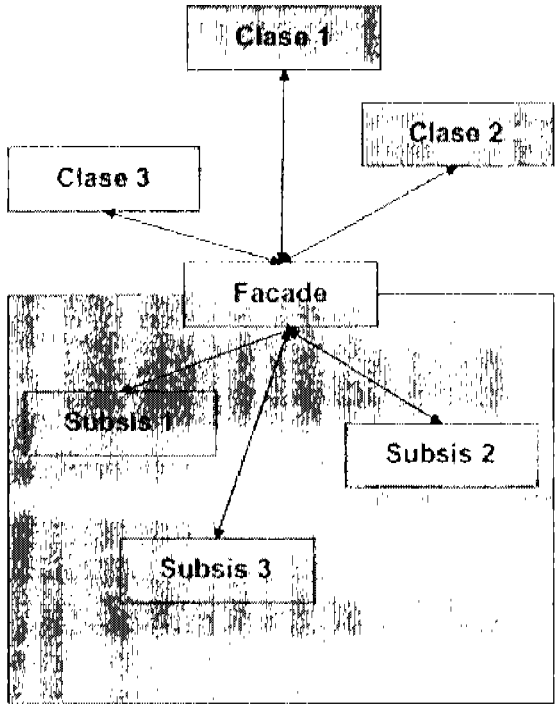

Figura 2.13: Diagrama de bloques de minimización de dependencias usando *Façade*

Façade ofrece los siguientes beneficios [15]:

1. Encapsula a los componentes del subsistema, reduciendo el número de objetos creados en las dependencias.

2. Promueve el bajo acoplamiento entre un subsistema y el exterior así como de los elementos del subsistema entre sí. Esto permite el cambio de clases en el subsistema sin afectar a los elementos del subsistema y del exterior. Esta característica es escencial en un sistema de software para evitar la recodificación de clases.

3. *Façade* se puede usar cuando se requiera. La decisión de usarlo queda abierto a elegir entre generalidad o fácil implementación.

2.3.3. Decorator

Problema

Se tiene un objeto con ciertas propiedades. Sin embargo, estas propiedades no son lo suficientemente descriptivas por tener datos codificados o que requieren algún postproceso para comprenderse. El postproceso puede implicar pérdidas de espacio en memoria y tiempo al crear nuevas estructuras que cumplen con las propiedades requeridas además del tiempo que implica su creación.

Solución

El patrón **Decorator**, también conocido como **Wrapper**, sirve para sumar propiedades a un objeto sin necesariamente asignarlo a toda la clase. Supóngase el ejemplo de un formato de fecha en la clase `Date` de Java. La forma en la cual Date tiene sus valores para representar la fecha es por medio de un número entero del tipo *long*. Este número puede ser modificado para obtener la fecha en cierto formato. Cuando el valor es "decorado" a partir de un método o clase, los datos de la fecha pueden ser manipulados para mostrar la fecha en otro formato mas allá de las capacidades de `Date` o `Calendar` de Java. La Figura 2.14 muestra el hecho anterior, haciendo un paso del formato nativo de Java a un formato más amigable en español, para ser mejor comprendido por un usuario.

Una forma de decorar a una clase es por medio de la herencia [15]. Esto, sin embargo, decora a todos los objetos de la clase implemetada sin elección si debe ser decorado un dato o no. Por tanto, una clase ajena debe hacer la decoración al recibir un objeto de la clase a decorar.

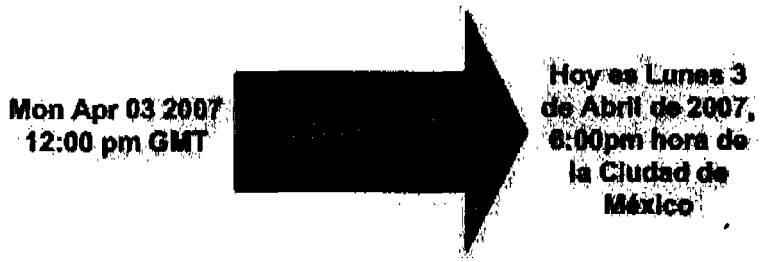

Figura 2.14: Diagrama de bloques mostrando el uso de Decorator en formato de fecha

Decorator brinda los siguientes beneficios [15]:

1. **Más flexibilidad.** En la implementación del decorador se puede elegir cuándo es posible hacer la decoración del objeto.

2. **Evita aumentar la jerarquía de clases.** La creación de subclases solo incrementa las dependencias entre ellas. En este caso, la jerarquía de clases no es una opción práctica.

3. **El objeto a decorar y el objeto decorado son los mismos.** La clase o método decorador sólo agrega o quita responsabilidades de un objeto sin quitarle su identidad.

4. **Muchos objetos pequeños.** Cuando un diseño usa Decorator, resulta como un conjunto de objetos que se parecen mucho entre sí. Los objetos solo difieren en la forma en la que están interconectados, mas no en su clase o las variables contenidas. Por ejemplo, se puede decorar a un objeto lo suficiente como para hacerlo parecer al objeto de otra clase diferente, sin embargo ambos objetos permanecen diferentes, conteniendo sus propios métodos y variables locales, dependiendo de la clase a la que representan.

Decorator tiene las siguiente desventajas [15]:

1. **Debe considerarse su uso cuando a un objeto se le agregan demasiadas propiedades.** Tal vez es más funcional crear una nueva estructura de datos con esas propiedades.

2. **El objeto decorado no pierde su identidad.** Aunque esto es un beneficio también en una desventaja. Las propiedades agregadas al objeto decorado no pueden ser manipuladas como si fueran las propiedades originales del objeto.

2.3.4. Dependency Injection

Problema

En los sistemas de *software* siempre existen dependencias. Cuando ocurren cambios en una clase, es necesario hacer el cambio en cada clase donde exista la dependencia. El uso de interfaces y clases abstractas ayuda a minimizar el problema ya que se trata con métodos abstractos y no con sus impementaciones. Sin embargo, las implementaciones deben de definirse en algúna parte del código del sistema de *software*. Si los cambios ocurren continuamente, por mantenimiento del código, se debe invertir tiempo en revisar y cambiar a las nuevas implementaciones en cada archivo que las defina.

La Figura 2.15 muestra un diagrama de paquetes que necesitan de un paquete implementación de una clase abstracta. Cuando los cambios ocurren en la implementación deben de modificarse en cada uno de los paquetes que lo necesitan.

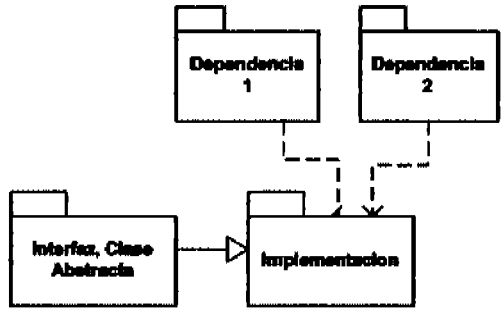

Figura 2.15: Diagrama de paquetes muestra la dependencia entre sus elementos

Solución

Dependency Injection, también conocido como **Inversion of Con-**

trol, es un patrón nacido en el lenguaje de programación PHP. La razón de su descubrimiento obedece a encontrar la forma de presentar las clases de manera legible, esto es, tener código fácil de leer, sin tener la necesidad de buscar las dependencias de clases en diferentes archivos. Se usan interfaces o clases abstractas y se definen en tiempo de ejecución gracias a *frameworks* como Spring. Esto se consigue, gracias a métodos *setters* y *getters* en la clase principal y un archivo de configuración con la ubicación de las clases que implementan esas clases abstractas.

La Figura 2.16 muestra el nuevo diagrama de paquetes tras aplicar *Dependency Injection*. Los paquetes dependen ahora de las interfaces que son inyectadas en momento de ejecución con los datos contenidos en un archivo de configuración. De esta manera, los cambios en las implementaciones sólo se hacen en el archivo de configuración, dejando sin cambios a las demás clases que dependen de los cambios pues estas reciben clases abstractas.

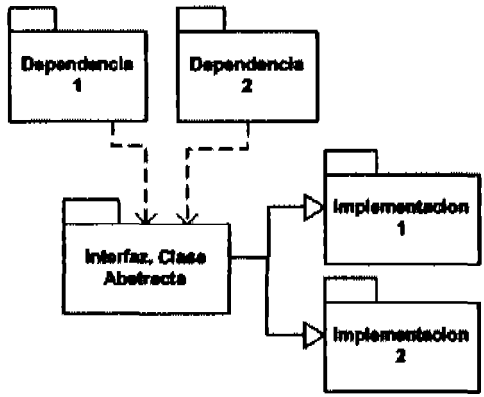

Figura 2.16: Diagrama de paquetes con *Dependency Injection*. Se puede cambiar la implementación sin afectar las dependencias

Estas son los beneficios de usar *Dependency Injection* [29]:

1. **El reuso del código y su independencia de la clase que lo manda a llamar.** Con esto es posible cambiar la implementación de una clase directamente manteniendo la estructura del sistema. Si se cuenta con un *contenedor de dependencias*, esto es, un archivo de configuración con el cual se haga la asignación de implementaciones desde el mismo momento de ejecución del sistema, será posible mantener el

código y cambiar implementaciones sin siquiera cambiar código en las clases. Por ejemplo, el contenedor de dependencias de Spring Framework (llamado *Spring Core Inversion of Control*), se encarga de esta labor.

2. **Un código difícil de probar es también difícil de entender y mantener.** Por tanto se debe buscar la forma de hacer al código más fácil de entender. Esto se consigue solo disminuyendo las dependencias o asignarlas por medio de interfaces implementadas.

Estas son la desventajas del uso de *Dependency Injection* [29]:

1. Ocurren errores en la ejecución de todo el sistema si no están disponibles las implementaciones. Esto ocurre porque las asignaciones se hacen en tiempo de ejecución de la aplicación. Todas las definiciones del archivo de configuración deben estar disponibles al momento de ejecución.

2. Si existen errores en el archivo de configuración también se detiene la ejecución del sistema. Aunque se minimiza la posibilidad de error por ser un solo archivo donde ocurren los cambios, una mala definición de las implementaciones implica la disponibilidad de la aplicación.

2.3.5. Proxy

Problema

La creación de objetos en una clase ocurre de manera aleatoria. Es por ello que no se tiene control de la creación. Además, si los objetos creados tienen el mismo comportamiento, la comunicación entre objetos implica alguna clase de gasto excesivo en la creación de objetos.

Solución

El patrón **Proxy**, también conocido como **Surrogate**, provee un "representante" de control para acceder a un objeto. Se usa cuando se trata de ahorrar el uso de recursos computacionales que conlleva la creación de un objeto [15]. De esta manera se tiene control del momento en el cual se requiere su creación y uso. Una clase *Proxy* sirve como intermediario entre una implementación y la clase que lo necesita.

La versatilidad de un *Proxy* puede activar desde un apuntador hasta un objeto. Es por ello que las capacidades de un *Proxy* pueden ser adaptadas

a diferentes situaciones. Existen los *proxies virtuales, de protección, de referencia inteligente y referencia remota.* Los **proxies virtuales** son aquellos que crean objetos bajo demanda desde una aplicación local. Se usan cuando se quiere hacer más rápido a un sistema en tiempos de respuesta. Los **proxies de protección** sirven para controlar el acceso de las clases a la clase implementada.

El **proxy de referencia inteligente** es para cambiar entre clases implementadas a partir de ciertas condiciones. Por ejemplo, si se requiere cargar una serie de usuarios llamados desde una clase llamada MAIN, el proxy puede hacer el cambio de implementaciones de manera indistinta dependiendo de las condicionales dadas por la clase MAIN. La Figura 2.17 muestra esta posibilidad. Este proxy permite hacer un postproceso de datos una vez que ha sido elegida la implementación a usar [15].

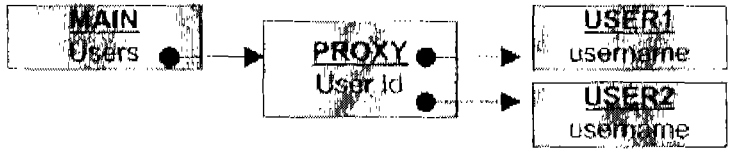

Figura 2.17: Diagrama de bloques de un Proxy de Referencia Inteligente

Por otro lado, el **proxy de referencia remota** sirve para controlar el acceso a un objeto localizado en algún punto de la red. Se usa cuando se requiere hacer una llamada a un recurso en la red pero se espera que la creación del objeto se haga sólo en el momento en que se haga uso de la red. Un *proxy de referencia remota* puede ocultar el hecho de que un objeto reside en algun lugar de la red diferente a la local [15]. En la Figura 2.18 se muestra el uso de la red para ocultar la localización de un objeto remoto de la clase USER.

Estos son los beneficios de *Proxy* [15]:

1. Cuando se usan los *proxies de referencia inteligente* se obtienen sistemas que cambian su implementación al momento mismo de la creación del objeto que lo llama.

2. Cuando se usan los *proxies de referencia remota* se puede mantener el control de la creación de los objetos remotos. De esta manera, las

Figura 2.18: Diagrama de bloques de un Proxy Remoto

llamadas remotas se llevan a cabo en el momento que se requiera. Esto tiene la posibilidad de usar la red sólo cuando se han juntado cierta cantidad de mensajes y se busque un ahorro en cantidad de información enviada.

Estas son las desventajas de *Proxy* [15]:

1. Las activaciones fallan cuando la configuración de los objetos por activar están mal definidos. Si al menos una de estas definiciones está errada, toda la aplicación está comprometida.

2. Los *proxies de referencia remota* están sujetos a disponibilidad remota, esto es, si el lugar físico o las condiciones de la red no permiten una activación remota o el representante no está disponible, la ejecución de la aplicación es detenida.

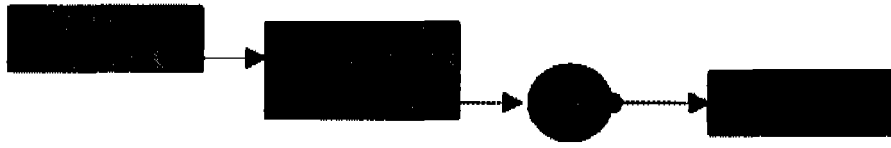

Figura 2.18: Diagrama de bloques de un Proxy Remoto

llamadas remotas se llevan a cabo en el momento que se requiera. Esto tiene la posibilidad de usar la red sólo cuando se han juntado cierta cantidad de mensajes y se busque un ahorro en cantidad de información enviada.

Estas son las desventajas de *Proxy* [15]:

1. Las activaciones fallan cuando la configuración de los objetos por activar están mal definidos. Si al menos una de estas definiciones está errada, toda la aplicación está comprometida.

2. Los *proxies de referencia remota* están sujetos a disponibilidad remota, esto es, si el lugar físico o las condiciones de la red no permiten una activación remota o el representante no está disponible, la ejecución de la aplicación es detenida.

Capítulo 3

Trabajos Relacionados

En este capítulo se muestran los trabajos relacionados con esta tesis. Se muestran ejemplos de arquitecturas orientadas a servicios. Es la forma en la que algunas compañias como Cisco Systems e IBM ven a la orientación a servicios y la forma en la que lo adoptan para su beneficio corporativo.

3.1. Service-Oriented Network Arquitecture

La **Arquitectura de Redes Orientada a Servicios** (SONA, en inglés), es la orientación a servicios desde el punto de vista de Cisco Systems [6]. Muestra una estructura de capas, que es similar a SOWA (Véase la Figura 3.1).

Analizando la estructura de SONA, la implementación es directa, implementando cada una de las capas que lo conforman y manteniendo la independencia de las capas en la construcción de un sistema [6].

La capa inferior, llamada **Infraestructura en Red**, trata todo lo concerniente al cliente y servidor de Web. En esta capa se maneja la administración de proveedores y consumidores. Se analiza la funcionalidad para saber la mejor forma de comunicarlos por servicios remotos.

La capa intermedia de la SONA es **Servicios Interactivos**. Todo aquéllo que rodea a los servicios ocurre aquí, a excepción de la definicion de los servicios. Cuando en la capa inferior están definidos el proveedor y el consumidor, la capa de interactividad se encarga de la exposición y almacenamiento de los servicios del proveedor. Estas acciones sólo se definen en esta capa. La

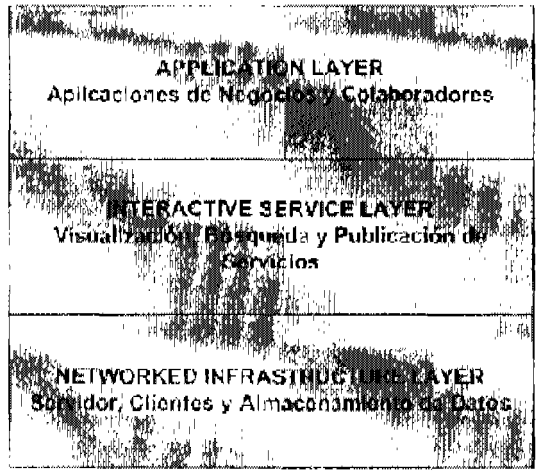

Figura 3.1: Diagrama de bloques de SONA de Cisco Systems

funcionalidad pertenece a la capa superior.

Finalmente la capa superior de la SONA es de **Aplicación**. Aquí ocurre la comunicación entre el servidor y el cliente. A diferencia de la SOWA, la localización remota de los servicios es más compleja. SONA interconecta la localización de los servicios, disponiendo de una red de servicios compartida llamada *Intelligent Information Network*. Cuando un cliente requiere de un servicio, la red de servicios provee la dirección, información e incluso conexión con el servicio remoto en el momento de ejecución del cliente. Por ejemplo, si los servicios están disponibles por WS, en SONA se interconectan las unidades UDDI de cada servicio.

Desventajas de SONA

La principal desventaja de SONA es la falta de apoyo por parte de las compañías importantes de desarrollo de software. Incluso compañías subsidiarias de Cisco e IBM como SAP AG han desacreditado la idea de Cisco. Según *IT World Canada* [24], "la idea de Cisco no determina la forma cómo la red compartida funciona en realidad para integrar cualquier servicio de manera directa". En el mismo documento, se menciona que Cisco supone

que la forma de comunicar a un cliente con un servidor es siempre la misma sin importar la aplicaciónes involucradas. La falta de apoyo, documentación y disponibilidad de SONA, dificulta encontrar condiciones en un ambiente de programación Web para obtener interoperabilidad entre sistemas, por lo que SONA no funciona para el objetivo de esta tesis.

3.2. Service-Oriented Modeling and Architecture

La **Arquitectura y Modelado Orientados a Servicios** (o SOMA, en inglés), es la orientación a servicios desde el punto de vista de IBM. También muestra una estructura de capas, esto porque considera la parte de diseño de los servicios remotos como la parte más importante del desarrollo de un sistema orientado a servicios. La SOMA contiene en su capa más alta a toda la definición de SOWA. El énfasis en los servicios es la característica principal de SOMA. En la Figura 3.2 se presenta la estructura de SOMA.

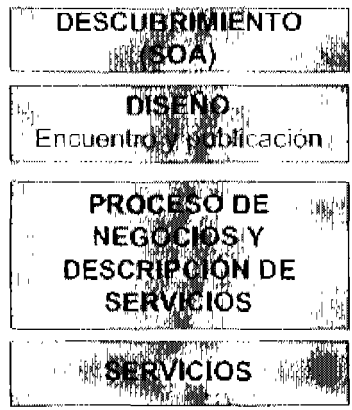

Figura 3.2: Diagrama de bloques de SOMA de IBM

La idea de IBM está en el modelado y diseño de los servicios. La creación de módulos, que posteriormente se convertirán en servicios, ocurre en la capa más baja llamada **Servicios**.

Una vez que se tiene el servicio remoto diseñado, es necesario describirlo. En la capa de **Proceso de Negocios y Descripción de Servicios**, los

servicios están formados por acciones y en esta capa se definen las condiciones a partir de las cuales estas acciones son expuestas como parte del servicio remoto.

En la capa de **Diseño** se encuentran las etapas de Encuentro y Publicación de los servicios remotos de SOWA. Se definen los mecanismos para exponer el servicio remoto. Cuando esta capa se completa, ya se tiene un servicio diseñado y con una estrategia definida para exponer sus acciones y consumirlos.

Finalmente en la capa llamada **Descubrimiento** ocurre finalmente la definición de proveedor y consumidor del servicio. Dado que estos últimos pueden cambiar con el tiempo, es importante la estandarización de la publicación y exposición del servicio en la capa anterior.

Con todo lo anterior ya se puede implementar el servicio remoto teniéndose todo lo necesario para exponer y consumir el servicio remoto. Esto se consigue solamente con la herramienta de desarrollo de software de IBM, Websphere [22].

Desventajas de SOMA

La arquitectura de IBM está firmemente ligada con la herramienta de desarrollo de software *IBM Websphere Business Integration Solutions*[1]. Con esta herramienta, es posible el modelado de sistemas de software, implementación y reportes de desarrollo además de monitoreo de los procesos que ocurren en el consumo o exposición de servicios.

Por esta misma razón, Websphere oculta el proceso de consumo y exposición de un servicio remoto. A pesar de que SOMA utiliza las ideas más importantes de la orientación a servicios, no permite reconocer los elementos involucrados en la explotación de servicios y por tanto, el estudio de SOMA, resulta inútil para el alcance de esta tesis.

[1] www-306.ibm.com/software/websphere

Capítulo 4

Aplicación de la SOWA para un caso estudio

En este capítulo se analiza la estructura de un módulo para ejemplificar el uso de la SOWA en un caso de estudio. En la primer sección se describen las características funcionales del módulo. En la segunda sección se muestra la justificación de las características. En la tercer sección se conocen los elementos del sistema y su relación con modelos en UML y diagramas de bloques. En la cuarta sección se hace énfasis en el patrón de software MVC para la explicación de los paquetes que forman el módulo. Finalmente en la última sección se explica la exposición del servicio representativo del sistema para su potencial consumo.

4.1. Características funcionales del módulo de evaluación

Para el caso de estudio se propone el análisis de un módulo de evaluación, construído en base a SOWA y con las siguientes características funcionales:

- Abstraer los aspectos importantes que comparten los sistemas Web dedicados a la evaluación. Estos aspectos se concentran en definiciones Grading de OKI (véase sección 2.2.4). Esta definición contiene datos referentes a personajes a ser calificados, objetos a calificar, como tareas, exámenes parciales o finales y finalmente la evaluación.
- Usar las definiciones OKI para la posible creación de nuevas formas de evaluación que incluyen nuevos tipos de calificación (numérico, por

letras, FALLA y PASA) y una escala (de cero a diez, de cero a cien). Asimismo, se pueden modificar y borrar definitivamente del sistema. Las evaluaciones asignadas pueden ser guardadas en una base de datos y estar disponibles para que sean vistas por los usuarios del módulo de enseñanza.

- Exponer la creación de objetos calificables y evaluaciones como un servicio remoto (véase sección 1.4). Aquellos consumidores que requieran usar el servicio remoto pueden hacerlo usando diferentes lenguajes (Java, .NET, PHP, etcétera) y usarlo desde diferentes plataformas o sistemas operativos (Windows, Linux, Unix, etcétera).

- Acceder al servicio remoto de evaluación a través de Internet y pasar a través de dispositivos de seguridad del sistema operativo conocidos como Firewalls (ver sección 2.2.3).

- El módulo también funciona como una aplicación Web (véase sección 2.1.3). El objetivo de la aplicación Web es demostrar que las calificaciones han sido guardadas, y por tanto, mostrar la interoperabilidad entre el cliente y servidor. Una vez que los datos son mandados desde el cliente, el servidor le manda al cliente la fecha de creación de la calificación.

- La aplicación Web cuenta con validaciones según la W3C como accesibilidad (WAI-AA), la forma de codificación de las páginas dinámicas (XHTML 1.0 Transitional) y las páginas de estilos (CSS)[1].

4.2. Estructura general del módulo de evaluación

La estructura general del módulo de evaluación se presenta en la Figura 4.1. En ella se especifican sus componentes desde el punto de vista de los patrones de software y las herramientas de software involucradas. Se muestra el patrón MVC para denotar la estructura básica del sistema Web, además de otros patrones como *Decorator, Dependency Injection, Façade* y *Proxy.*

[1]WAI-AA, XHTML y CSS son certificaciones de W3C para las páginas que cumplen con ciertas características mínimas para ser comprendidos por un usuario promedio o manipulados por un programador con conocimientos básicos de creación de páginas Web. Veáse más al respecto en `www.w3.org`

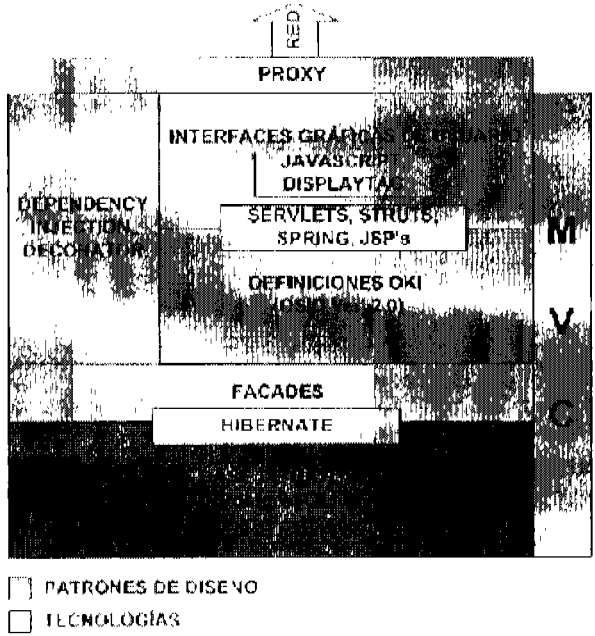

Figura 4.1: Diagrama de bloques del módulo de evaluación

Visto a nivel de paquetes que conforman al sistema, la Figura 4.2 muestra la forma en la que están relacionados. Las definiciones OKI están contenidas en un *jarfile* llamado `org.osid.Grading`. Las implementaciones de OKI están contenidas en el paquete `impl`. La definiciones de *Grading* son expuestas como servicio remoto con las clases contenidas en el paquete `server.axis`. Las implementaciones dependen del paquete `facades`, donde se pueden hacer cambios en la base de datos con ayuda de Java Beans contenidas en `beans`. Parte de la manipulación de los datos conseguidos por la implementación es usada por los paquetes `actions` y `decorators` para el manejo de las acciones Web y transformaciones de datos, respectivamente.

4.2.1. Justificación de decisiones generales

A continuación, la justificación del uso de la orientación a servicios para la creación de sistemas en Web. Se justifica el uso de la SOWA, WS y las definiciones de OKI para la creación de sistemas enfocados a la educación en línea.

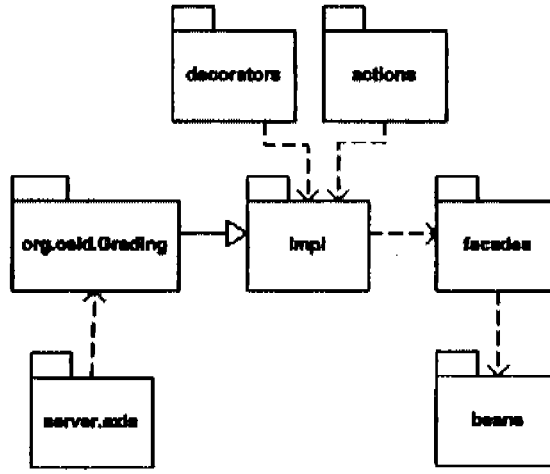

Figura 4.2: Diagrama de paquetes mostrando la estructura general del módulo de evaluación

Uso de SOWA

La SOWA es una descripción de Web para reusar módulos de *software* por medio de la red. La creación de módulos es común en el desarrollo de un sistema de software. Estos módulos deben ser independientes entre sí. Cuando esta idea se traduce en el entorno Web, los módulos son expuestos para ser usados desde cualquier ubicación en la red.

La idea de considerar a los módulos como servicios obedece a la necesidad de independizar a los módulos del sistema al que originalmente pertenencen y definir sus capacidades. La principal capacidad está en el reuso de código. De esta manera, integrar un servicio a un sistema se traduce en reuso del código de un módulo en un sistema cualquiera.

Las demás capacidades de la definición de los servicios están contenidas en la SOWA. Se definen fases para la exposición y consumo de servicios, así como el cliente y servidor como entidades necesarias para conseguir interoperabilidad. Además existen herramientas disponibles para facilitar la interacción con servicios.

Así pues, si se toma como objetivo el reuso, centralización de información y la interoperabilidad de módulos por medio de la red para beneficio general, entonces la orientación a servicios resulta una solución adecuada.

Uso de Web Services

Los WS son una forma de explotar y comunicarse con un servicio remoto con la características de comunicarse con servicios escritos en cualquier lenguaje de programación, así como el envío y recepción de mensajes por medio de protocolo HTTP (vease sección 2.2.3).

Estas características permiten a los clientes del servicio que lo consuman de diferentes maneras. Por ejemplo, en el capítulo 5 se hacen experimentos de consumo desde diferentes lenguajes de programación enfocados a Web como Java, AJAX, .NET y PHP. Aunque no es el objetivo de esta tesis, el consumo del servicio también ocurre desde diferentes sistemas operativos.

Uso de las definiciones OSID de OKI

La creación de un módulo de *software* dirigido a la evaluación está apegado a sistemas de enseñanza en línea. En esta tesis se busca mostrar que los servicios pueden ser integrados y usados de la manera que se requiera, sin importar hacia dónde esté enfocado el sistema Web. Las definiciones de OKI muestran la estructura mínima de construcción de un sistema de evaluación. Se ahorra en tiempo de diseño de la arquitectura general del sistema, permitiendo el paso directo a la implementación de las interfaces que proponen.

Otro beneficio está en la publicación de estas convenciones de implementación. Como es una iniciativa de varias universidades, las definiciones de OKI son previamente conocidas, favoreciendo el uso de los servicios una vez que son implementados. Cuando los servicios son construídos a partir de estas definiciones son mencionados en la página de la comunidad de OKI [2].

4.3. *Model*

En este componente se mantienen y cambian los datos del sistema. Las estructuras de datos usadas en las definiciones de OKI son mapeadas de alguna forma a las estructuras nativas del manejador de bases de datos

[2] okicommunity.org

PostgreSQL (véase sección A.8). Así, por ejemplo, la estructura *Type*, formada por un `authority`, `domain`, `keyword` y `description`, es mapeada como cadenas de texto.

En la base de datos existen dependencias como llaves foráneas para permitir la creación de *Grade Records* sólo cuando existen *Gradable Objects* que lo contengan. Además se crean identificadores a nivel base de datos para mantener orden de los valores que entran en cada una de las tablas. Esto se consigue por medio de funciones de secuencia nativas de PostgreSQL (Véase apéndice sección A.8).

El paso de información desde el manejador de bases de datos a clases Java es por medio de Hibernate (Véase apéndice sección A.4). Existen POJO's o Java Beans como resultado de la traducción de datos. Una vez que se tienen los Beans, ya pueden ser manipulados por las clases implementadas de OKI.

Para el uso de una herramienta de persistencia de datos como Hibernate (A.4), es necesario que existan Java Beans con los respectivos datos a guardar en la base de datos. Es por ello que existen estos Beans en un paquete del código del caso de estudio[3]. Los Beans tienen los datos a mapearse en la base de datos y para que esta acción ocurra es necesario dar a conocer estos Beans a Hibernate por medio de sus archivos de configuración. Cuando esto ocurre, Hibernate realiza una conexión a la base de datos vía JDBC (A.4).

Con las sesiones abiertas a la base de datos es necesario implementar cada acción posible como agregar, borrar y actualizar los datos. Así pues, algunas interfaces de OKI requieren realizar estas acciones o consultas directamente a la base de datos.

4.3.1. Façade

En esta sección se explica el uso del patrón Façade para explicar la forma en la que se encuentran guardados los datos en bases de datos. A pesar de que las acciones de insertar, borrar y actualizar son propias del controlador de MVC, se incluye este problema en esta sección para resolver el problema

[3]Contenido en el paquete edu.unam.okigrading.beans.

de persistencia de datos y explicar su interacción con la biblioteca Hibernate.

Problema: Controladores Hibernate

Las acciones de agregar, borrar, actualizar datos y consultas extras son parte de las acciones propias de las implementaciones de OKI. Como se ha elegido Hibernate para tener sesiones persistentes, significa que algunas clases como `GradingManagerImpl`, `GradableObjectImpl` o incluso `GradeRecordImpl` tienen la capacidad de crear *GradeRecords*. En la Figura 4.3 se denota este hecho. Los bloques denominados *GradableObject*, *GradingManager* y *GradeRecord* son objetos con la capacidad de realizar las acciones de *insert()*, *delete()* o *update()*. Las flechas representan esta capacidad.

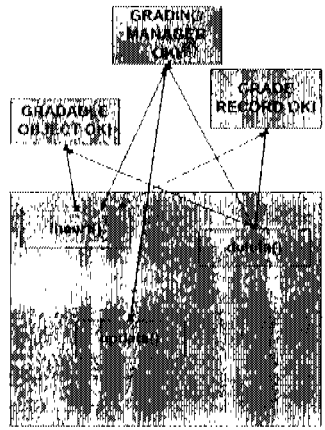

Figura 4.3: Diagrama de bloques de la estructura del modelo con múltiples dependencias

El número de flechas en la Figura 4.3 muestra las acciones compartidas por las tres clases. Esta situación de dependencia puede minimizarse con la creación de una sola clase capaz de realizar las acciones y, por medio de mensajes, ser accedidas desde las clases dependientes. La Figura 4.4 muestra la forma en que esta clase fachada (llamada "Clase Hibernate") funciona. Por un lado, se crean Java Beans con los datos disponibles y por los métodos de la biblioteca Hibernate se acceden los datos en la base de datos. Esto se consigue por medio del lenguaje de consultas nativo de Hibernate llamado

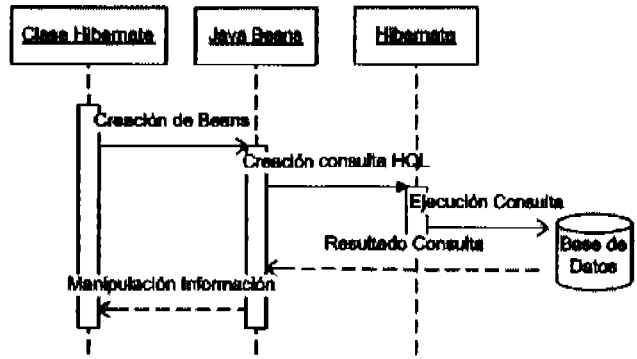

Figura 4.4: Diagrama de secuencia para la acción de una clase Hibernate

HQL (véase apéndice sección A.4). Cuando la consulta o acción en la base de datos ocurre, se obtiene un resultado el cual puede ser manipulado posteriormente por la fachada que finalmente se trata de la implementación del patrón de diseño Façade.

El resultado de la creación de estas clases es la minimización de dependencias. Comparando las Figuras 4.3 y 4.5 se nota una disminución en las flechas. En adelante las clases que requieran acciones directas sobre la base de datos deberán hacerlo desde fachadas pensando en la futura adición de funcionalidades.

Finalmente la clase Hibernate Façade es implementada por cada una de las estructuras del sistema y, por este medio, es posible insertar, borrar y actualizar. Las clases Hibernate están contenidas en el paquete `edu.unam.okigrading.facades`.

4.4. *Controller*

El componente **Controller** es la implementación de las definiciones de OKI para Grading, además de la implementación de las llamadas *compartidas*: *Type, Id* y *Agent*. Las estructuras básicas de *GradableObject, GradeRecord* y el manejador principal de la definición *GradingManager* están contenidas en el paquete `edu.unam.okigrading.impl`. Estas clases toman las "clases Hibernate" de la sección anterior para hacer permanentes los cam-

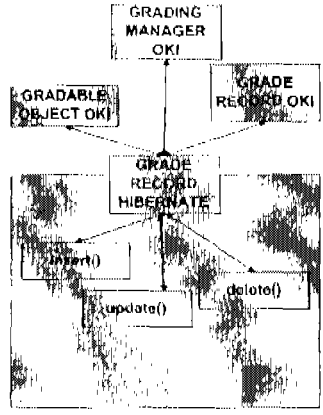

Figura 4.5: Diagrama de bloques del modelo con una sola dependencia

bios que se hacen desde el Controller.

Además, las clases contenidas en el paquete `decorators` y `actions` toman los datos proporcionados por la implementación de OKI para preparar los datos y que sean usados por *View* para ser mostrados.

4.4.1. Dependency Injection

En esta sección se revisa una aplicación del patrón Dependency Injection en un problema del módulo de evaluación. De esta manera se busca que las clases existentes en el código sean dependientes de las interfaces de OKI en lugar de sus implementaciones.

Problema: Dependencias de la Implementacón

Las clases contenidas en el paquete `actions` y `decorators` dependen de la implementación del paquete `impl` para hacer sus funciones en el sistema. Sin embargo, la implementación puede ser cambiada por cualquier otra. Los cambios en todas las clases que dependen de ella implican nuevas etapas de pruebas para comprobar si los cambios resultaron. La Figura 4.6 es un diagrama de paquetes para mostrar esta dependencia.

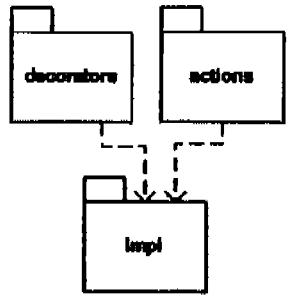

Figura 4.6: Diagrama paquetes muestra la dependencia de implementaciones

El problema se soluciona haciendo al sistema "dependiente de las interfaces" en lugar de "dependiente de implementaciones". Esto significa que las instancias de las clases se dan a partir de la interfaz a la que pertenecen (véase sección 2.3.4). Así, las dependencias pueden ser cambiadas sólo cuando los objetos son instanciados y no en cada llamada a sus métodos. La Figura 4.7 muestra el cambio de la estructura del sistema con interfaces en lugar de implementaciones. Las clases del paquete `impl` son las implementaciones de las interfaces OKI contenidas en el *jarfile* de Grading `org.osid.grading`.

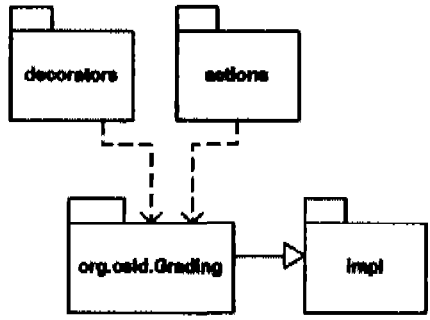

Figura 4.7: Diagrama paquetes muestra la dependencia de interfaces

Si la "inyección de dependencias" o asignación de implementaciones es por medio de un archivo de configuración y por la herramienta Spring Framework, entonces se evita el paso de cambiar las clases con cada instanciación.

Spring ve a las clases como un Java Beans, por lo que hace la asignación por medio de un *setProperty()* donde Property es el nombre de la interfaz. Por ejemplo, la asignación de la implementación de la intefaz *AgentManager*, llamada *AgentManagerImpl* en cualquier clase es por medio de un método llamado *setAgentManager()*. Spring toma estos métodos en cada clase y asigna las implementaciones en tiempo de ejecución. Si se requiere usar otra implementación de Agent Manager, se describe la ubicación de la nueva impleméntación en el archivo de configuración de Spring.

4.4.2. Decorator/Wrapper

En esta sección se utiliza el patrón Decorator para resolver la forma en la que son llamados y manipulados diferentes clases que tienen el mismo comportamiento aunque con diferentes estructuras de datos. La solucón radica en unificar todas estas clases y "sumarles" propiedades cuando sen llamados.

Problema: Múltiples Iteradores

Los iteradores de las definiciones de OKI funcionan similarmente aunque manejando diferentes tipos de estructuras de datos. Por ejemplo se tienen *GradableObjectIterator, GradeRecordIterator* o *TypeIterator*. Cada uno de ellos tiene los métodos de *next()* para obtener el elemento al que se apunta en el iterador y *hasNext()* para mover el apuntador a la posición siguiente del iterador.

Al crear un iterador se debe especificar el tipo de dato que va a contener. Sin embargo en el caso de *GradeType* o *GradeScale* el tipo de dato es siempre *Type*, pero aún así debe ser especificado. Esto significa la implementación de cada uno de los tipos posibles que existen en las definiciones OKI para iteradores.

La solución propuesta radica en la creación de una clase que reciba el nombre de la estructura de datos y los elementos que lo conforman y devuelva siempre una estructura del tipo *TypeIterator*. La idea puede extenderse a todas las estructuras de datos implementadas y regresar un iterador del tipo *Iterator* con la estructura indicada en su interior. En el paquete `utils` existe la clase `IteratorWrapper`, en la Figura 4.8 se muestra como funciona.

Usando esta clase se pueden acceder de la misma forma a los datos en el interior del iterador y dependiendo del caso, los datos pertenecen a *Gradable*

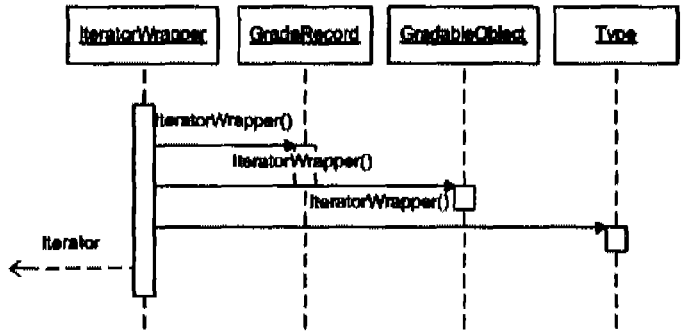

Figura 4.8: Diagrama de secuencia de la acción de la clase IteratorWrapper

Object, Grade Record o *Type* sin necesidad de cambios al momento de su invocación. A pesar de que los iteradores son de un tipo en particular, son "envueltos" por una capa de abstracción que mantiene los mismos nombres de métodos por lo que la invocación de alguno de estos es en realidad una llamada a la estructura en particular.

4.5. *View*

En este componente, las páginas Web son dinámicamente creadas por medio de Java Server Pages (o JSP, véase seccion 2). La información obtenida y manipulada por el *Model* y *Controller* es guardada en los diferentes niveles de memoria Web y usada para ser mostrada en las páginas dinámicas. En el caso de estudio, son tomadas las estructuras de *GradableObject, GradeRecord, Type* y sus iteradores (todos son parte de las interfaces de OKI).

4.5.1. Decorator/Wrapper

El patrón Decorator es usado una vez más para resolver otro problema. En esta ocasión, a un número `long` se le suma una propiedad para que sea visto como un `java.util.Date`, mientras que a partir de un identificador se obtiene el nombre respectivo a ese identificador.

Problema: Fechas, Gradable Objects y Agents

Se tienen las fechas de creación y los identificadores de algunas estructuras en la base de datos. Por ello, al momento de mostrar la información en el sistema, solo aparecen estos datos, generalmente, sin sentido para el usuario convencional. En el caso de listados de las estructuras *Gradable Object* y *Grade Record*, están formados por identificadores de la base de datos o estructuras *Id* de OKI (`org.osid.id`). En el caso de las fechas, el valor es un número a 64 bits que representa la fecha en milisegundos.

La solución propuesta a este problema está en la creación de una clase que reciba estos identificadores o fechas y devuelva valores reconocibles por un usuario. Esto significa que deben aparecer nombres de las estructuras en lugar de sólo los identificadores y las fechas en un formato comprensible como "dia/mes/año" en lugar de números a 64 bits.

La herramienta llamada DisplayTag (véase apéndice sección A.9) es una de muchas formas de iterar y mostrar los datos de una estructura por medio de tablas de HTML. Se usa en este módulo de evaluación porque también implementa el patrón de diseño *Decorator* usando una clase implementada para hacer esta labor de transformación de datos a mostrar. Estas clases para decoración están ubicados en el paquete `decorators`. DisplayTag hace llamados a las clases de este paquete para cambiar la vista de las estructuras de datos. En la Figura 4.9 se muestra la forma en la que cambian las fechas de formato.

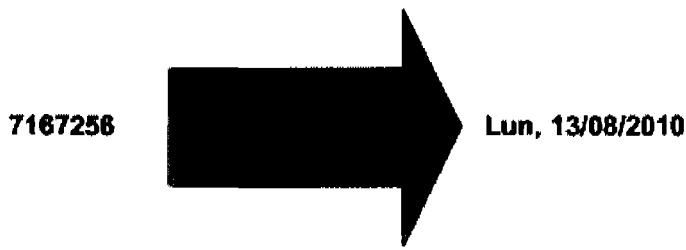

Figura 4.9: Diagrama de bloques del uso de decorator en fechas

En las Figuras 4.10 y 4.11 se muestran los cambios de formato a partir de los identificadores. Los datos son obtenidos desde una base de datos local. En la sección 4.7.5, la obtención de los datos provienen de la base de datos

del servicio remoto que es consumido.

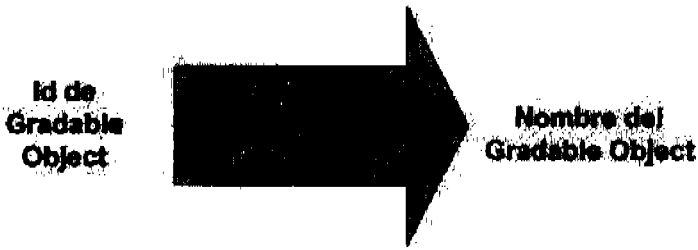

Figura 4.10: Diagrama de bloques del uso de decorator en gradable object

Figura 4.11: Diagrama de bloques del uso de decorator en agent

4.5.2. Proxy de Referencia Inteligente / Proxy Remoto

En esta sección se mezclan las ideas de Proxy de Referencia Inteligente y Remoto para resolver un problema de decoración de agentes de la sección anterior. Si los nombres están localizados en diferentes bases de datos, el módulo busca en qué base datos se encuentra con un proxy de referencia inteligente y establece una conexión con un proxy remoto para obtener la información.

Problema: Múltiples consumos de servicios

El servicio *AgentManagerService* es un servicio que una vez que es invocado con un número identificador, devuelve el nombre que corresponde

a ese identificador localizado en su base de datos. El sistema consume este servicio para la decoración de los agentes (véase sección anterior 4.5.1). El servicio de agentes existe en dos lugares remotos diferentes y se debe tener el consumo de los dos. Para fines de diferenciar ambos servicios serán llamados UXMAL y PALENQUE.

La solución propuesta está en la creación de clases que funcionen como *Proxy de Referencia Inteligente* y que puedan elegir cuál implementación usar de acuerdo con ciertas condiciones. Se realizan las descripciones de ubicación de cada servicio y se guardan en un *HashMap*. De acuerdo a una convención, los identificadores deben ser escritos en forma de URN, es decir, mencionando una palabra para conocer el servicio a usarse y el identificador mismo a buscar separados por dos puntos. Por ejemplo "PALENQUE:2" significa que el servicio a consumir es el llamado "PALENQUE" y el identificador a buscar en ese servicio es el "2".

El *Proxy* busca en el HashMap toda la configuración respectiva al servicio remoto y genera un *Proxy Remoto* para realizar las llamadas correspondientes. Como los listados contienen varios *Gradable Objects*, esta acción la realiza por cada uno de los elementos listados. La creación de los *Proxy Remotos* que hacen las llamadas remotas, las hace Spring por lo que la creación de estas clases se hacen en tiempo de ejecución ahorrando memoria en la creación de todos los *Proxies*. La Figura 4.12 muestra una llamada de este tipo que puede ser indistintamente a cualquiera de los dos servicios remotos disponibles.

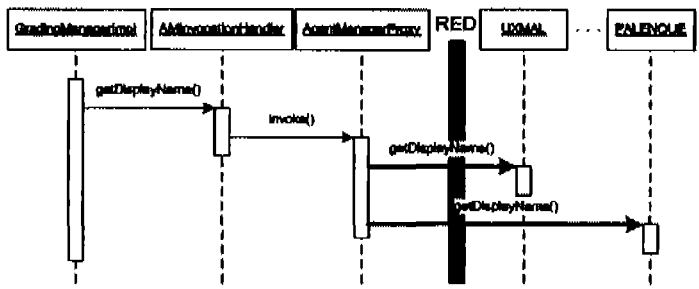

Figura 4.12: Diagrama de secuencia de la acción de los proxies

4.6. Exposición del servicio remoto

La exposición del sistema como un servicio remoto es la traducción de la implementación de las definiciones OKI en un formato XML en el llamado WSDL. Los métodos mínimos que deben ser expuestos para crear una calificación son *createGradableObject()* y *createGradeRecord()*. Para su mejor comprensión, se utilizan los diagramas de clases UML extendidos vistos en la sección 2.2.3. En la Figura 4.13 se muestra el diagrama general que representa el WSDL del servicio remoto.

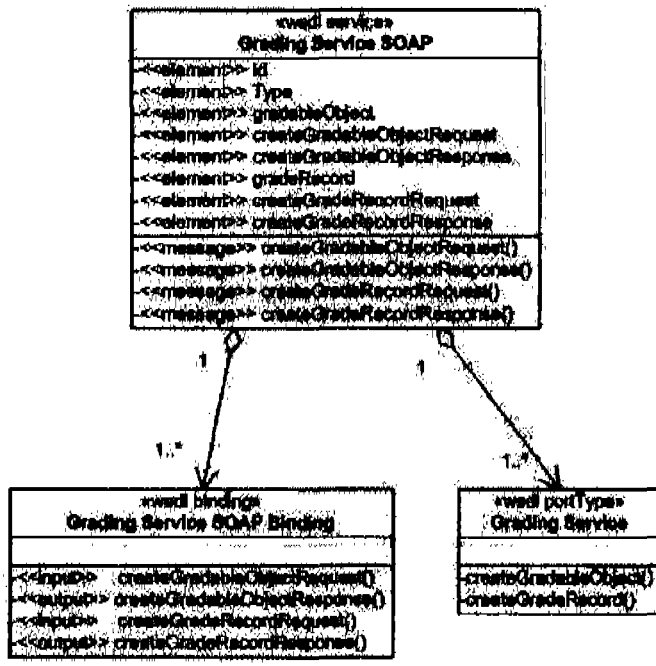

Figura 4.13: Diagrama de clases del WSDL

La clase principal es la definición de servicio. En ella se especifica su función con el estereotipo de *«wsdl service»*. Los elementos marcados como *«element»* son los atributos usados en la exposición del servicio remoto. Como estos elementos no están contemplados como nativos en el esquema de XML (como cadenas o números enteros), son considerados atributos

complejos y por tanto deben ser extendidos de las especificaciones de XML por medio de un esquema adicional. Nótese que los atributos de *Id, Type, GradableObject* y *GradeRecord* de OSID son mencionados en el documento. Además de los anteriores se deben especificar los nombres de los métodos. Para ello están los atributos *createGradableObjectRequest* y *createGradableObjectResponse* para crear un objeto calificable y los atributos *createGradeRecordRequest* y *createGradeRecordResponse* para crear finalmente la calificación. Estas estructuras de datos se presentan en las Figuras 4.14, 4.15, 4.16 y 4.17. La razón de crear atributos con el mismo nombre de los métodos a exponer es definir la estructura de datos a mandarse y recibirse durante el proceso de interoperabilidad.

Bajo el estereotipo de «*message*» se mencionan finalmente los métodos. Cuando el nombre de método tiene el sufijo *Request* se refiere a la serie de datos necesarios para que el servidor pueda realizar su función. En contraste, si el sufijo es *Response* se trata de los valores de regreso que tienen los métodos. Por ejemplo, para crear una nueva calificación, el *Request* necesitan identificadores de Gradable Object, Agent y Grade Record Type además del valor de la calificación. El *Response* del método regresa además la fecha de creación de la calificación como un número expresado en 64 bits[4].

En el «*wsdl binding*» se define la unión de los métodos y las estructuras de datos definidas como atributos. Bajo el estereotipo de «*input*» / «*output*» los atributos son unidos a un tipo de mensaje. En el binding se establece la forma de comunicación como tipo SOAP y HTTP.

Finalmente el denominado «*wsdlportType*» de la Figura 4.13, engloba los mensajes y los atributos de la definición del servicio. Los nombres definidos aquí son los mismos de las definiciones OKI en la interfaz *GradingManager*. Es decir, solo se mencionan los métodos *createGradableObject()* y el *createGradeRecord()*.

[4]También conocido como *long*.

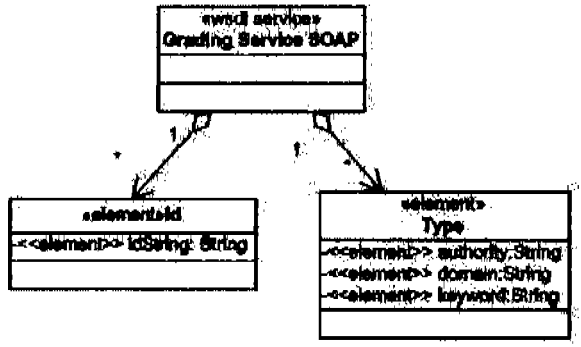

Figura 4.14: Diagrama de clases de Id y Type según OSID

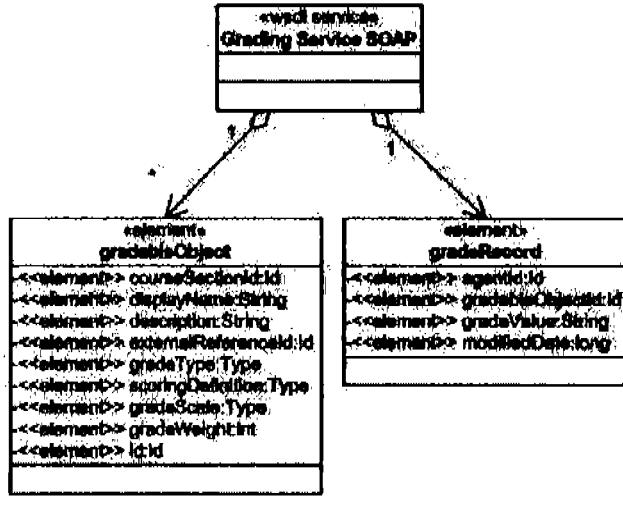

Figura 4.15: Diagrama de clases de GradableObject y GradeRecord según OSID

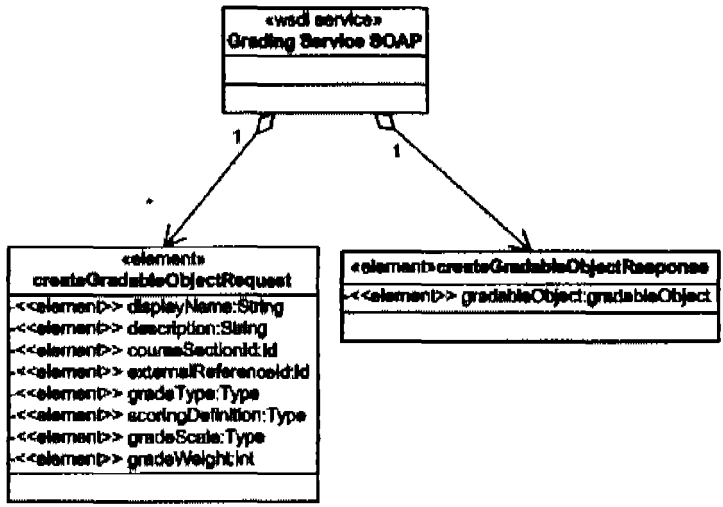

Figura 4.16: Diagrama de clases para atributos necesarios para crear un objeto calificable

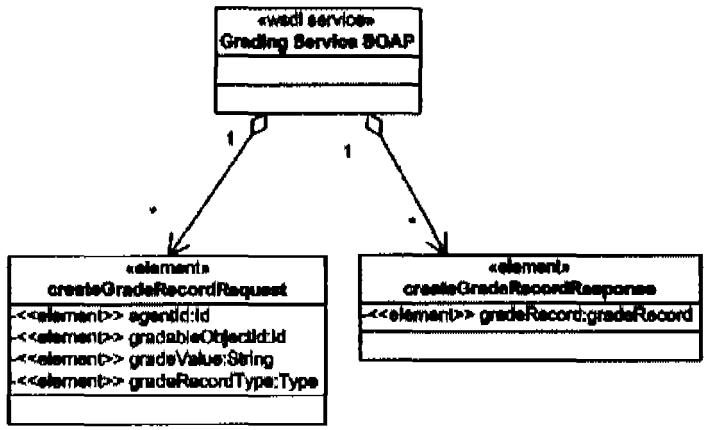

Figura 4.17: Diagrama de clases para atributos necesarios para crear una calificación

Hasta ahora se tiene el archivo WSDL describiendo al servicio remoto y la implementación por detrás del mismo. Solo falta su verdadera exposición en la red. Para ello se usa un servidor de WS. En este caso particular se usa el servidor Axis que se encarga de exponer el WSDL y responder a mensajes enviados a este servicio remoto. El servidor y el servidor de WS usualmente son el mismo elemento. Sin embargo, esto puede separarse para mantener cierta independencia entre módulos o diferentes versiones del mismo servicio remoto. La Figura 4.18 muestra esta forma de comunicación.

Figura 4.18: Diagrama de bloques de las acciones de consumo de un servicio remoto

4.7. Escenarios

A continuación se presentan algunas acciones comunes en el módulo de evaluación. Para su mayor comprensión, estas acciones están descritas de la manera más simple posible. Es por ello que, validaciones de datos y diálogos de confirmación de acciones han sido evitadas.

4.7.1. Creación de una calificación de manera local

La creación de una calificación es el punto mas importante del sistema. En la implementación de OSID (véase sección 2.2.4) en la clase `GradingManagerImpl` se llevan a cabo estas acciones. Para crear una calificación se necesitan los siguientes pasos:

1. Se crea un "objeto calificable" por medio de la función *createGradableObject()*. Para ello, se necesitan los valores de *scoring definition*, *grade scale* y *grade type*, nombre del objeto calificable y su respectiva descripción.

2. Una vez que está creado el objeto calificable, es momento de crear propiamente la calificación. Para ello se usa la función *createGradeRe-*

cord() con los datos identificadores del objeto calificable recién creado, el estudiante, grade record type y el valor de la calificación.

La Figura 4.19 muestra el diagrama de secuencia de la creación de un objeto calificable. En el mensaje de "Datos Iniciales" se refiere a la etapa a partir de los cuales se obtienen los identificadores de los *Types* necesarios. Esto es por medio de los métodos *getGradeScales()*, *getScoringDefinitions()* y *getGradeTypes()* en la clase `GradingManagerImpl`. Una vez que se tienen los argumentos, se llama a la clase `GradableObjectHibernate`. Esta clase debe de tener los mismos argumentos que `GradingManager`, pero con un "mapeo" de los valores a datos nativos del manejador de la base de datos. Este mapeo se consigue gracias a la clase `GradingUtils`, transformando los *Types* en cadenas de caracteres.

Una vez conseguida la traducción de datos, se ejecuta finalmente la creación de un objeto calificable, guardando todos los datos en la base de datos. Para ello, es necesario crear un Java Bean con esos datos pues es la única forma de ejecutar acciones con Hibernate. Con esto se devuelve un nuevo `GradableObjectImpl` para ser usado en la creación de la evaluación.

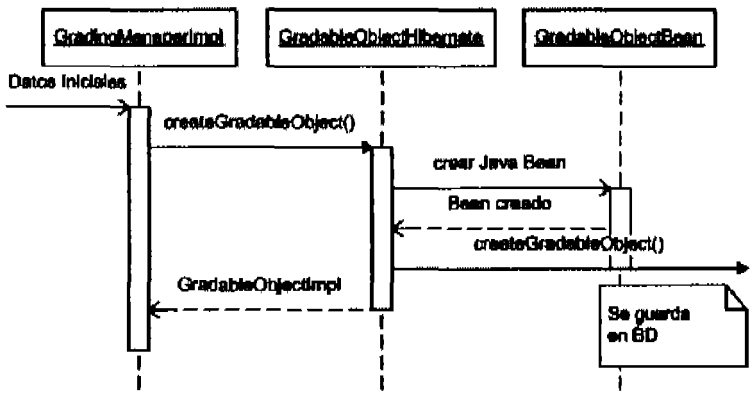

Figura 4.19: Diagrama de secuencia para la creación de un objeto calificable

El segundo paso de este escenario (véase Figura 4.20), comienza con otra etapa de "Datos Iniciales". Hasta este punto ya se tiene creado un objeto calificable. Dicho objeto tiene una serie de atributos del que interesa su iden-

tificador. También es necesario el identificador de un estudiante previamente insertado en la base de datos. Con estos valores ocurre una acción similar de la creación del objeto calificable: se llama a la clase `GradeRecordHibernate` que manda la creación de un Java Bean llamado `GradeRecordBean` el cual finalmente queda guardado en la base de datos. El resultado es un objeto `GradeRecordImpl` con ciertos atributos extras a los argumentos iniciales como la fecha de creación de la evaluación.

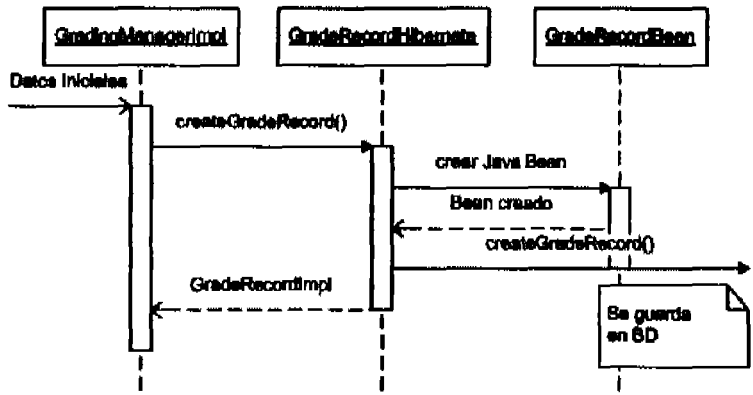

Figura 4.20: Diagrama de secuencia para la creación de una evaluación

4.7.2. Actualización de una calificación de manera local

En este escenario también se necesita la implementación de *Grading Manager*. Es una medida de seguridad para la creación de una evaluación si ésta ya existe en el sistema. Los identificadores del estudiante y el objeto calificable, así como el *grade type* forman una llave única, es decir, esta combinación es irrepetible en el sistema por lo que si se quiere cambiar alguno de estos datos, sólo puede ser *grade value* (es decir, la calificación en sí). El escenario sigue los siguientes pasos.

1. Una vez que se tienen los identificadores del estudiante y del objeto calificable, se siguen los mismos pasos de la creación de una calificación del escenario anterior ilustrados en la Figura 4.20.

2. Si la calificación ya existe en el sistema, entonces solo se encarga de actualizar el dato de *grade value*, es decir, la evaluación propiamente.

La operación se ilustra en la Figura 4.21. Cuando la clase `GradeRecord Hibernate` encuentra la llave única en la base de datos, crea el Java Bean con los datos y en lugar de hacer una inserción normal en base de datos, hace una actualización del único dato disponible para ser cambiado. Nótese que si se requiere hacer un cambio en la llave única se pretendería cambiar el significado mismo de una evaluación, por lo que se recomienda borrar la evaluación y el objeto calificable y reiniciar el proceso completo de inserción descrito en el escenario anterior.

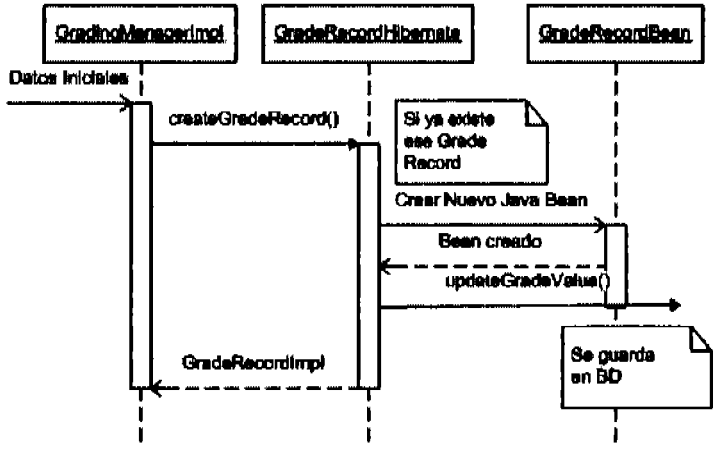

Figura 4.21: Diagrama de secuencia para la actualización de una evaluación

4.7.3. Borrado de una calificacion de manera local

El borrado de una calificación se consigue con los siguientes pasos.

1. El identificador de un *Grade Record* está formado por el identificador del estudiante, *gradable object* y el *grade record type*. Esta combinación es única por lo que una vez que se tienen estos datos, se llama a la clase `GradeRecordHibernate` y esta se encarga de borrarlo definitivamente de la base de datos.

2. Adicionalmente se puede borrar también el objeto calificable pues también se conoce su identificador.

La Figura 4.22 muestra las dos acciones por medio de un diagrama de secuencias. Cuando se borra una calificación no se afecta a las demás estructuras de datos guardadas en la base. Es por eso que el borrado es tan sencillo. Cuando se requiere borrar el objeto calificable es necesario que no exista alguna evaluación atribuída al él. Esto se traduce como una norma de seguridad para evitar que se pierdan datos de los demás estudiantes por borrar un objeto calificable.

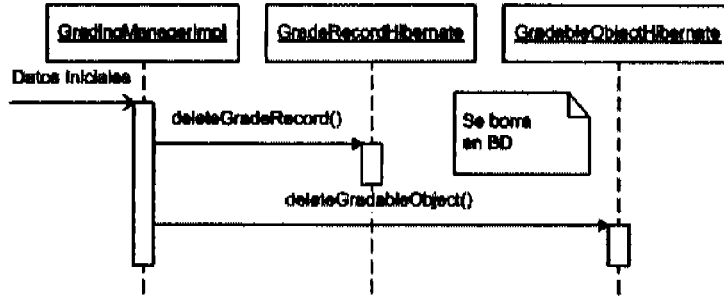

Figura 4.22: Diagrama de secuencia para el borrado de una evaluación

4.7.4. Creación de una calificación de manera remota

La exposición del servicio remoto vista en la sección 4.6 es sólo la implementación de la creación de calificaciones. Para ello es necesario la creación de un cliente del servicio quien hace las llamadas por medio de la red. Los pasos a seguir son los mismos de la creación de una calificación desde el sistema Web (sección 4.7.1) aunque con la variante de las llamadas remotas. En la Figura 4.23 se extiende un diagrama de secuencia de UML para mostrar estas llamadas a través de la red.

La clase `Cliente` es una aplicación sencilla en algún lenguaje de programación. `Proxy/Stub` es la implementación del patrón de software "Proxy Remoto". Finalmente el objeto `GradingManagerService` es la forma en la que se designa a la exposición de la interfaz `GradingManager` en la red.

Los métodos del `Proxy/Stub` tienen el mismo nombre y "casi" la misma funcionalidad de la clase original `GradingManager` a partir de la cual ocurren todas las acciones descritas en los escenarios anteriores. La mímina diferencia entre el `Proxy/Stub` y `GradingManager` radica en que el primero hace llamadas a través de la red mientras el segundo sólo las hace de manera local. El `Proxy/Remoto` implementa en algún momento *Sockets* y conexiones remotas para mandar y recibir datos en red.

En la Figura 4.23 la red se denota como una barra gris en medio de las llamadas entre `Proxy/Stub` y `GradingManagerService`. De esta manera el `Cliente` no está enterado de que se realizan llamadas remotas. La creación del objeto calificable ocurre de esta manera y así mismo devuelve la estructura *GradableObject*. Con esta estructura de datos, ya puede tomar el identificador necesario y usarlo en la creación de una nueva evaluación.

Finalmente existen diferencias mínimas entre el escenario de creación de una evaluación de manera remota y la que ocurre localmente. Las llamadas usan los mismos nombres, devuelven las mismas estructuras de datos

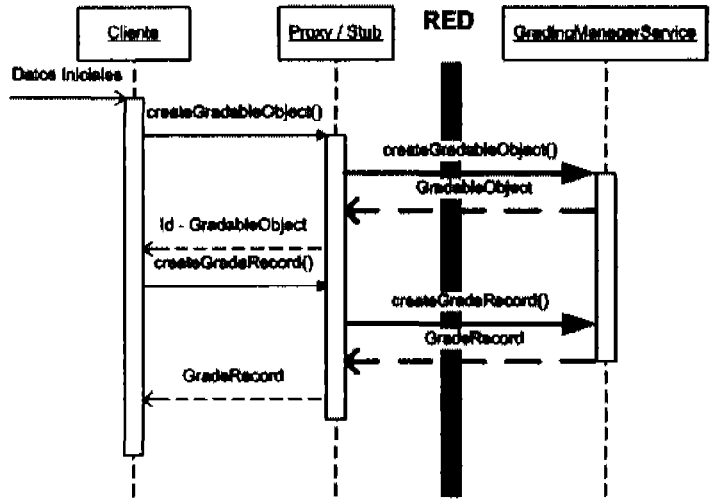

Figura 4.23: Diagrama de secuencia para la creación de una evaluación de manera remota

y mantienen la misma persistencia de datos, aunque en el caso remoto, las llamadas son por medio de la red.

4.7.5. Consumo del servicio remoto AgentManagerService

El servicio remoto AgentManagerService es consumido por el sistema del caso de estudio para obtener los nombres de los estudiantes desde una base de datos perteneciente a otro sistema Web. De esta manera los nombres de los estudiantes cambian dependiendo del servicio remoto de *AgentManager-Service* que sea consumido. La herramienta de software Spring Framework (véase apéndice sección A.3.2) ofrece la creación de la clase `Proxy/Stub` (visto en el escenario anterior) de manera explícita. Es decir, la clase `Proxy/Stub` se crea en tiempo de ejecución y es usada a manera de un Java Bean.

Los pasos del consumo se enuncian a continuación:

1. Spring Framework crea un `Proxy/Stub` que es usado por medio de Dependency Injection (véase sección 2.3.4) dentro del JSP o Servlet que lo necesita.

2. Spring realiza la llamada remota al servicio a través de la red.

3. El servicio devuelve el nombre del estudiante correspondiente al identificador enviado. Este nombre es sacado de la base de datos usado por el servicio. La localización de la base de datos es irrelevante.

4. El nombre es devuelto y usado por el *Servlet* para decorar los identificadores y mostrarlos en el JSP.

La Figura 4.24 muestra los pasos anteriores. A pesar de ser una estructura particular de este escenario, es también la forma general en la que un servicio es consumido usando Spring Framework. El JSP manda el identificador del estudiante y es recibido por el Servlet de decoración *Agent-Decorator*. Esta tiene una implementación de la interfaz *AgentManager* de OKI por *Dependency Injection*. La implementación que recibe ha sido creada por Spring y se encarga de hacer la llamada remota hasta el servicio. Esta responde con el nombre del estudiante. Cuando *AgentDecorator* recibe el nombre, se encarga de decorar el valor que a su vez es mostrado en el JSP.

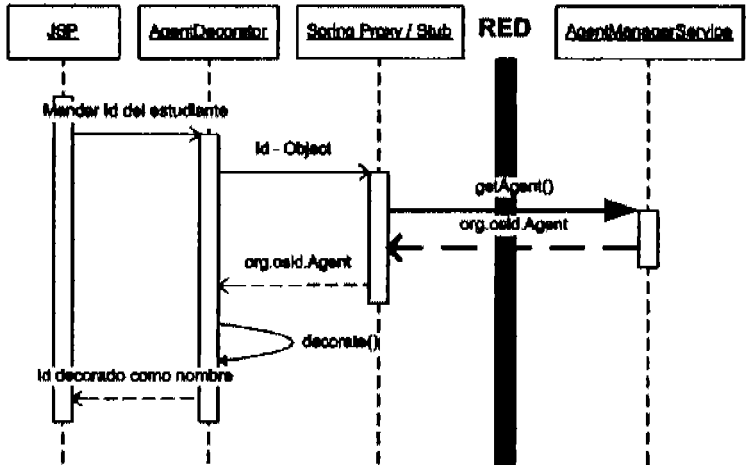

Figura 4.24: Diagrama de secuencia para el consumo del servicio remoto AgentManagerService

Capítulo 5

Experimentos de interoperabilidad

En este capítulo se documentan los resultados tras realizar dos series de experimentos de interoperabilidad. En la primera serie se toma al caso de estudio como un consumidor del servicio conocido como *AgentManager-Service* para obtener nombres de una base de datos de un sistema Web independiente y ser mostrados en el módulo de evaluación del caso de estudio. En la segunda serie se consume el servicio expuesto en el caso de estudio desde sistemas implementados en diferentes lenguajes de programación. Se hace una descripción de los experimentos mostrando los resultados esperados y los resultados obtenidos.

5.1. Consumo del servicio AgentManagerService

En la sección 4.7.5 se menciona la existencia de un servicio remoto llamado *AgentManagerService*. Se trata de un servicio que recibe un identificador único. La implementación de este servicio toma este identificador y busca en su base de datos el nombre del usuario correspondiente. Lo devuelve en forma de cadena nativa de XML. A continuación se muestran experimentos de cómo es usado en el módulo de evaluación. Primeramente con el método de *Hessian, Burlap*, y finalmente con los WS. Debe entenderse que los experimentos hechos con los métodos Hessian y Burlap tienen únicamente un fin comparativo con los WS. Como se mantiene la estructura de consumo Cliente-Servidor considérese al cliente (consumidor) con el nombre ALPHA y al servidor (proveedor) como UXMAL.

Las siguientes figuras son usadas durante los experimentos. En la primera imagen se muestra el listado de *Grade Records* antes de la acción de interoperabilidad (Figura 5.1). Nótese la columna llamada *Agent*. Muestra sólo los registros de activación[1] como identificadores de los agentes.

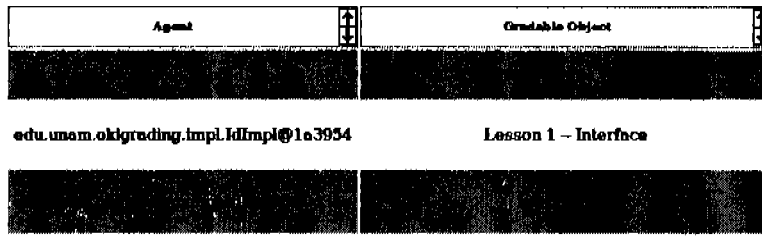

Figura 5.1: Captura de pantalla del módulo sin decoración

A continuación el listado de *Grade Records* con la vista de decoración en la Figura 5.2. Los nombres que no son encontrados en la base de datos del servidor son decorados con un nombre provisional como "Nombre generado localmente[x]". Contrariamente si los nombres son encontrados en el servidor UXMAL, manda el nombre del agent de la forma "Agente UXMAL id=[x]". En ambos casos se manda a llamar al servicio *AgentManagerService* pero la búsqueda de nombres es la que cambia de localización.

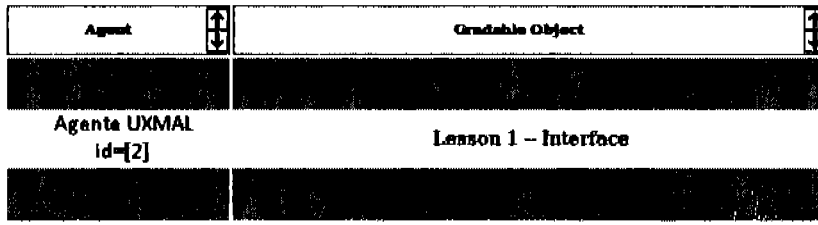

Figura 5.2: Captura de pantalla del módulo con decoración. Compárese con la Figura 5.1

[1]Una representación del objeto creado en memoria, en este caso se trata de la clase edu.unam.okigrading.impl.IdImpl

5.1.1. Método de Hessian

El método Hessian utiliza los serializadores y deserializadores de la compañia Caucho [31]. Usando este método significa, que al momento de las invocaciones remotas con *Sockets* por medio de Java RMI[2], utiliza los serializadores de Caucho en lugar de las RMI. A pesar de ser un método Java-a-Java[3], se analiza el experimento por su integración con Spring y para mantener el esquema de consumo visto en la sección 4.6. Hessian transforma un objeto en una representación de arreglo de *bytes* y no usa el formato XML.

Objetivo

1. Se busca el consumo del servicio remoto llamado *AgentManagerService*, implementado en Java, desde el módulo de evaluación para obtener los nombres de los estudiantes en el listado de las calificaciones.

2. Debe evitarse el uso de los serializadores de Java RMI pues los datos de envío y recepción del servicio son estructuras complejas de Java que no reconoce RMI.

Datos Importantes

1. La localización del servicio de agentes se encuentra en la dirección *http://uxmal.dgsca.unam.mx/strutsSpring/agent.service.*

2. El consumo del servicio puede hacerse desde Spring Framework evitando la creación de la clase `Proxy/Stub` de la sección 4.7.5.

3. El uso de Hessian no es WS. Esto significa que los mensajes para comunicación son binarios evitando la codificación de mensajes en SOAP vistos en la sección 2.2.3.

4. Se asegura el paso de mensajes síncronos pues la comunicación ocurre por HTTP y el servicio está disponible al momento de realizar el experimento.

Resultados Esperados

1. La interoperabilidad entre los dos sistemas es posible, es decir, existe un intercambio de información entre sistemas.

[2]Son las bibliotecas de Java para programación de cómputo remoto.

[3]Donde el servicio y el sistema consumidor están implementados en Java.

2. Los nombres que aparecen en el listado de *Grade Records* en la columna llamada *Agents* serán los nombres obtenidos en la base de datos del servicio *AgentManagerService*.

3. Si el identificador no aparece en la base de datos del servicio *AgentManagerService* debe mostrarse un nombre generado por el módulo de evaluación. Es decir, el sistema no detiene su funcionamiento si ocurren errores en el servidor.

Condiciones de Hardware y Software

Para el servidor del *AgentManagerService*:

1. Computadora personal Pentium IV, 1 GB de memoria RAM. Uso de red a 2 Gbps aprox. Dirección IP 132.248.200.18 (UXMAL).

2. Sistema Operativo Linux Debian 2.6 Testing, Contenedor de Servlets Apache Tomcat 5.5, Java SDK 1.5. Uso de serializadores Hessian 1.6

Para el cliente (consumidor) del servicio:

1. Computadora personal Pentium IV, 512 MB de memoria RAM. Uso de red a 2 Gbps aprox. Dirección IP 132.248.200.2 (ALPHA).

2. Sistema Operativo Linux Ubuntu 2.5 Stable, Contenedor de Servlets Apache Tomcat 1.6, Java SDK 1.6. Uso de deserializadores Hessian 1.6. Uso de Struts y Spring Framework 3.5. y DisplayTag 1.0 para la decoración.

3. El módulo Web está implementado en Java.

Desarrollo

Se define el consumo del servicio desde el archivo de configuración de Spring, especificando la ubicación del servicio. Esto crea un Java Bean con la implementación de la interfaz `AgentManager` de OKI. Por ello es posible aplicar *Dependency Injection* y usarse desde las implementaciones del módulo como si fuera clase local. Tras el evento que implica el listado de *Grade Records*, los estudiantes (o agentes según OKI), son tomados por un Servlet que evita que sean listados los identificadores y en su lugar lista los nombres. A continuación se muestra el código XML para definir el consumo del servicio desde Spring. Nótese que se menciona la ubicación del servicio y el deserializador a usar de Burlap.

```
<!-- se menciona la clase deserializadora -->
<bean id="agentManagerBean" class="org.springframework.
          remoting.caucho.HessianProxyFactoryBean">

<!-- se menciona la ubicacion del servicio -->
   <property name="serviceUrl">
      <value>http://132.248.200.18/strutsSpring/
            AgentService.service
      </value>
   </property>

<!-- se indica hacia donde deserializar -->
   <property name="serviceInterface">
      <value>
         mx.unam.dgsca.uxmal.AgentManagerService
      </value>
   </property>
</bean>
```

Para explicar mejor el proceso de intercomunicación entre los sistemas, la Figura 5.3 muestra la forma en la que se envía un mensaje al servidor. Esto se consigue con tan solo mandar el identificador del agente por medio de la red. El envío serializa el identificador en datos binarios que se transportan en la red. Cuando llegan al servidor son deserializados, procesados y reenviados de vuelta donde el deserializador Burlap se encarga de hacer datos entendibles por el cliente.

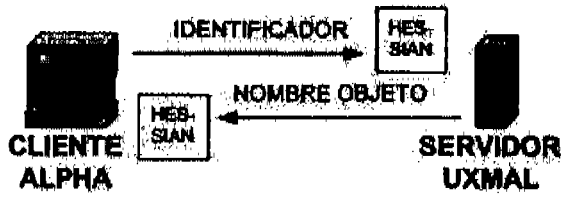

Figura 5.3: Diagrama de bloques de las acciones de consumo del servicio Java a Java con Hessian

Resultados Obtenidos

1. El intercambio de información ocurre, sin embargo el objeto obtenido por el cliente al final del experimento tiene propiedades `null`. Esto se debe a un problema de serialización o deserialización por parte de la biblioteca de Caucho. La solución es descargar los archivos que forman el servicio a manera de *jarfile* para mostrar de manera local la forma de deserializar los datos de llegada. Los resultados finales resultaron de acuerdo a las Figuras 5.1 y 5.2.

2. A pesar de cumplir las expectativas, descargar los archivos como *jarfile* muestra la deficiencia de los deserializadores. Otros deserializadores de Caucho que prometen arreglar estos problemas son de distribución comercial [36].

3. La solución sólo funciona en el consumo Java-a-Java. La deserialización debe hacerse sobre clases implementadas, es decir, las interfaces de OKI no funcionan en la deserialización.

5.1.2. Método de Burlap

Burlap es un método similar al Hessian pero que utiliza serializadores propios de Spring Framework para XML. A diferencia del método Hessian, Burlap construye un mensaje en XML en tiempo real (que no es SOAP) y lo serializa para su envío. Esto asegura que los mensajes sean entendidos por lenguajes de programación que entiendan XML aunque la forma en la que se escribe el XML podría no ser comprensible en la deserialización, sobre todo porque Burlap es una biblioteca disponible solamente para Java.

Objetivo

1. Consumir el servicio *AgentManagerService* desde el módulo de evaluación para obtener los nombres de los estudiantes en el listado de las calificaciones.

2. Enviar y recibir mensajes en forma de XML para que se pueda usar cualquier deserializador que comprenda XML como el usado por Spring.

Datos Importantes

1. La ubicación del servicio de agentes está en *http://uxmal.dgsca.unam.mx-/strutsSpring/agent.service.*

2. El consumo del servicio solo es posible a través de Spring Framework.

3. Hessian envia y recibe datos en XML que se genera en tiempo de ejecución. Los mensajes no usan SOAP ni la descripción del servicio del WSDL.

4. El servidor funciona sobre el protocolo HTTP y el cliente hace la petición usando el mismo protocolo. Esto es para asegurar los mensajes síncronos.

Resultados Esperados

1. Los nombres que aparecen en el listado de *Grade Records* en la columna llamada *Agents* son los nombres obtenidos en la base de datos del servicio *AgentManagerService*.

2. Si el identificador no aparece en la base de datos del servicio *AgentManagerService* debe mostrarse un nombre generado por el módulo de evaluación. Es decir, el sistema no detiene su funcionamiento si ocurren errores en el servidor.

Condiciones de Hardware y Software

Para el servidor del servicio de *AgentManager*:

1. Computadora personal Pentium IV, 512 MB de memoria RAM. Uso de red a 2 Gbps aprox. Dirección IP 132.248.200.18 (UXMAL).

2. Sistema Operativo Linux Debian 2.6 Testing, Contenedor de Servlets Apache Tomcat 5.5, Java SDK 1.5. Uso de serializadores comunes de Spring.

Para el cliente consumidor del servicio:

1. Computadora personal Pentium IV, 512 MB de memoria RAM. Uso de red a 2 Gbps aprox. Dirección IP 132.248.200.2. (ALPHA).

2. Sistema Operativo Linux Ubuntu 2.5 Stable, Contenedor de Servlets Apache Tomcat 1.6, Java SDK 1.6. Uso de deserializadores de Spring. Uso de Struts y Spring Framework 3.5. y DisplayTag para la decoración. Módulo de evaluación implementado en Java que utliza el servicio remoto.

Desarrollo

Al igual que Hessian, en Burlap se usa Spring para las llamadas remotas. Burlap crea mensajes en formato XML en tiempo de ejecución y los mensajes recibidos los deserializa al leer el XML y creando objetos con esa información. A continuación se lista el código del archivo de configuración de Spring. Nótese los cambios en la clase deserializadora.

```
<!-- se menciona la clase deserializadora -->
<bean id="agentManagerBean" class="org.springframework.
          remoting.caucho.
          BurlapProxyFactoryBean">

<!-- se menciona la ubicacion del servicio -->
   <property name="serviceUrl">
      <value>http://132.248.200.18/strutsSpring/
            AgentService.service
      </value>
   </property>

<!-- se indica hacia donde deserializar -->
   <property name="serviceInterface">
      <value>
         mx.unam.dgsca.uxmal.AgentManagerService
      </value>
   </property>
</bean>
```

Como ocurre con el experimento de Hessian, en Burlap solo cambia el deserializador de regreso de datos. Por ello, la Figura 5.4 muestra la misma estructura aunque con el cambio de deserializador.

Resultados Obtenidos

1. El intercambio de información es posible mostrando el mismo comportamiento de las Figuras 5.1 y 5.2.

2. Los deserializadores funcionan correctamente mostrando errores sólo a nivel *debbug* con mensajes de prevención mostrados desde los archivos de *logging* del contenedor de servlets. Sin embargo, cumple su función en este caso particular.

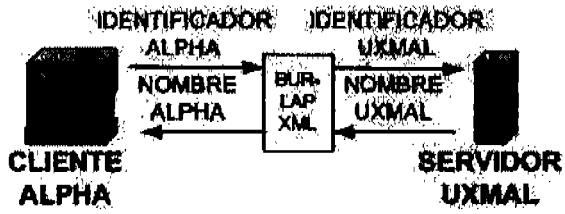

Figura 5.4: Diagrama de bloques de las acciones de consumo del servicio Java a Java usando Burlap

3. Al igual que Hessian, Burlap sólo funciona con consumos tipo Java-a-Java. Además la deserialización no pudo concretarse con las interfaces de OKI haciéndolo dependiente de la implementación y no de las definiciones.

5.1.3. Método de WS

El método de WS es implementado por diversas tecnologías para la comunicación entre sistemas por medio de servicios. Al igual que los experimentos de Burlap y Hessian, los WS pueden ser invocados desde Spring Framework, especificando los datos necesarios para realizar la conexión y la creación de clases `Proxy/Stub` que ejecuten las llamadas remotas. Estos *Proxies* encapsulan y ocultan la serialización y deserialización de datos, evitando los problemas de los anteriores experimentos. El WSDL indica exactamente cómo deben escribirse los mensajes XML para que sean correctamente deserializados por el cliente.

Objetivo

1. Consumir el servicio *AgentManagerService* desde el módulo de evaluación para obtener los nombres de los estudiantes en el listado de las calificaciones.

2. Evitar el uso de envío y recepción de datos binarios o el método de Hessian ya que los datos involucrados en los mensajes son estructuras complejas irreconocibles o mal entendidos por Burlap y Hessian.

3. La deserialización se requiere hacer en las interfaces de OKI para aplicarse *Dependency Injection* en las clases que conforman el módulo de evaluación.

Datos importantes

1. El **WSDL** del servicio (véase sección 2.2.3) se localiza en la dirección *http://132.248.200.18:8080/strutsSpring/services/AgentManagerService?wsdl*. A partir de los datos en este documento XML se deslindan los datos mínimos para generar las clases *Proxy* para llamadas remotas.

2. El **Port Type** del WSDL se llama "agentManagerService". En Spring se le conoce a este dato como **serviceName**.

3. El **Binding** del WSDL es la combinación de HTTP y SOAP como protocolo de envío de mensajes y la forma de los mensajes respectivamente. En Spring no se menciona este dato pues las clases *Proxy* son generadas con respecto a lo que se encuentra en WSDL en línea.

4. El **Port** del WSDL se llama "agentManagerService". En Spring se le llama a este dato **portName**.

5. Finalmente el **Target Namespace** es "http://uxmal.dgsca.unam.mx.AgentManagerService". En la configuración de Spring se le conoce a este dato como **namespaceUri**.

6. El servicio está disponible como un contexto en Jakarta Tomcat por lo que se accede al servicio solamenete por HTTP, lo que asegura los mensajes síncronos entre el cliente y el servidor.

Resultados Esperados

1. Los nombres que aparecen en el listado de *Grade Records* en la columna llamada *Agents* son los nombres obtenidos en la base de datos del servicio *AgentManagerService*.

2. Si el identificador no aparece en la base de datos del servicio *AgentManagerService* debe mostrarse un nombre generado por el módulo de evaluación. Es decir, el sistema no detiene su funcionamiento si errores ocurren en el servidor.

Condiciones de Hardware y Software

Para el servidor del *AgentManagerService*:

1. Computadora Personal Pentium IV, 512 MB de memoria RAM. Uso de red a 2Gbps aprox. Dirección IP 132.248.200.12 (UXMAL).

2. Sistema Operativo Linux Debian 2.6 Testing, Contenedor de Servlets Apache Tomcat 5.5, Java SDK 1.5. Uso del servidor Apache Axis para exponer el servicio remoto.

Para el cliente consumidor del servicio:

1. Computadora Personal Pentium IV, 1GB de memoria RAM. Uso de red a 2 Gbps aprox. Dirección IP 132.248.200.2 (ALPHA).

2. Sistema Operativo Linux Ubuntu 2.5 Stable, Contenedor de Servlets Apache Tomcat 1.6, Java SDK 1.6. Uso de Struts y Spring Framework 3.5 y DisplayTag 1.0 para decoración. Uso de Apache Axis para creación de *Proxies* y realizar llamadas remotas.

Desarrollo

A continuación el código de Spring que toma los datos para la creación de los *Proxies*.

```
<bean id="agentManagerBeanUXMAL" class="edu.unam.okigrading.
                    serviceconsumer.MappingRegisterUxmal" >
    <property name="wsdlDocumentUrl">
        <value>http://132.248.200.18:8080/strutsSpring/
                services/AgentManagerService?wsdl</value>
    </property>
    <property name="serviceInterface">
        <value>org.osid.agent.AgentManager</value>
    </property>
    <property name="namespaceUri">
        <value>http://uxmal.dgsca.unam.mx/AgentManager
                                        Service</value>
    </property>
    <property name="serviceName">
        <value>AgentManagerService</value>
    </property>
    <property name="portName">
        <value>AgentManagerService</value>
    </property>
    <property name="serviceFactoryClass">
        <value>org.apache.axis.client.Service
                            Factory</value></property></bean>
```

En este experimento se cambia un poco la estructura de comunicación. Empezando por el esquema de deserialización que es desplazado por el uso de mensajes SOAP en XML. Esto trae como consecuencia que tanto el cliente como el servidor tengan la obligación de entender los mensajes en XML y la capacidad de mandarlos a través de la red. El servidor por su parte se comunica específicamente con el servidor Axis (o servidor de WS) para crear el proceso de datos visto en experimentos previos. Véase Figura 5.5.

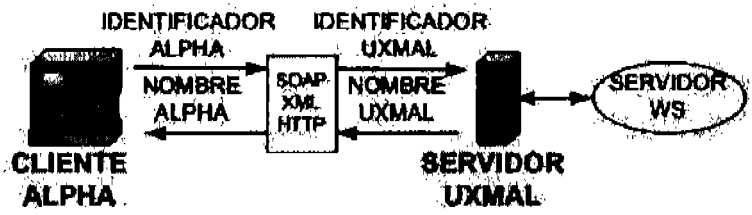

Figura 5.5: Diagrama de bloques de las acciones de consumo del servicio Java a Java con WS

Además se realiza la creación de una clase que menciona la ubicación del servicio por medio de "nombres calificados"[4]. De esta manera se describe la serialización y deserialización de datos. Esta manera determina el paso de XML de los mensajes SOAP a clase Java. El bean creado por Spring es inyectado por *Dependency Injection* en el servlet de decoración que aplica la funcionalidad en los identificadores mostrando los datos decorados en el momento de listar los *GradeRecords*.

Resultados

1. Se consigue la interoperabilidad de sistemas. Los datos son decorados apareciendo los datos con nombres de una base de datos remota en lugar de los identificadores.

2. Los serializadores funcionaron correctamente evitando los mensajes de *error* y *warning* mandados por el servidor Axis.

3. La deserialización se hace a partir de las interfaces de OKI en lugar de las implementaciones como ocurrió en Burlap y Hessian.

[4]También conocido como *Qualified Name* o simplemente QName. Son direcciones URI para ser entendidas por Java

5.1.4. Observaciones

A continuación se muestra una tabla comparativa con los resultados obtenidos, usando los métodos de los últimos experimentos. La columna llamada "Inter." indica si se lleva a cabo un intercambio de datos entre aplicaciones. Debe entenderse que la interoperabilidad ocurre entre sistemas Java-a-Java. La columna "Spring" señala si existe integración con Spring Framework. Esto ayuda cuando se requiere usar *Dependency Injection* y separar dependencias de implementación (véase sección 2.3.4). La columna "Deserial." muestra si hay problemas de deserialización de datos. Si al menos existen mensajes de precaución se considera errores de deserialización. Finalmente la columna "Interfaces" indica si la deserialización se hace sobre las interfaces de OKI.

Método	Inter.	Spring	Deserial.	Interfaces
Hessian	χ	$\surd$	χ	χ
Burlap	$\surd$	$\surd$	$\surd$	χ
Web Services	$\surd$	$\surd$	$\surd$	$\surd$

A partir de los experimentos y de la tabla comparativa se desprenden las siguientes observaciones:

- Hessian hace el intercambio de información en forma de binaria [31]. Es útil en los casos que involucran envío y recepción de objetos simples. En los experimentos de esta sección no ocurre la interoperabilidad por la complejidad de los objetos involucrados en la comunicación.

- Burlap hace una traducción de los objetos a XML para su transmisión por la red y que facilite la deserialización de los datos [31]. Los objetos complejos fueron enviados y se consiguió la interoperabilidad. La deserialización no genera objetos de implementaciones OKI. Burlap ofrece la interoperabilidad pero deserializa en objetos que a Burlap le conviene.

- WS permite tener control de qué datos y el orden en que se mandan en el mensaje XML [1]. De esta manera se puede conseguir la interoperabilidad como lo consiguió Burlap pero con deserialización en implementaciones OKI.

5.2. Consumo del servicio GradingManagerService

Esta es la segunda serie de experimentos, en la que el módulo de evaluación, analizado en el capítulo 4 es expuesto como un servicio remoto como se ve en la sección 4.6 para la creación de objetos calificables y evaluaciones. Tomando como base los experimentos con el servicio *AgentManagerService*, en esta sección se realizan experimentos usando WS con diferentes lenguajes de programación, siguiendo las ideas de la orientación a servicios. En secciones subsecuentes se muestran experimentos con Java, AJAX+PHP, .NET y PHP.

5.2.1. Java: La Prueba UNAM-MIT

En esta sección se trata de comunicar el servicio de *GradingManagerService* implementado en Java con un sistema Web implementado en la Dirección General de Servicios y Cómputo Académico de la UNAM. La comunicación ocurre cuando en el sistema Web llamado *SCORM Player* ocurren ciertas acciones, se manda a llamar el servicio *GradingManagerService* enviando evaluaciones al sistema Web de Grading como se ve en el escenario de la sección 4.7.4.

El experimento se trata de una demostración entre la UNAM y el Instituto Técnologico de Massachusetts como marco de la convención de interoperabilidad *open iWorld 2007* ganando el premio *Best Response to a Technology Challenge*[5]. Esta es una recreación de dicho experimento creando un cliente de WS en Java y mandando los mismos mensajes de creación que hace el sistema *SCORM Player*.

Objetivo

1. El consumo del servicio remoto *GradingManagerService* para crear calificaciones desde un módulo implementado en Java simulando las llamadas hechas por el sistema *SCORM Player*.

2. El resultado de la interoperabilidad se establece cuando los datos de las calificaciones aparezcan en el sistema Web, en los listados de objetos calificables y evaluaciones (*Gradable Objects* y *Grade Records*).

[5] www.openiworld.org/UNAM.html

3. Los datos mostrados desde el cliente deben de coincidir con los aparecidos en los listados, esto es, debende mostrarse pruebas de la coincidencia en los datos en cliente y en servidor.

Datos Importantes

1. El servicio remoto se ubica en la dirección *http://alpha.dgsca.unam.mx:8082/OGP/services/GradingManagerService?wsdl.*

2. El sistema Web que contiene los listados se encuentra en *http://alpha.dgsca.unam.mx:8082/OGP* en las ligas de *Gradable Objects* y *Grade Records.*

3. La recreación del experimento es una clase Java llamada *GradingConsumer.* Se encarga de mandar los datos gracias a las clases `Proxy/Stub` generadas por Apache Axis.

Resultados Esperados

1. La interoperabilidad existe entre los dos sistemas Web. Para ello, las evaluaciones son enviadas desde GradingConsumer y aparecen listados en el módulo de evaluación.

2. Los objetos calificables y las evaluaciones deben de guardarse en una base de datos para mantener la persistencia de los datos.

3. Si no se sigue el formato de los identificadores de los estudiantes, el sistema debe de proveer una forma de poner un nombre genérico.

4. Se debe devolver la fecha de creación de la evaluación desde el servidor en el mensaje de retorno del servicio remoto.

5. El nombre correspondiente al identificador de agentes con el número "1" es generado por el sistema y se llama "nombre generado localmente [1]". Esto significa que en la lista de *Grade Records* aparece este nombre en la columna de "Agents".

Condiciones de Hardware y Software

Para el servidor:

1. Computadora Personal Pentium IV, 1 GB en memoria RAM. Uso de la red a 2Mbps aprox. Dirección IP 132.248.200.2 (ALPHA).

2. Sistema Operativo Linux Ubuntu 2.5 Stable, contenedor de Servlets Apache Tomcat 6.0, Java SDK 1.6. El servicio remoto es expuesto con Apache Axis Server. El sistema Web usa Struts y Spring Framework 3.5 y DisplayTag para mostrar los valores desde los listados.

Para el cliente:

1. Computadora Personal Péntium IV, 2GB en memoria RAM, uso de la red a 2Mbps aprox. Dirección IP 132.248.200.12 (BALAM).

2. Sistema Operativo Linux Ubuntu 2.6 Stable, contenedor de Servlets Apache Tomcat 5.5 Bundle, Java SDK 1.5. El servicio remoto es consumido por Apache Axis.

Desarrollo

Se expone el servicio con ayuda de Apache Axis como se ve en la sección 4.6. La localización del servicio queda en una dirección URL indicada en los Datos Importantes. Del lado del cliente, se genera la clase `Proxy/Stub` con ayuda de Axis y se genera la clase `GradingConsumer`. A continuación parte del código de esta clase mandando a llamar clases como `GradingServiceSOAPStub`, `StringHolder`, `GradableObjectResponse` y `GradeRecordResponse` que son las versiones generadas por Axis que hacen directa la deserialización. Estas clases se encargan de hacer las llamadas pertinentes usando las bibliotecas para cómputo en red como Java RMI y JAX RPC.

```
GradingServiceSOAPStub g =
                        new GradingServiceSOAPStub();
StringHolder displayName = ''Prueba Java Tesis'';
                    .
                    .
                    .
GradableObjectResponse go =
                g.createGradableObject(displayName, ...);
GradeRecordResponse gr =
                g.createGradableObject(go.getId, ...);
System.out.println(gr.getModifiedDate());
```

Se mandan los siguientes datos para la creación de la evaluación:

- **Nombre del** *Gradable Object*: "Prueba Java Tesis"

- **Descripción del** *Gradable Object*: "Prueba Java Tesis"
- **Identificador del agente**: "1"
- **Evaluación** (*GradeValue*): "1000"

Con esto se obtiene la fecha de creación de la evaluación. A nivel de procesos, los datos son enviados en formato de XML en un mensaje SOAP. Una vez que se llega al servidor (ALPHA), éste llama al servidor Axis de WS, una vez ocurrido el conjunto de acciones en el servicio remoto. El resultado de estas acciones se manda como mensajes de regreso y desde el cliente es posible pedir los datos recien recibidos como la fecha. Esta viene en formato de número expresado a 32 bits y con un postproceso se transforma a una fecha de Java. La Figura 5.6 muestra un diagrama de bloques mostrando estas acciones.

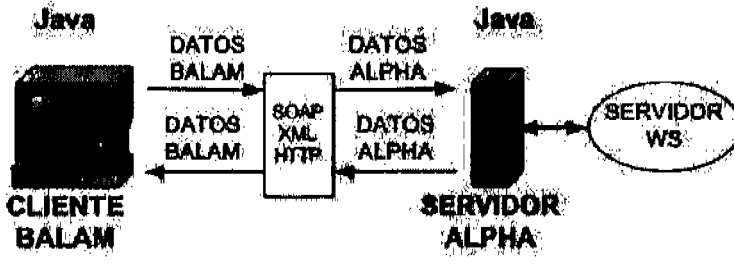

Figura 5.6: Diagrama de bloques de las acciones de consumo del servicio Java a Java

Resultados Obtenidos

1. Los datos se mandan satisfactoriamente desde el cliente en la computadora llamada BALAM. Véase la Figura 5.7 donde se ejecuta la clase `GradingConsumer`.

2. Se consigue la interoperabilidad entre los sistemas. Los datos aparecen mostrados en los listados del sistema. Véase las Figuras 5.8 y 5.9 en comparación con la Figura 5.7.

3. La deserialización de los datos funciona ya que Axis hace crea sus propias clases en las cuales deserializar los datos. Estos son usados desde el cliente que puede hacer llamadas de manera remota sin tener

pleno conocimiento del uso de la red y en su lugar puede usar estas clases para crear evaluaciones de manera local.

4. A nivel de números se manda la creación de un *Gradable Object* recibiendo su identificador (344 según la Figura 5.7) que coincide con el identificador dado por el listado del sistema en ALPHA.

5. Tras mandar la llamada de creación de *Grade Record* se recibe la fecha "Tue Oct 02 14:28:51 CDT 2007" (mostrado en la Figura 5.9) que coincide con la del listado como "Tue 02/10/2007" (mostrado al final del listado en la Figura 5.9).

```
sai@pub04:~/GConsumer/bin$ java -cp ../lib/axis.jar:../l
b/saaj.jar:../lib/spring.jar:../lib/wsdl4j-1.5.1.jar:. (
- Unable to find required classes (javax.activation.Date
------ RESPONSE RECIEVED ------
DISPLAYNAME : Prueba Java Tesis
DESC : Prueba Java Tesis
-GRADING GRADABLE OBJECT ID: 344
Resultado de la prueba

GradeRecord recibido en el sistema. Fecha de creacion:
Tue Oct 02 14:28:51 CDT 2007
sai@pub04:~/GConsumer/bin$
```

Figura 5.7: Captura de pantalla de la ejecución del Cliente WS

Figura 5.8: Captura de pantalla del listado de *Gradable Objects*

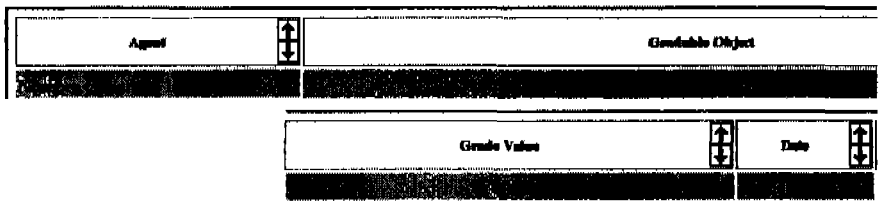

Figura 5.9: Captura de pantalla del listado de *Grade Records*

5.2.2. AJAX y PHP: Usando Web 2.0

En este experimento se consume el servicio de Grading para Web 2.0, especificamente con AJAX. Se recrea un mensaje SOAP y se envía al servidor de WS. La interoperabilidad queda comprobada al analizar el mensaje SOAP recibido. En esta prueba se nota más la estructura al interior de un WS pues se genera un mensaje SOAP y se espera una estructura previamente definida aunque con datos desconocidos.

Sin embargo, AJAX no soporta el consumo de WS. Esto ocurre porque los navegadores confunden nombres de dominios suponiendo que el servidor (hardware) y el servidor de WS (en este caso Axis) son elementos diferentes, ubicados en dominios de Internet distintos (véase Figura 5.10). Este problema esta documentado en diversos foros y páginas de Internet[6] donde se propone un script en PHP u otras soluciones para homogenizar los nombres de dominios.

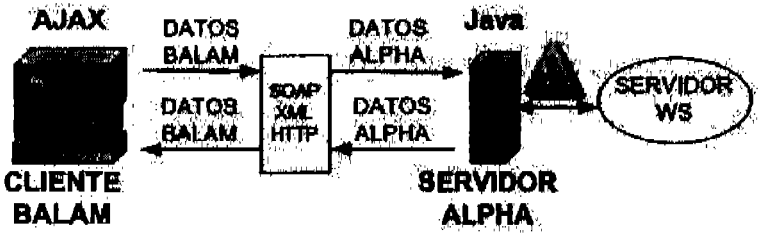

Figura 5.10: Diagrama de bloques del consumo del servicio remoto con AJAX

[6]Por ejemplo www.simple-talk.com/dotnet/asp.net/calling-cross-domain-web-services-in-ajax/ donde se le conoce como *Cross-Domain Exception* o *Cross-Domain Violation*.

Objetivo

1. Las computadoras cliente y servidor deben conseguir interoperabilidad usando un cliente implementado en Javascript y el objeto XMLHTTPRequest de AJAX para mandar y recibir mensajes SOAP.

2. El nombre del objeto calificable y la evaluación en deben aparecer en los listados correspondientes del módulo de evaluación.

3. La confusión de nombres de dominios debe evitarse con ayuda de algún mecanismo más allá de las limitantes de Javascript y AJAX. Se propone PHP.

Datos Importantes

1. El servicio está disponible en la dirección *http://alpha.dgsca.unam.mx:8082/OGP/services/GradingManagerService?wsdl.*

2. La llamada al servicio remoto no implica creación de clases `Proxy/Stub` pues el experimento se particuliza hasta el nivel de envío y recepción de mensajes en XML.

Resultados Esperados

1. El envío de un mensaje siguiendo la estructura de un mensaje SOAP visto en la sección 2.2.3 usando el objeto XMLHTTRequest.

2. La recepción de un mensaje con la estructura de código XML de SOAP con un tag llamado *date* y con un valor numérico que significa la fecha de entrada de la evaluación en el sistema en milisegundos.

3. La presentación de los datos enviados en el módulo Web en el listado de *Gradable Objects* y *Grade Records.*

El siguiente código en XML pertence al esqueleto de un mensaje SOAP de respuesta tras crear un *Gradable Object.* La diferencias entre este código y el obtenido del experimento debe ser solo en el tag llamado ***id*** en el identificador único asignado por el servidor tras su creación.

```
<soapenv:Envelope ...>
   <soapenv:Body>
     <ns:createGradableObjectResponse>
        <displayname>Prueba AJAX y PHP de javascript</displayname>
        <description>Prueba AJAX y PHP</description>
                                .
                                .
                                .
<!-- El siguiente dato demuestra la interoperabilidad -->
        <id>00000</id>
     </ns:createGradableObjectResponse>
   </soapenv:Body>
</sopaenv:Envelope>
```

Ahora, la supuesta respuesta del servidor de WS tras la creación de un *Grade Record*. La interoperabilida se demuestra con el tag <date> que menciona la fecha de creación en un número expresado a 64 bits.

```
<soapenv:Envelope ...>
   <soapenv:Body>
     <ns:createGradeRecordResponse>
        <agentid>1</agentid>
<!-- este dato se obtiene en la creacion del Gradable Object -->
        <gradableobjectid>00000</gradableobjectid>
                                .
                                .
                                .
<!-- El siguiente dato demuestra la interoperabilidad -->
        <date>00000</date>
     </ns:createGradeRecordResponse>
   </soapenv:Body>
</sopaenv:Envelope>
```

Condiciones de Hardware y Software

Para el servidor:

1. Computadora Personal Pentium IV, 1 GB en memoria RAM, Uso de la red a 2Mbps aprox. Dirección IP 132.248.200.2 (ALPHA).

2. Sistema Operativo Linux Ubuntu 2.5 Stable, contenedor de Servlets Apache Tomcat 6.0, Java SDK 1.6. El servicio remoto es expuesto con Apache Axis Server. El sistema Web usa Struts y Spring Framework 3.5 y DisplayTag para mostrar los valores desde los listados.

Para el cliente:

1. Computadora Personal Pentium IV, 2GB en memoria RAM, uso de la red a 2Mbps aprox. Dirección IP 132.248.200.12 (BALAM).

2. Sistema Operativo Linux Ubuntu 2.6 Stable, contenedor de Servlets Apache Tomcat 5.5 Bundle. Uso de Firefox como intérprete de código Javascript y el plugin FireBug para comprobar los mensajes mandados y recibidos

Desarrollo

Se crea el código Javascript que hace la conexión usando el objeto XMLHTTPResquest además del código PHP para evitar la confusión de nombres de dominios. La Figura 5.11 muestra la función de PHP en el módulo de prueba de Javascript para realizar la comunicación con la misma estructura de los demás experimentos.

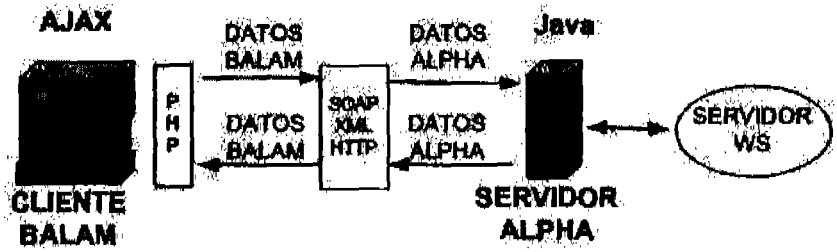

Figura 5.11: Diagrama de bloques del consumo del servicio remoto con AJAX y PHP

El experimento ocurre en dos pasos. En la primera parte, un mensaje es enviado para la creación de un *Gradable Object*. Los mensajes SOAP son enviados de uno por uno para tener control de los datos recibidos por la primer llamada y que sean usados en la segunda llamada.

```
<soap:Envelope ...>
   <soap:Body createGradableObject ...>
       <displayname>Prueba AJAX y PHP de javascript</displayname>
       <description>Prueba AJAX y PHP</description>
                    .
                    .
                    .
  </soapenv:Body>
<soapenv:Envelope>
```

En el segundo paso se llama al servicio mandando un mensaje para crear el *Grade Record*. Para ello es necesario el dato del identificador de la creación del *Gradable Object* y la evaluación. A continuación un fragmento de ese mensaje a mandar.

```
<soap:Envelope ...>
   <soap:Body createGradeRecord ...>
      <agentid>1</agentid>
      <gradableobjectid>2</gradableobjectid>
                    .
                    .
                    .
  </soapenv:Body>
<soapenv:Envelope>
```

En resumen, se mandan los siguientes datos:

- **Nombre del** *Gradable Object*: "Prueba AJAX y PHP de javascript"
- **Descripción del** *Gradable Object*: "Prueba AJAX y PHP"
- **Identificador del agente**: "1"
- **Evaluación** (*GradeValue*): "1000"

Resultados Obtenidos

1. Los resultados obtenidos se consiguen usando el plugin de Firefox llamado Firebug (véase apéndice sección A.9), esto es, además de monitorearse los mensajes SOAP de ida se consiguen los de regreso. La razón de usar esta herramienta obedece a las capacidades de Firebug para monitorear cambios en el código HTML de una página aunque existan ótras herramientas para conseguir el mismo objetivo [7].

2. Los listados en el módulo Web son mostrados en las Figuras 5.12 y 5.13, demostrando que los datos enviados desde el cliente aparecen en el servidor.

3. Las Figuras 5.14 y 5.15 para la creación de un Gradable Object, muestran el mensajes de ida y regreso en la llamada al servicio remoto. la Figura 5.15 muestra el identificador único dado por el sistema al Gradable Object (en este caso es 10, véase penúltimo renglón del XML).

4. La creación de un Grade Record muestra intercambio de datos. Por un lado, el mensaje de ida, mostrado en la Figura 5.16, muestra los datos enviados. Por otro lado la Figura 5.17 tiene la fecha de creación en forma de un número entero expresado a 64 bits.

5. La Figura 5.12 muestra el listado del sistema con los Gradable Objects. Los datos de envío coinciden con los mostrados al final de esta lista.

6. La fecha de creación del Grade Record mostrado al final del listado en la Figura 5.13 (Fri, 19/10/2007) coincide con la transformación del número recibido en el mensaje SOAP de la Figura 5.17 (1192831414981 en milisegundos es trasformado a Fri Oct 19 16:35:09 CDT 2007 tras un postproceso).

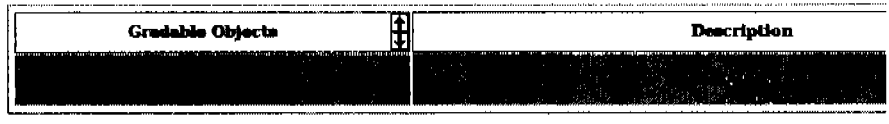

Figura 5.12: Captura de pantalla del listado de *Gradable Objects*

[7] Por ejemplo, HTTP Monitor o SOAP Monitor. Ambas herramientas son proyectos de Apache

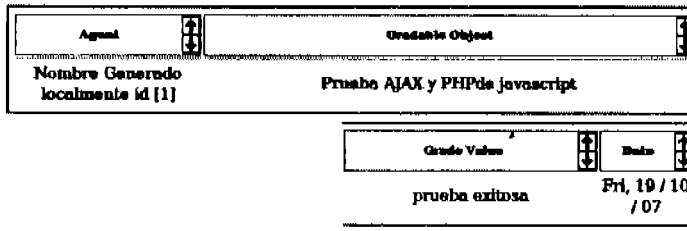

Figura 5.13: Captura de pantalla del listado de *Grade Records*

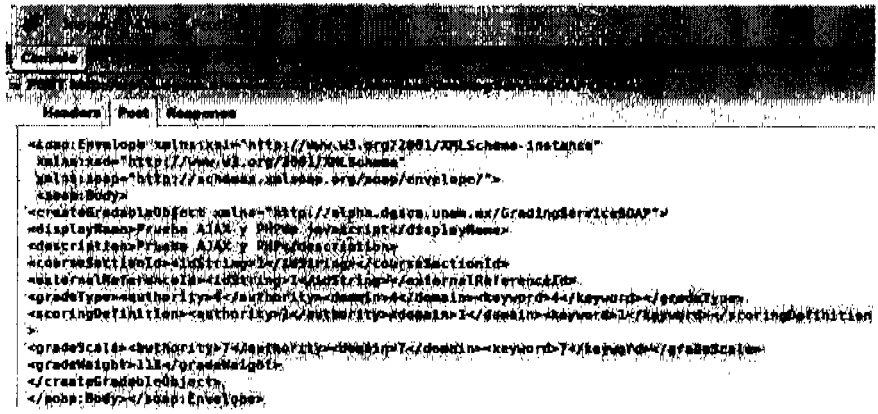

Figura 5.14: Captura de pantalla del mensaje de envío para el servicio en la creación de un *Gradable Object*

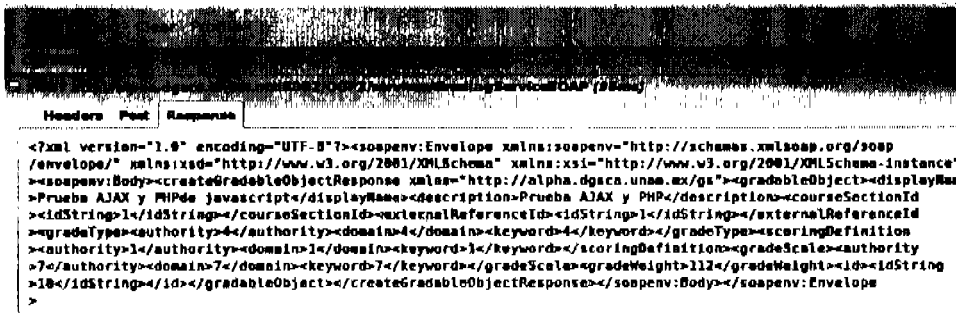

Figura 5.15: Captura de pantalla del mensaje obtenido del servicio en la creación de un *Gradable Object*

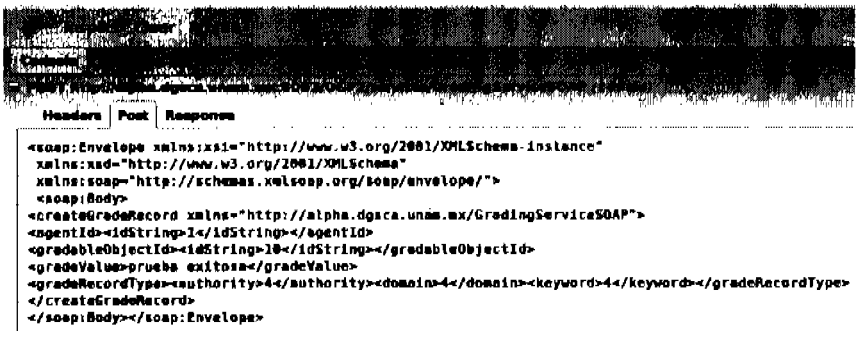

Figura 5.16: Captura de pantalla del mensaje de envío para el servicio en la creación de un *Grade Record*

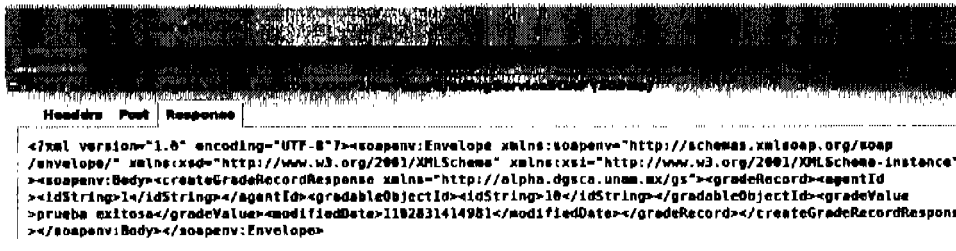

Figura 5.17: Captura de pantalla del mensaje obtenido del servicio en la creación de un *Grade Record*

5.2.3. .NET: Prueba con C#

En esta sección se realiza una prueba para consumir el servicio de Grading a partir de la tecnología de Microsoft .NET. Particularmente se realiza la prueba con un cliente consumidor implementado en C#. En .NET se utiliza un lenguaje común para los programas realizados en C# .NET, J# .NET y Visual Basic .NET llamado CLI (véase apéndice sección A.10). Por tanto, el experimento con C# .NET es equivalente a realizar los experimentos con los tres lenguajes de programación que conforman Microsoft .NET.

Objetivo

1. El consumo del servicio remoto *GradingManagerService* para crear calificaciones desde un módulo implementado en el lenguaje de programación C# .NET.

2. El resultado de la interoperabilidad queda establecido si el módulo Web tiene los datos enviados en los listados de *Gradable Objects* y *Grade Records* así como se obtiene la fecha de creación desde el cliente.

Datos Importantes

1. El servicio remoto se ubica en la dirección *http://alpha.dgsca.unam.mx:8082/OGP2/services/GradingManagerService?wsdl.*

2. El sistema Web que contiene los listados se encuentra en *http://alpha.dgsca.unam.mx:8082/OGP* en las ligas de *Gradable Objects* y *Grade Records.*

3. La creación de un archivo C# llamado *GradingConsumer*, se encarga de mandar los datos gracias a las clases `Proxy/Stub` generadas por .NET SDK Framework 1.1 (véase apéndice sección A.10).

4. El cliente debe contener los mecanismos mínimos para ejecutar aplicaciones .NET, es decir, algún sistema operativo de la familia Microsoft igual o superior a Windows XP Service Pack 2, .NET Framework 1.1 Redistributable o superior y el acceso a la red.

Resultados Esperados

1. La interoperabilidad existe entre los dos sistemas de software. Los datos enviados tras la ejecución de *GradingConsumer* se ven reflejados en los listados del módulo Web de *Gradable Objects* y *Grade Records.*

2. Tras las llamadas al servidor, el cliente obtiene un valor numérico a 64 bit que puede trasnformar a una fecha, la cual coincide con la ubicada en el listado de *Grade Records* en la columna de "Fecha".

Condiciones de Hardware y Software

Para el servidor:

1. Computadora Personal Pentium IV, 1 GB en memoria RAM. Uso de la red a 2Mbps aprox. Dirección IP 132.248.200.2 (ALPHA).

2. Sistema Operativo Linux Ubuntu 2,5 Stable, contenedor de Servlets Apache Tomcat 6.0, Java SDK 1.6. El servicio remoto es expuesto con Apache Axis Server. El sistema Web usa Struts y Spring Framework 3.5 y DisplayTag para mostrar los valores de los listados.

Para el cliente:

1. Computadora Personal Dell Inspiron Notebook 1150 Pentium Celeron a 2.60 GHz, memoria de 256MB en RAM. Uso de la red a 56 kbps aprox. Dirección IP dinámica (para fines del experimento será llamada WINDOWS).

2. Sistema Operativo Windows XP Service Pack 2, .NET Framework 1.1. Redistributable y .NET Framework SDK 1.1 para ejecutar apliaciones .NET.

Desarrollo

Con ayuda de .NET Framework SDK 1.1 se generan las clases `Proxy/Stub` y se implementa la clase `GradingConsumer`. Se hacen llamadas a las estructuras del `Proxy/Stub` para llamar al servicio remoto. A continuación la forma en que es llamado el programa wsdl.exe de .NET SDK para generar las clases `Proxy/Stub`:

```
wsdl.exe /n:ws /language:cs
/out:GConsumerdotNET/ GConsumerProxy.cs
http://alpha.dgsca.unam.mx:8082/OGP2/services/
GradingManagerService?wsdl
```

En la clase a implementar (GConsumer.cs) se especifican los valores a enviar. Estos datos son los siguientes:

- **Nombre del** *Gradable Object*: "Prueba dotNET Tesis"
- **Descripción del** *Gradable Object*: "Prueba dotNET Tesis"
- **Identificador del agente**: "1"
- **Evaluación** (*GradeValue*): "1000"

La Figura 5.18 muestra la forma en que el servicio es consumido usando los mensajes SOAP como contenido y a la red como medio. Cuando los mensajes salen de la computadora WINDOWS están en formato XML que ALPHA entiende y manda a su servidor WS Axis. Cuando se realiza el proceso pertinente, se manda el mensaje de regreso por el mismo medio y es comprendido por el cliente. Con la información, el cliente puede hacer un postproceso y mandar la fecha en un formato propio del lenguaje de programación.

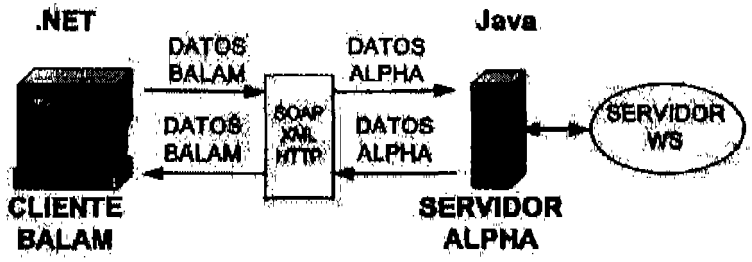

Figura 5.18: Diagrama de bloques del consumo del servicio remoto con .NET a Java

Finalmente, el archivo obtenido tras la compilación del archivo GConsumer.cs, llamado GConsumer.exe es ejecutado en consola.

Resultados Obtenidos

1. El cliente envía y recibe datos con el servidor de manera satisfactoria. La Figura 5.19 muestra la captura de pantalla con la ejecución de la aplicación .NET. Nótese la similitud entre las fechas encontradas en la ejecución del cliente en la Figura 5.19 y el listado de *Grade Records* de la Figura 5.21.

2. Desde el lado del servidor también puede notarse la comunicación. Los listados de los *GradableObjects* y *Grade Records* contienen los nombres

mandados del cliente. Véase Figuras 5.20 y 5.21. Compárece con los datos a enviar desde el planteamiento del experimento.

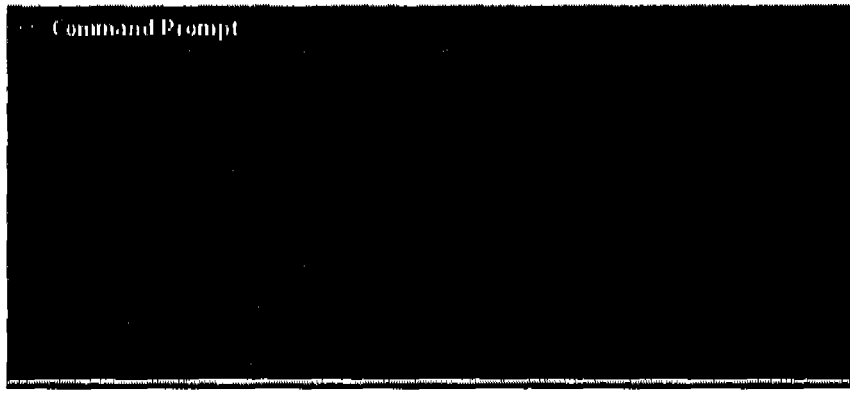

Figura 5.19: Captura de pantalla de la ejecución del cliente ws

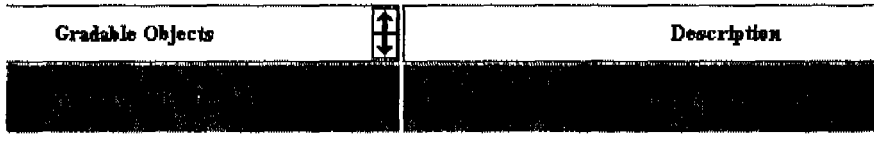

Figura 5.20: Captura de pantalla del listado de *Gradable Objects*

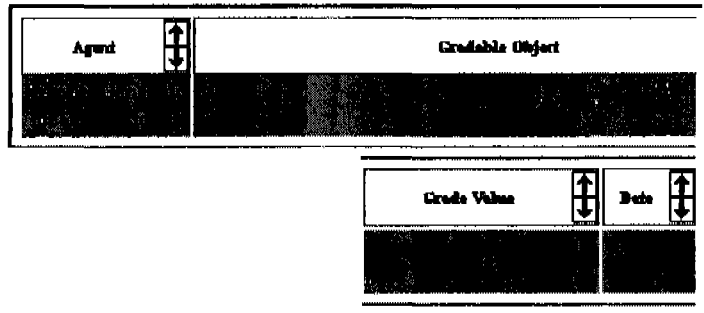

Figura 5.21: Captura de pantalla del listado de *Grade Records*

5.2.4. PHP: Prueba Web en Servidor Apache

El experimento final en el consumo de un servicio remoto. PHP utiliza una serie de bibliotecas implementadas en C aunque adaptadas al ambiente Web. Una de estas librerías es NuSOAP (véase apéndice sección A.11) y se usa en este experimento para consumir servicios remotos y probar la interoperabilidad con este lenguaje de programación.

Objetivo

1. El consumo del servicio remoto *GradingManagerService* para crear evaluaciones desde un módulo implementado en el lenguaje de programación PHP.

2. El resultado de la interoperabilidad se obtiene solo cuando los datos enviados desde el cliente aparecen en los listados del módulo de evaluación de *Gradable Objects* y *Grade Records*.

3. Los datos recibidos por el cliente consumidor deben de mostrar el identificador del objecto calificable recién creado asi como la fecha de creación de la calificación. Estos datos son mandados por el servidor.

Datos Importantes

1. El servicio remoto se ubica en la dirección *http://alpha.dgsca.unam.-mx:8082/OGP/services/GradingManagerService?wsdl.*

2. El sistema Web que contiene los listados se encuentra en *http://alpha.-dgsca.unam.mx:8082/OGP* en las ligas de *Gradable Objects* y *Grade Records*.

3. La creacion de un archivo en PHP llamado GradingConsumer.php, se encarga de mandar datos con ayuda de la librería de WS NuSOAP.

Resultados Esperados

1. La interoperabilidad existe entre los dos sistemas de software. Los datos enviados tras la ejecución de *GradingConsumer* se ven reflejados en los listados del módulo Web de *Gradable Objects* y *Grade Records*.

2. Tras las llamadas al servidor, el cliente obtiene un valor numérico a 64 bit que puede trasnformar a una fecha, la cual coincide con la ubicada en el listado de *Grade Records* en la columna de "Fecha".

Condiciones de Hardware y Software

Para el servidor:

1. Computadora Personal Pentium IV, 1GB en memoria RAM, Uso de la red a 2Mbps aprox. Dirección IP 132.248.200.2 (ALPHA).

2. Sistema Operativo Linux Ubuntu 2.5 Stable, contenedor de Servlets Apache Tomcat 6.0, Java SDK 1.6. El servicio remoto es expuesto con Apache Axis Server. El sistema Web usa Struts y Spring Framework 3.5 y DisplayTag para mostrar los valores desde los listados.

Para el cliente:

1. Computadora Personal Pentium IV, 2GB en memoria RAM, uso de la red a 2Mbps aprox. Dirección IP 132.248.200.12 (BALAM).

2. Sistema Operativo Linux Ubuntu 2.6 Stable, servidor Apache 2 y PHP 5. El servicio remoto se consume con ayuda de la librería NuSOAP [8].

Desarrollo

Se crea la aplicación PHP de *GradingConsumer*. La página resultante de llamar a la aplicación PHP muestra el identificador del *Gradable Object* generado tras el primer mensaje SOAP enviado y recibido. Además muestra

[8] www.sourceforge.net/nusoap

la fecha de creación del *Grade Record* tras el segundo envío y recepción de los mensajes SOAP. La interacción entre BALAM y ALPHA se hace por medio de mensajes SOAP en XML. Los mensajes son responsabilidad de ser entendidos por el cliente y el servidor. La Figura 5.22 muestra estas acciones.

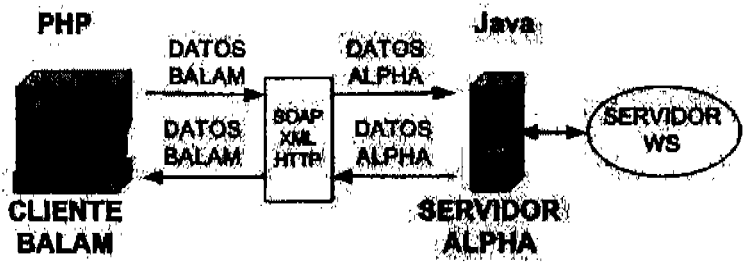

Figura 5.22: Diagrama de bloques del consumo del servicio remoto con PHP

Los datos a enviarse y que son visibles en el módulo Web, aparecen en los listados de *Gradable Objects* y *Grade Records* son los siguientes:

- **Nombre del** *Gradable Object*: "Prueba PHP"
- **Descripción del** *Gradable Object*: "Prueba PHP"
- **Identificador del agente**: "1"
- **Evaluación** (*GradeValue*): "1000"

A continuación parte del código de la aplicacion *GradingConsumer.php* mostrando la forma de llamar los métodos para crear un objeto calificable y la evaluación con los datos anteriores:

```
$req = array('displayName' => 'Prueba PHP',
             'description' => 'Prueba PHP' ... );
$idGO = $proxy->createGradableObject($req);//para crear el objeto califica
//se crea la evaluación con el objeto calificable recién obtenido (idGO)
$req1 = array('agentId' => 1,
              'gradableObjectId' => $idGO
              'gradeValue' => 1000);
$gradeRecord = $proxy->createGradeRecord($req1);
```

Resultados Obtenidos

1. La Figura 5.23 muestra la acción ocurrida desde el cliente. En esta captura de pantalla se muestra que hubo un intercambio de información.

2. En las Figuras 5.24 y 5.25 se listan los *Gradable Objects* y *GradeRecords*. Debe ponerse atención a los datos de los listados y su coincidencia con los datos enviados desde la computadora BALAM.

Prueba de PHP

DISPLAY NAME: Prueba PHP
DESCRIPTION: Prueba PHP
GRADABLE OBJECT ID:42

GRADE RECORD RECIBIDO
FECHA DE CREACION: 20/11/2007, 14:04:06 PM

Figura 5.23: Captura de pantalla de la ejecución del cliente WS

Gradable Objects	Description
Prueba PHP	Prueba PHP

Figura 5.24: Captura de pantalla del listado de *Gradable Objects*

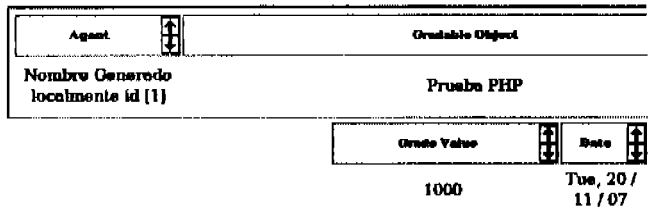

Figura 5.25: Captura de pantalla del listado de *Grade Records*

5.2.5. Observaciones

La siguiente tabla trata de condensar características funcionales de los lenguajes de programación tras el consumo del servicio *Grading Manager* usando WS. La columna "Inter." indica si es posible la interoperabilidad. En "XML" se establece si los lenguajes usan y manipulan el formato XML para conseguir la comunicación. En la columna "Serial." se indica si es posible la serialización y deserialización de los datos. En "Red" muestra si el lenguaje de programación hace estas llamadas a través de la red.

Método	Inter.	XML	Serial.	Red
Java	√	√	√	√
Javascript	√	√	√	√
C#	√	√	√	√
PHP	√	√	√	√

A partir de los experimentos realizados en esta sección y la tabla comparativa se tienen las siguientes observaciones:

- El servicio GradingManagerService envía diferentes identificadores al crearse los objetos calificables, mostrando la independencia de cada experiemento de interoperabilidad. El envío de la fecha demuestra la recepción de los datos para su almacenamiento en el servidor.
- El intercambio de mensajes ocurre sobre HTTP y sobre el servidor Tomcat. Los mensajes ocurren y son síncronos.
- El servicio GradingManagerService siempre envía respuestas con la misma estructura XML; no se adapta al lenguaje de programación que lo llama. Es responsabilidad de cada lenguaje la interpretación de esta información.

- Todos los lenguajes de programación involucrados en los experimentos cumplen las características funcionales de manipulación de XML, serialización y envío y recepción de datos por medio de la red. Todos los experimentos consiguieron la interoperabilidad.

Capítulo 6

Conclusiones

En este capítulo se muestran las conclusiones generales de la tesis. En la primera sección se revisa la hipótesis, basado en las pruebas realizadas en el Capítulo 5. En la siguiente sección se comparan la arquitectura orientada a servicios usada en las pruebas con los trabajos relacionados. Después, se hace un recuento de las contribuciones y los resultados obtenidos. Finalmente en la última sección se muestran temas de interés que tomen esta tesis como base para trabajos futuros.

6.1. Revisión de la hipótesis

En esta sección se hace una revisión de la hipótesis dados los resultados obtenidos en los experimentos. En la primera parte, se responde a la pregunta vista en el capítulo 2. En la segunda parte se interpretan los resultados de los experimentos.

6.1.1. Discusión

A continuación se vuelve a mencionar la pregunta a la que se intenta dar respuesta en esta tesis.

> *¿Cuáles son las condiciones de un entorno de programación Web bajo las cuales se obtiene una interoperabilidad entre aplicaciones, siendo estas descritas en términos de una Arquitectura Web Orientada a Servicios (SOWA)?*

La relevancia de esta pregunta radica en encontrar las características funcionales para definir una descripción, es decir, una arquitectura cuyo objetivo sea la interoperabilidad entre aplicaciones. Así pues, la definición de

los experimentos, usando estos lenguajes de programación, responde a la necesidad de experimentar con esta arquitectura y verificar su utilidad.

De esta manera se propone el uso de lenguajes de programación como Java y C# que pertenecen a corrientes de programación ampliamente usadas y documentadas en el entorno Web y que se consideran opuestas por las compañías creadoras y que apoyan estos lenguajes. El uso de PHP responde a la necesidad de usar un lenguaje enfocado al desarrollo de sistemas en Web y que cuenta con una serie de bibliotecas que permiten automatizar algunas rutinas complejas para otros lenguajes de programación. Finalmente con Javascript se busca mostrar la interoperabilidad con uno de los lenguajes con más auge en los últimos años. En Web 2.0 se busca aprovechar algunas situaciones en la Web para mejorar tiempos de carga. A pesar de ser un lenguaje emergente, esta arquitectura presenta compatibilidad aunque con unos cambios mínimos.

La tabla mostrada en la sección 5.2.5 muestra la abstracción de características funcionales de los lenguajes de programación tras experimentar la interoperabilidad. De esta manera se explica la existencia de la arquitectura orientada a servicios cuyo objetivo es la interoperabilidad y cuyas características son el uso de XML, la capacidad de serializar o deserializar y el envío y recepción de datos por la red. En la siguiente sección (6.1.2), se interpretan los resultados obtenidos con el objetivo de establecer las características de los lenguajes de programación.

6.1.2. Interpretación y análisis de resultados

En esta sección se da sentido y significado a los resultados obtenidos en los experimentos de interoperabilidad. En la primera parte se analizan los resultados de consumir el servicio *AgentManagerService*. En la siguiente parte se analizan los resultados del servicio *GradingManagerService*.

AgentManagerService

Los experimentos de la sección 5.1 sirven para mostrar las capacidades de WS sobre otros tipos de comunicación entre sistemas. Por ejemplo, en las tres formas de comunicación se consigue la interoperabilidad pero con diferencias en la deserialización de datos. En el caso particular de Hessian, los problemas de deserialización provocaron que los servicios tuvieran que ser descargados y usados como *jarfiles* para conseguir la deserialización. En el

caso de Burlap, la deserialización no ocurre como implementaciones de OKI.

En el caso de WS los problemas de deserialización se resuelven pues las clases conocidas en esta tesis como `Proxy/Stub` se encargan de crear clases basadas en las interfaces OKI y con la capacidad de realizar llamadas a nivel de sockets de Java. Los WS son entonces la forma que mejor satisface el objetivo de interoperabilidad usando las definiciones OKI como estándar.

En esta serie de experimentos se busca la interoperabilidad entre sistemas Java-a-Java por los que las alternativas de Hessian y Burlap resultan útiles a pesar de las consecuencias. En el caso particular de esta tesis, las soluciones no funcionan solamente por su incompatibilidad con las interfaces de OKI.

GradingManagerService

Una vez establecido el uso de WS por su utilidad en esta tesis, es momento de hacer pruebas más complejas con el uso de otros lenguajes de programación. En la sección 5.2 se hacen estas pruebas.

En la sección 5.2.5 se muestra una tabla comparativa de los lenguajes de programación usados en los experimentos. Por un lado, todos los lenguajes de programación consiguieron la interoperabilidad. En el caso de Javascript, existe la confusión de nombres de dominios. Esto solo muestra que el lenguaje aun sigue en adaptación a nuevas tendencias y en espera de una solución integral a este problema.

En esta tesis no se busca hacer una crítica a los lenguajes de programación sino de promover un paradigma producto de la abstracción de características funcionales. Estas características son obtenidas y presentadas en la tabla de la sección 5.2.5.

El uso de la red es una característica predecible pero necesaria para la comunicación de aplicaciones. Java usa las bibliotecas llamadas JAX-RPC. En .NET se usa el lenguaje común CLI WinSock (Windows Sockets). Al final estos dos lenguajes usan *sockets*[1]. Mientras tanto, PHP usa los sockets de las bibliotecas comunes del lenguaje C. En el caso de Javascript, el objeto XMLHTTPRequest de AJAX hace las llamadas en la red por medio del *browser* en donde sea creado (Firefox, MS Explorer, Mozilla, etc.).

[1] javax.rmi.Socket en el caso de Java y System.Net.Sockets para .NET.

En el caso del uso de serialización y deserialización se consigue desde todos los lenguajes. En el caso de Java, C# y PHP, las clases `Proxy/Stub` ocultan la deserialización en las clases generadas por las herramientas WS de cada lenguaje. Por un lado, Java y C# toman la información del mensaje SOAP y lo convierten en objetos nativos del lenguaje. Por otro lado, PHP convierte todo el mensaje SOAP en arreglos simbólicos propios del lenguaje. En el caso de Javascript, se hacen las llamadas directamente escribiendo los mensajes SOAP y mandándolo por el objeto XMLHTTPRequest por lo que la serialización y deserialización se traduce en obtener la información del mensaje SOAP en XML, obtenido del servidor de WS.

La manipulación de XML es una característica común en los lenguajes de programación actuales. Su uso se ha generalizado conforme el paso de los años. En el caso de Java y C# también son ocultos por el conjunto `Proxy/Stub` donde las llamadas a métodos se traducen en mensajes SOAP que son enviados a través de la red. Una vez que el servidor de WS toma la petición, la procesa y devuelve el mensaje de regreso en XML, ésta es obtenida por los `Proxy/Stub` y se obtienen los datos que considere útiles. En PHP se obtienen los mensajes y se transforman en estructuras de datos del lenguaje totalmente navegables. Para AJAX esto es directo, pues los mensajes XML de envío son manipulados directamente por el programador y una vez que se obtiene el mensaje de regreso se manipula como un archivo XML cualquiera.

6.2. Costos

En esta sección se revisan las ventajas y desventajas tras el uso de SOWA y WS vistas en la sección 2.2.3 para buscar la interoperabilidad entre aplicaciones basado en los resultados encontrados en los experimentos del capítulo anterior. De esta manera se fundamentan algunas beneficios de las arquitecturas orientadas a servicios mas allá del objetivo de la interoperabilidad, llegando a factores de diseño y reuso de código.

6.2.1. Ventajas comprobadas

A continuación una serie de beneficios del uso de la SOWA y WS, asi como de las herramientas y decisiones tras la implementación del objeto de estudio.

1. **Centralización de información**: La información capturada por un servicio es almacenada en un mismo medio independientemente de qué cliente lo llama. En los experimentos se puede ver, por ejemplo, la calificación obtenida en el cliente PHP listada junto a la calificación del cliente C#. Esta centralización es benéfica cuando se piensa en la información como un elemento compartido entre los clientes.

2. **Reuso de código**: Los servicios son módulos funcionales que atraviesan las limitantes de los lenguajes de programación. Esto significa que es posible acceder a una funcionalidad desde cualquier lenguaje de programación y desde cualquier punto de una red informática. No es necesario hacer cambios en el servicio para que sea compatible con diferentes lenguajes de programación. Los experimentos de la sección 5.2.5 muestran este hecho.

3. **Facilidad para generación de clientes**: El uso de herramientas en cada uno de los lenguajes de programación permite la generación de archivos necesarios para comunicar clientes y servidores en la SOWA, evitando la mayoría de la implementación y el pleno conocimiento de las clases o estructuras que permiten la comunicación por medio de la red. Como ejemplo de las herramientas usados en las pruebas, el uso de Apache Axis (véase apéndice sección A.5) para Java, .NET SDK (véase apéndice sección A.10) para .NET y NuSOAP (véase apéndice sección A.11) para PHP.

6.2.2. Desventajas comprobadas

La exposición y consumo de servicios remotos brinda interoperabilidad entre aplicaciones Web, sin embargo, existen ciertas variables que dificultan o entorpecen la comunicación. A continuación se muestra un compendio de estos costos tras la realización y documentación de los experimentos.

1. **Herramientas en vías de desarrollo**: Las herramientas de creación de clientes de consumo de servicios proveen la mayoría de la implementación necesaria para conseguir la comunicación. Sin embargo existen defectos en las clases que generan. Por ejemplo, la creación de clientes en .NET usando la herramienta wsdl.exe genera las clases en C# se conectan al puerto 80 en lugar del 8080. Esta diferencia se traduce en problemas de conexión haciendo suponer al programador del cliente que el servidor de WS no está disponible.

2. **Centralización de aplicaciones**: Cualquier cambio hecho en la estructura del servicio remoto significa un cambio en la funcionalidad del servicio. Todos los clientes del servicio son afectados por una simple corrección en la funcionalidad del servicio remoto.

3. **Inadaptación de nuevos lenguajes de programación**: Los lenguajes de programación podrían tener problemas de adaptación a la SOWA como ocurre con Javascript. La SOWA es la abstracción de características de los lenguajes de programación en Web pero debería de ser una abstracción de cualquier lenguaje de programación. Aunque en teoría esto sí ocurre, es necesaria la experientación para comprobar esta situación.

6.3. Comparación con el trabajo relacionado

En esta sección se hace un análisis desde el enfoque de los trabajos relacionados vistos en el capítulo 3. Por una parte se analizan las similitudes con la *Service-Oriented Network Arquitecture* de Cisco Systems. Posteriormente se revisa la *Service-Oriented and Modelling Architecture* de IBM.

6.3.1. Comparación con SONA

En la arquitectura orientada a redes de servicios de Cisco Systems (véase sección 3.1) se puede hacer una comparativa con respecto a sus componentes y conectores.

La capa de **Infraestrutura de Red** es comparable a SOWA en el cliente y el servidor. La capa de **Servicios Interactivos** puede considerarse como las acciones de Descubrimiento y Publicación de servicios remotos. Y la capa de **Aplicacion** es la acción de Descubrimiento, Publicación e Interacción aunque llevado a nivel de redes compartidas con la Red de Información Inteligente de la SONA.

La forma de conseguir la interoperabilidad es la misma en ambas arquitecturas siendo la mayor diferencia en la sección de descubrimiento. En SONA se espera que exista una red de servicios remotos actualizándose cada vez que un servicio remoto es expuesto en la red. Sin embargo, esta red aun es un proyecto de la compañia por lo que esta sección queda pendiente a ser comparada por la SOWA. La parte de descubrimiento de la SOWA no es revisada en esta tesis aunque la definición usada en los experimentos es más

genérica que SONA por lo que la comunicación con estas arquitecturas sería similar.

6.3.2. Comparación con SOMA

En la Arquitectura y Modelado Orientado a Servicios de IBM se hace un mayor énfasis al diseño de aplicaciones que pueden ser expuestos como servicios remotos. A continuación una comparativa con los conectores y componentes de la SOMA.

Como en SOMA todo es diseño, no existe una comparativa directa con los componentes de SOWA. Sin embargo, las etapas de SOMA, una vez que son implementadas se pueden comparar con SOWA de la siguiente manera.

Las etapas de **Servicios** y **Proceso de Negocio y Descripción de Servicios** se mapean como el componente de Descubrimiento en SOWA. En la etapa de **Descubrimiento** se determinan las funciones de Cliente y Servidor. El descubrimiento y comunicación ocurre aquí por cualquier método y medio de comunicación que se decida.

Con todo lo anterior se puede decir que ambas arquitecturas realizan el mismo tipo de comunicación entre sistemas aunque uno más enfocado al diseño que el otro. SOMA tiene un mejor control del diseño aunque desafortunadamente está ligado a la herramienta *Websphere Business Integration Solution*. No existen diferencias funcionales entre un servicio remoto implementado con ayuda de SOMA a uno implementado con SOWA pues la comunicación acurre igualmente en un protocolo de envío/recepción de datos y un tipo de datos como XML.

6.4. Revisión de las contribuciones

Con referencia a la sección 1.5, se hace una revisión de las contribuciones y los resultados de los experimentos realizados.

Esta tesis es una experiencia documentada. La arquitectura Web orientada a servicios ha sido probada con los experimentos documentados en el capítulo 5. Así pues, durante la tesis se define el objetivo de interoperabilidad. Se propone el uso de una SOWA para responder a ese objetivo y tras los experimentos realizados se consigue la comunicación entre sistemas.

La documentación de estas pruebas muestra las diferentes experiencias con los lenguajes de programación. Son el resultado de corroborar el consumo de un servicio remoto, usando las herramientas provistas por cada lenguaje de programación y mostrando sus efectos en el cliente y el servidor.

Por ejemplo, en el caso de Javascript y AJAX existen problemas de comunicación por una confusión de nombres de dominios. Este problema es resuelto haciendo uso del lenguaje de programación PHP. Esta experiencia puede ser tomado como un consejo al consumir un servicio remoto en lo sucesivo.

6.5. Investigación futura

En esta sección se argumentan temas de interés que toman a esta tesis como fundamento para nuevas investigaciones. Estos temas son puntos no alcanzados por esta investigación y que presentan por si mismos temas de estudio con fines no determinados.

Existen variables alrededor de la interoperabilidad no contabilizadas en esta tesis como los tiempos de comunicación entre sistemas, interoperabilidad entre diferentes sistemas de hardware o seguridad en la comunicación de aplicaciones. Estos temas conjuntan diversas áreas de la computación que pueden desembocar en propuestas de nuevas arquitecturas Web o diferentes formas de comunicación alternas a los WS.

En esta tesis se definen ciertas reglas para conseguir la comunicación entre sistemas Web, siendo estas probadas y documentadas. Se promueve el uso de servicios para resolver el problema de interoperabilidad de serialización de datos, quedando por demostrar los efectos de la SOWA en el problema de sincronía descrito en el contexto de esta tesis.

Apéndice A

Herramientas útiles para el desarrollo del sistema

A.1. Vista general de las herramientas

La tecnología es un factor importante en la creación de sistemas de software, particularmente en el módulo de evaluación mostrado en esta tesis. A pesar de la codependencia existente entre el diseño, teoría y tecnología es esta última la más cambiante con respecto al tiempo. Esto trae como consecuencia que las herramientas usadas para el desarrollo de un sistema Web sean el único medio para concretar los fundamentos establecidos por la teoría y el diseño. Por lo anterior, crean cambios estructurales, siendo en muchos casos, factores para cambiar las decisiones en el desarrollo de un proyecto. Por ello, las herramientas están ligadas al tiempo siendo susceptibles a ser remplazadas por alguna otra nueva.

Un ejemplo es Eclipse. Se trata de un conjunto de herramientas para la creación de proyectos de software. Dado que Eclipse está implementado en Java es posible el uso de *Plugins*. Con ayuda de estos módulos, Eclipse logra extender sus capacidades. En el diseño del caso de estudio se usa Eclipse para la creación de diagramas UML o representaciones gráficas de bases de datos y a partir de los mismos generar cierto código como archivos de configuración, clases Java y archivos SQL para un manejador de bases de datos.

Existen plugins de Eclipse libres y comerciales. MyEclipse [1], por ejemplo, contiene plugins a partir de los cuales es posible hacer la mayor parte

[1] www.myeclipse.com

de la implementación expuesta en el caso de estudio. Facilita el diseño de un sistema poniendo a disposición diagramadores para representarlo. Además provee del mecanismo a partir del cual es posible la traducción de los diagramas a clases Java, archivos de configuración para diferentes *frameworks* o librerías de Java disponibles, archivos HTML y sus diferentes versiones para páginas dinámicas de Internet así como conecciones con manejadores de bases de datos. Desafortunadamente MyEclipse es comercial con costo al usuario.

Otro ejemplo con respecto a las versiones de Eclipse es EasyEclipse [2]. Al igual de MyEclipse es la recopilación de varios plugins enfocados al diseño y desarrollo de sistemas en Web. La virtud de este sistema es su disponibilidad como software libre ya que está formado por plugins gratuítos y previamente probados. Aunque tiene una serie de errores de compatibilidad entre los mismos es una opción importante con respecto a los demás sistemas de edición e integración conocidos como *Integrated Development Environment* (IDE). A continuación se muestra una tabla de plugins de Eclipse usados en el caso de estudio. También pueden ser encontrados en Eclipse Plugin Central[3] y en EclipsePlugins[4]:

Plugins de Eclipse SDK

Tecnología	Categoría	Alternativas Gratuítas	Alternativas Comerciales
Omondo	Diagramas UML	EasyEclipse	MyEclipse
Azurri	Diagramas para SQL	EasyEclipse	MyEclipse
Hibernator	Generador de archivos hbm	EasyEclipse	MyEclipse
WTP	Creación archivos WSDL	EasyEclipse	No aplica

[2] www.easyeclipse.org/site/home/
[3] www.eclipseplugincentral.com
[4] eclipse-plugins.2y.net/eclipse

A continuación se muestra una tabla con información correspondiente de las tecnologías disponibles en la elaboración de un proyecto para Web y que han sido usadas en la elaboración del módulo de software correspondiente a esta tesis. Además se indican alternativas gratuítas y comerciales [18].

Tecnología	Categoría	Alternativas Gratuítas	Alternativas Comerciales
Spring Framework (springframework.org)	Contenedor de inversión, Framework para Web	Struts, JSF, Tapestry	No aplica
Struts (jakarta.apache.org/struts/)	Control MVC Framework para Web	Spring	No aplica
Hibernate (hibernate.org)	Framework de Persistencia	EJB, JDO, iBatis	TopLink de Oracle
Ant (ant.apache.org)	Manejo de configuraciones	make, gnumake, maven, jam	MS nmake, MKS nmake
jUnit (junit.org)	Framework de pruebas	Test NG, Fit	Mercury Load Runner
PostgreSQL (postgresql.org)	Manejador de bases de datos	MySQL, HSQLDB	Oracle, Sybase, DB2
Apache Tomcat (tomcat.apache.org)	Contenedor de Servlets, servidor HTTP	Jetty	IBM Websphere, BEA WebLogic, Caucho Resin
Firefox 2.0 (mozilla.org)	Web Browser	MS Explorer, Netscape	No aplica
Eclipse SDK (eclipse.org)	Kit para desarrollo de proyectos	NetBeans, jEdit, jCreator	IntelliJ, IBM Websphere Studio App Developer
Displaytag (displaytag.org)	Librería JavaScript	Logic e Iterate de Struts	No aplica
Axis (axis.apache.org)	Generador de servicios	No existe	Websphere
Castor (castor.apache.org)	Convertidor XML - POJO	No existe	Websphere

Tecnología	Categoría	Alternativas Gratuítas	Alternativas Comerciales
Log4J (apache.org/log4j)	Mensajes de Debug	Log	WebSphere Logging
.NET SDK Redist. (microsoft.com)	Aplicaciones en .NET y WS	Mono, Visual St. Express, Indigo (Vista)	Visual St. Professional
NuSOAP (sourceforge.net/ projects/nusoap)	Aplicaciones PHP y WS	MyEclipsePHP	NuSphere

A.2. Servlets, JSP's y sus contenedores

A.2.1. Servlets

Los *servlets* son clases Java unidos a cada acción que ocurre en una página Web. Las acciones o eventos son todas aquellas cosas que un usuario del sistema puede hacer, como presionar un botón o usar una de las ligas de la página. Los servlets tienen un método llamado *execute* y se pueden pasar como parámetros todos aquellos datos necesarios para ser manipulados desde la clase Java. Todos los datos que se mandan al entrar y al salir del servlet pueden ser guardados en estructuras de memoria en el servidor conocidos como *scopes*. Por la permanencia de los datos en la memoria del servidor, los scopes se dividen en:

- **Page:** Es la memoria más corta para los datos. Solo dura el tiempo en el cuál el usuario está visitando esa página. Su uso se recomienda para ser usadas por tecnologías como Javascript o Web 2.0 que buscan hacer uso de la información en el lado del cliente. Cuando el usuario cambia de una página a otra, los datos contenidos en la memoria Page se pierden.

- **Request:** Está apegada a las acciones o eventos de la página. Por ejemplo, al llenar un formulario de una página de Internet, la información dada se guarda en memoria Request. Con esto se asegura que los datos del formulario se mantienen constantes hasta el momento en el cual el servidor manipula los datos para ser guardados. Cuando el servidor recibe la información del cliente la memoria Request se vacía.

- **Session:** Esta memoria es un poco más duradera. Existe mientras el usuario esté usando el sistema. Si en algún momento el usuario se dirige a otra dirección de Internet los datos en la memoria Session se pierden.
- **Aplication:** Es la memoria más duradera de todas. Existe mientras esté disponible el sistema en la Internet. La única forma en la que la memoria Aplication se pierde es cuando el servidor tiene algún error o no está disponible.

A.2.2. Java Server Pages

Los *Java Server Pages* (JSP) son una forma de poner código Java en una página con formato HTML. Este código es llamado *scriplet* y desde él es posible llamar clases Java, definición de métodos, etc. La ejecución de un scriplet ocurre del lado del servidor por lo cuál se pueden manejar los eventos de manera dinámica y proveer diferentes comportamientos. Como los JSP's contienen código en lenguaje Java es necesario su compilación. Esto ocurre por medio del servidor justo al momento de mostrar la página.

A.2.3. Contenedores de Servlets/JSP's

El contenedor de servlets, es entonces un programa que se ejecuta del lado del servidor y en el cual se guardan los archivos HTML, JSP, biblioteca o .class referentes a un sistema Web. La estructura de los archivos está previamente definida para ser encontrados automáticamente por el contenedor. Otra de sus funciones es hacer una traducción de un JSP a servlet. Por medio de los datos escritos en el JSP es capaz de traducirlos a una clase Java con la estructura dada de un servlet y con la funcionalidad dada por el programador. Después compila el servlet autogenerado y muestra como un HTML común con la funcionalidad de manejo de eventos por los servlets de Java. Existen varios contenedores de servlets y JSP's disponibles en Internet. El primer contenedor en existir es Jetty pero el más usado en la actualidad es Jakarta Tomcat.

En Tomcat el conjunto de archivos referentes a un sistema Web se llama *contexto*. Tomcat maneja todos los contextos y sus archivos de configuración en XML. Programas de integración y desarrollo de proyectos Web como Eclipse y NetBeans contienen una versión del Tomcat para permitir que programadores prueben su sistema de manera local.

A.3. Spring Framework y Struts

Los frameworks son fragmentos de programas que proveen cierta funcionalidad dado un lenguaje de programación. Con ayuda de los frameworks es posible ahorrar trabajo en implementación. Como son un conjunto de funciones previamente implementados, pueden considerarse servicios locales. En Java, estos servicios locales están disponibles en archivos llamados *jarfiles* e integrarlos en el desarrollo de un proyecto Web se resuelve sólo por mensajes y llamados al archivo .jar que lo representa. Al ejecutarse los programas también deben ser llamados para que el compilador del lenguaje entienda la ubicación del framework. El *classpath* es el parámetro necesario para que el compilador de Java lo entienda y en él van todas las ubicaciónes de los frameworks.

A.3.1. MVC Módelo 2 y Jakarta Struts

El Modelo 2 se trata de una variación del patrón de software de Model-View-Controller visto en el capítulo 2. Struts promueve su uso [20], aunque no obliga a los desarrolladores a usarlo. La diferencia de Modelo 2 con respecto a MVC tradicional está en la total independencia de las partes Modelo, Vista y Controlador. Esto se ve reflejado en la implementación y la estructura del proyecto.

En Struts existen los archivos de configuración, los cuales tratan de mantener esa independencia. Los archivos están en formato XML y por las prácticas de Modelo 2, todas las acciones de la aplicación deben de pasar por al menos una de las definiciones de Struts.

Los archivos de configuración de Struts y una breve explicación se mencionan a continuación [20].

- **struts-config:** Aquí está toda la configuración de Struts. Se menciona la localización de todos los demás archivos de configuración. Además los nombres de las acciones en el sistema están definidas en este archivo.
- **tiles-defs:** Las definiciones "tiles" son los nombres y estructuras de los archivos JSP disponibles a traves de una acción. Es decir, las acciones ocurren en una página cualquiera, son captados en struts-config y son mandados a una definición contenida en tiles-defs.

- **validator:** Las validaciones son todas aquellas reglas que deben pasar los datos en un sistema. Supóngase el caso en la que se muestra un formulario para que un cliente escriba su número telefónico. Si el usuario comete un error y en lugar de número escribe letras, entonces el sistema debe de poner cierta protección para que esto no ocurra, mandando mensajes al usuario que le hagan ver su error. En el archivo validator están todas las posibles validaciones que pueden hacerse en un sistema. Al menos Struts tiene las más comunes aunque el programador puede proponer sus propias validaciones. Estas últimas pueden ser escritas en formato XML cuando se hacen del lado del servidor y en formato de Javascript cuando se desea hacer desde el lado del cliente.

- **validator-rules:** Finalmente en este archivo están dadas las reglas de las validaciones. Por medio de los nombres de los formularios de la página en HTML o JS,P es posible definir las reglas. Retomando el ejemplo anterior, si el formulario del número telefónico es llamado "teléfono", entonces en el validator-rules se especifica cuando exista un formulario de nombre "teléfono" deberá de aplicar la validación de número telefónico.

La idea original de Modelo 2 es la de mantener cualquier acción del sistema informado a los archivos de configuración [20]. Así se define la independencia de los módulos de Modelo-Vista-Controlador.

Existen otras capacidades de Struts. Los archivos llamados *tld* redefinen algunas de las banderas de HTML para hacerlos compatibles a funciones de Struts y evitar la codificación por medio de scriplets. Por ejemplo, existe el llamado `logic.tld` y `iterate.tld` que permite tener control de flujo de acciones como *if* o *while* e iterar listas guardadas en *scope*, respectivamente. Por otro lado es posible la integración con otros frameworks como Spring o jUnit con las clases implementadas de Struts.

Finalmente Struts cuenta con una interfaz dedicada a las pruebas unitarias de un sistema Web. Por medio de la interfaz `TestCase` es posible hacer pruebas de todas las acciones del controlador.

A.3.2. Spring y algunas de sus capacidades

Spring es uno de los frameworks mas completos y complejos de la actualidad. Es similar a Struts en la parte de la integración MVC. Sin embargo,

Spring es mucho más que eso. La gráfica A.1 muestra este hecho. La parte de Spring MVC y Spring Core son equivalentes a todo el framework de Struts.

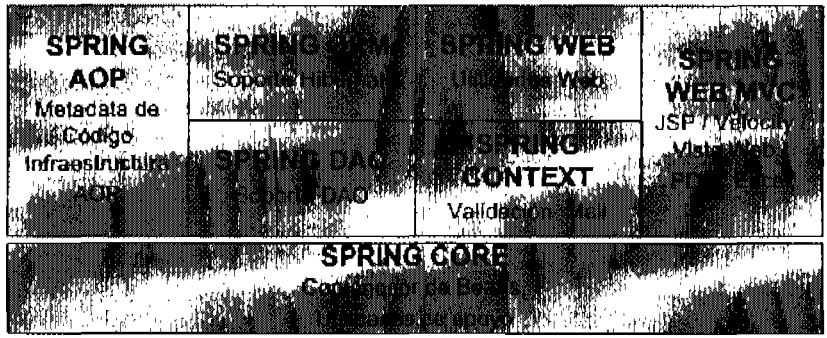

Figura A.1: Diagrama de bloques de Spring Framework

Spring complementa la idea de Java para Web [5] con el Spring Core que sirve para el control de conexiones a bases de datos, manejo de calendarización, transacciones, Dependency Injection y capacidades de correo electrónico.

Una de las características importantes de Spring es la de manejar *Dependency Injection*. Con ayuda de su archivo de configuración [6], es posible asignar las clases dependientes a implementar las interfaces de un sistema. En este archivo se escriben todas las dependencias y se asignan en el momento de ejecutar el sistema. De esta manera queda independiente de la ejecución del sistema y persistente durante la ejecución.

A.4. Hibernate

En Java, cuando una clase es implementada solo con atributos y los métodos para accederlos y establecerlos (atributos y métodos *setters y getters*) se le conoce como *Java Bean* o POJO (Plain Old Java Object). Al

[5] también conocido como Java 2 Enterprise Edition (J2EE)
[6] llamado *applicationcontext.xml*

implementarse una clase de conexión a base de datos es necesario hacer la traducción de los datos provenientes de un *Bean* hacia datos que comprenda el manejador de bases de datos. *Hibernate* es un *framework* dedicado a la persistencia de mapeo de datos de objeto-a-relacional (OR), esto es, la traducción de POJO a datos en la base de datos y viceversa. Además de la traducción, *Hibernate* hace la lectura-escritura de información de la base de datos. *Hibernate* funciona por medio de archivos de configuración escritos en XML. Estos archivos son llamados *hibernate mapping* en el caso de los mapeos de cada *Bean*. El archivo general de configuración de hibernate es el encargado de hacer la conexión a la base de datos. En él, se especifican los datos importantes para conseguir la conexión en la base de datos y los *mappings* a realizar en escritura y lectura de la base de datos. Cambiando este archivo de configuración es posible cambiar de manejador de base de datos.

A.5. Axis, Castor y Eclipse

Axis y *Castor* son proyectos de Apache. Con la ayuda de *Axis* es posible generar clases Java a partir de ciertas reglas. Estas clases Java son usados para comunicar dos sistemas por medio de WS. *Axis* crea las clases para cliente y servidor de servicios remotos a partir de un archivo WSDL. Además *Axis* sirve como contenedor de servicios remotos. Esto es, una vez que se generan las clases de servidor, *Axis* los expone en la red como sistema Web. Por otro lado, la característica especial de *Castor* radica en la capacidad de controlar la serialización y deserialización de estructuras de datos. Cuando *Castor* recibe un archivo XML como por ejemplo los mensajes SOAP, puede generar estructuras con esos datos. Como *Castor* funciona en Java, las estructuras de datos generadas son POJO's y ser usados con el objetivo que se requiera.

A.6. Ant

Otro proyecto de Apache. Es una herramienta basada en XML para describir "tareas". Estas tareas pueden ser tan complejas como las defina el programador y pueden ir desde una simple compilación de los archivos Java a partir de cierto *classpath*, la creación de directorios o lugares de origen de las clases compiladas, hasta la creación de tareas para hacer pruebas del sistema o creación de tablas en un manejador de bases de datos. Otra característica de *Ant* es ser independiente de plataforma, esto es, puede ser ejecutado en

cualquier sistema operativo sin necesidad de hacer cambios en su estructura. Debe entenderse que las tareas de *Ant* están hechas exclusivas para el sistema en el que está alojado por lo cual, los cambios de un sistema operativo a otro implican cambiar el contenido de algunas de las tareas. Por tanto los datos realmente importantes que pueden cambiar al ejecutar un a tarea pueden estar en otro archivo llamado *properties*. Se trata de archivos que pueden ser usados por cualquier archivo del sistema con datos simples, guardados como texto plano. La idea es cambiar solamente el archivo *properties* y no tener que buscar la localización exacta de la tarea en el archivos de *Ant* (conocido como build.xml).

A.7. jUnit

Es un *framework* hecho para hacer pruebas en las aplicaciones. Por medio de este *framework*, es posible automatizar una serie de pruebas visto desde una clase Java. Con jUnit se reconoce cuales métodos de una clase fallaron. Con las pruebas automatizadas es posible rehacer toda la tanda de pruebas al mas mínimo cambio estructural en el sistema. Existen dos presentaciones de jUnit en la red: la primera que puede ser por modo de interfaz gráfica de usuario y la segunda es por plugins de Eclipse o módulos de Netbeans.

A.8. PostgreSQL

Es un manejador de bases de datos. Se encarga de guardar los datos en un modo confiable. Al igual de todos los demás manejadores de bases de datos relacionales, guarda los registros en tablas. Los datos a guardar están disponibles solo a partir de una conexión con la misma. La seguridad de *PostgreSQL* es por cifrado por lo cual, valores importantes como palabras de acceso sólo están disponibles a usuarios con el rol de administrador. La forma de conexión a una base de datos puede ser de varias formas. La conexión desde una clase Java es por medio de *Java Data Base Conection* (JDBC). Con ella es necesario el uso de un *driver*, nombre de usuario y password. El *driver* es un programa java capaz de reconocer la codificación nativa del manejador de bases de datos y hacer el paso de información entre datos nativos de Java y la base de datos. Por ejemplo, hace la traducción del nativo `int` de Java a `integer` de *PostgreSQL*.

Dado que los datos para realizar la conexión son más o menos sencillos de obtener es posible hacer la conexión desde *Hibernate* o desde una clase Java

cualquiera. Esto significa que la conexión es independiente con el manejador de base de datos. Dependiendo del diseño es posible decidir la forma de conexión o bajo cuáles términos se debe hacer la conexión.

A.9. Firefox, Javascript y Displaytag

Firefox es un *browser* de los creadores de Mozilla. La utilidad de Firefox en el desarrollo de un sistema es importante pues contiene una serie de plugins a su alcance, facilitando las pruebas o validación. Por ejemplo, por medio de plugins como *Web Developer* es posible tener acceso a información de una página Web como las hojas de estilo que lo conforman, los archivos javascript y si existen errores en el mismo. La validación de un sistema es importante. La World - Wide - Web Consortium o W3C provee la validación de un sistema desde el punto de vista de sus hojas de estilos, validación de enlaces en la página o de la codificación en HTML. En el caso de las páginas de Internet, XHTML (HTML con estilo de XML) es la validación mas común. Por otro lado, se puede tener conocimiento de imágenes, información en formularios, y *cookies* (datos utiles para el programador en la memoria de *Scope*).

Javascript es un lenguaje de programación que ocurre en el lugar del cliente. Toda la información provista por el cliente puede ser manipulada antes de ser enviada hasta el servidor. Para ello Javascript es un lenguaje que se ejecuta en los *browsers*. Dependiendo del *browser*, el intérpete Javascript puede cambiar, aunque a últimos años la diferencia entre los intérpretes es cada vez menor. Por ejemplo, la forma de presentar una página con MS Internet Explorer tiene ligeras diferencias con respecto al de Netscape u Opera. Esta diferencia también afecta el desempeño de una página de Internet. La manipulación de datos desde Javascript también tiene esas diferencias entre un *browser* y otro. Por tanto el uso de librerías para Javascript tiene repercusiones dependiendo del browser en el cual el cliente ve la información.

Displaytag es una librería de Javascript. Por medio de este, es posible mostrar tablas a partir de listas guardadas en *scope*. Anteriormente se menciona la misma funcionalidad con respecto a *logic* e *iterate* de Struts. Tiene la capacidad de auto ordenar la información de sus tablas y hacer un manejo previo de los datos antes de ser mostrados. Para este último, Displaytag tiene los medios para implementar el patrón de diseño de software *Decorator*. Por medio de una clase Java es posible tomar un dato y "decorarlo" a

un formato que interese y mostrarlo en pantalla. Además, Displaytag tiene predefinidos nombres de clases en hojas de estilos asi pues, mostrar una lista de datos en una página se consigue con solo unas cuantas líneas de código HTML.

A.10. .NET Redistributable y .NET SDK 1.1

El .NET Redistributable es una serie de funcionalidades instalables en el sistema operativo Microsoft Windows para poder ejecutar aplicaciones en .NET. Las aplicaciones hechas en lenguajes como Visual Basic .NET, C# .NET y J# .NET pueden ejecutarse en Windows [28].

Por otro lado, el .NET SDK 1.1 es un conjunto de programas que permiten la compilación y ejecución de programas escritos en los diferentes lenguajes de programación de .NET. Los programas hechos en Visual Basic .NET, C# .NET y J# .NET son traducidos a un lenguaje común llamado CLI. Es por ello que cualquier programa escrito en estos lenguajes funcionan comosi hubieran sido escritos en el lenguaje único CLI. Además de los compiladores, cuenta con otras características como el llamado "wsdl.exe" para generar clases *Proxy/Stub* para poder comunicarse con un servicio remoto a partir de un WSDL ubicado en la red. Asi pues, comunicarse con un servicio remoto se hace solo a partir de las clases generadas por esta herramienta [28].

A.11. NuSOAP

Asi como .NET tiene su herramienta dedicada a WS, PHP tiene la suya [11]. NuSOAP es la herramienta de más fácil uso para comunicarse con servicios remotos. Cuenta con clases predefinidas para llamar a los servicios desde un script en PHP y usarlos de manera transparente.

Bibliografía

[1] Andrieux, Alain, K. Czajkowski, A. Dan. *Web Services Agreement Specification (WSAG).* Web Services Official Draft. Versión 2005/09.

[2] Armstrong, Chris. *Modeling Web Services With UML.* OMG Web Services Workshop 2002. Disponible en www.atcenterprices.com

[3] Baresi, Luciano, E. Di Nitto y C.Ghezzi. *"Toward Open-World Software: Issues and Challenges" en Computer Society's E-News.* IEEE Computer Society. Octubre 2006.

[4] Beck, Kent. *Extreme Programming Explained, Embrance Change.* Addison-Wesley. 2000.

[5] Burke, Eric M., B.M. Coyner. *Java Extreme Programing Cookbook.* O'Reilly and Associates, Inc. 2003.

[6] Cisco Systems White Papers. *Service-Oriented Network Architecture.* Cisco Publishing. Disponible en `www.cisco.com/application/pdf/en/us/guest/netsol/ns477/c643/cdccont_0900aecd8039b324.pdf`

[7] Cooper, James W. *The Design Patterns Java Companion.* Addison-Wesley. 1998.

[8] CORBA. http://en.wikipedia.org/wiki/CORBA. 19 de Mayo de 2008.

[9] Daconta, Michael, C.L. Obrst y K.T. Smith. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management.* Wiley Publishing Inc. 2003.

[10] DCOM. http://en.wikipedia.org/wiki/Distributed_Component_Object_Model. 19 de Mayo de 2008.

[11] Diez, Ricardo. *"Servicios Web PHP con NuSOAP" en Solo Programadores.* Sección Redes. Año 12, Segunda época, número 137. 2006.

[12] Farley, Jim. *Java Distributed Computing.* O'Reilly and Associates, Inc. 2001.

[13] Ford, Neal. *Art of Java Web Development: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, Webwork.* Manning Publications Co. 2004.

[14] Gabrick, Kurt A., D.B. Weiss. *J2EE and XML.* Manning Publications. 2002.

[15] Gamma, Erich, R. Helm, R. Johnson y J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* HardBack. 1995.

[16] Garcia-Pelayo, Fernando. *Pequeño Larousse Ilustrado.* Libraire Larousse. 1964.

[17] Greenfield, Jack, K. Short, S. Cook y Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools.* Wiley Publishing. 2004.

[18] Hemrajani, Anil. *Agile Java Development with Spring, Hibernate and Eclipse.* Sams Publishing. 2006.

[19] Hessian. http://en.wikipedia.org/wiki/Hessian_(protocol). 19 de Mayo de 2008.

[20] Hightower, Richard. *Jakarta Struts Live.* Source Beat, LLC. 2004.

[21] Ibargüengoitia, Guadalupe, H. Oktaba. *Ingeniería de Software para Principiantes: Para trabajar en equipo y con elementos de MoProSoft.* Facultad de Ciencias UNAM. 2005.

[22] IBM White Papers. *IBM Service Oriented Modeling and Architecture.* IBM Service Consulting Services. 9 de noviembre de 2004. Disponible en `www-935.ibm.com/services/us/gbs/bus/pdf/g510-5060-ibm-service-oriented-modeling-arch.pdf`

[23] Interoperability. http://en.wikipedia.org/wiki/Interoperability. 19 de Mayo de 2008.

[24] IT World Canada. *Analysts confused with Cisco's SONA initiative.* 21 de Septiembre de 2006. Disponible en `http://www.itdirection.net/it-news-0004/ 092106-00080-it-news.shtml`

[25] Krol, Ed. *The Whole Internet.* Segunda edición. O'Reilly and Associates, Inc. Abril 1994.

[26] Mak, Gary. *Spring Recipes: A Problem-Solution Approach.* Appress. 2008.

[27] McLaughlin, Brett. *Java and XML.* O'Reilly and Associates, Inc. 2000.

[28] MS Dot Net. http://en.wikipedia.org/wiki/.NET_Framework. 19 de Mayo de 2008.

[29] Moore, Jeff. *"Dependency Injection" en PHP Architect: Total Eclipse of PHP Development.* Volumen 5 Tomo 6 Junio 2006.

[30] POSIX. http://en.wikipedia.org/wiki/POSIX. 19 de Mayo de 2008.

[31] Potts, Stephen, M. Kopack. *Sams Teach Yourself Web Services in 24 Hours.* Sams Publishing. 2003.

[32] Richard, Robert. *Pro PHP XML and Web Services.* Apress Publishing. 2006.

[33] Salvadori, Martin. *Serie de Reportajes "Universo SOA" en .code Comunidad de Desarrolladores.* Volumen 24. Primera Edicion. Buenos Aires, Argentina. MP Ediciones. 2006.

[34] Thorne, Jeff. *OKI Architectural Concepts.* Disponible en http://okiproject.org/filemgmt-data/files/OkiArchitecturalConcepts.pdf. 2007.

[35] Tong, Ka lok. *Developing Web Services with Apache Axis.* Tip Tec Development. Primera Edición. 2005.

[36] Walls, Craig y Ryan Breidenbach. *Spring in Action.* Manning Publications. 2005.

[37] Web Services. http://en.wikipedia.org/wiki/Web_service. 19 de Mayo de 2008.