Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Ingeniería en Ciencias de la Compuación

Tesis de licenciatura

ESTUDIO Y APLICACIÓN DEL DESARROLLO BASADO EN MODELOS (MDA) EN REDES SOCIALES

Por

Autor: Luis Josué Calva Rosales

Asesor: Dr. Abraham Sánchez López

Puebla, 2013

Para mi familia, amigos y asesor.

Índice general

Agradecimientos

A mi madre Dulce Roció Rosales Gómez que siempre ha estado hay para mí y con la cual he tenido las mejores charlas sobre cualquier tema en mi vida, y que cuando nadie confiaba en mí siempre me motivaba con una sonrisa para salir adelante.

A mi padre Luis Antonio Calva Diez que me han enseñado la importancia de ser fuerte y proteger a las personas que me rodean, que me ha motivado a seguir teniendo metas y que siempre existen las oportunidad solo hay que saberlas buscar.

Agradezco a mis hermanas Delia Denice Calva Rosales y Dulce Valera Calva Rosales que me han motivado han ser una mejor persona, que me han enseñado lo importante que es el respeto a las mujeres y con las cuales me divertí durante mi infancia entre peleas y risas.

Agradezco a mi tía Rosa Ana Rosales Gómez que aun con su enfermedad me sigue recibiendo con una sonrisa y que es un vivo ejemplo de lucha y amor a la vida.

A mis abuelos por enseñarme el valor de vivir y la experiencia que da la vejez. A mi abuela Delia Gómez Lara que siempre me ha querido y a la cual le estoy muy agradecido por sus consejos.

Agradezco a toda mi familia que me ha acompañado y motivado en todos estos años de lucha para hacerme lo que soy en especial a María Del Rosario Calva Diez, Guadalupe Carmona Calva, Andrés Arturo Calva López, Roxana López Rosales, Teresa Ylenia Rosales Ramírez, Ximena Rosales Ramírez, Pedro Ramírez Calva por haberme permitido sentir que tengo un hogar y más que una familia.

A mis profesores que me han ayudado a crecer y explotar mis habilidades para todos ellos mi respeto y admiración. A mi asesor DR. Abraham Sánchez López que me ha permitido crecer como estudiante y que confió en mí en el desarrollo de este trabajo.

Agradezco a mi novia Victoria Niño Pineda que me ha acompañado en estos últimos 3 años a seguir luchando y que me ha enseñado la importancia de no darse por vencido con la cual he aprendido el verdadero valor del amor de pareja.

A mis amigos que siempre han estado hay y que se han chutado mis buenos y malos momentos a todos ustedes que saben quiénes son todo mi cariño.

Índice de figuras

Índice de cuadros

Capítulo 1

Estado del Arte

1.1. Introducción

Debido a la evolución que día a día se lleva a cabo en el internet y las redes sociales, es necesario crear sistemas Web independientes a las plataformas, esto con el fin de agilizar los procesos y que tengan la capacidad de evolucionar cómo evoluciona la tecnología.

El crecimiento de los servicios Web ha generado un cambio de paradigma en la programación de los sistemas de información, en un principio el paradigma utilizado era el estructurado, el cual se basaba principalmente en la utilización de funciones como el punto principal del sistema, a continuación surgió el paradigma orientado a objetos, el cual pretende adoptar clases y objetos como la columna vertebral del desarrollo de software; en este trabajo se utilizara el computo orientado a servicios (SOC) el cual nos permitirá la evolución de la plataforma y adaptarlo a la red de una manera natural.

Las redes sociales en internet están siendo utilizadas por las empresas y personas como herramientas de comunicación libres que permiten a los usuarios expresarse, informar y divertirse, sin embargo mucha gente deja de lado la información y se enfoca más en quien lo dice y no en como lo dice. Nuestro proyecto de red social de nombre AnonymousRemark dejara de enfocarse en el usuario poniendo más atención al contenido de los mensajes por lo cual se seguirán ideas y no personas haciendo que los mensajes estén respaldados por usuarios. Para la realización de esta red social utilizaremos servicios Web gracias a que cuentan con estándares en la Internet, y MDA para guiar el desarrollo de la misma.

En estos días gracias al crecimiento del Internet, las redes sociales, los diferentes lenguajes de programación y las muchas tecnologías que existen, han causado problemas de compatibilidad. Anteriormente muchos sistemas no tenían la posibilidad de compartir información con otros o era muy ineficientes, hasta el nacimiento de los servicios Web.

Antes del nacimiento de los servicios Web se gestaron diversos tipos de tecnologías que lograron comunicarse entre diferentes tipos de plataformas, sin embargo muchas de estas tecnologías eran privativas o no daban el rendimiento necesario en la Internet como lo eran RPC, CORBA, RMI, etc.

Lo que ha permitido que los servicios Web crezcan como una manera de desarrollar sistemas complejos, es la utilización de estándares y herramientas con las que cuenta: XML, SOAP, UDDI, WSDL de los cuales se hablara en este trabajo de tesis.

Para la realización de cualquier proyecto de software es importante la realización de

los proyectos utilizando ingeniería de software. En este trabajo de tesis hablaremos de MDA (Model Driving Architecture) con el que dirigiremos el desarrollo de este proyecto. MDA divide el proyecto en los PIM (Modelos Independientes de la Plataforma) y PSM (Modelos Específicos de la Plataforma) de los cuales se habla más a fondo después.

Aunque la utilización de las redes sociales es prácticamente joven, estos sigues siendo sistemas de información, por lo tanto tiene una arquitectura y un diseño; en la siguiente sección se hablara acerca de los sistemas de información orientados al Internet.

1.2. Los sistemas de información: de la distribución al Internet

Abordamos los sistemas de información distribuidos, resaltando principalmente los siguientes aspectos: el diseño, la arquitectura, y los patrones de comunicación [4, 28]. A continuación abordaremos los Middleware que son uno de los soportes indispensables para los sistemas de información distribuidos.

1.2.1. El diseño de un sistema de información

El diseño de un sistema de información puede organizarse en forma general en 3 capas (layers): presentación, lógica de la aplicación y administración de recursos. La Figura 1.1 presenta estas capas.

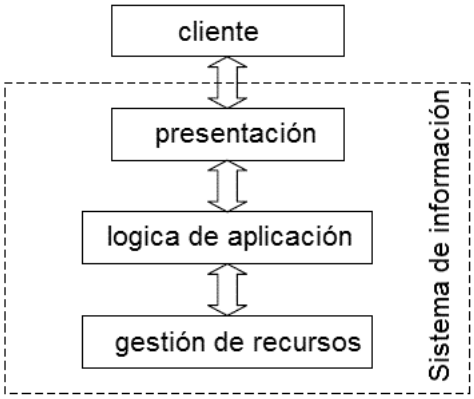

Figura 1.1: Las diferentes capas de un sistema de información

En general, las capas pueden definirse como sigue [4, 28]:

- **Presentación**: todo sistema de información necesita comunicarse con entidades externas que pueden ser usuarios finales u otras computadoras.Esta tarea se realiza por la capa de presentación que puede implementarse de diferentes maneras, por ejemplo, puede implementarse por una interfaz de usuario grafica o por un modulo que presente un conjunto de datos de-acuerdo a una presentación especifica. En el caso de la Web, la capa de presentación puede ser implementada por un Servlet Java que realiza la creación de paginas HTML.
- **Lógica de la aplicación**: en general, los sistemas de información efectúan el tratamiento de los datos antes de entregar los resultados. La capa de la lógica

de aplicación es por lo tanto la responsable del tratamiento de los datos. Estos programas representan frecuentemente los servicios proporcionados por el sistema de información.

- **Administración de recursos**: un sistema de información accede y administra un conjunto de datos. Estos datos puede residir en una base de datos, un sistema de archivos o algún otro tipo de depósito.

El diseño de un sistema de información puede ser de tipo Top-down o Bottom-up. La Figura 1.2 ilustra los 2 casos y pone en evidencia las relaciones entre las etapas y las capas.

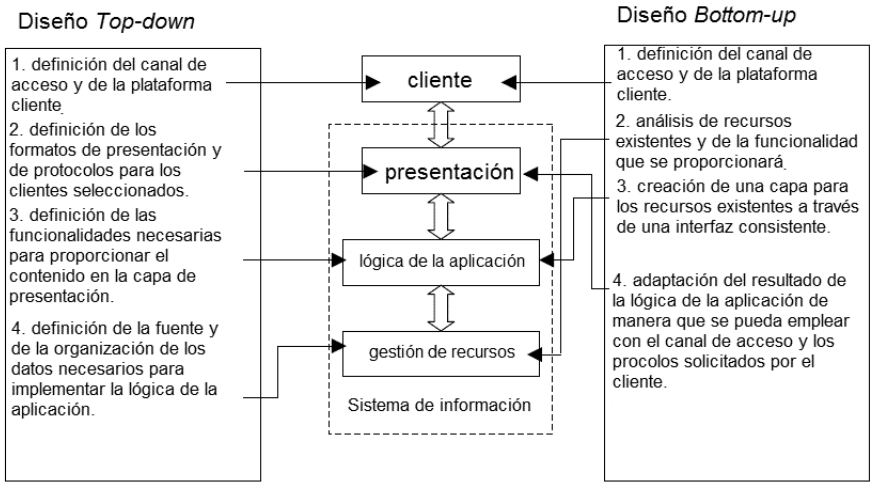

Figura 1.2: El diseño de tipo Top-down y Bottom-up [4].

1.2.2. La arquitectura de un sistema de información

Las tres capas presentadas anteriormente son construcciones conceptuales que separan lógicamente las funcionalidades de un sistema de información. En el plano real, estas capas se concretizan por los tiers. Se pueden distinguir 4 tipos de sistemas de información basados en la noción de tiers: 1-tier, 2-tiers, 3-tiers y 3-tiers cliente ligero.

La Figura 1.3 presenta la arquitectura 1.- tier. Esta arquitectura es la más antigua y se remonta a lo años de gloria de los mainframes. En este caso, la administración de recursos, la lógica de la aplicación y la presentación se realizan por un mainframe. La presentación se prepara en el mainframe y se visualiza en una terminal pasiva (dumb terminal).

La Figura 1.4 ilustra la arquitectura 2-tiers. Esta arquitectura apareció al mismo tiempo que las computadoras personales. De hecho, las PC y las estaciones de trabajo han proporcionado la potencia en el tratamiento necesario para separar la presentación de la lógica de la aplicación y de la administración de recursos.

El desplazamiento de la capa de presentación hacia las PC tiene 2 ventajas importantes. La capa de presentación puede utilizar la potencia del tratamiento disponible en las PC y, por consecuencia, los recursos que esta utiliza estarán disponibles para las capas

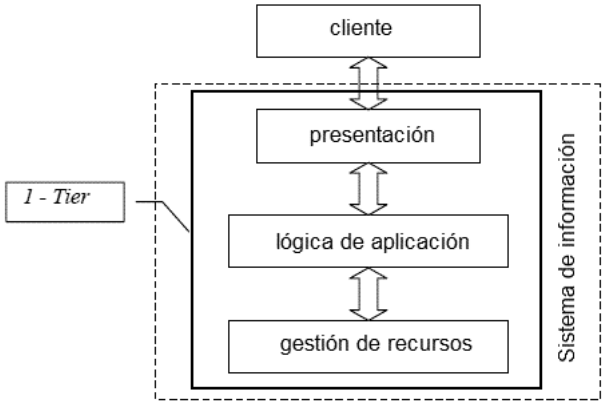

Figura 1.3: La arquitectura 1-tier.

de la lógica de la aplicación y de la administración de recursos. Enseguida, esto facilita la creación de la presentación con diferentes objetivos, sin aumentar la complejidad del sistema.

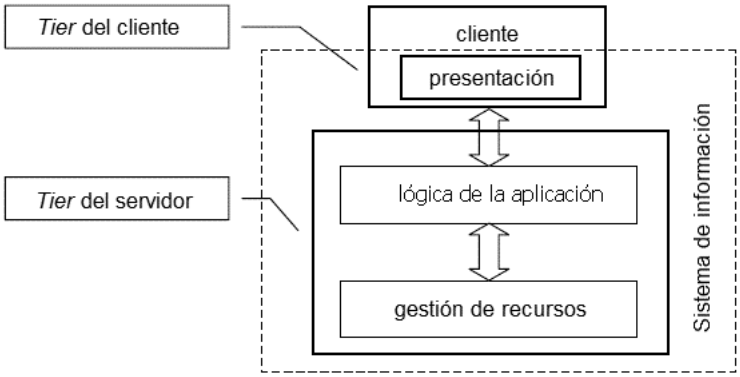

Figura 1.4: La arquitectura 2-tier.

La arquitectura 2-tiers se volvió muy popular en particular para las arquitecturas cliente-servidor. En esta arquitectura, el cliente es responsable de la capa de presentación, y el servidor se ocupa de la lógica de la aplicación y de la administración de recursos.

Los sistemas cliente-servidor han representado una evolución significativa de los sistemas computacionales. Como las PC y las estaciones de trabajo son más potentes, la capa de presentación es cada vez más sofisticada. En esta evolución, la llamada de procedimientos remotos (RPC - Remote Procedure Call) ha sido un paso adelante.

Sin embargo, la arquitectura 2-tiers ha creado islas de información donde un conjunto de clientes pueden comunicarse con su servidor, pero estos no pueden comunicarse con otros servidores. Así, la integración de servidores es actualmente un tema de debates en la computación, y por lo tanto es necesaria una nueva arquitectura que responda a estas nuevas demandas.

La Figura 1.5 presenta esta nueva arquitectura llamada 3-tiers. En esta arquitectura,

la capa de presentación reside en el cliente como en la arquitectura 2-tiers. La lógica de la aplicación reside en el middle tier o se sitúa en el middleware. Un Middleware facilita y administra las interacciones entre aplicaciones que funcionan generalmente en plataformas heterogéneas. Podemos citar como ejemplos de Middleware: CORBA [67], Java RMI [76] y DCOM [16].

La administración de recursos reside en el tier de las bases de datos (back end).

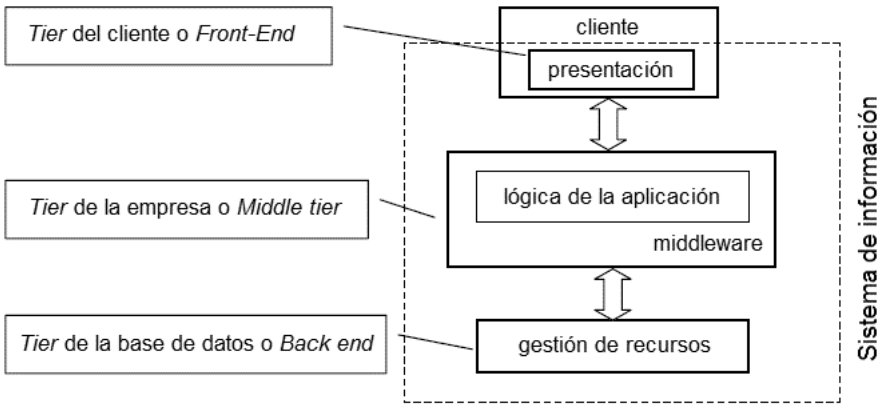

Figura 1.5: La arquitectura 3-tier.

En relación a la arquitectura 2-tiers, la arquitectura 3-tiers presenta ciertas ventajas e inconvenientes. En la arquitectura 2-tiers, las capa de la logia de la aplicación y de la administración de recursos se sitúan en el mismo nivel: la ventaja es que la comunicación entre estas 2 capas es eficaz, la contraparte es que un servidor con mejor desempeño se tiene que utilizar para ejecutar la lógica de la aplicación y la administración de los recursos, lo que implica entonces un costo material más elevado.

Los sistemas 3-tiers han introducido conceptos importantes como las interfaces uniformes para que la lógica de la aplicación pueda acceder a la capa de administración de recursos, como ODBC (Open DataBase Connectivity) y JDBC (Java DataBase Connectivity).

Las arquitecturas 3-tiers se explotan de manera más importante cuando se utilizan para integrar diferentes recursos. En este caso, los middleware modernos proporcionan no solamente un lugar para desarrollar la lógica de la integración que constituye el middle tier si no también las funcionalidades necesarias para proporcionar al middle tier, propiedades adicionales como: la transacción entre los administradores de recursos, el balance de carga, la capacidad de logging, la replicación, la persistencia, la concurrencia, etc.

Una posibilidad de utilizar los sistemas 3-tiers con el internet se muestra en la Figura 1.6. En la literatura, esta variante de la arquitectura 3-tiers se conoce bajo el nombre de 3-tiers cliente ligero, ya que la presentación no se crea en el cliente, sino en un servidor de aplicaciones Web dedicado. Este genera la presentación utilizando la creación dinámica de paginas HTML que enseguida se visualizan mediante un navegador Web. Podemos citar como ejemplos de tecnologías para crear páginas Web dinámicas: Java Servlets, JSP (Java Server Pages), ASP.NET (Active Server Pages dotNET) y PHP (Personal Home Page).

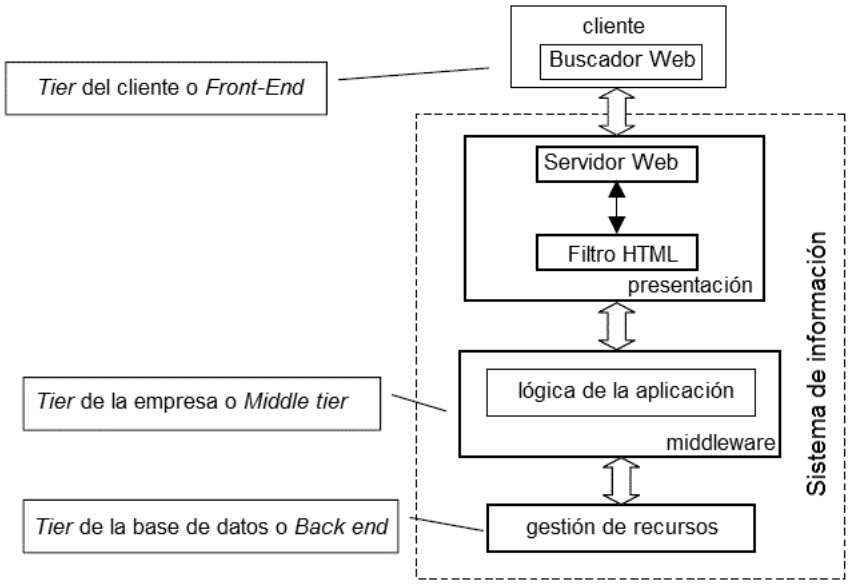

Figura 1.6: La arquitectura 3-tier con cliente ligero.

1.2.3. Comunicación en los sistemas de información

Las interacciones en los sistemas de información distribuidos se implementa por los modelos de comunicación: llamadas síncronas y llamadas asíncronas.

- En el caso de las llamadas síncronas entre los procesos, el que genera la solicitud debe atender la llegada de la respuesta para poder continuar su ejecución.
- En el caso de las llamadas asíncronas, un proceso envía la llamada y continúa su ejecución hasta el momento donde solicita el resultado de la llamada.

La Figura 1.7 ilustra la llamada síncrona y asíncrona.

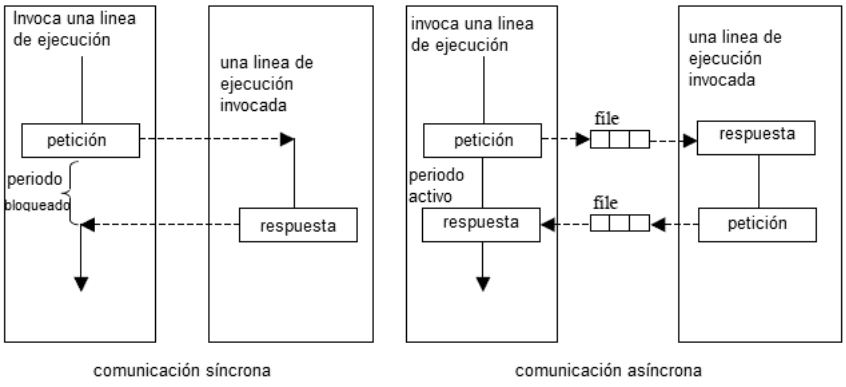

Figura 1.7: Comunicaciones síncronas y asíncronas.

1.3. Middleware

Los middleware se utilizan ampliamente en las arquitecturas de los sistemas de información [20, 90].

El middleware es un software de conexión formado por un grupo de servicios que permiten la ejecución de diversas aplicaciones sobre una o varias maquinas conectadas en red. Esta tecnología proporciona una capa de abstracción que permite la interoperabilidad entre diferentes lenguajes de programación, sistemas operativos y arquitecturas de computadora [101].

Los middleware se pueden clasificar en diferentes categorías:

- **Sistemas basados en RPC (Remote Procedure Call)**: RPC es la forma más básica de middleware. Proporciona la infraestructura necesaria para transformar las llamadas de procedimientos locales en llamada de procedimientos a distancia de manera uniforme y transparente [102]. Actualmente, los sistemas RPC se utilizan como base para casi todos los tipos de middleware, por ejemplo CORBA y los servicios Web.
- **TP monitors (monitores transaccionales)**: son la forma más antigua y más conocida de middleware. Son también los más confiables, mejor probados, y la tecnología más estable en EAI (Enterprice Application Integration) [9].
- **Object brokers**: son similares a los RPC pero se han introducido para tomar en cuenta a los sistemas orientados a objetos. El ejemplo más popular es CORBA del OMG.
- **Object monitors**: son la fusión de los TP monitors y de los Object brokers.
- **Message-oriented**: este tipo de middleware se conoce como MOM (Message Oriented Middleware) basado en los mensajes asíncronos. Este proporciona un mecanismo de acceso transaccional para archivos, persistencia para los archivos, y un número de primitivas para leer y escribir en un archivo local o a distancia [49].
- **WfMS (Workflow Managment System)**: un sistema de administración de Workflow (flujo de trabajo) permite el control del envió de información entre los participantes de un proceso, asi como la definición de la lógica de negocios necesaria para integrar los sistemas heterogéneos y distribuidos [37].
- **Servicios Web**: es el tipo más reciente de middleware, se ha concebido para soportar la interoperabilidad en Internet [107].

  La Figura 1.8 ilustra de manera más detallada a un middleware.

De hecho, los middleware tienen como objetivo principal, permitir la interoperabilidad entre sistemas de información. Una de las primeras tecnologías que considero la interoperabilidad fue EDI (Electronic Data Interchange) [20]. Además, cada middleware se basa en un tipo específico de EDI. Por ejemplo, CORBA está basado en CDR (Common Data Representation) y los servicios Web se basan en XML.

En el contexto del Internet, XML se ha utilizado como un formato de datos para EDI. Además, es bien conocido como el estándar tradicional de EDI para los protocolos de Internet. Por ejemplo OBI (Open Buying on the Internet) utiliza el protocolo HTTP para transferir los mensajes en el Internet cuyo contenido se basa en el estándar X12.

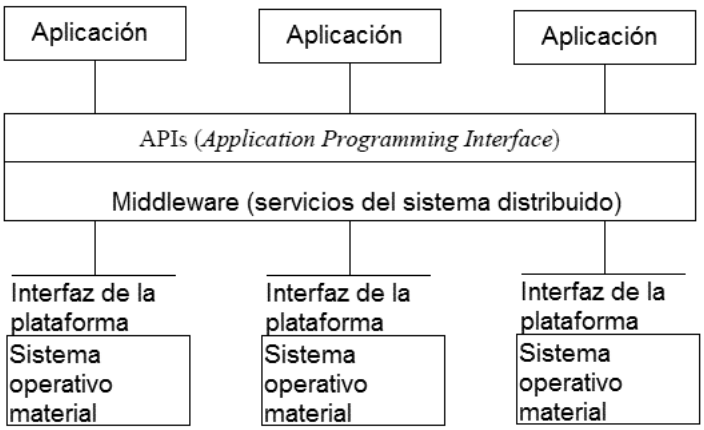

Figura 1.8: Middleware [101].

1.4. Contribución

En este trabajo se investigara sobre MDA y cómo implementar este enfoque para realizar e integrar los servicios Web en el mismo para realizar una red social. Para ello en los capítulos siguientes hablaremos de los servicios Web, el enfoque MDA y cómo aplicarlo en la creación de una red social. A continuación se hace una breve descripción delo que se verá en cada capítulo de esta tesis.

- **Servicios Web**: Este capítulo explicara que es un servicio, como estos se aplican a la Web, los estándares, manera de utilizarlos y trabajos futuros sobre los mismos.
- **Modelos y lenguajes de modelización**: Con el fin de entender el enfoque MDA se describirán los diferentes lenguajes de modelización existentes para el desarrollo de las arquitecturas de software.
- **MDA para los servicios Web**: En este capítulo se hablará cómo aplicar el enfoque MDA para desarrollar sistemas con los servicios Web.
- **Aplicación de MDA en redes sociales**: Se llevará a cabo la descripción de la implementación de la red social AnonymousRemark con el enfoque MDA y los Servicios Web.
- **Aplicación MDA con los servicios Web**: En este capítulo se mostrará la aplicación realizada ya con los diferentes plataformas objetivo (JAVA, C# y PHP).
- **Resultados**: Se mostrará un conjunto de resultados obtenidos al realizar este trabajo de tesis.
- **Conclusiones y trabajo futuro**: Se mostrarán las conclusiones obtenidas sobre la utilización del enfoque MDA con los servicios web y trabajos que se realizarán a futuro en la red social.

Capítulo 2

Servicios Web

En esta parte, describimos de manera más detallada los servicios Web. Destacamos, la definición, los actores y las tecnologías de los servicios Web.

2.1. Servicios Web: Definiciones y actores

En esta sección, presentaremos los servicios Web respondiendo a las siguientes preguntas:

- **¿Qué es un servicio Web?**
- **¿Porqué los servicios Web?**
- **¿Para qué se han concebido y normalizado?**

2.1.1. ¿Qué es un servicio Web?

Para saber que es un servicio Web necesitamos primero definir el término "Servicios Web ". Definiremos de manera separada los términos "Servicio "y "Web ", para después presentar las definiciones más comunes utilizadas para identificar que es un servicio Web.

Un servicio es un recurso abstracto que representa las posibilidades de realizar tareas que aseguran una funcionalidad coherente desde el punto de vista de las entidades proveedor y solicitante. Para ser empleado, un servicio se debe realizar por un agente proveedor concreto. La noción de servicio existe desde la aparición de los servicios Web: de hecho se utiliza desde hace algunos años por la DCE (Distributed Computing Envaironment) de la OSF, por CORBA de la OMG, por Java RMI de SUN y por DCOM de Microsoft. Sin embargo la noción de servicio se ha vuelto de vital importancia con la aparición de las Arquitecturas Orientadas a Servicios (SOA) [107].

La Web o World Wide Web es uno de los servicios más importantes del Internet. La Web se compone de servidores Web conectados al Internet que alojan páginas Web. Estas páginas son documentos multimedia que contienen texto, imágenes, animaciones y videos. El Internet es la red más grande del mundo, que se compone con más de 100 millones de computadoras en más de 100 países y realiza esfuerzos importantes para promover el comercio, actividades académicas y gubernamentales. Las tecnologías que soportan la Web son HTTP, HTML, y más recientemente XML.

El termino servicios Web se utiliza frecuentemente pero no siempre con el mismo significado. Un servicio Web es más frecuentemente visto como una aplicación accesible a otras aplicaciones en la Web. Esta es una definición muy abierta, ya que permite considerar toda cosa que utiliza una URL (Uniform Resource Locaitor) como un servicio Web, por ejemplo los scripts CGI (Common Gateway Inteface).

Una definición más especifica la proporciono el consorcio UDDI, que definió un servicio Web ‟Como una aplicación de negocios modular que forma un todo, con interfaces basadas en estándares y orientada a Internet ″, pero sobre todo abierta.

Algunas otras definiciones más precisas de servicios Web fueron propuestas por el W3C [106]. El W3C define un servicio Web como ‟Una aplicación software identificada por una URI, cuyas interfaces y vínculos están definidos, descritos y descubiertos como objetos XML. Un servicio Web soporta las interacciones directas con otros agentes de software utilizando el paso de mensajes basado en XML y utilizando los protocolos de Internet ″. Esta definición es más precisa que las anteriores, y evidencia la manera de identificar un servicio (Utilizando las URI), los elementos de base para contemplar la posible interoperabilidad (Utilizando una interfaz y los vínculos basados en XML), las interacciones a través de los protocolos basados en XML.

Sin embargo, esta definición evidencia únicamente el estándar XML, pero cualquier aplicación orientada a Internet que utiliza las tecnologías basadas en XML es también un servicio Web.

[209], el W3C define un servicio Web como ‟Un sistema de software concebido para soportar la interacción interoperable de máquina a máquina en la red. Este posee una interfaz descrita en un formato explotable por la maquina, es decir descrita en WSDL. Otros sistemas interactúan con el servicio Web de una manera predefinida por su descripción utilizando los mensajes SOAP, típicamente utilizando HTTP con una serialización XML al mismo tiempo con otros estándares de la Web ″.

Una definición más precisa de los servicios Web la proporciona el diccionario Webopdia (http://www.Webopedia.com). Define un servicio Web como ‟Una manera estandarizada de integración de las aplicaciones basadas en la Web, utilizando los estándares abiertos XML, SOAP, WSDL, UDDI y los protocolos de transporte del Internet. XML se utiliza para representar los datos, SOAP para transportar los datos, WSDL para describir los servicios disponibles, y UDDI para listar los proveedores de servicios y los servicios disponibles. ″.

En nuestro trabajo, adoptamos esta última definición para identificar un servicio Web, ya que no enfatiza sobre las tecnologías y su interdependencia. Estas tecnologías estándares son interdependientes ya que se han concebido de manera independiente. La Figura 2.1 presenta estas tecnologías de los servicios Web evidenciando su interdependencia.

La Figura 2.2 presenta la utilización de los servicios Web o los actores proveedores y los solicitantes.

De acuerdo a la Figura 2.2, los actores que solicitan servicios, los proveedores de servicios y el anuario mantienen una relación por medio de las tecnologías de los servicios Web:

1. El proveedor de servicios publica sus servicios Web en el anuario utilizando UDDI.

2. El solicitante busca un servicio Web con las características X, Y y Z.

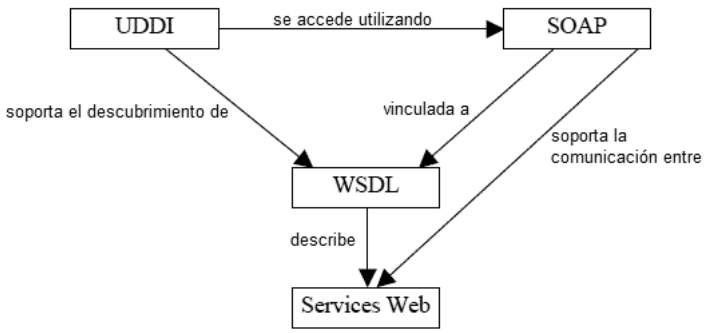

Figura 2.1: Las principales tecnologías de los servicios Web y sus relaciones.

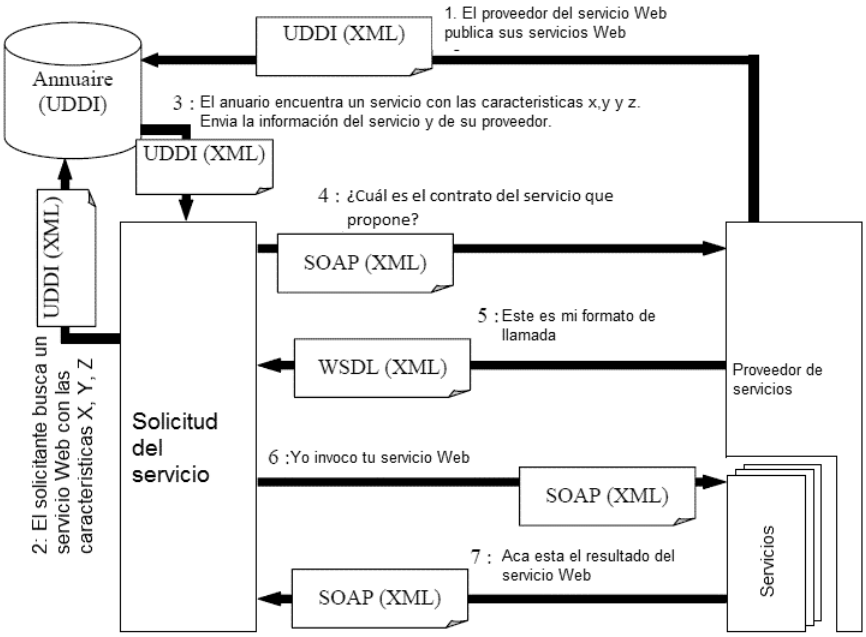

Figura 2.2: Las tecnologías de los servicios Web y los principales actores.

3. El anuario encuentra un servicio con las características X, Y y Z, y envía las informaciones sobre el proveedor de servicio y sobre el servicio.

4. El solicitante del servicio solicita el contrato de servicio Web al proveedor.

5. El proveedor envía el contrato de servicio (WSDL).

6. El solicitante del servicio invoca el servicio Web según su contrato (Utilizando SOAP).

7. El servicio Web regresa el resultado de la llamada (SOAP).

Los estándares SOAP, WSDL y UDDI constituyen la primera generación de los servicios Web. Otros estándares más recientes, como WS-Coordination, WS-Security, WS-

Atomic Transaction y BPEL4WS (Business Process Execution Language for Web Services) constituyen la segunda generación de servicios Web.

2.1.2. ¿Por qué utilizar los servicios Web?

Los servicios Web se han concebido en primer lugar para responder a los problemas de interoperabilidad en la Internet, que los Middleware convencionales como CORBA y DCOM han sido incapaces de resolver de manera satisfactoria.

Estos Middleware convencionales se desarrollaron en una época donde los sistemas distribuidos estaban limitados a una red local o privada de una empresa, de un gobierno o de una universidad. Se concibieron igualmente para responder a necesidades específicas y en un contexto bien limitado. La idea de crear un Middleware bastante robusto para adaptarse a cualquier situación o problema se considero en la creación de CORBA. Por ejemplo, cuando el Internet estuvo disponible para la utilización civil, CORBA se adapto rápidamente para trabajar sobre TCP/IP y el tunneling HTTP. Técnicamente esto era posible gracias al protocolo GIOP/IIOP. Sin embargo, se trata de una adaptación y no de una solución hecha para considerar el contexto de la Internet. Además, CORBA no tenía un consenso universal. Microsoft desarrollo y estimulo la utilización de DCOM. Los Gateway entre CORBA y DCOM se propusieron, pero introdujeron más problemas que soluciones. Así, durante los años 90 se paso del periodo Middleware al periodo Middlewar donde una fuerte competición se estableció entre CORBA y DCOM sin tener ganador ni perdedor.

Al final de los años 90, los investigadores en computación percibieron que ni CORBA ni DCOM eran convenientes para soportar los sistemas distribuidos en la Internet. Se propusieron muchas iniciativas para crear una nueva solución más adaptada al Internet, de hecho los actores del mundo computacional deseaban contar con un soporte de la interoperabilidad de manera uniforme y extensible.

También consideraron que un nuevo formalismo para representar la información no era suficiente para desarrollar las aplicaciones en el contexto de la Internet: era importante por lo tanto cambiar de paradigma. Así, el paradigma de servicios parecía ventajoso para responder a las necesidades de la Internet. Era igualmente conveniente concebir soluciones que pudieran ser integradas a los estándares de Internet, como HTTP y HTML, y/o utilizados entre ellos.

Entre estas iniciativas, podemos citar: ActiveXML, XML-RPC, SOAP, WSDL y UDDI. Todas estas iniciativas están basadas en XML como norma para representar la información. Además, la noción de servicio está presente en las soluciones propuestas. Los servicios Web son el resultado de la convergencia de estas iniciativas.

Al inicio, Se crearon muchos mitos alrededor de los servicios Web, solo había que considerar que los servicios Web eran la solución más adaptada para cualquier contexto. En el contexto de la Internet, los servicios Web presentan varias ventajas en relación a los otros Middleware. Para elaborar un comparativo más preciso, comparamos la pila de estándares de Java RMI, CORBA y XML-RPC con la pila de los servicios Web como se muestra en la Cuadro 2.1.

Java RMI es un Middleware específico en la plataforma Java. Las interfaces están definidas en Java. El anuario es muy limitado y se resume a un servicio de nombres. El

Función	Java RMI	CORBA	XML-RPC	Servicios Web
Descripción	Interfaces en Java	IDL-CORBA	-	WSDL
Anuario	+/- rmiregistry	CORBA Services	-	UDDI
Protocolo de comunicación	JRMP	GIOP	Protocole XML-RPC	SOAP Message
Representación de datos	Serializar	CDR	XML	XML
Protocolo de transporte	RMI	HTTP	HTTP	HTTP

Cuadro 2.1: Un comparativo entre los servicios Web y los otros Middleware.

protocolo de comunicación JRMP (Java Remote Method Protocol), La representación de información y el protocolo de transferencia son específicos a Java.

De hecho, Java RMI es una adaptación de RPC al mundo java. En [44], el autor presenta un estudio detallado del desempeño entre Java RMI y los servicios Web. Este estudio demuestra que la utilización de Java RMI es 8.5 veces más poderoso que la utilización de los servicios Web sobre una red local. Sin embargo, el autor señala que sobre la Internet, la utilización de Java RMI necesita frecuentemente una técnica de tunneling como HTTP-to-port, HTTP-to-CGI y HTTP-to-Servlet. El esfuerzo para desplegar y configurar Java RMI y los componentes del tunneling es más significativo que en el caso de los servicios Web. Además, Java RMI y las técnicas de tunneling presentaron un desempeño inferior a los servicios Web. La utilización de Java RMI y HTTP-to-port fue 4 veces más potente que los servicios Web. Y la utilización de Java RMI y HTTP-to-Servlet fue 4 veces menos potente que los servicios Web.

Más adelante el mismo autor demostró que en ciertos contextos y con la utilización de técnicas de optimización, los servicios Web pueden ser más poderosos que Java RMI (sin tunneling).

CORBA puede soportar varios tipos de aplicaciones distribuidas, pero no está eficazmente adaptado al Internet. Define de hecho, la noción de servicio, así como una descripción de una interfaz independiente de la plataforma, es decir IDL-CORBA. El protocolo GIOP es un protocolo concebido para adaptarse a otros protocolos específicos en el dominio. CDR es una representación binaria de información; esto permite la interoperabilidad, pero sin ser extensible. IIOP es el protocolo de transporte definido en GIOP para considerar el TCP/IP, que más adelante se estandarizo y se utiliza como tal. Por un lado, a través de los servicios CORBA, se proporciona un anuario para los servicios. Sin embargo, el naming y el trading service no se han considerado fuertemente. Por otro lado, este anuario es muy limitado ya que no considera la semántica de las propiedades y de la descripción del servicio.

En [8] los autores presentan un caso de estudio que demuestra el interés y la necesidad de la cohabitación de los servicios Web y CORBA. Además, ellos hacen énfasis en que los servicios Web representan un Middleware para un Middleware, es decir un Middleware universal que permite la integración de diferentes Middleware con el fin de asegurar la interoperabilidad. Así, la coexistencia de los servicios Web y CORBA puede ser justifi-

cada en ciertas situaciones. En otros contextos una de estas tecnologías parece ser mas adaptada suficiente para soportar el desarrollo de los servicios.

Además, los servicios Web presentan ventajas significativas en relación a CORBA. El WSDL contiene 2 partes una abstracta y otra concreta. La parte abstracta contiene el contrato del servicio, y la parte concreta contiene los vínculos (bindings) y la localización del servicio. IDL-CORBA no proporciona más que la parte abstracta, es decir el contrato del servicio.

Los tipos de datos codificados en IDL-CORBA están predefinidos, ya que existe la posibilidad de crear tipos complejos (por ejemplo, estructuras, uniones) y tipos dinámicos con el constructor ‟any ″. WSDL permite la descripción de tipos que no están predefinidos en su especificación, esto gracias a la extensibilidad de XML y los esquemas XML.

SOAP es un protocolo de comunicación más rico que IIOP. Como SOAP está basado en XML, permite la creación de tipos de datos más flexibles y no predefinidos. Además, SOAP puede extenderse para responder a nuevas exigencias (por ejemplo, seguridad, transacción) sin modificar la normalización de SOAP. De hecho, SOAP es más simple porque se adapta a las necesidades de un contexto.

Así como las otras tecnologías de los servicios Web, UDDI también se puede extender para responder a nuevas exigencias. En relación a CORBA, UDDI proporciona además la posibilidad de clasificar un servicio por sus características técnicas.

XML es un mecanismo más rico que CDR para representar los datos. XML se concibió para ser extensible y comprensible para los humanos y para las maquinas. En contraparte, CDR es una representación binaria que no es extensible ni comprensible para los humanos aunque sí para las maquinas. Es necesario señalar que la utilización de XML permite una mayor interoperabilidad y extensibilidad que CDR. En contraparte, la utilización de XML exige un tratamiento más elaborado que CDR, lo que ocasiona un impacto sensible en el desempeño de los servicios Web.

XML-RPC es solamente un RPC basado en XML y HTTP. Además, es muy limitado para soportar aplicaciones complejas en la Internet.

Los servicios Web son la más reciente tecnología utilizada para soportar el desarrollo de sistemas de información distribuidos, en particular las aplicaciones B2B sobre la Internet.

Los servicios Web están basados en tecnologías estandarizadas, lo que reduce la heterogeneidad y proporciona un soporte para la integración de aplicaciones. Además, los servicios Web son el soporte para nuevos paradigmas, como el tratamiento y la arquitectura orientada a servicios. De hecho, el tratamiento orientado a servicios existe desde hace tiempo, pero es la primera vez que alcanza un nivel de importancia significativo.

Para resumir, los servicios Web tienen varias ventajas como la utilización de estándares universales, la independencia de la plataforma, un ambiente universal para los sistemas de información distribuidos, la utilización de varios protocolos de transferencia (por ejemplo HTTP, SMTP y FTP), el cifrado de los mensajes en XML, un comportamiento compatible con los fireware, una facilidad de adaptación a los sistemas heredados, y la localización por URI.

2.1.3. ¿Quien concibió y normalizo los servicios Web?

Los servicios Web fueron concebidos por diferentes instituciones, pero no podemos restringir a los diseñadores citando solamente los organismos que los estandarizan y las

empresas que los promueven.

Los organismos más importantes que efectúan la estandarización de los servicios Web son el W3C (World Wide Web Consortium), la OASIS (Organization for the Advancement Structured Information Standards), y el WS-I (Web Service Interoperability Organization). El W3C es el responsable del desarrollo de varios estándares, entre ellos XML, WSDL, SOAP y WSA. La OASIS es la responsable del desarrollo de estándares como UDDI, WS-Security y más recientemente WS-BEPEL (Web Services Business Process Execution Languages). La organización WS-I es un tipo de organización diferente, se encarga de las herramientas y directivas para ayudar a los desarrolladores a crear software en conformidad con los estándares de los servicios Web. La WS-I libera diferentes productos, entre ellos perfiles, software de demostración y herramientas de prueba. La WSI Basic Profile representa un avance en los servicios Web, ya que proporciona una guía esencial para asegurar la interoperabilidad entre las implementaciones de los servicios Web. Esta especificación es única y demuestra la importancia de desarrollar productos verdaderaente interoperables, lo que no había sido posible en la creación de CORBA.

Entre las empresas que participan en la creación de los estándares de servicios Web, podemos citar IBM, Microsoft, Ariba, Developmentor, BEA Systems, HP, Sun Microsystems, SAP, Canon, Xerox, Oracle e Intel.

Uno de los factores que ha contribuido al éxito de los servicios Web ha sido el entendimiento entre los principales actores. De hecho, es la primera vez que las grandes empresas de computación como IBM, Microsoft y Sun(actualmente ORACLE) están de acuerdo para utilizar y sostener un estándar en común.

2.2. XML (Extensible Markup Language)

XML (Extensible Markup Language) es un metalenguaje de representación de datos, que compone un conjunto de reglas para crear y validar los datos. XML es texto plano y flexible que hereda de SGML (Standard Generalized Markup Language) (ISO 8879) [104].

El documento XML se basa en marcas o etiquetas y atributos los cuales representan información. En el cuadro 2.2 se muestra un ejemplo de un documento XML.

```
<?xml version='1.0' encoding='ISO-8859-1'>
<Materias>
<Materia id='10' nombre='Disciplina Computacional'>
<Maestro¿Juan Antonio </Maestro>
<Idioma>Español </Idioma>
</Materia>
<Materia id='10' nombre='Matemáticas Elementales'>
<Maestro>Carlos Gutiérrez </Maestro>
<Idioma>Ingles </Idioma>
</Materia>
</Materias>
```

Cuadro 2.2: Ejemplo de documento XML.

El poder de XML radica no solo en que es rápido y sencillo en su implementación, también cuenta con estándares que nos permite especificar los documentos, transformaciones y busquedas. En la Figura 2.3 veremos estos estándares.

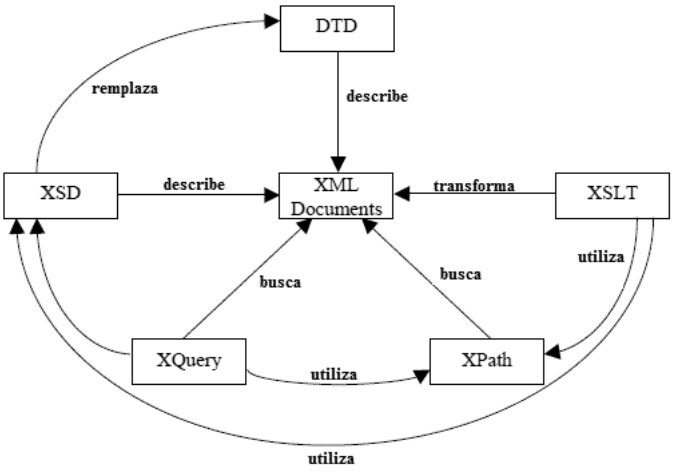

Figura 2.3: Especificación XML.

XML y sus estándares son utilizados en los servicios Web ya que especifican con WSDL, UDDI y SOAP. Lo que se utiliza para especificar es:

- **XSD (XML Schema)**: es un lenguaje que sirve para describir formalmente un vocabulario [112].
- **XSLT (Extensible Stylesheet Language Transformations)**: es útil para transformar un documento XML basado en un esquema a un documento en diferentes formatos [207].
- **XPath (XML Path Language)**: forma un conjunto de expresiones útiles para crear localización de datos [111].

2.3. WSDL (Web Services Description Language)

WSDL proporciona un modelo y un formato XML para describir servicios Web. WSDL separa la descripción de la funcionalidad de un servicio Web, es decir separa las funcionalidades en cómo y en donde este se ofrece [109]. Un documento WSDL se compone principalmente de las definiciones. Cada definición consta de interfaces, mensajes, links (enlaces) y servicios. En la Figura 2.3 se muestra la estructura de los WSDL.

2.4. UDDI (Universal, Description, Discovery and Integration)

La especificación UDDI organiza la información sobre los servicios Web en tres categorías:

```
<?xml version='1.0' encoding='ISO-8859-1'? >
<definitions >
<types >
...
</types >
<message name= >
...
</message >
<interface name= >
...
</interface >
<binding name= >
...
</binding >
<service name= >
...
</service >
</definitions >
```

Cuadro 2.3: Muestra la estructura básica de un documento WSDL.

- **Las páginas blancas**: direcciones, contactos e identificadores conocidos de la empresa (en sentido amplio: empresa comercial, administración, agencia gubernamental, asociación, organización sin ánimo de lucro);
- **Las páginas amarillas**: categorías industriales basadas en taxonomías estándares (productos, empresas, geográficas);
- **Las páginas verdes**: referencias técnicas sobre los servicios ofrecidos por la empresa (referencias a especificaciones de servicios Web, referencias hacia distintos recursos).

El modelo de información UDDI correspondiente, especificado en forma de esquema XML,define cinco tipos de estructuras de datos. La Figura 2.4 describe las relaciones entre estas estructuras:

Los cinco tipos estructurados, definidos por la especificación UDDI 2.0 son los siguientes:

- **El tipo entidad negocio (Business Entity)**: equivalente de las páginas blancas (existe desde UDDI 1.0);
- **El tipo servicio negocio (Business Service)**: equivalente de las páginas amarillas (existe desde UDDI 1.0);
- **El tipo modelo de conexión (Binding Template)**: equivalente de las páginas verdes (existe desde UDDI 1.0);
- **El tipo servicio modelo (Model)**: descripciones de especificaciones de servicios o de taxonomías referenciadas por los modelos de conexión (existe desde UDDI 1.0);
- **El tipo aserción de administrador (Publisher Assertion)**: descripciones de relaciones entre entidades de negocio, afirmadas por el administrador (<<editor >>)

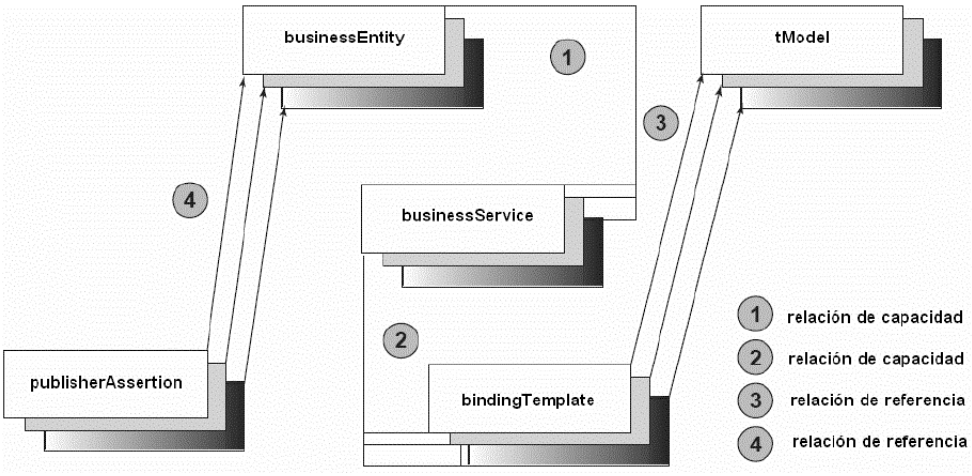

Figura 2.4: Relaciones entre las principales estructuras de datos UDDI.

de una de las entidades de negocio en cuestión (nuevo tipo introducido por UDDI 2.0).

2.5. SOAP (Simple Object Access Protocol)

SOAP proporciona una definición de información representada en XML que puede ser utilizada para el intercambio de información estructurada y escrito entre los participantes en un entorno distribuido descentralizada [105]. SOAP es un protocolo independiente de cualquier plataforma y lenguaje de programación. Un mensaje SOAP presenta una estructura normalizada (Figura 2.5). Se constituye siempre de un elemento «documento»(raíz), es decir el envelope (SOAP-ENV:Envelope), que contiene un elemento encabezado (SOAP-ENV:Header) opcional y un elemento cuerpo (SOAP-ENV:Body) obligatorio, seguidos de posibles elementos aplicativos específicos. La Figura 2.5 muestra esta estructura.

2.6. SOA (Service Oriented Architecture)

SOA es un conjunto de componentes que pueden ser llamados, sus descripciones de interfaces pueden ser editadas y descubiertas [110].

En nuestros días, los servicios Web proporcionan las tecnologías más adaptadas para ser posible la creación de la arquitectura orientada a servicios. El desarrollo de las aplicaciones utilizando SOA exige la adopción de un proyecto orientado a servicios, lo que es diferente de un proyecto orientado a componentes. Un proyecto orientado a servicios se concentra en las demandas determinadas en el nivel de la estrategia y del proceso de negocios, entonces los proyectos orientados a componentes se concentran en los componentes del programa utilizado para suministrar los servicios. SOA es también una arquitectura de sistemas distribuidos basada en los conceptos de servicio y caracterizada por las siguientes propiedades [107]:

- **Vista lógica**: un servicio es una vista lógica de un sistema.

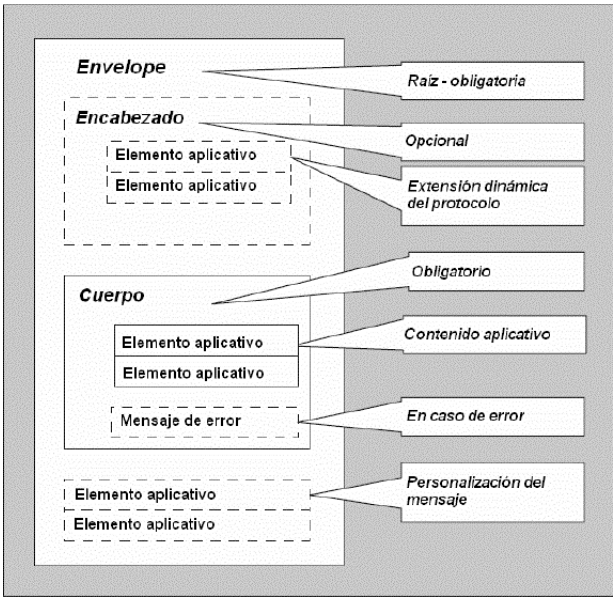

Figura 2.5: La estructura del mensaje SOAP

- **Orientado a mensajes**: la comunicación entre un agente proveedor y un agente solicitante está definida en términos de intercambio de mensajes.
- **Orientado a la descripción**: un servicio se describe por los metadatos.
- **Granularidad**: los servicios se comunican utilizando un número reducido de mensajes que son generalmente grandes y complejos.
- **Orientado a red**: los servicios tienden a ser utilizados en una red. Sin embargo, esto no es una exigencia absoluta.
- **Plataforma neutra**: los servicios se comunican utilizando mensajes cifrados en una representación independiente de la plataforma. XML se utiliza como una representación de datos universal y también independiente de la plataforma, es extensible y representa mejor la información.

La Figura 2.6 ilustra los elementos fundamentales de esta arquitectura. Un agente proveedor contiene los servicios. Estos servicios se describen a través de una representación que utiliza los metadatos, es decir la descripción del servicio. Enseguida, el agente proveedor registra la información de sus servicios en el anuario. Un agente solicitante busca en el anuario los servicios según ciertos criterios específicos. El anuario regresa al solicitante la información de un servicio solicitado. El agente solicitante recupera los metadatos de este servicio y los utiliza para intercambiar mensajes con el servicio.

Las tecnologías de los servicios Web pueden utilizarse para implementar una SOA, pero se deben realizar siguiendo las propiedades antes descritas, es decir vista lógica, orientado a mensajes, etc. En este caso, UDDI se utiliza para publicar o buscar los servicios, WSDL para describir un servicio, SOAP es el protocolo de comunicación.

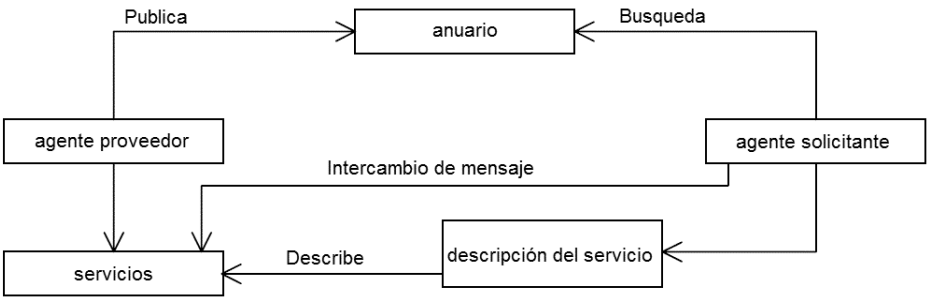

Figura 2.6: La arquitectura orientada a servicios (SOA) (Fragmentada).

De hecho, el conjunto XML, SOAP, WSDL y UDDI es el más adaptado para utilizarse con SOA. Sin embargo no hay que confundir los servicios Web con SOA. Los servicios Web proporcionan el soporte para la descripción, la publicación/búsqueda y la infraestructura de comunicación para los servicios, mientras que SOA describe como se puede construir un sistema compuesto de servicios.

2.6.1. Composición de los servicios Web.

La composición de servicios Web es un proceso mediante el cual podemos crear un servicio Web a partir de otros servicios. Este nuevo servicio se denomina servicio compuesto. En otras palabras, un servicio compuesto es un conjunto unificado de otros servicios. En este caso, los servicios que se han utilizado para crear el servicio compuesto están ocultos.

La creación de una aplicación distribuida compleja puede obtenerse a partir de la composición de servicios Web. Sin embargo, la creación de un servicio a partir de otros servicios está lejos de ser una tarea trivial. Para ayudar a los desarrolladores a crear servicios compuestos, el Middleware de composición de servicios Web debe proporcionar una abstracción y una infraestructura que faciliten la definición y ejecución de un servicio compuesto.

La composición de servicios Web tiene algunas similitudes con la tecnología de Workflow. Los dos tienen por objetivo especificar el proceso de negocios mediante la composición de entidades autónomas y de fuerte granularidad. Su diferencia reside en la naturaleza de la entidad. En el caso del Workflow, las entidades son aplicaciones convencionales mientras que en los servicios Web, las entidades son servicios.

Todos los trabajos alrededor de la composición de servicios Web [6, 52, 97] se han desarrollado utilizando los conceptos básicos desarrollados a través de los años en el contexto de los sistemas de gestión de Workflow. Por un lado, estos últimos han tenido un éxito limitado debido a que los sistemas de soporte de la composición eran complejos, difíciles de desplegar y mantener, compuestos de diferentes aplicaciones, a menudo demandan un esfuerzo significativo de desarrollo, sobre todo cuando la lógica de integración implicaba sistemas heterogéneos y distribuidos. Por otro lado, la composición de servicios Web es diferente en el hecho de que estos servicios parecen estar formados de componentes más adecuados. Es decir, los servicios Web tienen interfaces bien definidas y sus comportamientos están "Especificados "como parte de la información proporcionada por el anuario de servicios. Además, se basan en estándares aceptados como universales. Esto nos lleva a pensar que los servicios Web podrían adaptarse para la composición y

parecería que la composición de servicios tendría una posibilidad mayor de éxito que los sistemas de gestión de Workflow.

Un modelo de composición de servicios puede ser relativamente complejo. En general, las diferentes dimensiones de un modelo de composición de servicios Web pueden ser agrupados como sigue [4]:

- **Modelo de componente**: define la naturaleza de los elementos que serán utilizados en la composición.

- **Modelo de la orquestación**: define la abstracción y el lenguaje utilizado para definir el orden en el cual los servicios serán invocados. Entre las posibilidades de los modelos de la orquestación, podemos citar los diagramas de actividad, las redes de petri, los diagramas de estado, y las jerarquías de actividades.

- **Modelo de datos y de acceso de datos**: define como los datos se especifican y como estos se intercambian entre los componentes.

- **Modelo de selección de servicio**: define si los servicios están unidos estática o dinámicamente. En el caso de la composición estática, los servicios se seleccionan durante el diseño. En el caso de la composición dinámica, los servicios se seleccionan y componen en tiempo de ejecución.

- **Transacciones**: definen que semánticas de transacción pueden asociarse a la composición, y como se efectúa esta asociación.

- **Manipulación de secciones**: define como las situaciones excepcionales que producen la ejecución de un servicio compuesto podrían ser administradas, Sin que esto conduzca a la interrupción del servicio compuesto.

Con el fin de soportar la composición de servicios se han propuesto diferentes lenguajes de composición de servicios como XLANG [97] y WSFL (Web Services Flow Language) [52]. Estos lenguajes, aunque no se han utilizado ampliamente, representan un avance en la especificación de los procesos de negocio para la composición de servicios.

La composición de servicios Web no está lo suficientemente desarrollada, pero existen algunas propuestas importantes, como la convergencia concurrente de estándares como WSFL y XLANG en BPEL4WS.

BPEL4WS define un modelo y una gramática para describir el comportamiento de un proceso de negocios como la composición de un servicio Web. Este contiene varias características de XLANG y de WSFL, y está definido como un lenguaje estructurado en bloques y grafos directos. Además se basa en XML y extiende de WSDL. BPEL4WS contiene elementos que permiten la creación de procesos de negocios abstractos y ejecutables.

Recientemente, muchos investigadores han mostrado el interés de utilizar la Web semántica y las ontologías para soportar la composición de servicios Web, principalmente en el caso de la composición dinámica de servicios [63, 80, 95]. Sin embargo, la composición dinámica de servicios Web esta aun en desarrollo y existen aun varios problemas abiertos.

2.6.2. Los mitos alrededor de los servicios Web.

A través de los años se han detectado diferentes mitos alrededor de los servicios Web. Entre estos mitos tenemos [5]:

- **Servicios Web y los estándares**: a priori, los estándares de servicios Web deberán ser aceptados como universales, pero estos no remplazaran completamente los estándares convencionales. Sobre todo en el mundo de las aplicaciones B2B, donde las aplicaciones se construyen utilizando varias tecnologías. Desde un punto de vista evolutivo, los servicios Web son una capa más suplementaria para permitir a las aplicaciones inter-operar en la Internet.

- **Servicios Web y las aplicaciones convencionales**: hay una tendencia a considerar que todas las aplicaciones estarán construidas con la tecnología de los servicios Web, como una reservación de vuelos por ejemplo, aplicaciones que realizan el cargado automático de las actualizaciones y las aplicaciones que envían los reportes de sus estados de ejecución se pueden prever varios escenarios en la utilización de los servicios Web, entre estos escenarios, la utilización de tecnologías convencionales puede ser más adaptada que la utilización misma de los servicios Web.

- **Conectividad directa entre empresas**: inicialmente, los servicios Web se concibieron utilizando el mecanismo de RPC. En este caso, Estos no presentan ventajas significativas en relación a los otros Middleware, el único punto importante sería la interoperabilidad. Los servicios Web están por lo tanto fuertemente acoplados. Esto es difícil de aceptar para las empresas ya que los componentes pueden provenir de diferentes empresas. Es por esto que la mayoría de las interacciones B2B se efectúan en modo asíncrono o batch. Contrariamente a las invocaciones directas, las consultas y respuestas se envían en lotes y se conducen a través de hilos. Así los elementos de la interacción (cliente, servidor) se desacoplan en la medida de lo posible. Utilizando este modo de comunicación, los elementos pueden desarrollarse, mantenerse y evolucionarse de forma independiente uno de otro. En este escenario, SOAP utilizado conjuntamente con RPC en el modo asíncrono es esencial.

- **UDDI y el ligado dinámico (Dynamic Binding)**: un registro UDDI es conceptualmente un servidor de nombres y de direcciones. Se ha concebido como un anuario para los servicios Web. La información almacenada por estos registros está destinada, a una interpretación humana, y no a una interpretación por parte de las computadoras. De este hecho, el ligado dinámico no es posible, ya que las computadoras no sabrían automáticamente descubrir un servicio y construir la llamada en tiempo de ejecución. En este caso, los servicios Web conservan el mismo problema que sus predecesores. La semántica de los servicios no se ha considerado por UDDI. Además, una operación y sus parámetros pueden tener significados ambiguos y hacer imposibles el ligado dinámico. En este contexto, el OWL-S y otras iniciativas intentan resolver los problemas ligados a la semántica. Por otro lado, los factores jurídicos y culturales hasta nuestros días no se han discutido; como la responsabilidad de los servicios propuestos por un proveedor, el contrato, el pago, etc. A priori, el ligado dinámico podría preverse en el interior de una empresa. Sin embargo, el ligado dinámico entre empresas es aun un tema de investigación y de discusiones.

- **Todos los datos serán codificados en XML**: en ciertos casos, el cifrado y la transmisión de datos en XML no es deseable. Como en el caso de la transmisión de imágenes, música, etc. El cifrado binario puede resultar deseable por razones de desempeño. Así, SOAP y otras soluciones pueden ser esenciales para aplicaciones que requieren de un alto desempeño.

2.6.3. El futuro de los servicios Web

Inicialmente, los servicios Web eran considerados como simples y muy poderosos. Sin embargo, están compuestos de un conjunto muy limitado de tecnologías, XML, WSDL, UDDI y SOAP. En la actualidad se han añadido muchas especificaciones, tales como WSA, WS-Security, SAML, WS-Federation, WS-CDL, BPEL4WS y WS-I Basic Profile. Estas especificaciones colocan a los servicios Web como una tecnología muy completa, pero igualmente muy compleja.

Recientemente, el número de proposiciones de estándares para los servicios Web ha aumentado significativamente, lo que demuestra su interés. Sin embargo el número de proposiciones de estándares concurrentes es también significativo. En las empresas, la utilización de 2 o más estándares para resolver un mismo problema hace que la implementación de los servicios Web sea complicada. En el futuro, estas proposiciones concurrentes deberían converger hacia una solución con el fin de que las empresas puedan desarrollar aplicaciones basadas en un estándar universal. El riesgo para la tercera generación de servicios Web es el establecimiento poco probable de un conflicto entre los estándares concurrentes, similar al sucedido con los Middleware CORBA y DCOM en los años 90.

En el momento de la aparición de los servicios Web, había una duda importante sobre el futuro de los servicios Web. De hecho, fue necesario igualmente considerar la semántica de los servicios con el fin de realizar la composición dinámica. Lo que complico la evolución de UDDI y WSDL, Así como la evolución de los estándares en el afán de considerar la semántica.

La arquitectura de los servicios Web (WSA-Web Service Architecture) comenzó a cambiar la manera de crear aplicaciones que se desarrollaban de acuerdo a la especificación de SOA [107].

La seguridad también debe desarrollarse con mayor énfasis en el estudio de los servicios Web para asegurar la creación de servicios más seguros que los que existen actualmente [65].

Los servicios Web podrían ser una referencia para asegurar la integración de las aplicaciones de las empresas (EAI- Enterprise Application Integration). Igual en aplicaciones que exigen un buen desempeño en el tratamiento de la información, los servicios Web son cada vez más utilizados. Por ejemplo, la investigación científica basada en el cómputo en la grid, lo que significa que los servicios Web se utilizaran cada vez más para soportar la interoperabilidad entre las aplicaciones.

Desde el punto de vista económico, la IDC (International Data Corporation) ha constatado que la inversión en servicios Web en 2003 fue de aproximadamente 1.1 miles de millones de dólares y que hasta 2008 dicha inversión supero los 11 miles de millones de dólares en proyectos de software basados en servicios Web. Otro estudio importante muestra que la participación de 110 empresas, 54 % de ellas trabaja o están en vías de implementar servicios Web en sus aplicaciones.

En la actualidad, pensamos que los servicios Web son la solución del futuro para implementar los sistemas distribuidos en la Internet. Sin embargo, la historia ha demostrado que un solo Middleware no es capaz de soportar todos los tipos de aplicaciones. En consecuencia, en un futuro más lejano, aparecerán nuevas demandas que sobrepasaran las capacidades de los servicios Web y por lo tanto deberá aparecer alguna otra propuesta.

Capítulo 3

Modelos y lenguajes de modelización

Los lenguajes de modelización son uno de los ejes principales del desarrollo orientado a modelos, ya que permiten dar un nivel de abstracción sobre comportamientos o propiedades de los sistemas. A continuación mencionaremos los principales lenguajes de modelización.

3.1. UML 2.0

UML se define como un lenguaje de modelización gráfico y textual destinado a comprender y describir requisitos, especificar y documentar sistemas, resumir arquitecturas computacionales, concebir soluciones y comunicar puntos de vista (opiniones).

No se trata de una simple notación gráfica, ya que los conceptos transmitidos por un diagrama tienen una semántica precisa y son portadores de sentido de la misma forma que las palabras de un lenguaje. UML unifica a la vez las notaciones y los conceptos orientados a objetos.

UML 2 se basa en trece tipos de diagramas, cada uno de ellos está dedicado a la representación de los conceptos particulares de un sistema informático. Estos tipos de diagramas se distribuyen en dos grandes grupos:

- **Seis diagramas estructurales**:
  - **Diagrama de clases**: muestra los ladrillos de la base estática: clases, asociaciones, interfaces, atributos, operaciones, generalizaciones, etc.
  - **Diagrama de objetos**: muestra las instancias de los elementos estructurales y sus vínculos en la ejecución.
  - **Diagrama de paquetes**: muestra la organización lógica del modelo y las relaciones entre paquetes.
  - **Diagrama de estructura compuesta**: muestra la organización interna de un elemento estático complejo.
  - **Diagrama de componentes**: muestra estructuras complejas, con sus interfaces proporcionadas y requeridas.

- **Diagrama de despliegue**: muestra el despliegue físico de los "artefactos"sobre los recursos materiales.

▪ **Siete diagramas comportamentales:**

- **Diagrama de caso de uso**: muestra las interacciones funcionales entre los actores y el sistema en estudio.
- **Diagrama de vista del conjunto de interacciones**: fusiona los diagramas de actividad y de secuencia para combinar fragmentos de interacción con decisiones y flujos.
- **Diagrama de secuencia**: muestra la secuencia vertical de los mensajes pasados entre los objetos en el sentido de una interacción.
- **Diagrama de comunicación**: muestra la comunicación entre objetos en el plano, en el sentido de una interacción.
- **Diagrama de tiempo**: fusiona los diagramas de estados y de secuencia para mostrar la evolución del estado de un objeto en el transcurso del tiempo.
- **Diagrama de actividad**: muestra la secuencia de las acciones y decisiones en una actividad.
- **Diagrama de Estados**: muestra los distintos estados y transiciones posibles de los objetos de una clase.

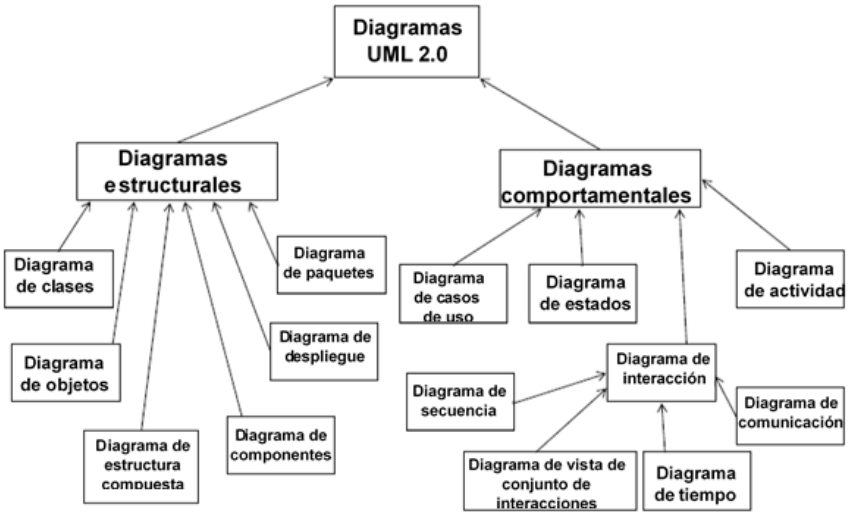

Figura 3.1: Tipos de diagramas UML2.0.

3.2. EDOC (Enterprice Distributed Object Computing)

EDOC es una especificación para el desarrollo de componentes basados sobre los sistemas EDOC a través de un framework de modelización [70]. La especificación EDOC es independiente de una plataforma tecnológica, lo que permite realizar su implementación en diferentes Middlewares, como los servicios Web, CORBA, EJB o DCOM/COM.

Esta especificación presenta los meta-modelos y los perfiles UML para modelizar las aplicaciones de las empresas. Está compuesta de la ECA (Enterprice Collaboration Architecture), los patrones, los modelos específicos a las tecnológicas y las correspondencias hacia las tecnologías.

La ECA proporciona 5 meta-modelos y perfiles:

- **CCA (Component Collaboration Architecture)**: utiliza las clases así como los grafos de actividad y colaboración para modelar la estructura y el comportamiento de los componentes.
- **Entity**: utiliza un conjunto de extensiones UML para modelizar las entidades.
- **Events**: es un conjunto de extensiones UML para modelizar los eventos de un sistema.
- **Business Process**: complementos de CCA y del modelo de comportamiento del sistema.
- **Relationships**: extiende las facilidades del núcleo de UML para responder a las demandas de las relaciones en general, La modelización del negocio y la modelización del software.

Los patrones proporcionan los modelos estándares que pueden utilizarse para elaborar modelos bien definidos para los sistemas EDOC.

Por un lado, la EDOC se centra en la especificación de partes estructurales y comportamentales de los sistemas EDOC. Por otro lado, los modelos específicos de la tecnología y las correspondencias de las tecnologías forman parte de los sistemas EDOC, que considera la correspondencia entre la especificación de ECA y un modelo específico de la tecnología.

Nos interesamos particularmente en la parte CCA de la especificación. Es decir, esta parte de la especificación presenta un meta-modelo de composición general que resume el estado del arte sobre los componentes, la recursividad en la composición, la especificación de los puertos de comunicación entre componentes y la noción de protocolo para representar los diálogos complejos entre componentes. La Figura 3.2 presenta un fragmento del meta-modelo de EDOC.

Entre los elementos de este fragmento del meta-modelo EDOC-CCA, distinguimos:

- **ProcessComponent**: representa el contrato para un componente que realiza las acciones. El ProcessComponent define el contrato externo del componente en término de los puertos y de la coreográfica de actividades de puertos (que envían o reciben mensajes o inicializan sub-protocolos).
- **PortOwner**: contiene el conjunto de los Ports que realizan el contrato del ProcessComponent.
- **Port**: define un punto de interacción entre los ProcessComponents. Las interacciones más complejas entre los componentes utilizan un ProtocolPort, el cual hace referencia a un Protocol, es decir una conversación completa entre componentes.
- **PropertyDefinition**: define un parámetro de configuración de componentes que puede ser un conjunto.

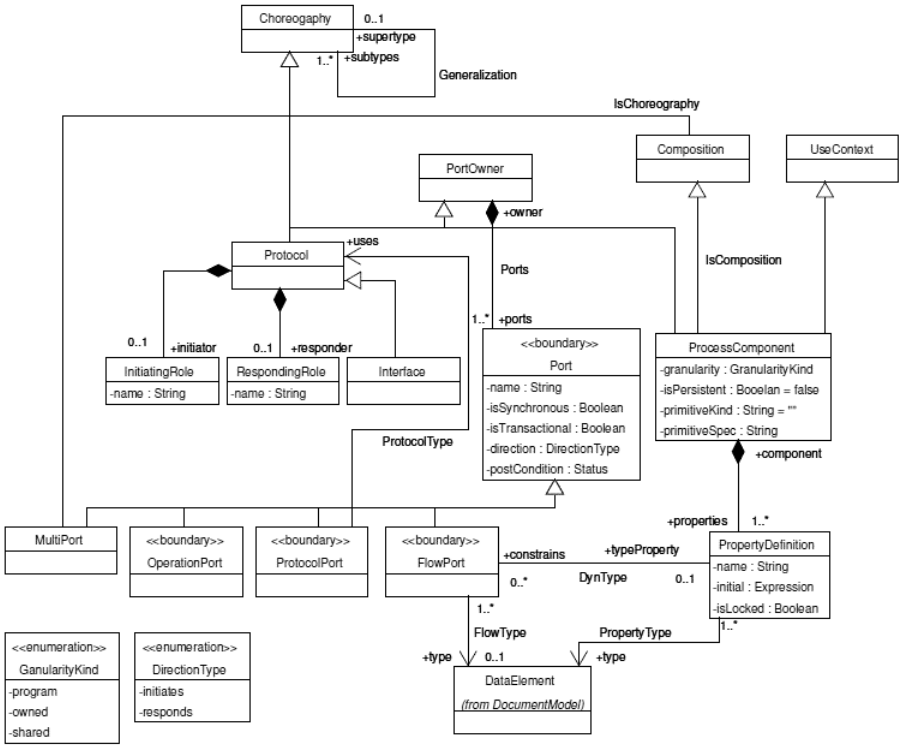

Figura 3.2: El meta-modelo de la especificación estructural de EDOC (un fragmento).

- **InitiatingRole**: representa el rol del protocolo que envía un primer mensaje.
- **RespondingRole**: representa el rol del protocolo que recibe un primer mensaje.
- **Interface**: representa una interfaz de objeto simple. Esta puede contener las OperationPorts que reperesentan las semánticas de call, return, así como las de los FlowPorts que representan las operaciones one-way.

Un cierto número de herramientas permiten definir y manipular los meta-modelos basados en EDOC. Por ejemplo, Component-X proporciona una interfaz grafica para crear y editar modelos, utilizando la notación CCA.

3.3. Lenguajes de meta-modelización

EDOC y UML son lenguajes de modelización. Estos tienen una sintaxis y una semántica adecuada en la creación de modelos, estos últimos están especificados por un meta-modelo.

En la literatura, podemos encontrar diferentes definiciones de meta-modelo:

- Un meta-modelo, es un modelo de un lenguaje de modelos [56].
- Un meta-modelo, es un modelo que define el lenguaje para expresar un modelo [72].

- Un meta-modelo utiliza otros lenguajes conocidos como lenguajes de meta-modelización. Cada lenguaje de meta-modelización corresponde a un meta-modelo, es decir, lo que precede el meta-modelo.
- Un meta-modelo es un modelo que define el lenguaje para expresar un meta-modelo. La relación entre un meta-meta-modelo y un meta-modelo es análoga a la relación entre un meta-modelo y un modelo [72].

La relación entre un meta-meta-modelo, un meta-modelo, un modelo y la información están definidos en la arquitectura de meta-modelización (framework de meta-modelización). Esta relación se basa en la arquitectura de 4 niveles que se presenta en la Figura 3.3.

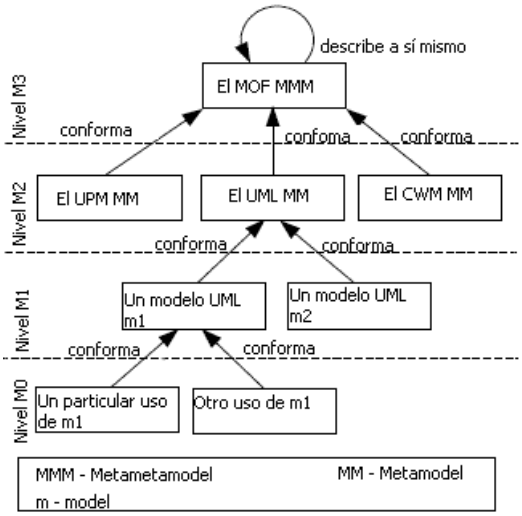

Figura 3.3: La arquitectura de 4 niveles.

En este documento, definimos que las relaciones entre el nivel M3 y M2, el nivel M2 y M1, y el nivel M1 y M0 son del tipo ConformsTo (conforme a). Podemos citar como ejemplos de lenguajes de meta-modelización, a MOF (Meta Object Facility) [68] y Ecore [27, 26]. Ambos están basados en las nociones de la orientación a objetos. En este caso, un objeto es una instancia de clase. Así, el termino instancia (es decir, una ocurrencia de cualquier cosa) relaciona una clase y un objeto.

Esta arquitectura considera los siguientes niveles [68, 74]:

- **M3 (meta-meta-modelo)**: el nivel M3 constituye la base de la arquitectura de meta-modelización. La función primaria de este nivel es definir el lenguaje para especificar un meta-modelo.
- **M2 (meta-modelo)**: un meta-modelo se corresponde a un meta-meta-modelo. La función primaria del nivel M2 es la de definir un lenguaje para especificar los modelos.
- **M1 (modelo)**: un modelo se corresponde a un meta-modelo. La función primaria de este nivel es la de definir un lenguaje para describir un dominio de información (es decir, el nivel de los objetos).

- **M0 (información)**: los objetos del usuario se representan por un modelo. La primera responsabilidad de los objetos del usuario es la de describir un dominio de información específica.

En esta arquitectura, remarcamos que existen pocos meta-meta-modelos (MOF y Ecore), muchos meta-modelos (UML, EDOC, UEML), y un gran número de modelos y por ende una infinidad de información.

3.4. El formato de intercambio normalizado: XMI

XMI (XML Metadata Interchange) permite el intercambio de modelos serializados en XML. XMI se concentra en el intercambio de metadatos en concordancia con MOF [73]. El objetivo de XMI es permitir serializar e intercambiar los meta-modelos MOF y los modelos basados en estos modelos en forma de archivos y utilizando dialectos XML. Permite también serializar los meta-modelos que se crean con Ecore. Esto es posible, ya que XMI no define un único dialecto XML, pero si un conjunto de reglas que permiten crear un DTD o un XML-Schema para diferentes meta-modelos.

Así, los meta-modelos en concordancia con MOF o con Ecore y sus respectivos modelos pueden portarse al utilizar XMI.

La Figura 3.4 presenta un modelo simple UML y el Cuadro 3.1 la correspondiente representación en XMI.

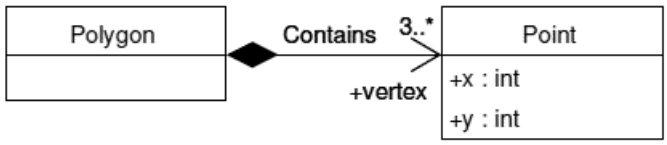

Figura 3.4: Un modelo simple de un polígono.

3.5. El desarrollo tradicional de software y XP (eXtreme Programming)

El desarrollo tradicional de software (es decir, antes de MDA) está más orientado hacia la generación de código. Un proceso típico de este enfoque tradicional se puede dividir de acuerdo a las siguientes etapas:

1. La conceptualización y la recopilación de exigencias (requerimientos).
2. El análisis y la descripción funcional (análisis).
3. El diseño (diseño).
4. La codificación.
5. Las pruebas.
6. El despliegue.

```
<?xml version = '1.0' encoding = 'UTF-8' ? >
<XMI xmi.version = '1.2' xmlns:UML =
'org.omg.xmi.namespace.UML' timestamp = 'Tue Nov 23 21:00:33
CET 2004' >
***
<XMI.content >
<UML:Model xmi.id = 'a1' name = 'model 1' isSpecification =
'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' >
***
<UML:Class xmi.id = 'a4' name = 'Polygon' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract
= 'false' isActive = 'false' >
</UML:Class >
<UML:Class xmi.id = 'a6' name = 'Point' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract
= 'false' isActive = 'false' >
***
<UML:Classifier.feature >
<UML:Attribute xmi.id = 'a8' name = 'x' visibility = 'public'
isSpecification = 'false' ownerScope = 'instance' >
***
</UML:Attribute >
<UML:Attribute xmi.id = 'a12' name = 'y' visibility = 'public'
isSpecification = 'false' ownerScope = 'instance' >
***
</UML:Attribute >
</UML:Classifier.feature >
</UML:Class >
<UML:Association xmi.id = 'a15' name = 'Contains' isSpecification
= 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false' >
***
</UML:Association >
***
</XMI.content >
</XMI >
```

Cuadro 3.1: La representación correspondiente en XMI del modelo del polígono..

El problema del enfoque tradicional es que los documentos y diagramas creados después de las 3 primeras etapas pierden rápidamente su validez cuando inicia el codificado. Además, cuando el sistema se modifica muchas veces, la discrepancia entre el código, la documentación y los diagramas aumenta [47].

XP es un enfoque deliberado y disciplinado para el desarrollo de software. Este enfoque se basa en reglas simples y prácticas [25]. Una característica de XP es la interacción intensiva entre los clientes y los desarrolladores durante un proyecto, el cliente se vuelve un socio activo en el desarrollo. La Figura 3.5 presenta el ciclo de vida de un proyecto XP.

XP se basa en la utilización de código para dirigir el desarrollo de software. Las principales etapas son la codificación y las pruebas. Sin embargo, la codificación y las pruebas hacen que el mantenimiento de los sistemas sea difícil. Además, la utilización

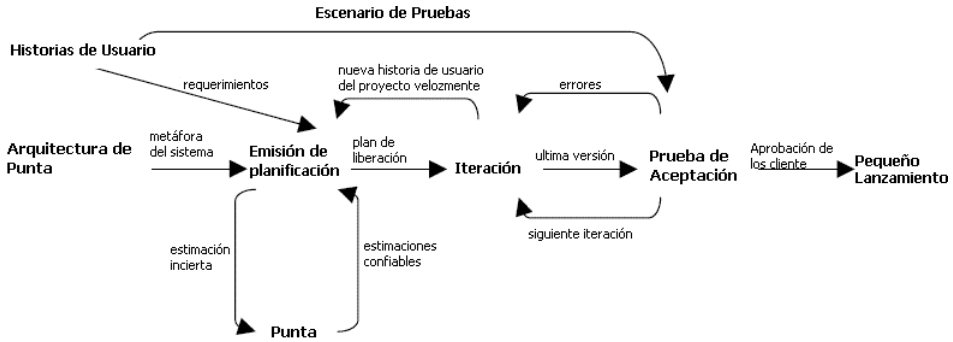

Figura 3.5: El ejemplo de un proyecto XP [113].

de XP se aconseja en la realización de proyectos de tamaño razonable, y con un ciclo de vida corto. A final de cuentas, la extensión o el mantenimiento de un proyecto realizado con el paradigma XP siempre implicara problemas, ya que los desarrolladores expertos en XP frecuentemente cambian de empleo.

3.6. La arquitectura dirigida por los modelos (MDA)

En esta sección, describimos a detalle la arquitectura dirigida por los modelos o MDA (Model Driven Architecture), basada en las tecnologías estándares que presentamos anteriormente (UML, MOF y XMI). Aprovechamos para destacar, la definición, los actores y las técnicas de desarrollo de MDA. Sin embargo, es necesario comprender que no hay una especificación precisa de MDA, pero si un conjunto de especificaciones que forman a MDA.

3.6.1. MDA: definiciones y actores.

En esta parte, describimos MDA respondiendo las siguientes preguntas: Qué es MDA? Por qué MDA? Cuáles son los beneficios del enfoque MDA? Por quién se concibió y normalizo?

¿Qué es MDA?

En la literatura, encontramos varias definiciones de MDA. Presentamos las más representativas en los siguientes párrafos.

MDA es "una evolución de la OMA (Object Management Architecture) que asegura la integración y la interoperabilidad para cubrir el ciclo de vida de un sistema después del modelo de negocios y su diseño, hasta la construcción de componentes, el ensamblado, la integración, el despliegue, la gestión y la evolución"[66]. Los fundamentos de MDA son el resultado de la experiencia al crear estándares como UML, MOF, CWM, XMI e IDL. MDA define igualmente una arquitectura para estructurar los modelos con el fin de permitir la integración, la interoperabilidad y la portabilidad.

Es un enfoque que aumenta el potencial de los modelos para el desarrollo de sistemas orientados a objetos. Se dice que está dirigida por los modelos "ya que proporciona un

enfoque de utilización de modelos para dirigir la comprensión, el diseño, la construcción, el despliegue, la operación, el mantenimiento y la modificación de sistemas"[147].

MDA es un enfoque para la especificación de los sistemas de las tecnologías de la información. Separa la especificación de las funcionalidades de la especificación de la implementación de estas funcionalidades en una plataforma tecnológica específica.

El estándar MDA del OMG es un caso particular del enfoque MDE (Model Driven Engineerig) [56].

En resumen, definimos MDA como un planteamiento de desarrollo basado en los modelos y en un conjunto de estándares del OMG. Este planteamiento permite separar las especificaciones funcionales de un sistema (el PIM, Platform Independent Model) de las especificaciones de su implementación (el PSM, Platform Specific Model). Los estándares fundamentales de MDA son IDL, UML, MOF, CWM y XMI. En el caso de MDA, el OMG destaca a la transformación como operador fundamental antes de ser aplicado entre PIM y PSM o entre PIMs o entre PSMs.

MDE es un enfoque más general que MDA: considera la existencia de varios modelos en un amplio sentido (es decir, meta-meta-modelos y los meta-modelos son también modelos) sobre los cuales se pueden realizar operaciones especificas. Las operaciones pueden ser muy elementales (por ejemplo, create, update, delete, select, enumeration) o complejas (por ejemplo, matching, differencing, merge, transformation) [10, 54]. Además, MDE es un enfoque abierto que considera a diversos espacios tecnológicos con la finalidad de crear una armonía.

En el planteamiento de MDA, todo es considerado como modelo, igualmente los esquemas el código fuente o binario. El modelo independiente de la plataforma (PIM) y el dependiente de la plataforma (PSM) constituyen los principales tipos de modelos en el estudio de MDA.

Una plataforma está definida como "un conjunto de subsistemas y tecnologías que proveen un conjunto coherente de funcionalidades a través de interfaces y el uso de patrones especificados, con lo cual cualquier aplicación soportada por la plataforma puede usarse sin preocupación de los detalles de cómo las funcionalidades que provee dicha plataforma se han implementado"[71]. Sin embargo, esta definición de plataforma es ambigua.

En [71], los autores intentan clarificar el término plataforma, describiendo algunos ejemplos:

- **Tipos de plataformas genéricas**:
  - **Object**: plataforma que soporta el estilo arquitectónico de los objetos con interfaces, consultas individuales de servicios, realización de servicios desencadenados por estas consultas, y respuesta a estas consultas.
  - **Batch**: plataforma que soporta una serie de programas independientes y cuya ejecución es secuencial.
  - **Dataflow**: plataforma que soporta un flujo continúo de datos entre las partes que constituyen el software.
- **Tipos de plataformas especificas a la tecnológica**:
  - **CORBA**: plataforma objeto que permite la invocación a distancia y los estilos de arquitecturas basadas en los eventos.

    - **CORBA Components**: plataforma objeto que permite el estilo arquitectónico basado en los componentes y los depósitos.

- **Tipos de plataformas especificas al vendedor**:

    - **CORBA**: Iona Orbix, Borland Visbroker, JDK-CORBA.

Estos ejemplos ayudan a comprender lo que es una plataforma. Sin embargo, se presenta otro problema. En estos ejemplos, los autores mencionan los tipos de plataforma genéricos, entre ellos los objetos. En este caso, un modelo de negocios en UML o en EDOC puede ser independiente de las plataformas específicas a la tecnología o al proveedor de software, pero no es independiente del objeto, ya que UML y EDOC están basados en la noción de objeto. Por lo tanto, el hecho de que un modelo sea independiente o no de una plataforma es relativo. En realidad, un modelo debe de ser siempre independiente de una plataforma, cualquiera que sea el objeto o de otra plataforma. Existen otras propuestas para clasificar los modelos que parecen ser más interesantes o coherentes como la que se presenta en [42].

Las operaciones de transformación son también la base del planteamiento MDA [69]. En este caso, las transformaciones pueden agruparse en:

- **PIM hacia PIM**: estas transformaciones se efectúan para agregar o eliminar información a los modelos. El paso de la fase de análisis a la fase de diseño es la transformación más natural de este tipo. Estas transformaciones no son siempre automáticas, y a menudo exigen la intervención del desarrollador para agregar o eliminar la información.

- **PIM hacia PSM**: estas transformaciones se efectúan cuando las PIM son suficientemente ricas con la información para poder ser transformadas sobre una plataforma tecnológica. Un ejemplo de esta clase es la transformación de un modelo de negocios en UML hacia un modelo UML que considera la lógica de negocio utilizando una plataforma como los servicios Web.

- **PSM hacia PIM**: estas transformaciones se efectúan para separar la lógica de negocios de la plataforma tecnológica. Por ejemplo, una transformación de un PSM (por ejemplo, un modelo UML que utiliza perfiles CORBA) que elimina la plataforma (por ejemplo, servicios Web) y devuelve la lógica de negocios. Particularmente, esta clase de transformación es la más solicitada en el proceso de ingeniería inversa [23].

- **PSM hacia PSM**: estas transformaciones se efectúan en las fases de refinamiento de las plataformas tecnológicas, de despliegue, de optimización o de reconfiguración.

Las relaciones entre PIM, PSM, meta-modelos, infraestructura (es decir plataforma) y otras tecnologías se describen por el meta-modelo MDA (ver la Figura 3.6) [66].

Analizando la Figura 3.6, podemos remarcar que MDA proporciona el proceso mediante el cual un modelo se transforma en otro modelo. La Figura 3.7 presenta otra perspectiva para comprender los elementos de una transformación y las relaciones entre ellos [39].

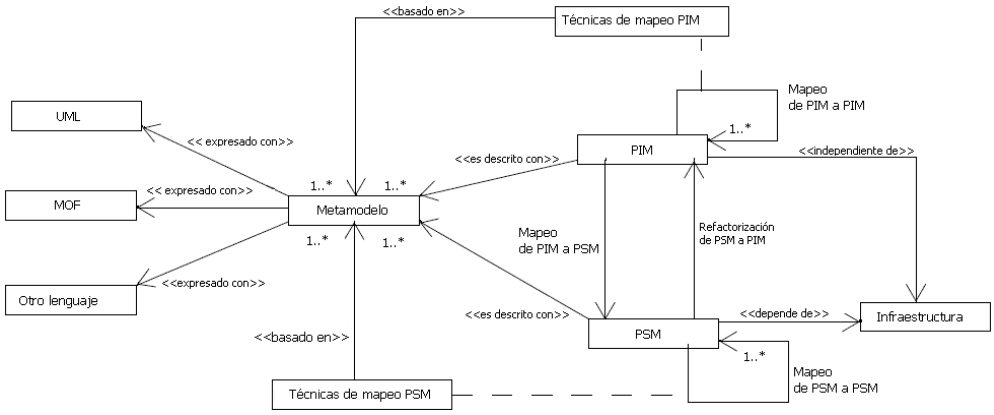

Figura 3.6: La descripción del meta-modelo MDA [66].

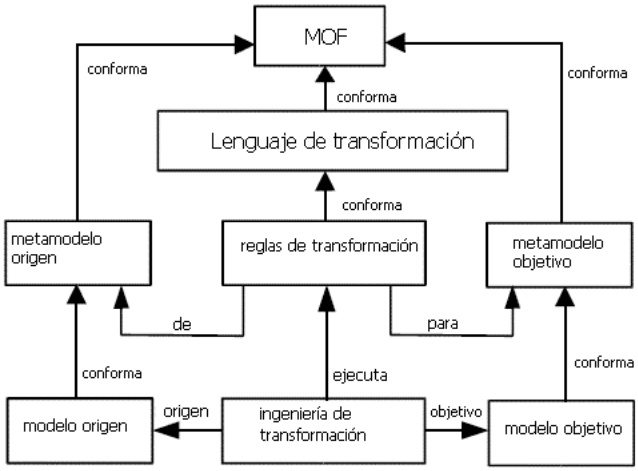

Figura 3.7: La transformación de modelos en MDA [39].

¿Por qué MDA?

En los años 80, la noción de objeto era considerada como el núcleo alrededor del cual se articulaba el desarrollo del software para los sistemas de información. Este concepto también origino el nacimiento de los métodos de análisis y diseño orientados a objetos, como el de Booch. La adopción de la tecnología de objetos ha permitido desde entonces avances incuestionables en el desarrollo de software, pero no ha resuelto del todo algunos problemas.

Más tarde, la noción de objetos no podía responder por si misma a todas las exigencias del desarrollo y mantenimiento de los sistemas de información, ya que estos se volvieron cada vez más complejos. Aparecieron entonces otras tecnologías, como los objetos distribuidos (por ejemplo, CORBA), los patrones de diseño y los componentes.

En este contexto, los Middleware son el desarrollo más importante para asegurar la cooperación de los objetos o de los componentes producidos por diferentes empresas. Así, a mediados de los años 90, el OMG creyó que era posible estandarizar un solo y único Middleware capaz de resolver todos los problemas de interoperabilidad y de porta-

bilidad. Sin embargo, otros Middleware como DCOM también surgieron. Igualmente se propusieron puertas de enlace entre estos 2 Middlewares, pero esto no representaba una resolución satisfactoria, y contribuía a otros problemas. Un poco después, la aparición de otros Middleware como EJB, CCM, .Net y los servicios Web volvieron aun más complicada esta situación.

Así, el concepto único y simplificador de "Todo es objeto"del inicio de los años 80, evoluciono hacia un "Todo es un componente"en los años 90. En este contexto, los Middleware jugaron un rol fundamental para resolver los problemas de interoperabilidad y portabilidad. Estas soluciones eran satisfactorias en la medida de que eran capaces de responder a las necesidades de los sistemas computacionales, en particular a los de la gestión de la complejidad.

Todas estas tecnologías se han desarrollado para responder a necesidades específicas y, por otro lado, a medida que una tecnología respondía a una necesidad, aparecían otras necesidades. Por lo tanto existen varias soluciones, en el caso de los Middleware estos no son capaces de responder a las nuevas necesidades. En el caso de la Internet, los servicios Web no son una solución completa, pero por el momento es un Middleware que responde a las necesidades de desarrollo de las aplicaciones Web.

La unificación de todos los métodos en UML demostró la importancia y la viabilidad de la modelización como soporte fundamental para desarrollar y mantener los sistemas computacionales.

El OMG ha comprendido que un Middleware como CORBA no era la solución más adaptada para enfrentar este nuevo contexto en el cual los sistemas computacionales son cada vez más complejos, a continuación se resume algunas de las razones de este hecho:

- **La integración de nuevos aspectos**: la seguridad, la disponibilidad y la fiabilidad se deben considerar desde el inicio del desarrollo de un sistema computacional.
- **La aparición de nuevas tecnológicas**: actualmente, los sistemas evolucionan más rápido que en el pasado, no solamente porque las necesidades del negocio y de las aplicaciones cambian, sino también porque las plataformas técnicas están en constante y rápida evolución. Por lo tanto, la protección de las inversiones en software contra la obsolescencia debido a la integración de nuevas plataformas técnicas es un punto importante en este contexto.
- **La compatibilidad con las antiguas tecnologías**: las nuevas tecnologías aparecen a gran velocidad, pero las antiguas tecnologías no desaparecen tan rápidamente. Este tipo de tecnología están presentes en los sistemas heredados (legacy systems), lo que impone un problema de heterogeneidad importante.
- **El gran numero de tecnologías**: Los sistemas computacionales distribuidos se construyen generalmente con diversas tecnologías además, estos sistemas deben siempre comunicarse con otros sistemas de manera transparente.

Estos factores reunidos hacen que la productividad y la calidad de los sistemas software sea difícil de asegurar. Además, el tiempo de entrega de estos sistemas es cada vez más corto, por consecuencia las empresas se tienen que adaptar a las nuevas exigencias lo más rápido posible.

Así, el OMG tomo la decisión de invertir en el enfoque dirigido por los modelos para hacer frente a la complejidad de desarrollo, mantenimiento y evolución de los sistemas computacionales.

La idea principal de MDA está representada en su logotipo (ver la Figura 3.8). Los modelos son las entidades capaces de unificar y de soportar el desarrollo de sistemas computacionales asegurando la interoperabilidad y la portabilidad. Los modelos representan el núcleo (por ejemplo, MOF, UML y CWM), los Middleware representan el nivel de ejecución de las aplicaciones (por ejemplo, CORBA, servicios Web y .Net), y los servicios normalizados proporcionan el soporte a la ejecución de estas aplicaciones (por ejemplo, la seguridad, transacciones y eventos). Todo este conjunto de tecnologías se utiliza para dar soporte a diferentes aplicaciones como las finanzas, el comercio electrónico o las telecomunicaciones.

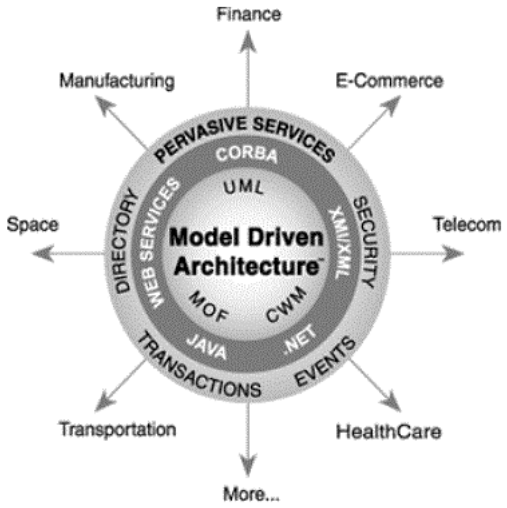

Figura 3.8: La arquitectura dirigida por los modelos.

MDA se desarrollo para soportar la evolución y administrar la complejidad de los sistemas distribuidos, pero también para armonizar las tecnologías. La aparición de la Internet y la utilización masiva de los sistemas computacionales han igualmente originado nuevas exigencias jamás imaginadas. En este caso, los modelos constituyen la capa necesaria para proporcionar el nivel de abstracción demandado en la actualidad.

El enfoque dirigido por los modelos (MDA o MDE) no es una solución que va a resolver todos los problemas, pero parece ser en la actualidad la única idea capaz de proporcionar respuestas satisfactorias a todas estas nuevas exigencias. Sus límites no son aún conocidos, pero debemos estar conscientes que todo enfoque tiene sus límites.

¿Cuáles son los beneficios del enfoque MDA?

Ciertos beneficios del enfoque MDA aún no se han verificado, pero los resultados actuales son prometedores. Se han identificado los siguientes beneficios.

- El mismo PIM puede utilizarse varias veces para generar los modelos sobre plataformas diferentes (PSMs) [30].
- Las diferentes vistas de un mismo sistema, es decir diferentes niveles de abstracción o detalles de implementación [66].

- El aumento de la portabilidad y de la interoperabilidad de los sistemas a través de los modelos.
- La protección de la lógica de negocios contra los cambios o la evolución de las tecnologías [66].
- La evolución simultanea de los modelos de negocio y de las tecnologías.
- La protección contra los factores inherentes al desarrollo manual de los sistemas [17].
- El aumento del retorno de las inversiones en tecnologías.
- El soporte para la reingeniería y la ingeniería inversa [23] que hace posible la recuperación de la lógica de negocios a partir del código fuente o de los ambientes de implementación.
- El aumento de los niveles de abstracción con los cuales los humanos pueden comunicarse con otros humanos de manera más productiva.
- La reutilización y la composición de modelos hace posible la construcción de sistemas complejos.
- El aumento de la interacción y de la migración entre diferentes espacios tecnológicos [48].

Además, podemos mencionar que un enfoque dirigido por los modelos obliga a los arquitectos y desarrolladores a centrarse en la arquitectura y en el modelo del sistema en desarrollo. Por el contrario, un enfoque centrado en el código se enfoca en la atención de los arquitectos y de los desarrolladores en el código, por consecuencia descuidan las propiedades principales del sistema.

MDA es una solución más poderosa que el enfoque tradicional o XP, ya que el modelo y el código están siempre en sincronía, la información sobre los proyectos se guardan de manera formal y bien documentada. Además, asegura el desarrollo rápido de los sistemas computacionales.

¿Por quién ha sido concebido y normalizado?

MDA es una iniciativa del OMG, debido a la constatación de que las nuevas exigencias no se podían resolver utilizando CORBA. Esta proposición se ha reforzado con las diferentes universidades, industrias y dependencias gubernamentales que han visto en el enfoque dirigido por los modelos, una solución necesaria y viable en el contexto actual.

Hasta estos días, no hay una especificación MDA, sino un conjunto de estándares (UML, CWM, XMI, MOF) y documentos que establecen la base y la dirección para la creación y por consiguiente la utilización del planteamiento MDA.

3.6.2. Correspondencia

Los conceptos de correspondencia (mapping) y de transformación se confunden frecuentemente en la literatura de MDA. Muchos documentos utilizan los términos mapping y transformación de manera intercambiable. Los siguientes párrafos ilustran algunas definiciones.

Mapping es "un conjunto de reglas y técnicas utilizadas para modificar un modelo con el fin de obtener otro modelo. Los mappngs se utilizan para la transformación de PIM hacia PIM, PIM hacia PSM, PSM hacia PSM y PSM hacia PIM"[138].

Un mapping en MDA proporciona "Las especificaciones para la transformación de un PIM en un PSM en una plataforma particular. El modelo de la plataforma determina la naturaleza del mapping. Un mapping se especifica utilizando un lenguaje que permite describir la transformación de un modelo en otro modelo. La descripción puede ser en lenguaje natural, lenguaje algorítmico, lenguaje de acción, o en lenguaje de transformación de modelo."[147].

Un mapping es la "Aplicación o la ejecución de una función de mapping con el fin de transformar un modelo en otro. Una función de mapping es una colección de reglas o de algoritmos que definen como un mapping especifico se realiza"[122].

No compartimos el punto de vista de la literatura sobre MDA, más bien estamos de acuerdo con la literatura de las bases de datos, en la cual existe una distinción bien explicita entre mapping y transformación. En este dominio el termino mapping se utiliza en el sentido de correspondencia, y el termino transformación se utiliza para hacer referencia a la accion de transformar un modelo en otro [10, 54]. Por lo tanto, utilizamos el término correspondencia (traducción de mapping) para referenciar las interrelaciones que existen entre los elementos de un modelo (o meta-modelo) y aquellos del otro modelo (o meta-modelo).

Por un lado, la correspondencia describe cuales son los elementos equivalentes o similares entre 2 meta-modelos y por otro lado, la transformación utiliza la correspondencia para completar la creación de un modelo objetivo a partir de un modelo fuente.

3.6.3. La transformación de modelos

La transformación de modelos es "el proceso de conversión de un modelo a otro modelo del mismo sistema"[71].

Durante estos últimos años, la transformación de modelos, en particular la creación de un lenguaje de transformación normalizada, se ha vuelto el centro de reflexiones alrededor de MDA. El OMG ha hecho propuestas a través de RFP QVT [69] para que se realicen propuestas de un lenguaje de transformación. Hasta nuestros días, existen varias propuestas, pero estas se encuentran en análisis.

Algunos otros lenguajes de transformación, han demostrado la viabilidad de los conceptos alrededor de MDA, por ejemplo: ATL (Atlas Transformation Language) [41], Mtrans [84], YATL (Yet Another Transformation Language) [82], UMLx [115], etc.

Estos lenguajes de transformación se basan en diferentes paradigmas: imperativos o declarativos, con tipos o sin tipos sintácticos o con tipos semánticos, textuales o gráficos [24]. Es entonces posible que un lenguaje de transformación sea más adaptado que otro en un contexto especifico. Sin embargo, en general, la tendencia es proponer lenguajes mixtos que integren diferentes paradigmas.

Sea como sea, estos lenguajes de transformación representan una evolución significativa para MDA. Además se prevé que el lenguaje de transformación normalizado sea un elemento fundamental en el futuro de MDA.

Una herramienta de transformación de modelos toma como entrada un modelo y lo transforma en otro modelo. En el nivel actual de MDA, el modelo de entrada es frecuentemente el PIM y el modelo objetivo es frecuentemente el PSM. Sin embargo,

una transformación puede tener un PSM como fuente y otro PSM como objetivo. Una transformación puede también tener un PSM como el modelo fuente y un PIM como el modelo objetivo. Además, una transformación de PIM hacia PIM es igualmente posible, aun cuando necesite la intervención de un experto humano.

A continuación, mencionamos 3 conceptos ligados al término transformación:

- Inicialmente, una transformación es la generación manual, semiautomática o automática de un modelo objetivo a partir de un modelo fuente, en conformidad con una definición de transformación.
- Una definición de transformación (transformation specification) es una colección de reglas de transformación que describen en conjunto como un modelo especificado en un lenguaje fuente puede transformarse en otro modelo especificado en un lenguaje objetivo.
- Una regla de transformación es una descripción de la forma de cómo una o varias construcciones en un lenguaje fuente pueden transformarse en otra construcción en un lenguaje objetivo.

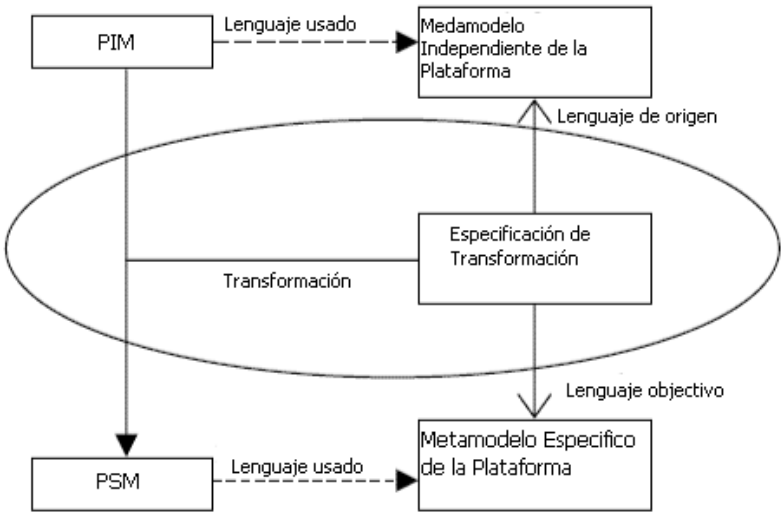

Figura 3.9: La transformación del meta-modelo [71].

En este trabajo de tesis, no se introducirá en las transformaciones de meta-modelos. Pero se muestra un ejemplo de transformacion en la Figura 3.9. Sin embargo, distinguimos explícitamente la etapa de especificación de transformación en 2 fases que denominamos respectivamente como la especificación de correspondencia y la definición de transformaciones.

Adicional a estos enfoques, en la actualidad, el enfoque de weaving se propuso en el estudio de MDA [43, 40]. La Figura 3.10 presenta la técnica de weaving comparada con la técnica de marking.

La técnica de marcado (marking) consiste en agregar marcas (marks) en la transformación de un PIM en un PSM. Las marcas son extensiones de un modelo inicial y

no intrusivo que especifica la información necesaria para la transformación del modelo sin contaminar el modelo original [60]. Están definidas por los modelos de marcado, los cuales describen la estructura y la semántica de un conjunto de tipos de marcas.

En el estudio de MDA, las marcas se efectúan gracias a la utilización de perfiles UML. Así, el meta-modelo de una plataforma específica se representa como una extensión de UML a través del perfil de esta plataforma. Las marcas también pueden utilizarse para refinar un PIM con el fin de agregar otro tipo de información como la seguridad, pero sin especificar la plataforma la cual se va a implementar.

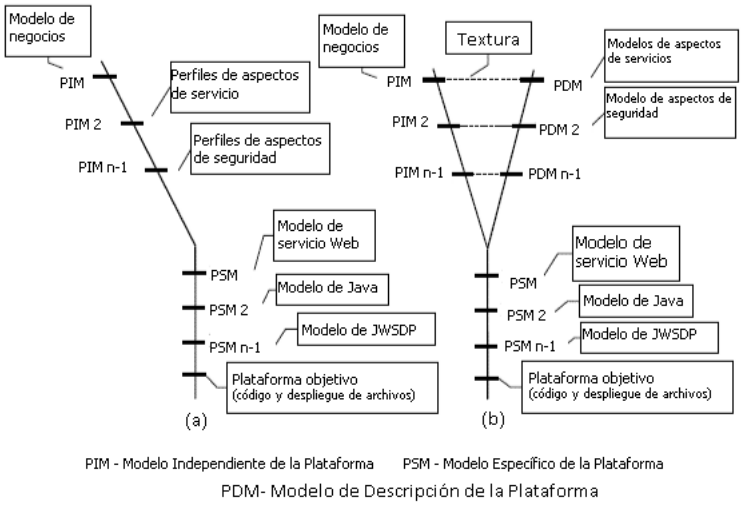

Figura 3.10: Las técnicas, a) marking y b) weaving.

Weaving es la técnica más reciente, se basa en la idea del tejido (texture mapping) entre el modelo independiente de la plataforma (PIM) y el modelo de descripción de la plataforma (PDM, Platform Description Model), Luego su transformación en un modelo especifico de la plataforma (PSM).

3.6.4. Los mitos alrededor de MDA.

Podemos enumerar algunos de los mitos alrededor de MDA:

- **La independencia total de la plataforma**: la noción de plataforma es relativa, y no absoluta. Por ejemplo, IDL-CORBA puede ser considerado como independiente de la plataforma ya que se ha concebido de forma independiente del sistema operativo y de los lenguajes de programación. En este caso, la plataforma de referencia es el sistema operativo y los lenguajes de programación. En el caso de un modelo de negocios basado en UML, es independiente de la plataforma si consideramos un tipo de middleware como referencia (por ejemplo, CORBA). Sin embargo, este modelo UML es especifico a la plataforma orientada a objetos, ya que UML es precisamente orientado a objetos.

- **La utilización de UML para modelar todo tipo de sistema**: UML es un lenguaje para los ingenieros y los arquitectos de software. Sin embargo, UML no es razonablemente el mejor lenguaje para modelar ciertos aspectos de sistemas

particulares. Se pueden desarrollar y/o considerar otros lenguajes que consideren estas especificidades, es decir, considerar fuertemente la tendencia actual a este respecto, DSL (Domain Specific Language) [18]. En el framework basado en los cuatro niveles de modelos, un aspecto importante es la presencia de varios lenguajes de modelización (meta-modelos) como lo sugiere MDE. Esto último, abre el espacio de los formalismos de modelización adaptados al dominio.

- **La generación automática y completa de código a partir de los modelos**: como regla general, es posible generar un código a partir de un modelo, sólo si la información necesaria a este respecto se ha considerado en dicho modelo. En el estado actual de la ingeniería de los modelos y de MDA; no es posible generar la integralidad del código, puesto que los modelos no definen la semántica y la lógica con un nivel de detalle suficiente. Para resolver este problema, el OMG propone la utilización de action semantics de UML 2.0. La action semantics (acción semántica) es una proposición que tiene aspectos positivos, pero es aún muy reciente y poco experimentada para asegurar que su uso permitirá la generación de la integralidad del código en un enfoque MDA [92].

- **Los problemas del enfoque MDA se resolverán con un lenguaje de transformación normalizado**: la transformación es sin duda la parte importante de MDA, que en la actualidad es el foco de atención de todos los estudios que realizan en torno a esta propuesta. Sin embargo, algunos resultados obtenidos, parecen ser prometedores. Una visión más futurista de MDA podría estar ligada a la gestión de modelos [10, 119].

- **La utilización de modelos simplificará a los sistemas complejos**: es cierto que los modelos pueden facilitar la gestión de la complejidad, sin embargo no puede transformar un sistema complejo en un sistema simple. La diferencia reside en el hecho de que la utilización de los modelos nos permite estudiar un aspecto específico de este sistema, de comprenderlo, y dominarlo. Un modelo no puede representar más que un número limitado de aspectos, pero si se conjuntan varios modelos, estos podría representar un sistema en su conjunto. La ventaja principal de la utilización de modelos es que esta nos permite tratar un problema por fragmentos con el fin de mejorar el aprendizaje y por lo tanto proporcionar una solución. Así, la complejidad de un sistema se divide con el fin de que las personas, o las maquinas, puedan comprender los fragmentos y presentar soluciones viables. Por consecuencia, el conjunto de soluciones permite resolver la complejidad global del sistema.

3.6.5. El futuro de MDA

Hacer predicciones en el tema de las tecnologías es muy difícil, sobre todo si estas están ligadas a los sistemas de información y a su evolución.

Pensamos que el futuro de MDA dependerá de su utilización (para construir y mantener los sistemas computacionales) y de los resultados obtenidos. Sin embargo, es necesario saber que MDA no es simplemente una teoría, sino una realidad que ya ha aportado resultados positivos.

Otros trabajos igualmente han demostrado la viabilidad de aplicación de MDA; por ejemplo la aplicación del enfoque MDA a los servicios Web [39, 22], a los sistemas em-

barcados [31], a CORBA [7], a las aplicaciones militares [98, 99], y a los sistemas de información [92].

El enfoque dirigido por los modelos ha probado su viabilidad en muchos casos de estudios pero se aconseja también cambiar la cultura de los arquitectos y de los desarrolladores quienes se encargan primordialmente de la creación y el mantenimiento de los sistemas computacionales. En otros términos, MDA debe aún sobreponerse a las resistencias culturales; es decir, situar a los modelos en detrimento de los conocimientos y las practicas de la codificación de las aplicaciones.

En este nuevo contexto, los arquitectos y los desarrolladores deben adquirir nuevas formas de saber hacer las cosas para expresar las funcionalidades de lo sistemas a través de los meta-modelos, las correspondencias entre los metamodelos y las transformaciones de modelos. Las herramientas de diseño y de desarrollo de las aplicaciones deben también evolucionar para soportar a MDA (modelización, transformación y generación de código ejecutable).

Las respuestas a MOF/QVT [69] y la estandarización de un lenguaje de transformación representarán el paso de una etapa crucial para el futuro de MDA, no será lo único. Otros estándares se demandaran a medida de que MDA evolucione. Entre ellos, la normalización de meta-modelos específicos al dominio [18], la creación de un proceso MDA más sofisticado [56], la creación de herramientas que permitan especificar los meta-modelos y editar los modelos basados en estos, la normalización de meta-modelos de plataformas y la interoperabilidad entre las herramientas.

Desde el punto de vista económico, muchas empresas ya han mostrado su interés y la cifra de inversiones para adoptar un enfoque MDA, ha empezado a aumentar de forma considerable.

Capítulo 4

MDA para los servicios Web

Los servicios Web han evolucionado a partir de un conjunto limitado de tecnologías (SOAP, WSDL, UDDI y XML) hacia un número de tecnologías complementarias dedicadas a las aplicaciones para la Internet. Estas tecnologías complementarias tienen como objetivo proporcionar los servicios esenciales para el funcionamiento de los servicios Web o recomendaciones para la creación de aplicaciones orientadas a servicios. Entre estos servicios, podemos citar la seguridad [65], el aspecto transaccional [15], la composición de servicios [6] y los eventos.

4.1. Introducción

Los ejes de investigación que abordan los servicios Web son igualmente numerosos. Podemos citar:

- La composición de servicios Web basados en la tecnología de Workflow [2, 96, 38] y en las redes de Petri [35].
- La composición de los servicios Web basados en la semántica de los servicios Web [55], la ontología [58, 114] y la planificación [1].
- La seguridad de los servicios Web [14, 51, 50, 64].
- El aspecto transaccional y los servicios Web [53, 79].
- El tratamiento orientado a servicios (Service Oriented Computing) [78] y el computo en la grid [88].
- Las aplicaciones móviles basadas en los servicios Web [83, 85].
- Las aplicaciones Internet (B2B e-commerce, B2C) soportadas por los servicios Web [14, 19].
- La gestión de sistemas computacionales basados en los servicios Web (Web Services Management) [29].
- La aplicación de los servicios Web y XML para facilitar la gestión de datos distribuidos y semi-estructurados [3].

En este trabajo, nos interesáremos en la aplicación de MDA en la elaboración de redes sociales en la plataforma de los servicios Web.

En la literatura sobre el enfoque MDA y, más generalmente, en la modelización de los sistemas, existen dos tendencias complementarias y no exclusivas que se refieren a las técnicas de transformación. La primera estudia los problemas ligados a la creación de correspondencias entre meta-modelos (o modelos). La segunda se concentra en el diseño y la realización de un lenguaje de transformación según las recomendaciones del OMG.

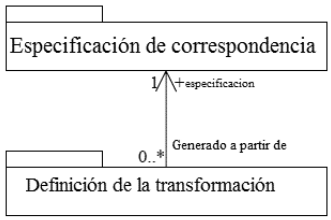

Figura 4.1: Relación entre especificación de correspondencia y definición de transformaciones.

La Figura 4.1 ilustra dicha propuesta y enfatiza la especificación de correspondencias y la definición de transformaciones. Por un lado, la especificación de correspondencias presenta la lógica utilizada para establecer las relaciones entre dos meta-modelos. Por otro lado, la definición de transformaciones contiene reglas precisas para transformar un modelo en otro, utilizando un lenguaje de transformación ejecutable. Es decir, la especificación de correspondencias puede ser considerada como un PIM, y la definición de transformaciones como un PSM.

Por un lado, la especificación de correspondencias nos permite analizar y tomar decisiones referentes a las correspondencias entre dos meta-modelos. Por otro lado, la definición de transformaciones se escribe en un lenguaje de transformaciones que es una plataforma específica en la realización y la ejecución de las transformaciones.

Las siguientes secciones detallan esta prospectiva: separación del estudio de la especificación de correspondencias en la creación de un lenguaje de transformación. Esta separación entre especificación de correspondencias y definición de transformaciones se podría utilizar en nuestro planteamiento.

4.2. Correspondencia entre meta-modelos

En esta sección, presentamos varios trabajos que han abordado la especificación de correspondencias en la perspectiva de explicitar la problemática, establecer una formalización y encontrar una teoría para considerar este tipo de problemática.

G. Caplat y J. Sourrouille definen el mapping en el sentido de la transformación de modelos de la siguiente manera [117]: Un mapping es un conjunto de reglas y técnicas usadas para modificar un modelo de tal manera que se obtenga otro modelo.

Sea $m(s)/f$ el modelo del sistema s en el formalismo *f*. Un mapping es una transformación general *m1*(s)/*f1* ->*m2*(s)/*f2*. El modelo de un formalismo se denomina un metamodelo.

Los autores tratan el problema general del mapping en el cual *f1 = UML + Perfil*, *m(f1)/MOF*, donde f1 se describe en MOF, y donde *m1(s)/f1* es un PIM y *m2(s)/f2* es

un PIM enriquecido con los elementos del PSM. Además, estos representan dos tipos de mappings: directos (o endógenos) e indirectos (exógenos). El mapping directo se realiza cuando los modelos se describen por un mismo formalismo. El mapping es indirecto cuando los modelos se describen por formalismos diferentes. La Figura 4.2 ilustra los dos tipos de mappings.

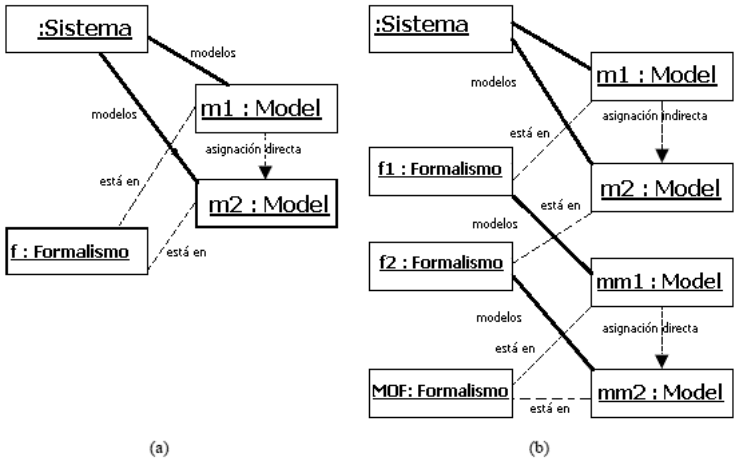

Figura 4.2: a)El mapping directo y b) el mapping indirecto [117].

Un mapping directo es una transformación simple en un proceso de refinamiento de modelos. En el caso del mapping indirecto, una comparación entre los formalismos f1 y f2 es necesaria y una relación debe forzosamente definirse de manera que cada expresión primitiva y abstracta en f1 se traduzca en una expresión primitiva y abstracta en f2. En la Figura 4.2, el meta-lenguaje del formalismo permite la expresión de tales traducciones y es naturalmente MOF.

En relación al mapping indirecto, G. Caplat y J. Sourrouille señalan lo siguiente: "En el caso general, el mapeo indirecto de modelos m1(s)/f1 ? m2(s)/f2 requieren el mapeo directo de formalismos: mm1(f1)/MOF ? mm2(f2)/MOF. Este mapeo puede ser muy costoso y no siempre posible. Afortunadamente, la situación mejora cuando uno de los formalismos provee un mecanismo de extensión tal como como UML. Cualquiera f1 o f2 (o ambos) pueden proveer un mecanismo de extensión".

En relación a las técnicas de transformación, G. Caplat y J. Sourrouille sugieren que la utilización de diferentes formalismos (meta-modelos) puede propiciar que la creación del mapping sea difícil, si no imposible. Esta sugerencia privilegia la técnica de marcado, la cual se basa en los perfiles UML. Sin embargo, en la técnica de transformación de meta-modelos y en nuestro planteamiento, la utilización de diferentes meta-modelos para la creación de modelos es indispensable para la realización de transformaciones.

La Figura 4.3 presenta la proposición G. Caplat y J. Sourrouille para describir los roles de cada elemento del enfoque MDA. Esta Figuratambién ilustra el hecho de que un mapping (en el sentido de transformación) relaciona un modelo fuente y un modelo objetivo. Un mapping también se basa en un modelo (Model of mapping) que está vinculado a un formalismo (es decir, MappingFormalism). Los mappings endógenos y exógenos son extensiones de un mismo mapping de base. Un formalismo hereda las características de un sistema.

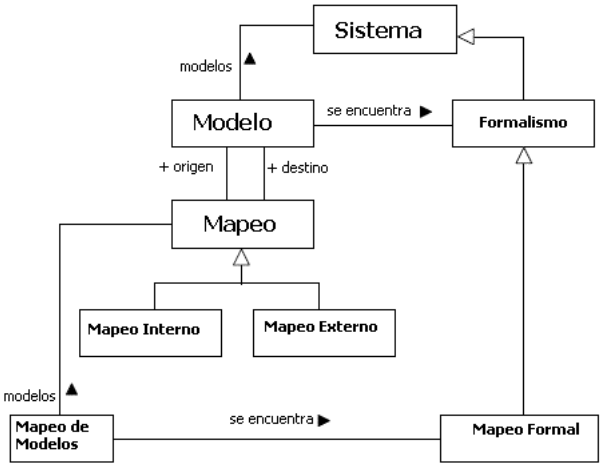

Figura 4.3: Las relaciones entre mapping y formalismo [118].

En [56], J. M. Favre retoma esta Figuray la denomina mega-modelo, que se define como una teoría para los conceptos alrededor del MDE (Model Driven Engineering). En este caso, es importante recordar que MDA es un caso particular de MDE. Según J. M. Favre, este mega-modelo (ver Figura 4.3) no es suficiente para responder a todas las preguntas en el área de MDE [56]. En esta mega-modelo, un "Model of Mapping"no es un modelo. Este no puede ser transformado en otro "Model of Mapping". Además, este mega-modelo considera a UML como el único formalismo, contrariamente a lo que MDE propone, es decir la utilización de varios lenguajes de modelización.

G. Caplat y J. Sourrouille señalan también las diferencias semánticas y sintácticas entre los meta-modelos en el momento de la creación de los mappings. La Figura 4.4 presenta el ejemplo que ellos utilizan para exponer las diferencias sintácticas, y la Figura 4.5 presenta el ejemplo que utilizan en lo que corresponde a las diferencias semánticas.

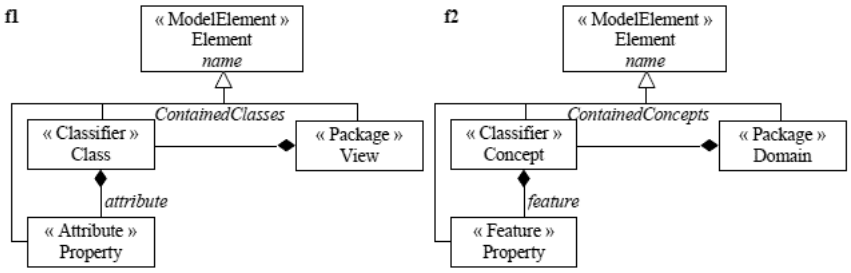

Figura 4.4: Dos meta-modelos simples (multiplicidad no especificada) propuestos en [118].

En la Figura 4.4, las clases estereotipadas (notación de acuerdo a UML) con ModelElement, Classifier, Package, Attribute, Feature hacen referéncia a una meta-instancia de MOF. Esta Figurademuestra que los elementos de los meta-modelos y modelos pueden tener la misma semántica, pero poseen una sintaxis diferente. El elemento Classe en f1 es el equivalente de Concept en f2. La View en f1 es el equivalente de Domain en f2. La

meta-clase Attribute utilizada para crear Property en f1 es el equivalente de la meta-clase Feature utilizada para crear Property en f2.

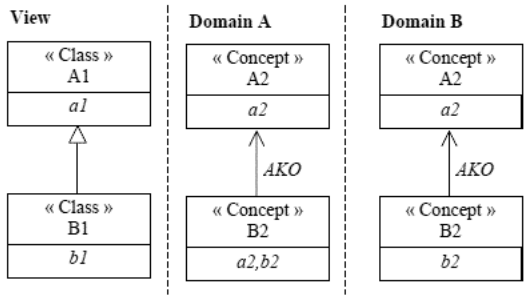

Figura 4.5: Traducción de la relación de la herencia propuesta en [118].

La Figura 4.5 presenta 3 modelos basados en los meta-modelos de la Figura 4.4 así como la notación de herencia en f1 y la noción de asociación binaria en f2. La noción de herencia en f1 debe traducirse en una expresión en f2 con una semántica equivalente. Contrariamente a la Figura 4.4 donde el mapping solo se refiere a la forma, la traducción de un modelo View en Domain debe considerar la semántica. En este caso la herencia puede traducirse en una asociación denominada AKO. Si B1 hereda de A1, las propiedades de A2 deben agregarse a las propiedades de B2. Una primera solución es agregar todas las propiedades de A2 a B2 (el modelo View y Domain A). El modelo se construye fácilmente, pero una modificación del valor de A2.a2 no se propaga a B2.b2.

Si esta modificación se hace dinámicamente, m1 y m2 no tiene el mismo comportamiento, es decir que no son semánticamente equivalentes. Una segunda solución es la transformación del acceso de las propiedades de Concept en una operación que considera la propagación de la alteración de las propiedades. Concretamente la operación ref_value(x) de MOF puede redefinirse en f2 de la siguiente manera:

```
1 ref_value(x)
2 if x 2 self.properties() // self is the current concept
3 result = self.ref_value(x)
4 else if x 2 self.associations(00AKO00).properties()
5 result = self.associations(00AKO00).ref_value(x)
```

J. H. Hausman y S. Kent definen el mapping (en el sentido de correspondencia) de la siguiente manera [36]: " Usamos el termino mapping para encausar el entendimiento general de la 'conexión entre modelos' y referirse a la relación, haciendo siempre la distinción más sutil entre definición e instancia ".

Los mappings deben de estar de acuerdo con las siguientes exigencias [36]:

- Estos deben representar la conexión implícita entre sus partes.
- Una notación para mapping debe ser fácilmente comprensible para permitir su definición y su adopción por los usuarios humanos.
- No debe interferir en la definición de modelos.
- Deben tener una instancia persistente.

La Figura 4.6 ilustra el patrón del modelo para especificar la relación entre modelos propuestos en [82].

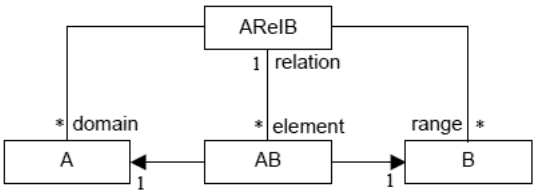

Figura 4.6: El patrón del modelo para las relaciones.

Para especificar las relaciones entre modelos, este patrón establece una relación de (ARelB) y un par de clases (AB) para cada correspondencia entre los elementos de los modelos. Las expresiones OCL se pueden utilizar para definir el dominio (domain), el codominio (range) y las restricciones adicionales de la relación.

La Figura 4.7 ilustra una sintaxis tipo UML para formalizar las relaciones entre los elementos de los modelos. Esta Figuramuestra que cada lenguaje de modelización (UML y Java) se define de forma separada en su package. Gracias a la utilización de las relaciones, cada conexión puede expresarse sin influenciar los contenidos de estos paquetes. Las relaciones se representan por líneas punteadas. De hecho, el mapping es también un modelo.

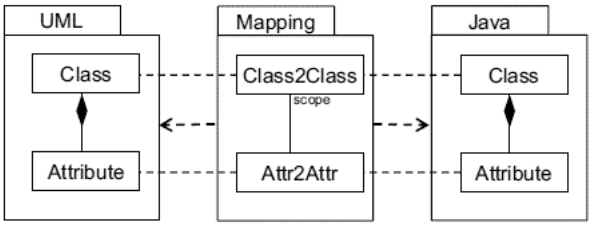

Figura 4.7: Relaciones entre modelos [36].

En la Figura 4.8, la relación R1 une a A1 con B1, y contiene una relación especificada por la relación R2 que une A2 a B2. Además, R1 está asociada a R3 quien une A3 con B3. En este último caso, R3 contiene una cardinalidad del tipo 1:n, el dominio contiene un elemento, y el co-dominio contiene dos elementos representados por la n-tupla x y y.

En el área de las bases de datos y más recientemente en el proyecto de gestión de modelos [10], P. A. Bernstein define el mapping de la siguiente manera: " .. las transformaciones entre modelos, requieren de una representación explicita de mappings, la cual describe como dos modelos se relacionan uno con otro ".

P. A. Bernstein aporta una contribución significativa con su teoría de gestión de modelos y su aplicación en la integración y la evolución de los esquemas [10]. Esta teoría se basa en los operadores para administrar los modelos, en donde las técnicas de correspondencia y de transformación se colocan de forma importante. Entre estos operadores, podemos citar los siguientes:

- **Match**: toma dos modelos como entrada y regresa un mapping entre ellos como resultado.

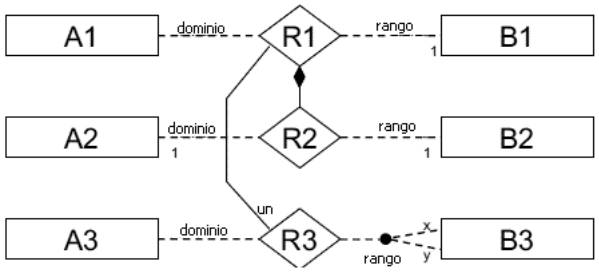

Figura 4.8: Ejemplo de sintaxis de relaciones [36].

- **Compose**: toma un mapping entre los modelos A y B y un mapping entre los modelos B y C, y regresa un mapping entre A y C como resultado.
- **Diff**: toma un modelo A y un mapping entre A y un modelo B, y regresa un sub-modelo de A que no participa en el mapping.
- **ModelGen**: toma un modelo A, y regresa un nuevo modelo B basado en A y un mapping entre A y B.
- **Merge**: toma un modelo A y B y un mapping entre ellos, y regresa la unión C de A y A.
- **Apply**: toma un modelo y una función abstracta f como entrada y aplica f para cada objeto de este modelo.
- **Copy**: toma un modelo como entrada q y regresa una copia de este modelo.
- **ModelGen**: toma un modelo en conformidad a un meta-modelo y genera otro modelo en conformidad con otro meta-modelo.
- **Enumerate**: toma un modelo como entrada y regresa un "cursor"como resultado. Cuando la operación Next se aplica al cursor, este regresa un elemento del mismo modelo de entrada o null si el cursor llega al final de la enumeración.

En [89], E. Rahm y P. A. Bernstein definen el mapping de la siguiente manera: " Definimos un mapping como un conjunto de los elementos de mapping, cada uno de los cuales indica que ciertos elementos del esquema S1 se han mapeado a ciertos elementos en S2. Además, cada elemento del mapping puede tener una expresión de mapping que especifique como se relacionan los elementos S1 y S2".

En [86], R. A. Pottinger y P. A. Berstein presentan el mapping como uno de los elementos fundamentales para realizar entre otras cosas, la fusión (merging) de los modelos. Además, los autores presentan diferentes tipos de relaciones entre los modelos y sus implicaciones, lo que es muy importante para encontrar los elementos equivalentes o similares en la manipulación de modelos.

Los tipos de relaciones se ilustran en la Figura 4.9. Estas relaciones son muy conocidas en el modelado de sistemas. Los tipos de relaciones son [86]:

- Associates: *A(x, y) significa que x está asociada a y (asociación simple en el sentido de UML).*

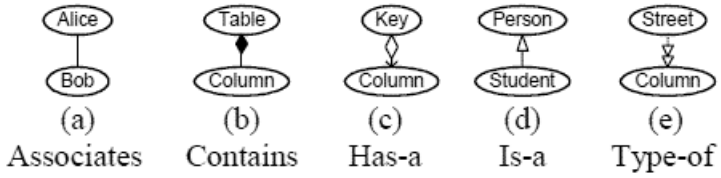

Figura 4.9: Diferentes tipos de relaciones entre modelos [86].

- Contains: *C(x, y) significa que x contiene y (composición simple en el sentido de UML).*
- Has-a: *H(x, y) significa que x tiene un subconjunto y (agregación en el sentido de UML).*
- Is-a: *I(x, y) significa que x es una especialización de y (herencia en el sentido de UML).*
- Type-of: *T(x, y) significa que x es del tipo y (tipeado en el sentido de UML).*

Las implicaciones entre los tipos de relaciones se presentan como sigue [86]:

I. Si T(q, r) e I(r, s) entonces T(q, s)

II. Si I(p, q) y H(q, r) entonces H(p, r)

III. Si I(p, q) y C(q, r) entonces C(p, r)

IV. Si C(p, q) e I(q, r) entonces C(p, r)

V. Si H(p, q) e I(q, r) entonces H(p, r)

La Figura 4.10 ilustra de manera gráfica las implicaciones entre los tipos de relaciones. Esto proporciona una visión de tal manera que los modelos puedan ser manipulados.

Para aplicar estas relaciones y estas implicaciones (con el objetivo de encontrar similaridades o equivalencias entre los modelos), un modelo L puede considerarse como una tripleta (EL, RootL, ReL). EL es el conjunto de los elementos de L, RootL ? EL es el elemento raíz de L, y ReL es el conjunto de las relaciones en L.

R. A. Pottinger y P. A. Berstein definen que dos modelos son equivalentes o similares como sigue:

" Definimos que dos modelos son equivalentes si estos son idénticos después de que sus relaciones implicadas se añaden a cada uno de ellos hasta que se alcanza un punto fijo (es decir, aplicando cada regla que resulta en ninguna nueva relación)" [86].

Y. Velegrakis, R. J. Miller y L. Popa han estudiado la adaptación y la evolución de los mappings entre esquemas cuando estos últimos cambian. Ellos definen el mapping como sigue [103]: " Un mapping especifica cómo las instancias de los datos de un esquema corresponden a las instancias de los datos de otro. Los Mappings se especifican a menudo en una manera declarativa, de forma independiente de los datos (por ejemplo, como consultas o definiciones de vistas). Sin embargo, dependen necesariamente de esquemas que se relacionan".

En el trabajo de Y. Velegrakis, R. J. Miller y L. Popa, el mapping se formaliza como sigue [103]:

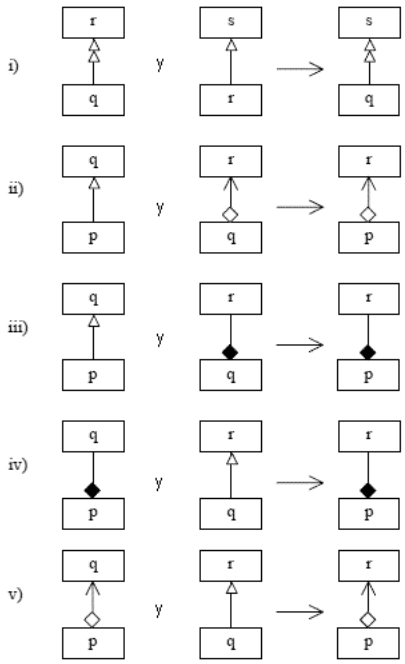

Figura 4.10: Las implicaciones entre los tipos de relaciones.

- Un mapping m de un esquema S (esquema fuente) en esquema T (esquema objetivo) es una aserción de la forma:

$$Q^S \Rightarrow Q^T$$

, donde

$$Q^S$$

es una consulta sobre S y

$$Q^T$$

es consulta sobre T.

- Un sistema de mapping es una tripleta <S, T, M>, donde S es el esquema fuente, T es el esquema objetivo, y M es un conjunto de mappings entre S y T.

- Los tipos de mappings pueden clasificarse en sound, complete y exact. En sound mappings, las consultas regresan un conjunto de n-tuplas, y la relación $\Rightarrow$ es la relación $\subseteq$ (sub-conjunto o igualdad). En complete mappings, las consultas regresan un conjunto de n-tuplas, y la relación $\Rightarrow$ es la relación $\supseteq$ (super-conjunto o igualdad). En exact mappings, la relación $\Rightarrow$ es la relación $=$ (igualdad).

Uno de los problemas con el que se debe enfrentar el mapping (en el sentido de la especificación de correspondencias) es la distancia semántica entre meta-modelos. La noción de distancia semántica existe en varias áreas de la computación [33, 32].

En [33], G. Booch, A. Brown, S. Iyengar, J. Rumbaugh y Bran Selic destacan la importancia de considerer la distancia semántica en la ingeniería de software: " Reduciendo la distancia semántica entre el dominio del problema y la representación nos lleva a un

acoplamiento más directo de las soluciones a los problemas, lo cual proporcionaría diseños más precisos e incrementaría la productividad" [33].

Así, un enfoque MDA y su automatización deberían permitir la reducción de la distancia semántica entre los conceptos de un dominio y las tecnologías de implementación: " Uno de los propósitos primarios de la automatización en MDA es llenar el vacío semántico entre el dominio de conceptos y la tecnología de implementación al modelar explícitamente ambas cosas, el dominio y la tecnología en frameworks y después explotando el conocimiento incorporado en un framework de aplicación particular (...) Quizás la dificultad más grande asociada al desarrollo de software es el enorme boquete semántico que existe entre los conceptos de dominio especifico encontrados en las aplicaciones computacionales modernas, tales como la administración del proceso de negocios o el procesamiento de las llamadas telefónica, y las tecnologías de programación estándar usadas para implementar dicho software".

En [92], O. Sims recalca que el problema de la distancia semántica en la ingeniería de software se resuelve frecuentemente de manera informal y es más bien un asunto de los expertos del dominio (heurísticas): " El hueco a menudo se rellena por todos los desarrolladores de aplicaciones de negocios que aprenden mucho sobre tecnologías de programación, o por algunos tecnólogos expertos del software dentro de un equipo de proyectos o compartido de forma informal a través de los equipos de proyecto" [92].

En [10], P. A. Bernstein recuerda que la distancia semántica entre meta-modelos puede comprenderse por:

- La construcción de meta-modelos más expresivos con la utilización de relaciones tales como is-a, data-types o keys.
- La utilización de expresiones (tal como las consultas queries) en los elementos de mappings.
- La creación de herramientas que permitan agregar la semántica en el mapping con el fin de reducir la distancia semántica entre los meta-modelos.

El objetivo de esta sección es el de mostrar que el problema de la especificación de mapping (en el sentido de la correspondencia) no es un problema nuevo, pero que ya se ha estudiando desde hace un buen tiempo en otras áreas como las bases de datos. En lo que nos interesa, pensamos que la definición de transformaciones debe precederse de una proposición que considere la especificación de correspondencias en el contexto de MDA.

4.3. La transformación de modelos

En esta sección, presentamos los planteamientos propuestos para proporcionar un lenguaje de transformación según las recomendaciones del OMG [69]:

Desde la aparición de MDA, la transformación de modelos ha tomado una mayor atención. De hecho, MDA está basado en la definición de modelos independientes de las plataformas (PIM) que se transforman en modelos específicos de las plataformas (PSM). MOF y UML proporcionan una base bien establecida para la definición de transformaciones para PIM y PSM. En 2002, el OMG inicio un proceso de normalización a través de su RFP MOF 2.0 Query/Views/Transformations (QVT) [69] para estimular y crear un lenguaje de transformación normalizado. Este RFP no está solamente destinado a la

transformación, sino también a la expresión de consultas (Query) y la definición de vistas (view).

En [100], T. Gardner et al., definen los elementos principales del RFP QVT como a continuación se detalla:

- Query (Consulta): Una consulta es una expresión que se evalúa en un modelo. El resultado de una consulta es una o varias instancias de tipos definidos en el modelo fuente, o definidos por el lenguaje de consulta. En el contexto de MDA, OCL (Object Constraint Language) es el lenguaje más utilizado.

- View (Vista): Una vista es un modelo que se deriva completamente de otro modelo (el modelo de base). En el caso donde la vista pueda ser editada, los cambios en la vista deben reproducirse en el modelo de base. El meta-modelo de la vista no es exactamente el mismo meta-modelo fuente. Una vista debe ser completa y tener el mismo contenido de información que la fuente, pero reorganizada para una tarea o un usuario en particular. Una consulta es un tipo restringido de vista. Las vistas se generan mediante las transformaciones.

- Transformation(Transformación): Una transformación genera un modelo objetivo a partir de un modelo fuente. Las transformaciones pueden llevar a modelos dependientes o independientes. En el primer caso, la transformación acopla el modelo fuente con el modelo objetivo. En el segundo caso, no hay relaciones entre el modelo fuente y el modelo objetivo una vez que este se ha generado. Entre los términos utilizados en el RFP QVT, podemos mencionar:

  - Lenguaje declarativo: termino general para un lenguaje relacional o funcional, opuesto a un lenguaje imperativo. Los lenguajes imperativos especifican explícitamente las secuencias y las etapas a seguir para producir un resultado.
  - Lenguaje imperativo: todo lenguaje de programación que especifica la manipulación del estado del sistema computacional.
  - Lenguaje hibrido: combinación de construcciones declarativas e imperativas. Típicamente, el enfoque declarativo se utiliza para seleccionar las reglas y aplicarlas, y el enfoque imperativo se utiliza para implementar los detalles de las reglas que no se expresan completamente de manera declarativa.

- Reglas: unidades en las cuales se definen las transformaciones. Una regla es responsable de la transformación de una selección especifica del modelo fuente en elementos correspondientes del modelo objetivo. Una transformación se especifica por un conjunto de reglas.

- Declaración: especificación de una relación entre los elementos del modelo de izquierda y aquellos del modelo de derecha.

- Implementación: especificación detallada que precisa como crear los elementos objetivo a partir de los elementos fuente.

- Match: se produce durante la aplicación de una transformación, más precisamente cuando los elementos del modelo de izquierda y del modelo de derecha se empatan en las especificaciones de restricciones proporcionadas por una regla.

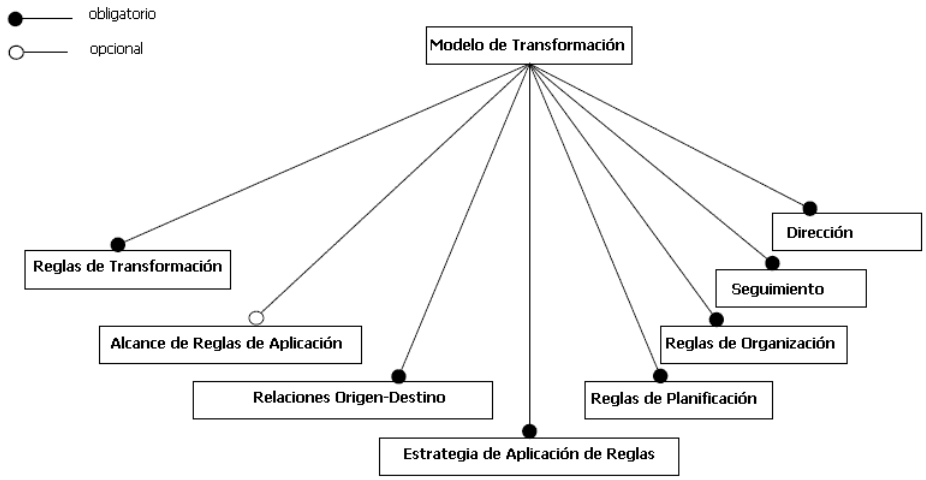

Figura 4.11: El diagrama de características de lenguajes de transformación de modelos [24].

Después de la divulgación del RFP QVT, se propusieron varios lenguajes de transformación en la literatura [41, 57, 82] o se sometieron al OMG.

Estos lenguajes presentan diferentes enfoques para resolver el problema de definición de transformación de modelos. En [24], K. Czarnecki y S. Helsen proponen una taxonomía basada en las características presentes en la mayoría de las proposiciones de lenguajes de transformación. La Figura 4.11 contiene un diagrama de características que define esta taxonomía para los lenguajes de transformación de modelos.

Según la Figura 4.11, los lenguajes de transformación pueden distinguirse en función de varios parámetros:

- Reglas de transformación: una regla de transformación tiene obligatoriamente dos partes distintas, un LHS (Left Hand Side) y un RHS (Right Hand Side). El LHS permite acceder al modelo fuente, y el RHS permite acceder al modelo objetivo. Los LHS y RHS pueden ser una combinación de:
  - Variables: las variables se utilizan para manipular los elementos del modelo fuente y/o objetivo. Las variables pueden no tener tipos, sintácticamente tipeadas o semánticamente tipeadas.
  - Patrones: los patrones pueden ser clasificados siguiendo la forma, la sintaxis y el tipo. La forma de un patrón puede ser strings, términos o grafos. Los patrones string se utilizan en modelos textuales (textual templates). Los términos y los grafos se utilizan en la tranformación modelo-hacia-modelo. En cuanto a la sintaxis, esta puede ser abstracta o concreta. Un patrón puede no tener tipos, sintácticamente con tipos o semánticamente con tipos.
  - Lógica: la lógica puede ser ejecutable o no ejecutable. La lógica no ejecutable se utiliza para especificar una relación entre modelos. La ejecutable puede ser imperativa o declarativa. La forma declarativa comprende las consultas OCL para la recuperación de los elementos de un modelo y la creación implícita de los elementos objetivo por medio de las restricciones. La forma imperativa

contiene un tipo de lenguaje de programación que utiliza una API para manipular los modelos directamente, por ejemplo JMI (Java Metadata Interface) [87] proporciona una API para acceder a un deposito MOF (repository MOF).

- Además, una regla de transformación puede tener las siguientes características:
  - Separación sintáctica: la sintaxis de una regla puede realizar la distinción explicita entre un RHS y un LHS o esta distinción puede no efectuarse.
  - Bidirecionalidad: una regla puede ser ejecutable en una sola dirección o en las dos direcciones (bidireccional). La regla bidireccional es más compleja y difícil de implementar.
  - Parametrización: las reglas de transformación pueden tener parámetros de control adicionales para permitir la configuración o la adaptación.
  - Estructuras intermedias: algunos enfoques demandan la construcción de estructuras de modelos intermedios.
- Alcance de la aplicación de la regla: permite a una transformación restringir las partes de un modelo que participa en la transformación.
  - Fuente: en este caso, podemos establecer un alcance menor que el del modelo fuente completo.
  - Objetivo: en este caso, podemos aumentar el alcance del modelo objetivo.
- Relación fuente-objetivo: varios enfoques imponen la creación de un nuevo modelo objetivo que debe separarse del modelo objetivo. Otros enfoques imponen que el modelo fuente y el modelo objeto sean el mismo modelo (estos enfoques permiten únicamente la actualización en el mismo sitio).
- Estrategia de aplicación de la regla: una regla debe aplicarse desde una localización específica en el interior del alcance del modelo fuente. En el caso donde esta esté activa desde una localización, la estrategia de la aplicación de la regla debe especificarse. Esta puede ser:
  - Determinista: la estrategia sigue un protocolo determinado y conocido desde el inicio.
  - No determinista: una localización puede elegirse o varias se pueden aplicar de manera concurrente.
  - Interactiva: el usuario debe decir cual localización debe aplicarse a la regla.
- Ordenamiento de una regla: determina el orden en el cual las reglas individuales se aplican. El mecanismo de ordenamiento puede clasificarse en cuatro grupos:
  - Forma: el aspecto del ordenamiento puede ser implícito y explicito. El ordenamiento implícito implica que el usuario no tenga un control explicito en el algoritmo de ordenamiento. El ordenamiento explicito proporciona construcciones dedicadas al control del orden de ejecución.
  - Selección de regla: las reglas pueden seleccionarse por una condición específica, o elegirse de una manera no determinista, o seleccionadas mediante un mecanismo de prioridad, o incluso seleccionadas de manera interactiva.

  - Iteración de regla: la recursividad, el ciclo y el punto fijo se encuentran entre los mecanismos de interacción de reglas.
  - Phasing: este aspecto se refiere a la organización del proceso de transformación en varias etapas.

- Organización de reglas: determina la composición y la estructuración de varias reglas de transformación. Se puede clasificar en:
  - Mecanismo de modularidad: permite la organización de reglas en módulos. La creación de módulos permite a un módulo importar los elementos de otro modelo.
  - Mecanismo de reutilización: proporciona una manera de definir una regla basada en una o en varias reglas. Entre los mecanismos de reutilización, podemos citar: composición lógica, herencia entre reglas, derivación, extensión, especialización, herencia entre módulos.
  - Estructura de organización: las reglas pueden organizarse en conformidad con la estructura del meta-modelo fuente, o del meta-modelo objetivo, o aún con una organización específica.

- Trazabilidad: las transformaciones pueden registrar los vínculos entre los elementos fuente y objetivo. Estos vínculos pueden ser útiles para el análisis del desempeño, la sincronización entre modelos, los depuradores orientados al modelo, y la determinación del objetivo de una transformación.

- Direccionalidad: indica la dirección de una transformación, y puede ser:
  - Unidireccional: la transformación se ejecuta en una sola dirección, en la cual un modelo objetivo se deduce a partir de un modelo fuente.
  - Bidireccional: la transformación puede ejecutarse en las dos direcciones. Esta proporciona igualmente el soporte necesario para la sincronización entre modelos.

En la creación de esta taxonomía, K. Czarnecki y S. Helsen consideraron los enfoques modelo-hacia-modelo y modelo-hacia-código de manera uniforme. El primer enfoque está aún en evolución y en experimentación mientras que el último está más difundido y se ha utilizado ampliamente. Sin embargo, estos presentan varias similaridades. En general, la transformación de modelo hacia el código puede verse como una transformación modelo hacia modelo, puesto que el código es también de hecho un modelo. Sin embargo, por razones prácticas de reutilización de la tecnología de compiladores existentes, el código es frecuentemente generado como texto simple, que puede ser compilado.

El enfoque modelo-hacia-código puede ser:

- Orientado al visitante: este enfoque es el más básico y consiste en proporcionar un mecanismo de visitante para liberarse de la representación interna de un modelo y escribir el código como un texto.

- Orientado al modelo: un template consiste habitualmente en un texto orientado que contiene meta-código para recuperar la información de un modelo fuente y utilizarla para rellenar el texto objetivo.

El enfoque modelo-hacia-modelo puede ser:

- Manipulación directa: este enfoque proporciona una representación interna de la representación del modelo así como una API para manipularla. Se realiza habitualmente como un framework orientado a objetos.
- Relacional: esta categoría agrupa los enfoques declarativos en los cuales las relaciones matemáticas constituyen el concepto principal. La idea subyacente es establecer un tipo de relación entre un elemento fuente y un objetivo y especificarla utilizando restricciones.
- Orientada a la transformación de grafos: esta categoría utiliza la teoría de grafos.
- Orientada a la estructura: en esta categoría, se pueden distinguir dos etapas: una se refiere a la creación de la arquitectura jerárquica del modelo objetivo, y la otra al establecimiento de los valores de los atributos y de las referencias en el modelo objetivo.
- Hibrido: es una composición de los enfoques declarativos e imperativos.
- Enfoques específicos: en la especificación del CWM (Common Warehouse Metamodel) del OMG, se propuso un framework de transformación. Este framework proporciona un mecanismo para vincular los elementos fuente y objetivo, pero la derivación de los elementos objetivo debe realizarse en un lenguaje concreto, el cual no está definido por el CWM. Otro enfoque significativo es XSLT. Desde que los modelos pueden serializarse a la manera XML (por ejemplo, XMI), XSLT puede utilizarse para realizar la transformación. Sin embargo, este enfoque no está adaptado a la transformación de modelos ya que este es fuente frecuente de errores en la definición, presenta una cantidad importante de verbos, y ofrece una visibilidad reducida debido a la débil compacidad de XML.

4.4. Aplicación de MDA para la plataforma de los servicios Web

La aplicación de MDA para crear modelos dependientes de plataformas a partir de modelos independientes de plataformas no es una tarea simple. Hay preguntas aún sin respuesta, pero la aparición de varios lenguajes de transformación [41, 91] ha servido para validar en parte el enfoque MDA. Sin embargo, muchos investigadores perciben que hace falta una teoría bien establecida que soporte a MDA.

Para responder a varias preguntas aún abiertas, la experimentación de MDA con varios lenguajes de modelización y plataformas objetivo debe permitirnos aprender más sobre estos problemas existentes y proponer soluciones viables. Entre los lenguajes de modelización, EDOC y UML son los más conocidos y más adaptados para modelar los sistemas distribuidos.

Los servicios Web son una plataforma muy importante en la actualidad. Es la plataforma objetivo que parece ser la más estudiada y la más deseada para construir las nuevas aplicaciones Internet y para permitir a las aplicaciones del pasado (legacy systems) ser adaptados a la Internet.

Presentamos en el párrafo 4.4.1, los enfoques que privilegian los aspectos estáticos de servicios Web (estructura) y, en el párrafo 4.4.2 , discutiremos los enfoques que privilegian sus aspectos dinámicos (comportamiento).

4.4.1. Aspectos estáticos

D. Frankel y J. Parodi son de los primeros investigadores en discutir el enfoque MDA para desarrollar aplicaciones que tienen a los servicios web como plataforma objetivo [22]. Ellos proponen en particular: Una arquitectura four-tiers para los sistemas distribuidos en la cual, los servicios Web tienen un rol fundamental para las aplicaciones orientadas a Internet.

La creación de modelos formales para soportar el enfoque MDA.

Un estudio para la correspondencia entre un modelo de negocios en UML y los servicios Web.

La Figura 4.12 presenta la arquitectura four-tiers propuesta por D. Frankel y J. Parodi.

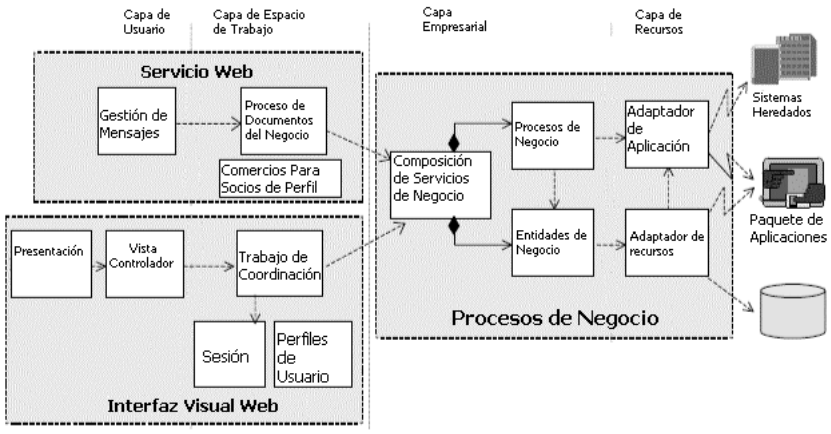

Figura 4.12: La arquitectura four-tiers [22].

Según la Figura 4.12, el Resource Tier es responsable de la adaptación de la lógica de negocios (Business Process) en los sistemas heredados (legacy systems) y de la adaptación de los datos (Resource Adapter) en las bases de datos. El Entreprise Tier contiene los servicios de negocio bajo la forma de servicios Web compuestos de asociaciones con los procesos de negocio (Business Process) y los datos (Business Entity). El Workspace Tier puede organizarse de dos maneras: como un tratamiento de documentos (Business Document Processing) y como un coordinador de trabajo (Work Coordinator). El User Tier puede organizarse también de dos maneras: como un manipulador de mensajes (Message Handling) y como un browser (Presentation + View Controller). Por ejemplo, un navegador Web que toma el control y la presentación de los datos (es decir, interacción con el usuario). En [59], se propuso otra arquitectura fourt-tiers muy similar a la de la Figura 4.12. La Figura 4.13 presenta (a) un modelo UML impreciso e incompleto y (b) otro modelo UML más preciso y completo. Esta Figuratrata de ilustrar que un enfoque MDA puede ser exitoso, sí y solo si, los modelos independientes de la plataforma son también precisos y completos para transformarse en un modelo dependiente de la plataforma. Esto no significa que el modelo deba tener todos los conceptos de la plataforma objetivo, pero si debe tener una descripción abstracta de la solución de problema. Por consecuencia, la descripción de la lógica del negocio debe contener la estructura y la semántica de la resolución del problema.

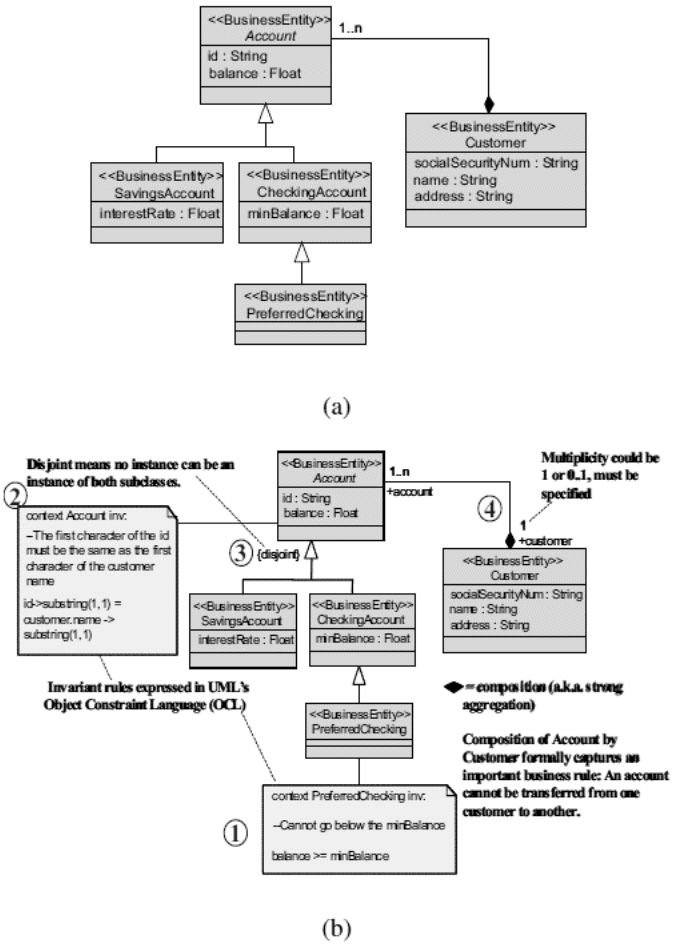

Figura 4.13: a: un modelo de negocios impreciso e incompleto; b: un modelo de negocio más preciso y completo [22].

Por ejemplo, la Figura 4.13(a) no contiene información precisa sobre las cardinalidades y restricciones. Esto complica la realización de este modelo en un lenguaje como Java y demanda al programador de estar siempre en contacto con el diseñador. Por el contrario, la Figura 4.13(b) es más precisa: permite conocer más sobre la lógica de negocio de resolución del problema y demanda menos interacción entre el programador y el diseñador. Por ejemplo, un modelo puede ser preciso por medio del uso de Action Semantics de UML.

La Figura 4.14 presenta un fragmento de correspondencia entre un modelo UML y WSDL. Una clase estereotipada BusinessService en UML corresponde al PortType en WSDL. Un método en UML corresponde a una Operation en WSDL. un parámetro de entrada o de salida en UML corresponde a un Message de entrada o de salida en WSDL.

Se han propuesto igualmente otros trabajos para estudiar la creación de meta-modelos para las plataformas objetivo como los servicios Web. Entre estos trabajos, B. Bordbar y A. Staikpoulos [12] estudian la creación de un meta-modelo lo más preciso posible

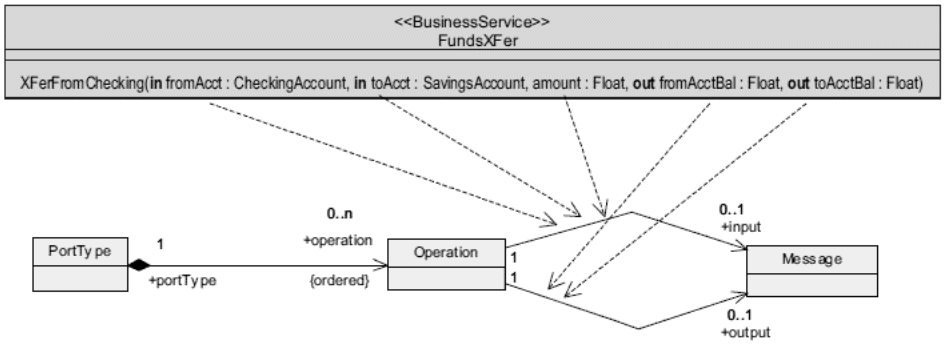

Figura 4.14: Una correspondencia entre un modelo de negocios en UML y WSDL [22].

para los servicios Web, a partir de esquemas o de otras especificaciones existentes. Por ejemplo, ellos proponen un meta-modelo para WSDL a partir de su esquema [109]. La Figura 4.15 presenta este meta-modelo generado a partir del esquema de WSDL. Este meta-modelo contiene Definitions que tienen una referencia para cero o más elementos anyTopLevelOperationalElement (con el estereotipo <XSDgroup>). Esto contiene los elementos Import, Message, PortType, Binding y Service. El elemento Message contiene los elementos Part. El elemento PortType contiene los elementos operation. El elemento Binding contiene los elementos BindingOperation. El elemento Service contiene los elementos Port. En este meta-modelo, los diferentes tipos de operaciones se modelan también como "request-response"y "solicit-response".

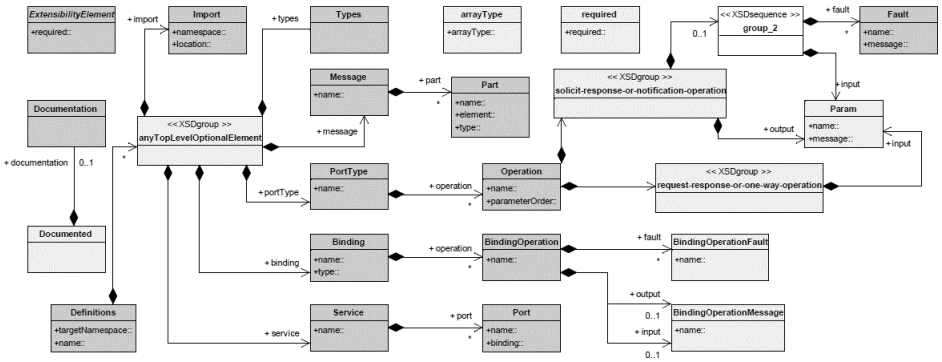

Figura 4.15: Un meta-modelo para WSDL propuesto en [12].

Entre los trabajos de O. Patrascoiu, una validación del lenguaje YATL se realiza por la aplicación de MDA a la plataforma de servicios Web [55]. El autor parte de un modelo conforme a EDOC y lo desarrolla en la plataforma de servicios Web. Además proporciona un meta-modelo para WSDL (figura 4.15), la transformación de EDOC-CCA en WSDL, y utiliza un caso de estudio para demostrar la validez de YATL.

El meta-modelo de WSDL de la Figura 4.15 y el de la Figura 4.16 presentan varias diferencias. Por ejemplo, en el primer meta-modelo, el elemento Message contiene elementos Part, mientras que en el segundo meta-modelo, el elemento Message tiene una referencia para los elementos Part. Una diferencia más importante puede verificarse en

el elemento Operation. En el primer meta-modelo, el elemento Operation contiene los elementos para cada tipo de operación, por ejemplo "oneway"y "request-response". En el segundo meta-modelo, el elemento Operation contiene los elementos Input, Output y Fault, no explicita el tipo de operación.

Una correspondencia entre WSDL y EDOC propuesta en [81] se puede resumir en la siguiente tabla:

EDOC	**WSDL**
Datatype	XML Schema SimpleType
ComplexType	XML Schema ComplexType
Attribute	XML Schema Attribute
CCA FlowPort	Message
CCA OperationPort	Operation
CCA ProtocolPort	PortType
CCA ProccesComponent	Service

Cuadro 4.1: Una correspondencia entre EDOC y WSDL (fragmento).

Dadas las correspondencias (ver Cuadro 4.1) entre el meta-modelo de EDOC y WSDL, el Cuadro 4.1 presenta los fragmentos de la definición de transformación en YATL propuesto por O. Patrascoiu.

Para validar su lenguaje de transformación, O. Patrascoiu se baso en el caso de estudio de la agencia de viajes. La Figura 4.17 presenta la agencia de viajes modelada en EDOC. Además, el autor comenta la utilización de reglas de transformación en YATL para obtener el WSDL correspondiente al modelo de la agencia de viajes en EDOC.

4.4.2. Aspectos dinámicos

Otros estudios han ido más lejos e igualmente han considerado los aspectos dinámicos para crear modelos independientes de la plataforma y desarrollarlos en servicios Web.

R. Gronmo et al., proponen una metodología para desarrollar los servicios Web y una correspondencia entre un modelo UML y WSDL [34]. La Figura 4.18 ilustra la metodología para el desarrollo de servicios Web dirigido por los modelos.

Esta metodología puede organizarse siguiendo estas etapas:

Discover Web Services: El desarrollador utiliza un navegador Web y un cliente de anuario para buscar un servicio Web candidato para la composición. Se obtiene una lista de servicios Web y las localizaciones de su descripción (representada en WSDL).

Import Web Service Descriptions: El desarrollador importa la descripción del servicio y la traduce en UML. El resultado es un modelo UML conforme a WSDL.

Model composite Web Service: El desarrollador utiliza una herramienta UML para revisar e integrar los modelos importados bajo la forma de modelo de un servicio compuesto. Esta etapa se compone de la modelización de la estructura del servicio (Service modeling) y de la modelización del comportamiento del servicio compuesto (Workflow modeling).

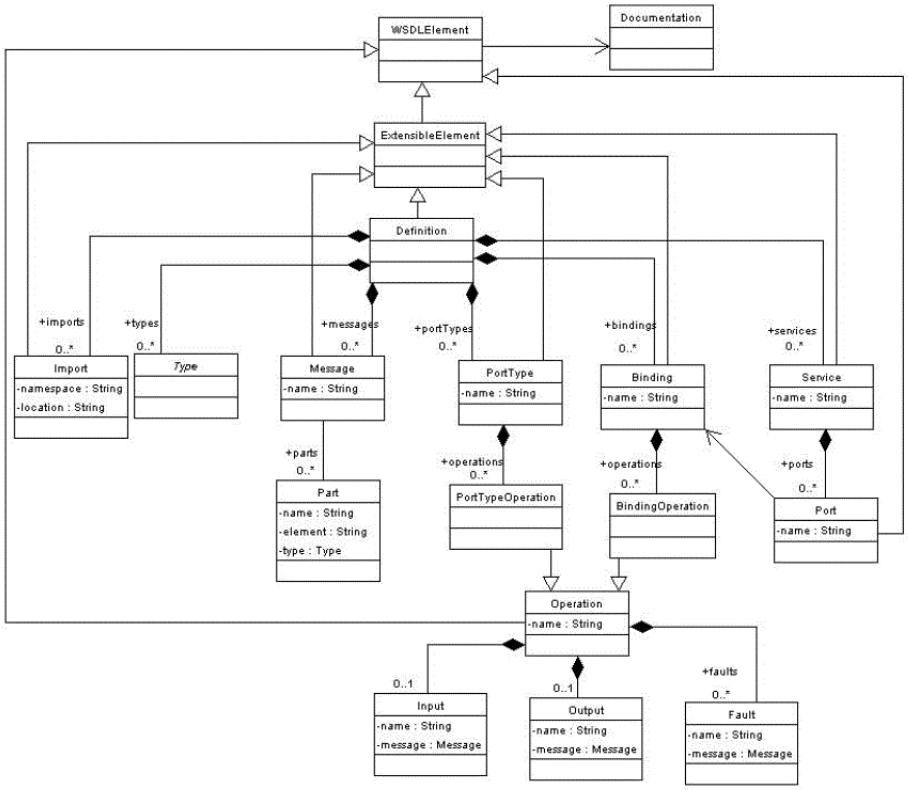

Figura 4.16: Un meta-modelo de WSDL propuesto en [81].

Export Web Services Description: El modelo del servicio Web compuesto se exporta con el fin de obtener un documento WSDL del servicio Web compuesto. Implement Composite Web Service: El servicio Web se realiza a partir de un documento WSDL.

Publish Composite Web Service: Finalmente, el servicio Web compuesto se publica en un anuario.

La Figura 4.19 presenta un modelo y su equivalente WSDL. Según esta Figura, una Class en UML es el equivalente de Types en WSDL. Una Interface en UML es el equivalente de PortType y Binding en WSDL. Un método UML es el equivalente de Operation en WSDL. Una clase estereotipada por Business Services es el equivalente de Service en WSDL. A continuación, R. Gronmo et al., utilizan UMT (UML Transformation Tool) para crear las reglas de transformación de UML en WSDL [93].

D. Skogan, R. Gronmo e I. Solheim proponen los diagramas de actividades de UML para realizar la composición de servicios. El servicio compuesto enseguida se exporta como un documento WorkSCo o BPEL4WS [94]. La Figura 4.20 presenta la conversión de UML en BPEL4WS. Según esta Figura, una actividad estereotipada WebServiceCall está asociada a los elementos Invoke y a PartnerLink en BPEL4WS. El objeto de entrada plumeInfo se transforma en Variable en BPEL4WS. El objeto de salida plumeLayer también se transforma en Variable. Varios valores marcados en las actividades (service, portType y operation) son referencias de definiciones de documentos WSDL, y corresponden a los atributos de Invoke en BPEL4WS. El flujo de secuencia de las actividades corresponde a Sequence en BPEL4WS. Las transformaciones de datos entre plumeLayer

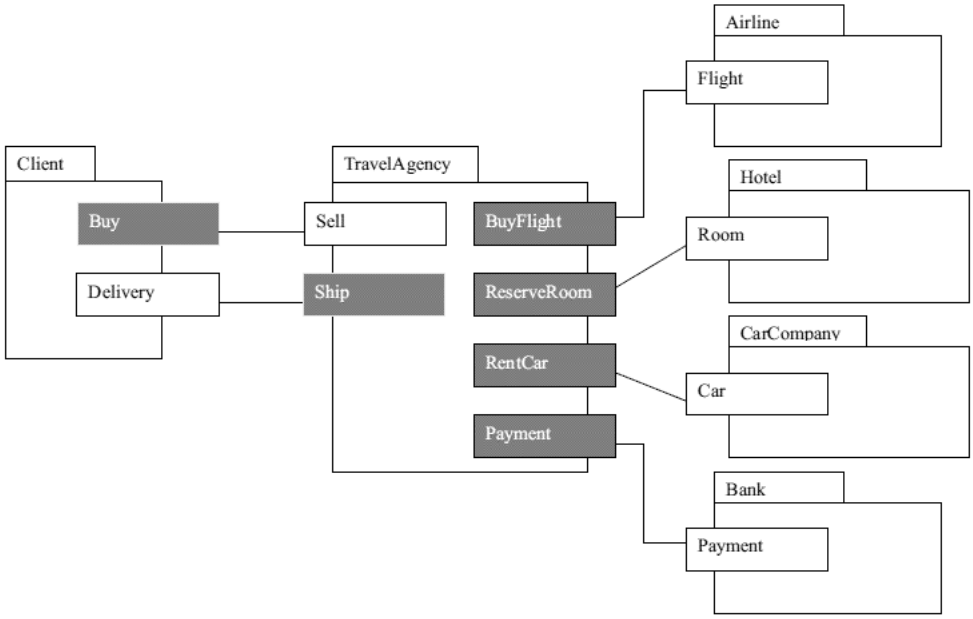

Figura 4.17: El estudio de la agencia de viajes en EDOC se propone en [81].

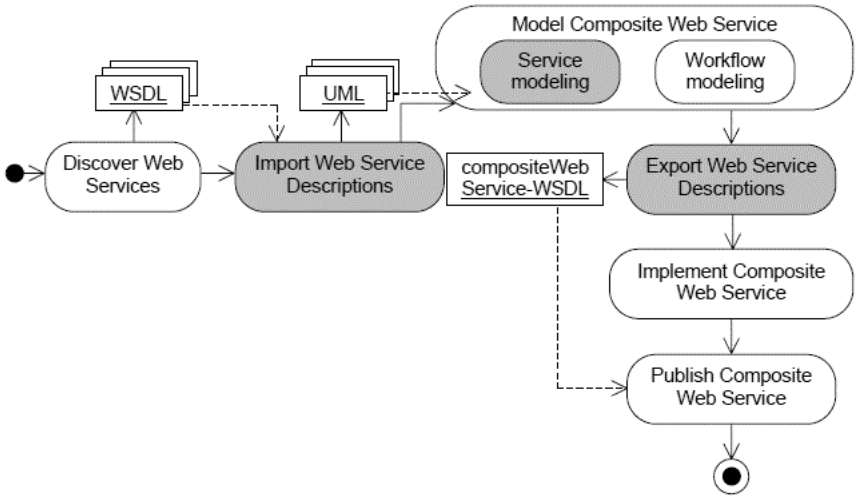

Figura 4.18: Las etapas de desarrollo de servicios Web dirigidos por los modelos: metodología [34].

y plume se efectúan en BPEL4WS por medio de la instrucción Assign.

B. Bordbar y A. Staikopoulos también estudiaron los aspectos dinámicos de los diagramas de actividades de UML y de los servicios Web, en particular la correspondencia entre los diagramas de actividades de UML y WSCI, y entre los diagramas de actividades de UML y BPEL4WS [11].

En [116], B. Bordbar y A. Staikopoulos presentan otro caso de estudio y su realización con los servicios Web. Los autores proponen un meta-modelo para BPEL4WS (ver Figu-

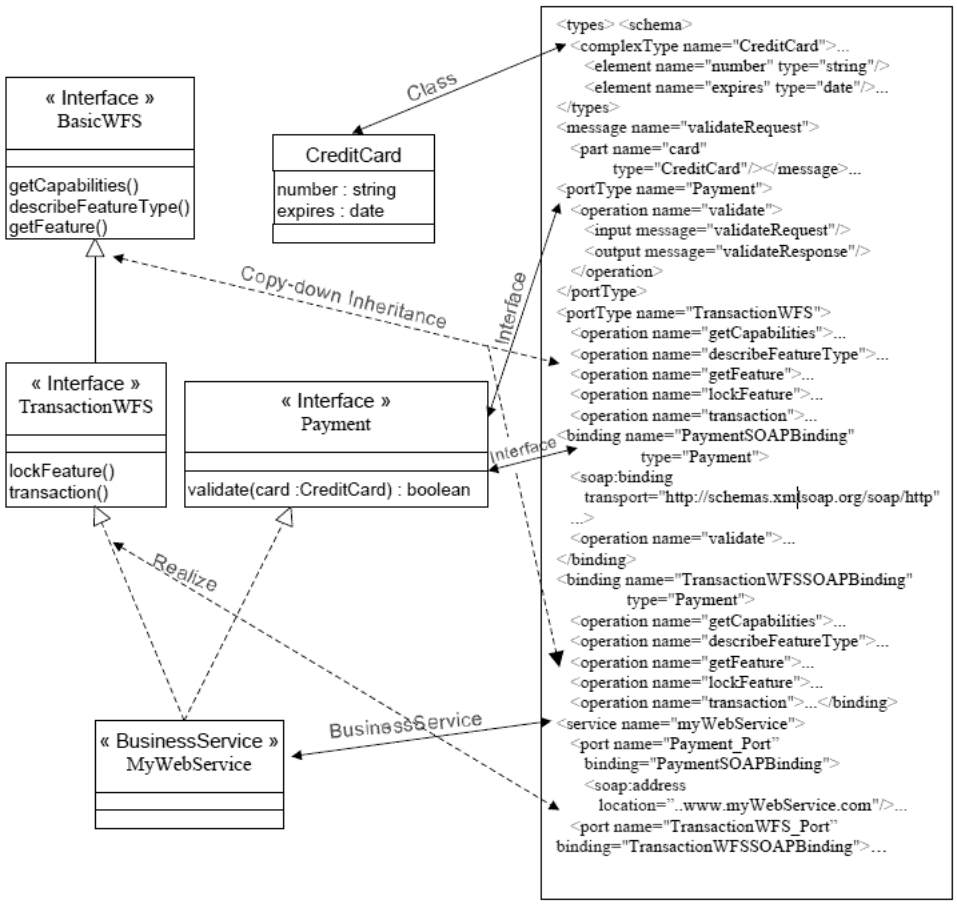

Figura 4.19: Una conversión entre un modelo UML y un documento WSDL [93].

ra 4.21), especifican también la correspondencia entre el diagrama de actividad de UML y BPEL4WS (ver Cuadro 4.2).

En el meta-modelo de la Figura 4.21, el elemento Process contiene otros elementos como Activity, Variable, Partner, PartnerLink y Role. Entre los diferentes tipos de acciones, podemos mencionar las secuencias (Sequence), ciclos (while) y llamadas (Invoque).

Kath et al., proponen una infraestructura para implementar el enfoque MDA y presentan experimentos con EDOC y los servicios Web [46]. La infraestructura propuesta se constituye de diferentes herramientas que se ensamblan para soportar el enfoque MDA. En su enfoque, los autores presentan un meta-modelo para los servicios Web (WSDL), una correspondencia entre EDOC y WSDL, un meta-modelo para BPEL4WS y una correspondencia entre EDOC BPEL4WS. La Figura 4.22 presenta el meta-modelo de WSDL. El Cuadro 4.2 ilustra una correspondencia entre UML y BPEL4WS.

La Figura 4.23 presenta la correspondencia entre EDOC y WSDL. según la Figura 4.23, ProcessComponent en EDOC-CCA es el equivalente de Service en WSDL. El Protocol-Port en EDOC-CCA es el equivalente de Port en WSDL. El Protocol/Inter-

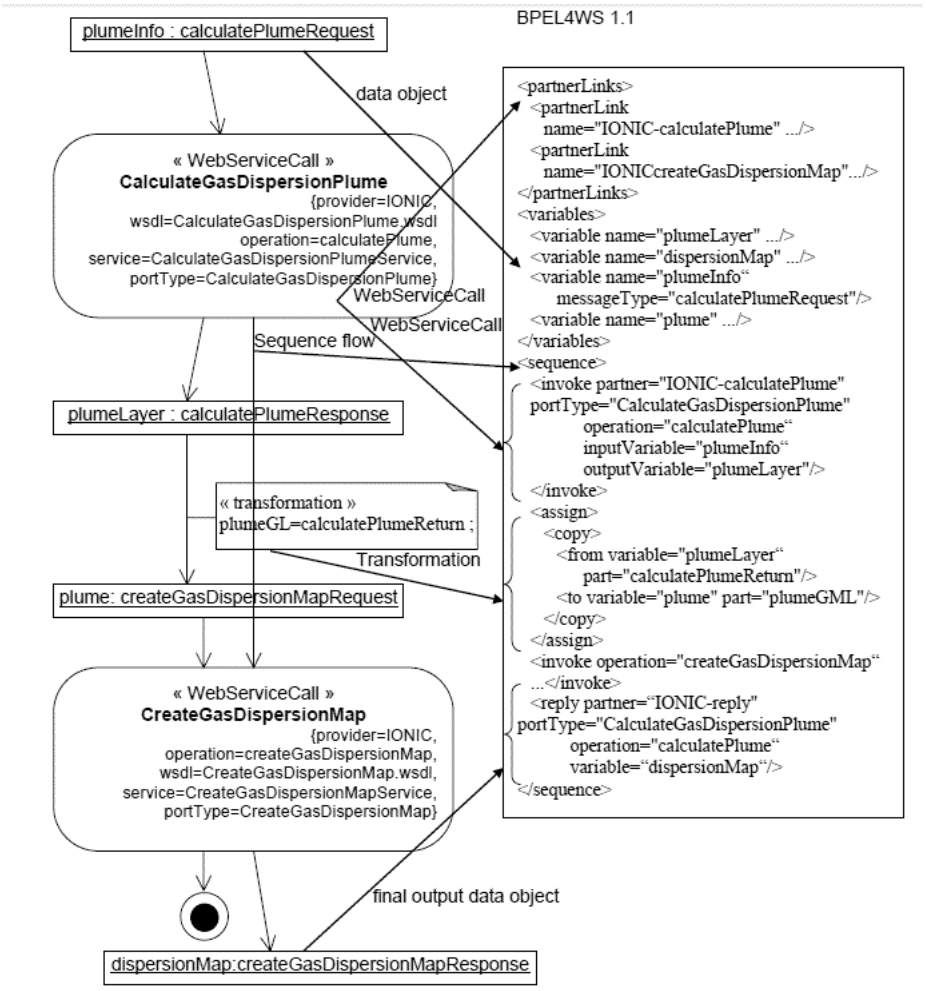

Figura 4.20: Conversión de UML en BPEL4WS propuesta en [94].

face en EDOC-CCA es el equivalente de Binding, BindingOperation y PortType. La OperationPort es el equivalente de Operation.

En [46], los autores especifican las correspondencias entre EDOC y BPEL4WS como a continuación se detalla: Una interfaz proporcionada para un ProcessComponent de EDOC corresponde a PaternLink.

Los parámetros in y out (FlowPorts) de las Operation-Ports de una Interface corresponden a las variables en BPEL. Los puertos de comunicación de EDOC corresponden a las activities en BPEL. Para cada elemento de una coreografía de EDOC un elemento invoke (reply o receive) se genera en BPEL.

K. Baina et al., proponen un estudio para el desarrollo de servicios Web dirigido por los modelos [45], pero fuera del contexto del enfoque MDA. Los autores proporcionan un framework que genera automáticamente los modelos de composición BPEL4WS a partir de las especificaciones de protocolos basados en la descripción de servicio y sobre el modelo de composición (una variación de stakcharts) de Self-Serv [13]. Así, este modelo de composición enfatiza los estados y las transiciones. La Figura 4.24 presenta un ejemplo

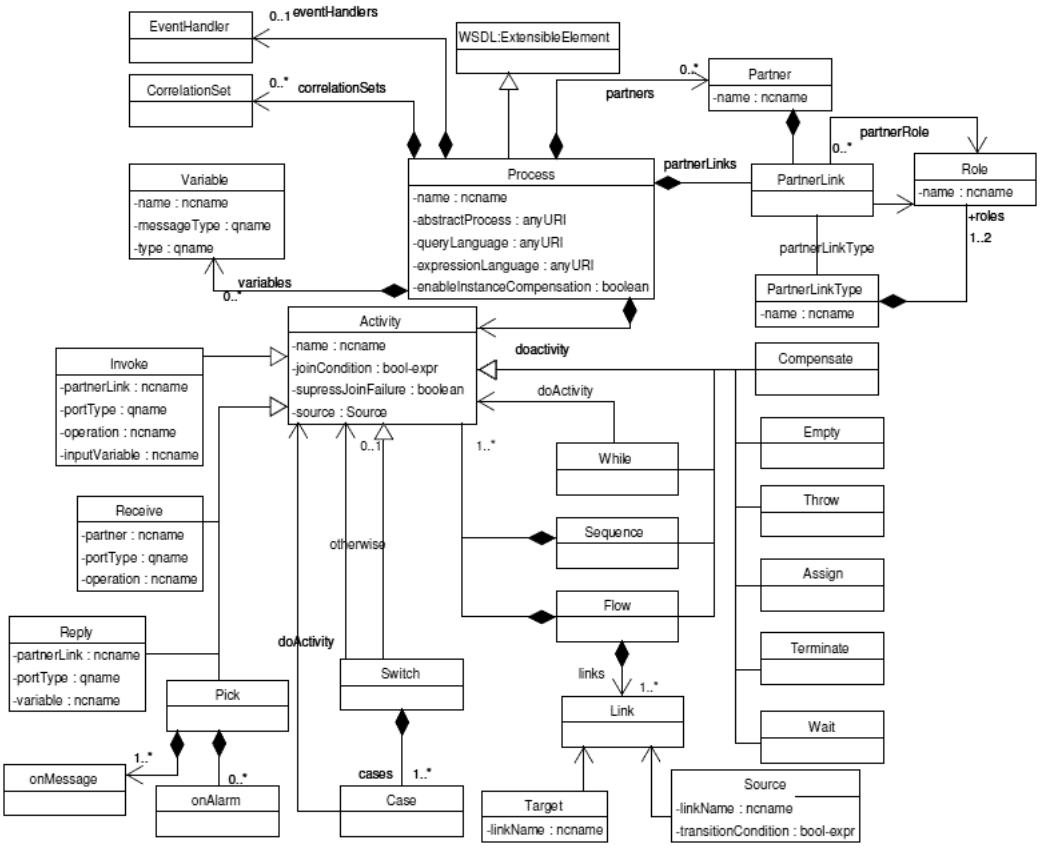

Figura 4.21: Fragmento del meta-moelo de BPEL4WS propuesto en [116].

de utilización de este modelo de composición basado en la conversación.

La Figura 4.25 ilustra la metodología propuesta por K. Baina et al., según esta Figura, su metodología se articula alrededor de tres fases de transformación: inicialmente, cada transición se transforma en un esqueleto de proceso; enseguida, cada estado se transforma en un esqueleto de estado agregando la transición subyacente al estado; finalmente, una vez que los esqueletos individuales de las transiciones y de los estados se han realizado, los esqueletos de estados se vinculan en conjunto en un esqueleto de proceso general según la topología del grafo statechart.

La Cuadro 4.26 presenta las reglas de generación del modelo de composición en BPEL4WS a partir de la especificación del protocolo. Como lo muestra esta tabla, los autores distinguen las operaciones en cuatro categorías: internal, singleton, serial y parallel. Para cada una de estas operaciones, los autores definen los elementos de BPEL4WS correspondientes. Así, si una operación de una interfaz op asociada a una transición t es una operación del tipo singleton, entonces el esqueleto de BPEL4WS correspondiente incluirá una actividad <invoke ../>para llamar a la operación opj del servicio Si. Si la transición t es compensable (compensable transition), la operación de compensación de la operación op, denominada opc, puede inferirse automáticamente, pero solamente si Si.opj es compensable en conformidad con el protocolo de conversación del servicio Si. En este caso, el <compensationHandler>del esqueleto de BPEL4WS de t incluirá una actividad

UML 2.0	**BPEL4WS 1.1**
Class comme BehavioredClassifier, Activity, StructuredActivityNode	<process >
Datastore, StructureNode Variable, ObjectNode, Class Attributes	<variable >
ControlFlow	<sequence >
AcceptCallAction	<receive >
ReadVariableAction, WriteVariableAction	<assign >
Variable, ObjectNode, OutputPin	<from >
Variable, ObjectNode, InputPin	<to >
CallOperationAction	<invoke >
ReplyAction	<reply >

Cuadro 4.2: Una correspondencia entre UML y BPEL4WS [116].

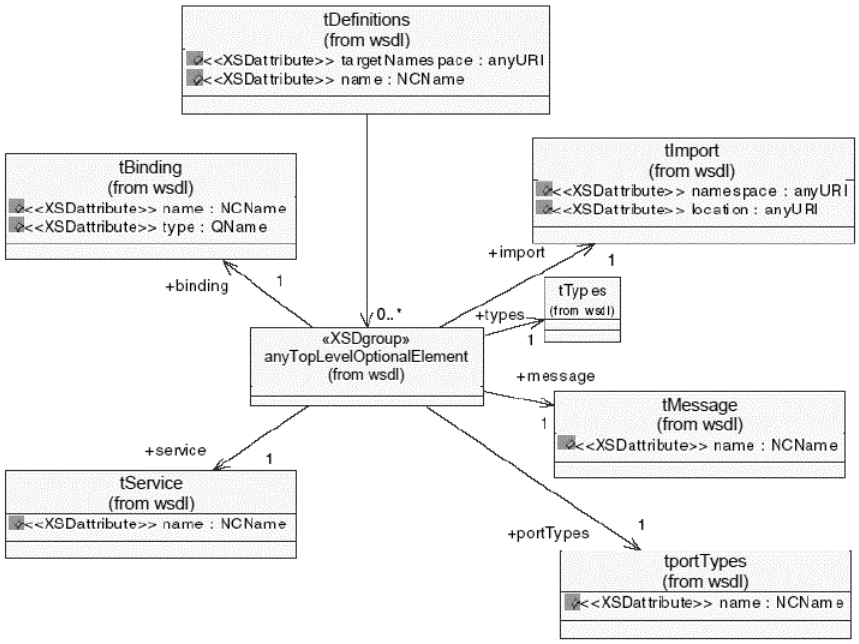

Figura 4.22: Un meta-modelo para WSDL [46].

<invoke ../>para llamar a la operación de compensación opj. En el caso de una operación serial, el equivalente en BPEL4WS incluirá una actividad <sequence>(<sequence> </sequence>). En el caso de una operación paralela, el equivalente en BPEL4WS incluirá una actividad flow(<flow></flow>).

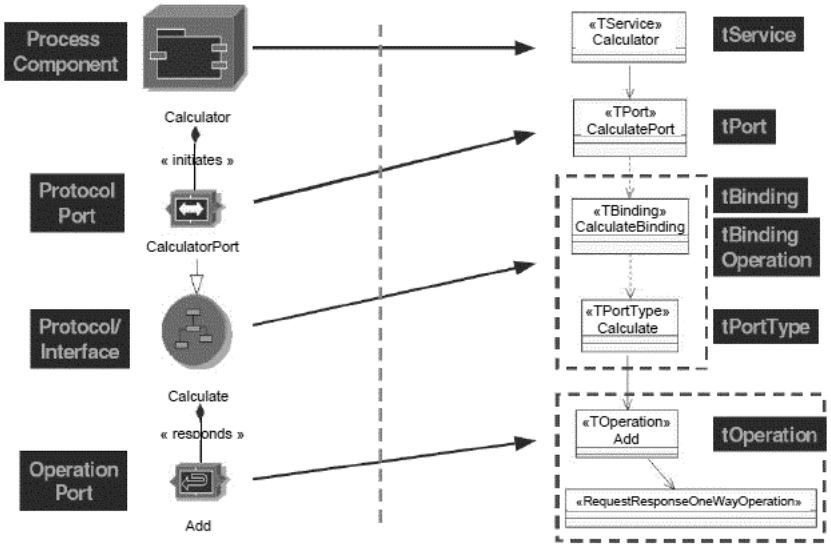

Figura 4.23: Una correspondencia entre EDOC y WSDL [46].

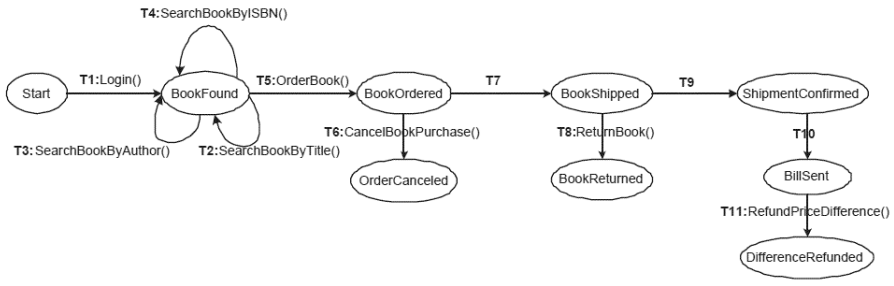

Figura 4.24: Un ejemplo de modelo de protocolo de conversación de servicio: EbookShop [45].

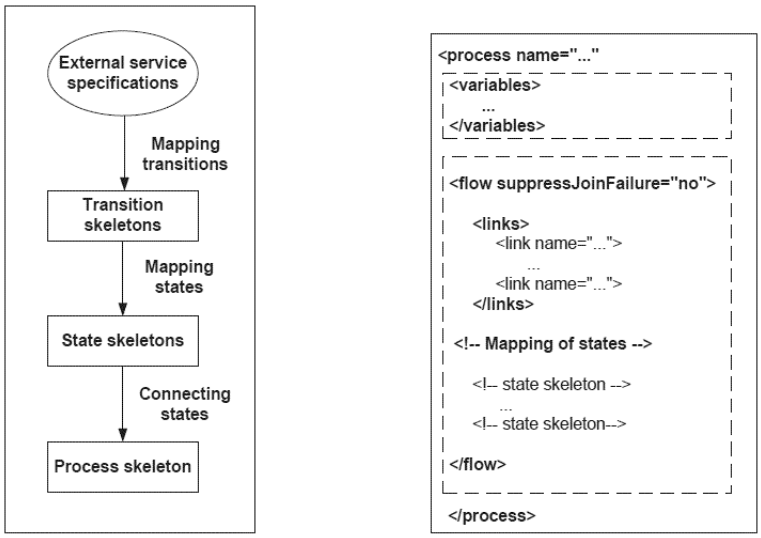

Figura 4.25: Un proceso de generación de esqueletos [45].

Entidad	**Metodo**	**Elemento de BPEL**
Internal	invoke	<!- - empty implementation logic- ->
	compensation	<!- - empty compensation logic - ->
Singleton $S_i.op_j$	invoke	<invoke partner="Si" operation="opj"../>
	compensation	op_j is related to a `Definite` or `Effect-less` transition in S_i protocol <!- - empty compensation logic - ->
		op_j is related to a $Compensable$ transition in S_i protocol <invoke partner="Si" operation="opjc" ./>
Serial $OP_j;OP_k$	invoke	<sequence> <!- - generating operation implementation logic of OPj - -> <!- - generating operation implementation logic of OPk - -> </sequence>
	compensation	<sequence> <!- - generating OPk compensation logic - -> <!- - generating OPj compensation logic - -> </sequence>
Parallel $OP_j//OP_k$	invoke	<flow> <!- - generating operation implementation logic of OPj - -> <!- - generating operation implementation logic of OPk - -> </flow>
	compensation	<flow> <!- - generating OPj compensation logic - -> <!- - generating OPk compensation logic - -> </flow>

Figura 4.26: Las reglas para generar un modelo de composición en BPEL presentadas en [45].

Capítulo 5

Aplicación de MDA en redes sociales

5.1. Enfoque MDA para la plataforma servicios Web: modelización

Este capítulo presenta el un enfoque MDA para la plataforma Servicios Web y concierne la parte dedicada a la modelización. Este enfoque se utilizara para la creación de la red social `http://anonymousremark.com` . Con lo cual se plantea utilizar MDA desde diferentes plataformas.

Se eligió el lenguaje de modelización UML para modelar la red social. De igual manera, se presenta un modelo independiente de la plataforma (PIM) creado en UML.

La plataforma de los servicios Web se eligió para crear un primer modelo dependiente de la plataforma (PSM). Además, se adopto J2EE (más precisamente, Java y JWSDP Java Web Services Developer Pack) para crear un segundo modelo dependiente de la plataforma, .NET (más precisamente, C# y WCF) para crear un tercer modelo dependiente de la plataforma y PHP (más precisamente, PHP y NUSOAP.). El primer PSM puede considerarse como una plataforma abstracta [42], o en otras palabras utilizando un lenguaje de implementación independiente del ambiente (implementation language environment independent) [66]. El segundo, tercero y cuarto PSM se pueden considerar como una plataforma concreta [42].

Se ha hablado de diferentes lenguajes de modelización en este trabajo de tesis para dar un enfoque general de los multiples lenguajes que existen. Para el desarrollo de esta utilizaremos UML como lenguaje de modelización para crear los PIMs y varias plataformas objetivos para crear las PSMs con el fin de identificar los problemas ligados en el enfoque dirigido por los modelos y proponer enseguida soluciones viables. De esto procede la experimentación de la aplicación de MDA para la plataforma de los servicios Web para la realización de la red social AnonymousRemark.

5.1.1. Red social

La red social AnonymousRemark proporciona al usuario la capacidad de registrarse, publicar ya sea de manera anónima o con identidad, compartir archivos, calificar contenido y publicar en otras redes sociales.

Para que la red social evolucione de una manera rápida, esta tiene que tener la capacidad de ser escalable, por lo cual los servicios Web deben ser consumibles desde

diferentes plataformas y en diferentes tipos de dispositivos.

El principio general de la red social se puede resumir como sigue:

- Los usuarios pueden inscribirse a la red social.
- Los usuarios pueden calificar o compartir el contenido con los demás.
- Se puede publicar información en la página sin ser usuario y siendo usuario se puede publicar contenido y archivos.
- Los usuarios pueden cambiar sus datos personales.
- Los usuarios pueden compartir información a otras redes sociales.

La red social pretende mostrar cómo es posible realizar un desarrollo orientado a servicios Web con MDA para la creación de redes sociales. Esta red debe permitir a múltiples usuarios conectarse a la misma, actualizar mensajes, calificar mensajes y compartir archivos de manera automatica por lo cual es un trabajo distribuido en Internet.

5.2. Etapas de desarrollo de la red social anonymous remark

Las etapas de desarrollo de la red social AnonymousRemark se presentan en el cuadro 5.1. Para realizar estas diferentes etapas, hemos adoptado un enfoque MDA. Por consecuencia, la metodología propuesta debe basarse en la creación de modelos de negocio (PIM) que serán transformados en modelos dependientes de las plataformas tecnológicas (PSM). La metodología se detallará posteriormente. La modelización de la red social se realizará en UML (PIM).

Tarea	**Descripción**
Elegir una metodología	La metodología mejor adaptada para este desarrollo es XP (Extreme Programming) por ser una metodología incremental.
Elegir una arquitectura en capas	La arquitectura de la aplicación sera 3-tiers por ser una aplicación Web.
Modelar la lógica de negocio de la red social y como se utiliza.	En estas se modelara las actividades que se pueden realizar en AnonymousRemark.
Elegir las plataformas para la implementación.	Las plataformas debe adaptarse fácilmente a la Internet y ampliamente promovidas en el mercado del cómputo. Los lenguajes más utilizados para desarrollar web son Java, PHP y C# por lo cual se desarrollaran los modelos en estas plataformas.
Entrega	Se realizaran los modelos de la aplicación en los diferentes lenguajes objetivos Java, PHP y C# y la aplicación desarrollada en PHP.

Cuadro 5.1: Etapas del desarrollo de la red social AnonymousRemark.

DC = Diagrama de Clase de UML.
DA = Diagrama de Actividad de UML.
CCA = Component Collaboration Architecture de EDOC.

PIM PSM	**Servicio Web**	**J2EE**	**.NET**	**PHP**
UML	DC WSDL y DA BPEL4WS	DC (Java + JWSDP)	DC (C# + API Framework)	DC (PHP + Framework (NUSOAP))

Cuadro 5.2: La experimentación propuesta para analizar el planteamiento MDA.

Las especificidades de la red social AnonymousRemark nos conducen a privilegiar una arquitectura 3-tiers cliente ligero. Simplificamos la presentación de desarrollo de la aplicación sólo considerando el middle tier y la capa de recursos, la cual se presentará como un ORM. Las capas de presentación no se abordaran aquí.

5.3. Metodología

El desarrollo de AnonymousRemark se tomara desde cero ya que se desea crear un producto que sea escalable a diferentes tecnologías de una manera fácil y rápida.

El planteamiento MDA que se propone puede presentarse por el siguiente diagrama de estado-transición de la Figura 5.1.

El proceso de la Figura 5.1 define las siguientes etapas:

- La etapa 1 puede realizarse ya sea con UML o EDOC como PIM. Otros enfoques pueden optar por la creación de meta-modelos específicos para modelar un negocio específico, tal como DSL (Domain Specific Language) [18].
- La etapa 2 corresponde a la elección de un meta-modelo pre-existente en la literatura o la creación de un nuevo meta-modelo especifico a las plataformas objetivos como los servicios Web, J2EE, .NET y PHP.
- La etapa 3 está dedicada a la correspondencia. Esta consiste en especificar cuáles elementos del PIM (por ejemplo de UML) son equivalentes o similares a los elementos del PSM (por ejemplo cuáles elementos de UML son equivalentes o similares a los elementos del WSDL).
- La etapa 4 utiliza la correspondencia definida en la etapa 3 para generar la definición de la transformación en un lenguaje de transformación. De hecho, la especificación de correspondencias puede tratarse como un lenguaje abstracto. Los lenguajes abstractos pueden especificar las relaciones entre los meta-modelos, pero estos no son ejecutables. Por ejemplo, los lenguajes abstractos propuestos en [11] y [47] están basados en OCL.
- La etapa 5 se realiza para la aplicación de definiciones de transformaciones para transformar un modelo de entrada (PIM) en un modelo de salida (PSM).

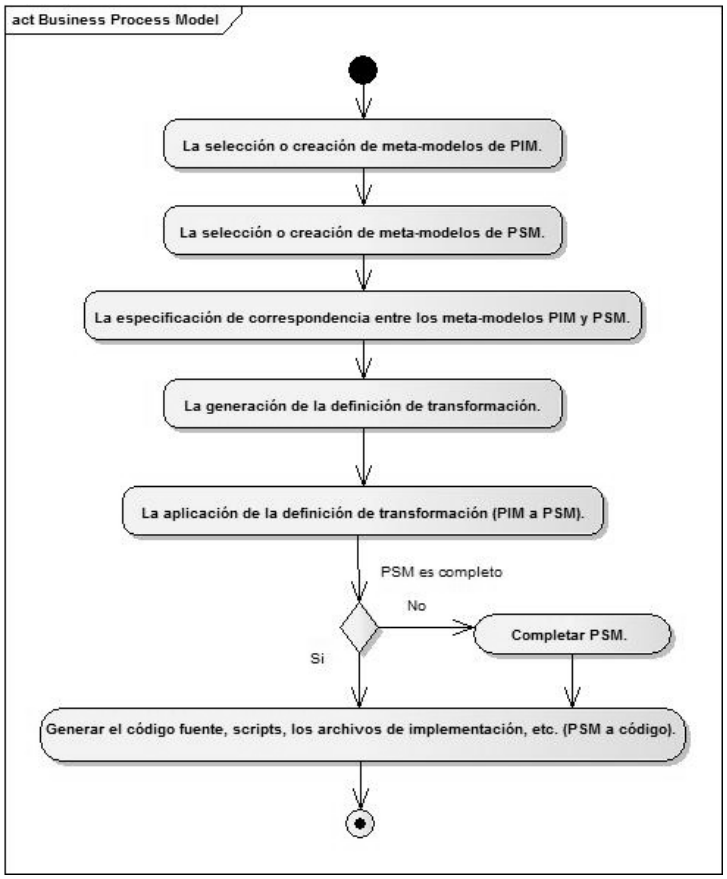

Figura 5.1: Un diagrama de actividades mínimo para la aplicación del planteamiento MDA.

- La etapa 6 verifica si el modelo generado durante la etapa anterior es completo.
- La etapa 7 necesita la intervención del desarrollador para completar el PSM.
- La etapa 8 consiste en generar el código fuente, los scripts y el archivo de despliegue a partir del PSM final creado en la etapa 5 o 7. En este caso, es necesaria una transformación de tipo modelo hacia código.

Las etapas 1 y 2 tienden a permitir la elección o la creación de meta-modelos para construir un PIM o un PSM. Existen dos posibilidades para crear nuevos meta-modelos:

- La utilización del concepto de perfiles UML para extender el meta-modelo de UML y considerar otros aspectos no previstos previamente.
- La creación de nuevos meta-modelos basados directamente en un lenguaje de meta-modelización como MOF [68] o Ecore [27].

Generalmente, los desarrolladores crean un meta-modelo y proponen enseguida un perfil UML conforme a este meta-modelo, puesto que la mayoría de las herramientas solo soportan la creación y la edición de modelos UML.

Una arquitectura tipo para la transformación de modelos puede ilustrarse en la Figura 5.2. En esta arquitectura, el modelo de correspondencia (la especificación de correspondencias mapping M) se separa del modelo de transformación (la definición de transformaciones transformation M).

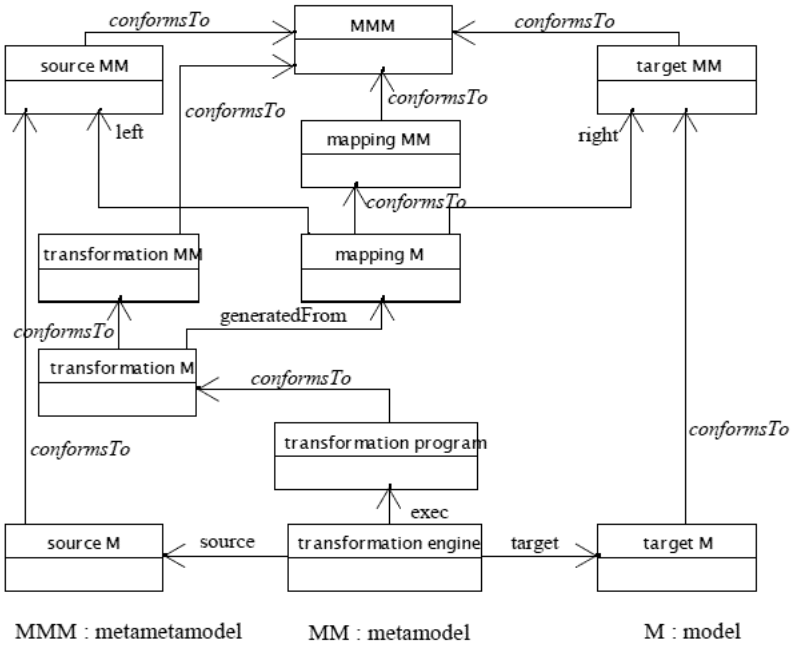

Figura 5.2: Una proposición de arquitectura tipo para la transformación de modelos [21].

Esta proposición de arquitectura tipo para la transformación de modelos contiene las siguientes entidades:

- MMM (un meta-meta-modelo): por ejemplo, MOF o Ecore.
- source MM y target M (un meta-modelo fuente y objetivo): por ejemplo, el meta-modelo de UML o de UML.
- source M y target M (un modelo fuente y un objetivo): desarrollo de la red AnonymousRemark en UML que debe transformarse en un modelo Java.
- mapping MM (un meta-modelo de correspondencia): el lenguaje para modelar las correspondencias entre los elementos de un meta-modelo fuente y los elementos de un meta-modelo objetivo.
- mapping M (un modelo de correspondencia): el modelo contiene las correspondencias entre dos meta-modelos.
- transformation MM (un meta-modelo de transformación): el formalismo que permite la creación precisa de definiciones de transformaciones de un modelo fuente en un modelo objetivo.
- transformation M(un modelo de transformación): describe la transformación de un meta-modelo en otro.

- transformation program (un programa de transformación): un programa ejecutable para realizar la transformación.
- transformation engine (un motor de transformación): ejecuta un programa de transformación.

El modelo de correspondencia especifica las correspondencias entre un meta-modelo fuente y un meta-modelo objetivo. Un modelo de transformación se genera a partir de un modelo de correspondencia. Un programa de transformación se corresponde a un modelo de transformación. La transformación se hace entonces mediante un motor de transformación que ejecuta un programa de transformación. El motor de transformación toma como entrada un modelo fuente, ejecuta el programa de transformación, y proporciona un modelo objetivo como resultado.

5.4. Modelización en UML

Los requerimientos pueden expresarse bajo la forma de diagramas de casos de uso de UML. Este diagrama es un lenguaje próximo del lenguaje natural y permite la descomposición del sistema de acuerdo con las funcionalidades demandadas. Cada caso de uso aparece entonces como una clase de situación.

La Figura 5.3 proporciona el diagrama de casos de uso para la red social Anonymous-Remark en la cual se puede ver los distintos tipos de usuarios que existen que son: Usuario anónimo, usuarios registrado y administrador. Cada usuario tiene distintos permisos por lo que permite modular la red.

A continuación mostraremos un diagrama de secuencia sobre el caso de uso acceder, cabe mencionar que se realizo un diagrama de secuencia por cada caso de uso. La Figura 5.4 muestra el caso de uso acceder.

La Figura 5.5 muestra un diagrama de clases para la red social AnonymousRemark. Estas clases son las que permitiran almacenar usuarios, crear, calificar, compartir publicaciones y compartir archivos.

La Figura 5.6 y la Figura 5.7 muestran los diagramas de usuario y de mensaje, los cuales muestran los métodos de cada uno de los servicios. Cada método en realidad tiene su propio diagrama de actividades los cuales se omitirán por ser demasiados.

Con el fin de modelar el proceso de negocio de anonymous remark como una composición de servicios , utilizamos los valores marcados de UML para ligar el diagrama de actividad al diagrama de clase y permitir su transformación hacia BPEL4WS. En cada iteración, los valores marcados se utilizan para realizar la referencia a una clase o a una operación del diagrama de clase. La tabla 5.3 la cual representa las acciones de usuario se presenta en la primera columna las acciones, en la columna de en medio las marcas y a la derecha los valores.

La utilización de valores marcados es necesario porque hay una diferencia sintáctica y semántica entre el meta-modelo de UML y de BPEL4WS.

En el meta-modelo de BPEL4WS, cada acción se realiza por una operación de un servicio, y cada servicio se asocia a un partnerLink. Este contiene la información sobre el rol y la interfaz (PortType) del servicio que realiza esta operación.

En este enfoque, se utilizan las siguientes marcas activity (aceptando los valores receive, assign, invoque y reply), class (aceptado el nombre de una clase) y operation (aceptando el nombre de una operación).

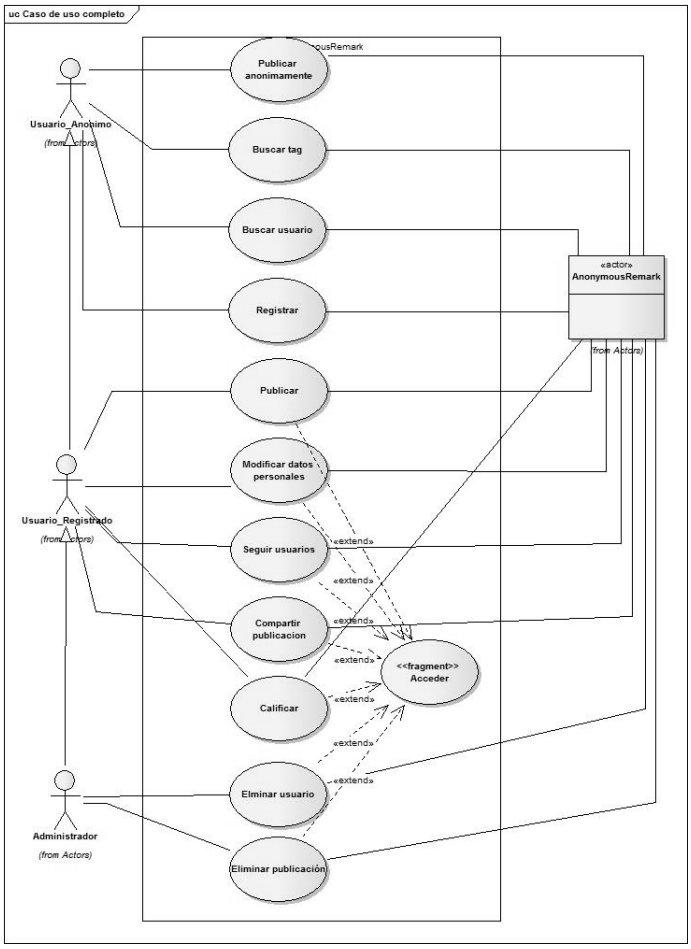

Figura 5.3: Los casos de uso de AnonymousRemark.

La información agregada en las acciones por medio de los valores marcados se utilizará más adelante para realizar la transformación del diagrama de actividad en BPEL.

5.5. Los servicios Web como plataforma objetivo

La plataforma de servicios Web está constituida de diferentes tecnologías. La Figura 5.8 ilustra estas tecnologías y sus relaciones.

La utilización de servicios como plataforma objetivo en el contexto de MDA debe estar en conformidad con el proceso de aplicación de MDA presentado anteriormente. Debemos por lo tanto elegir o crear los meta-modelos para las plataformas objetivo. Entre las tecnologías que constituyen los servicios Web, las más importantes para nuestro trabajo son WSDL, UDDI y BPEL4WS. Hasta nuestros días, no hay un modelo oficial de WSDL, UDDI y BPEL4WS.

La Figura 5.9 presenta un meta-modelo propuesto para WSDL obtenido a partir de su esquema [108]. En el apéndice A (Figura A.2, se proporciona un meta-modelo más

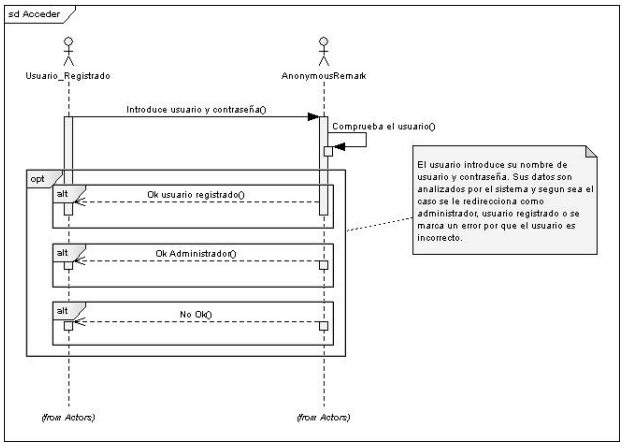

Figura 5.4: Diagrama de secuencia de acceder en AnonymousRemark.

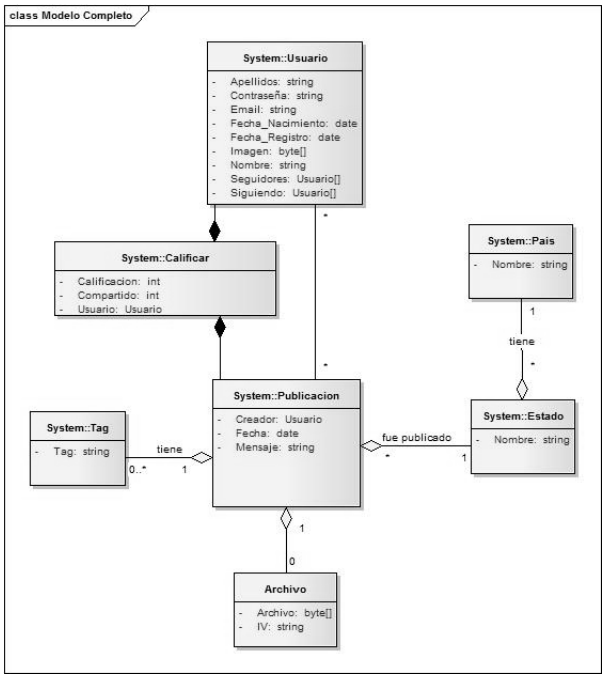

Figura 5.5: El diagrama de clases de la red AnonymousRemark.

completo para WSDL).

Este meta-modelo de WSDL se compone de los siguientes elementos:

- **Definition**: elemento principal de este meta-modelo que contiene un conjunto de Import, Type, Message, PortType, Binding y Service.
- **Import**: permite la asociación de un espacio de nombres (namespace) en la localización de un documento XML:
- **Type**: utilizado para definir un tipo de datos abstractos simples o complejos en conformidad con un esquema XML.

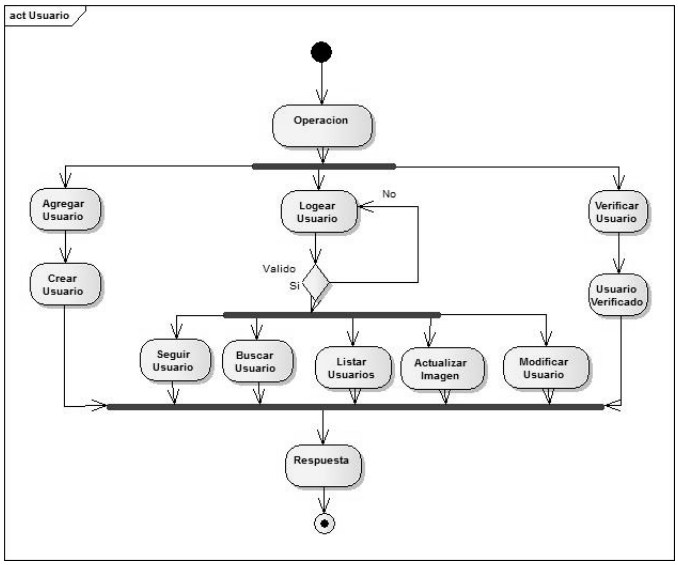

Figura 5.6: El diagrama de gestion de usuarios. (fragmento).

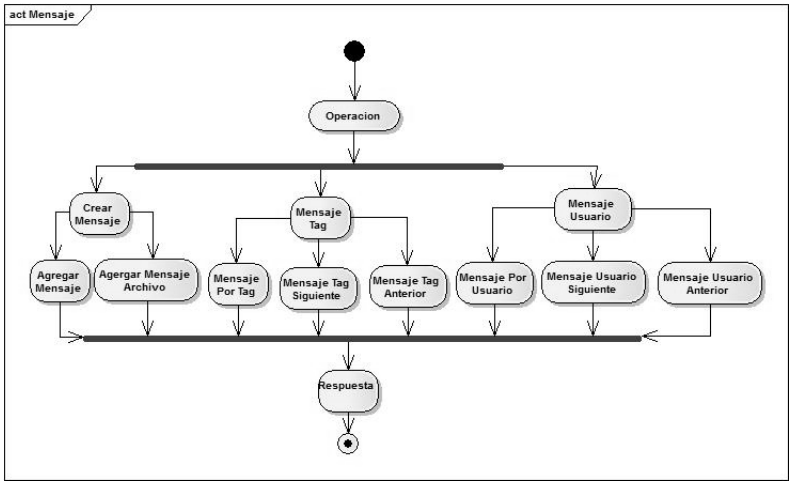

Figura 5.7: El diagrama de gestion de mensajes. (fragmento).

- **Message**: describe un formato abstracto de un mensaje particular que el servicio Web envía o recibe. Contiene las partes (por ejemplo Part) que describen cada parte de un mensaje.

- **portType** (se le llama Interface a partir de WSDL v1.2): define la interfaz de un servicio. Contiene un conjunto de operaciones que un servicio envía y/o recibe. Estas operaciones se caracterizan por el elemento Operation que describe los tipos de llamadas de manera abstracta. Los tipos de llamada se caracterizan por los elementos OneWayOperation, RequestResponseOperation, SolicitResponseOperation y NotificationOperation. El elemento ParamType identifica cuales son los mensajes de input, output y fault.

Acción	Marca	Valor
Agregar Usuario	activity class operacion	Receive Usuario Crear Usuario
Logear Usuario	activity class operacion	assign Usuario Logear Usuario
Verificar Usuario	activity class operacion	assign Usuario Verificar Usuario
Seguir Usuario	activity class operacion	assign Usuario Seguir Usuario
Buscar Usuario	activity class operacion	assign Usuario Buscar Usuario
Listar Usuarios	activity class operacion	reply Usuario Listar Usuarios
Actualizar Imagen	activity class operacion	reply Usuario Actualizar Usuarios
Modificar Usuario	activity class operacion	assign Usuario Modificar Usuarios

Cuadro 5.3: Los valores etiquetados de cada acción del diagrama de actividad (Figura 5.6).

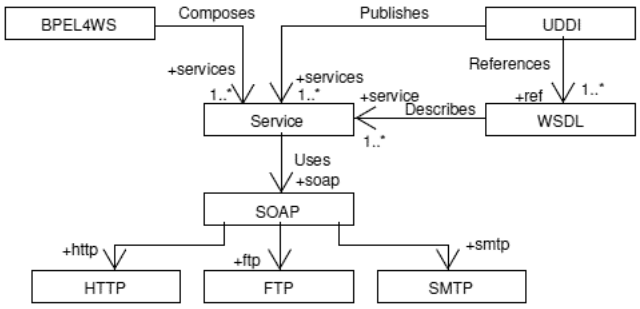

Figura 5.8: Las principales tecnologías de los servicios Web.

- **Binding**: describe un vínculo concreto de los componentes de una interfaz con el protocolo de comunicación utilizado, es decir, la manera en la cual una llamada puede concluirse de manera concreta (por ejemplo, utilizando SOAP con HTTP). El elemento BindinOperation contiene las input, output, y fault que se describen según la realización concreta de la llamada.

- **Service**: describe un servicio, identifica su interfaz (PortType) y sus localizaciones (endpoints).

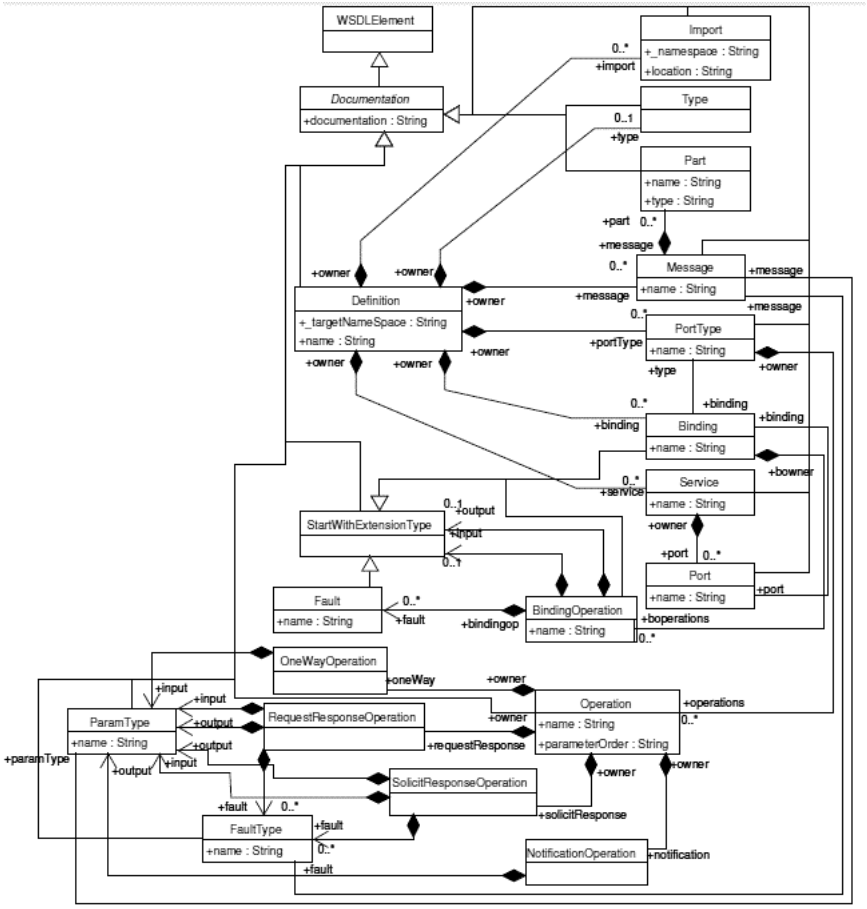

Figura 5.9: Un meta-modelo de WSDL (fragmento).

La Figura 5.10 presenta un meta-modelo de UDDI. Este meta-modelo se basa en el que se propuso en el UDDI Executive White Paper [132] con algunas modificaciones.

Este meta-modelo se compone de los siguientes elementos:

- **BusinessEntity**: elemento principal que soporta la publicación y el almacenamiento de la información relativa a un proveedor de servicio y a su negocio. Este contiene el nombre, una descripción, las taxonomías y los contactos relativos al servicio de negocio o a una organización que proporciona los servicios Web.

- **BusinessService**: descripción de un conjunto de servicios Web conforme al tipo de negocio o a la categoría de servicios. Jerarquiza los servicios en categorías de industria, de producto y servicio.

- **BindingTemplate**: contiene una información técnica para utilizar un servicio Web individual. Proporciona el punto de acceso o un mecanismo de localización (indirection mechanism) para el punto de acceso.

- **TModelInstaceInfo**: contiene los detalles específicos de una entidad de BindingTemplate para cada TModel referenciado.

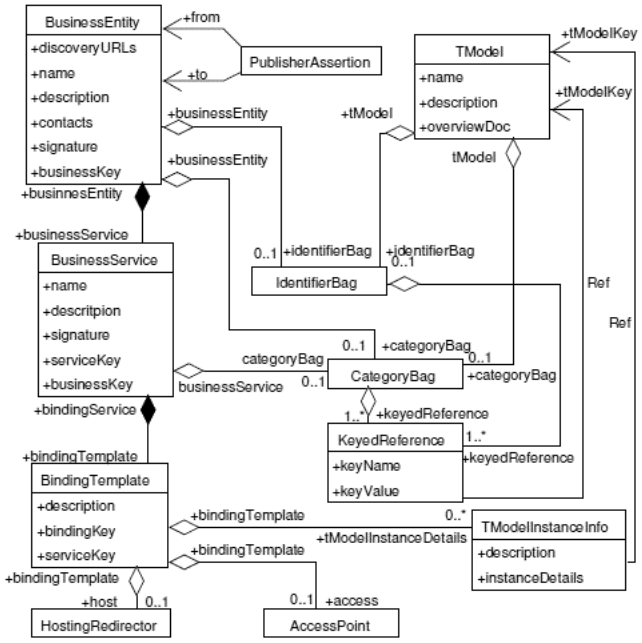

Figura 5.10: Un meta-modelo de UDDI.

- **TModel**: contienes los meta-datos de servicios que pueden ser una especificación, un protocolo de transporte o un espacio de nombres.

- **PublisherAssertion**: describe las relaciones entre BusinessEntities.

BPEL4WS tiene una relación estrecha con WSDL ya que hace referencia a un portType de servicios utilizados en el proceso. La Figura 5.11 presenta el fragmento de un meta-modelo de BPEL4WS obtenido a partir de su esquema [6]. Un meta-modelo de BPEL4WS más completo se presenta en el apéndice A (Figura A.2).

Este meta-modelo de BPEL4WS está constituido de los siguientes elementos:

- **Process**: contiene un conjunto de actividades (Activity), Partners, Variables, CorrelationSets, FaultHandlers, CompensationHandlers y EventHandlers.

- **PartnerLinks**: define las diferentes partes que actúan unas sobre las otras durante la ejecución de un proceso. Se caracteriza por uno o varios PartnerLink que especifican las relaciones de conversación entre dos servicios por medio de la declaración de sus roles. Cada rol especifica un portType de WSDL que un socio realiza.

- **Partner**: sub-conjunto de PartnerLink (referencias), introduce una restricción de la funcionalidad que un socio debe proporcionar.

- **Variables**: define las variables de datos utilizadas por el proceso, y permite al proceso mantener los datos y los históricos basados en el intercambio de datos. Una variable puede definirse en términos de tipo de mensaje WSDL (Message), tipo simple (type) o complejo (element) descrito en esquema XML.

- **CorrelationSets**: conjunto de propiedades utilizadas para considerar las correlaciones. Especifica los grupos de operaciones que tienen una correlación en una instancia de servicio.
- **FaultHandlers**: administrador de defectos de la ejecución de servicios que definen las actividades que serán ejecutadas en reacción a la falta.
- **CompensationHandler**: wrapper para las actividades de compensación.
- **EventHandlers**: este administrador se llama cuando un evento determinado llega y resulta en la invocación de las actividades.
- **Activity**: actividad estructurada en varias partes como el flujo de control (por ejemplo, Switch y While), mensajes (datos transferidos entre las actividades a través de Assign), transacciones (transacciones de larga duración soportadas en un proceso único) y la extensibilidad (el espacio de nombre en forma de URI puede utilizarse en varios elementos de BPEL4WS).

5.6. La plataforma J2EE

La plataforma J2EE está constituida de diferentes tecnologías, de manera especial el lenguaje Java, JDBC (Java DataBase Connectivity), JWSDP (Java Web Services Developer Pack), EJB (Enterprise Java-Beans) y JSP (Java Server Pages). Entre estas tecnologías, nos interesa especialmente el lenguaje Java y JWSDP. Este último framework del mundo Java permite la programación y el despliegue de los servicios Web. Así, se presenta un meta-modelo de Java y otro de JWSDP (se usa Tomcat 5.0 como el servidor de las aplicaciones Web).

La Figura 5.12 expone un posible meta-modelo de Java. Otros meta-modelos de Java pueden encontrarse en la literatura (por ejemplo, el que se propone en la especificación de UML profile for EDOC [70]) o en Internet (por ejemplo, el que propones NetBeans).

Este meta-modelo de Java está constituido de los siguientes elementos:

- **JElement**: elemento raíz de este meta-modelo.
- **JPackage**: contiene JClass, JInterface, etc.
- **JClassifier**: contiene JMember (JField y JMethod).
- **JClass**: especialización de JClassifier, implementa una JInterface.
- **JInterface**: otra especialización de JClassifier, que contiene solamente los prototipos de los métodos (sin el cuerpo) y atributos de tipo final static (constante).
- **JField**: contiene solamente un JPrimitiveType o JClass o JInterface.
- **JMethod**: contiene las operaciones (es decir, el comportamiento) de una clase o interface en Java. Hay un parámetro de retorno y uno o varios parámetros de entrada.
- **JParameter**: especifica los parámetros de un método Java.

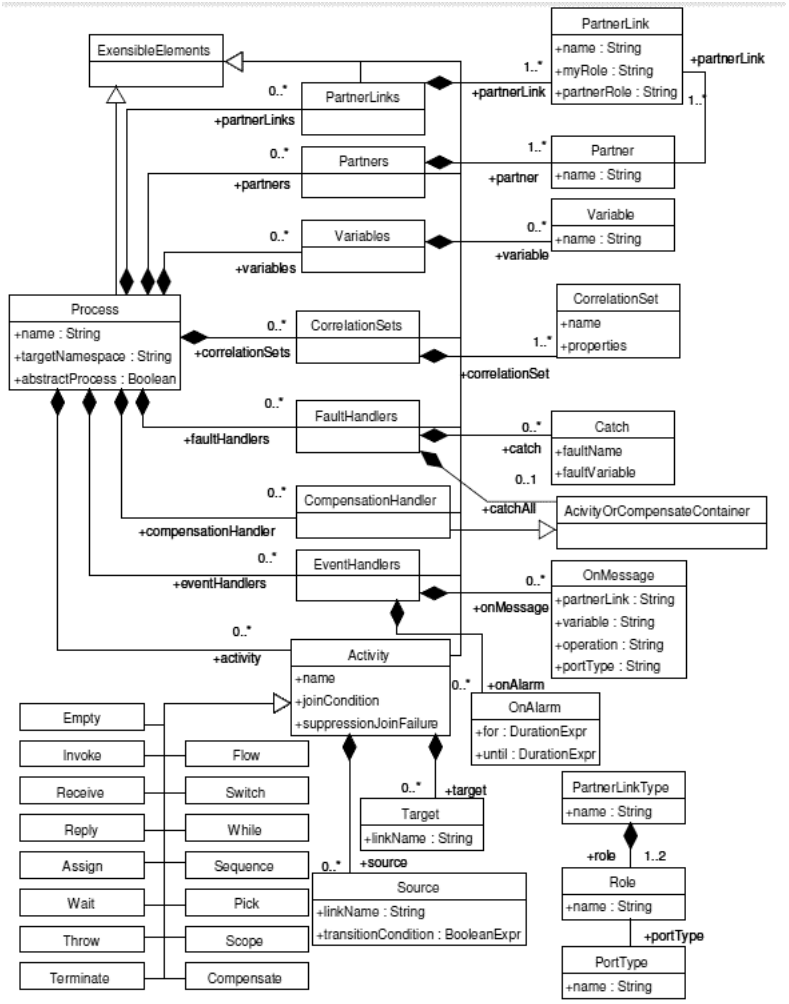

Figura 5.11: Un meta-modelo de BPEL4WS (fragmento).

La Figura 5.13 presenta (a) un template (modelo) y (b) un meta-modelo para JWSDP. Este template describe la realización de un servicio Web en JWSDP. En realidad, este template especifica la utilización de la API de JWSDP que se sitúa en el nivel modelo en la arquitectura de cuatro niveles de metadatos [68]. Un servicio Web se crea en JWSDP a través de una WSClass (conforme a JClass) que implementa una WSInterface (conforme a JInterface). Esta interfaz debe heredar de java.rmi.Remote y sus métodos deben lanzar una java.rmi.RemoteException en caso de excepción. La WSClass realiza el servicio y la WSInterface se utiliza por el cliente para llamar a este servicio. El meta-modelo de JWSDP que se propone se construyo a partir de esquemas XML y de DTDs de la especificación de JWSDP versión 1.3 [77]. Este meta-modelo de JWSDP está constituido de cuatro meta-paquetes: configInterface, jaxrpc, web y configWsdl. Estos contienen los elementos que representan la información utilizada para generar los archivos de despliegue y de configuración de JWSDP: config-interface.xml, jaxrpc-ri.xml, web.xml, config.xml.

Los archivos config-interface.xml y jaxrpc-ri.xml se utilizan para la compilación y la generación de WSDL y de Tie (stubs del lado del servidor) de un servicio Web. el archivo

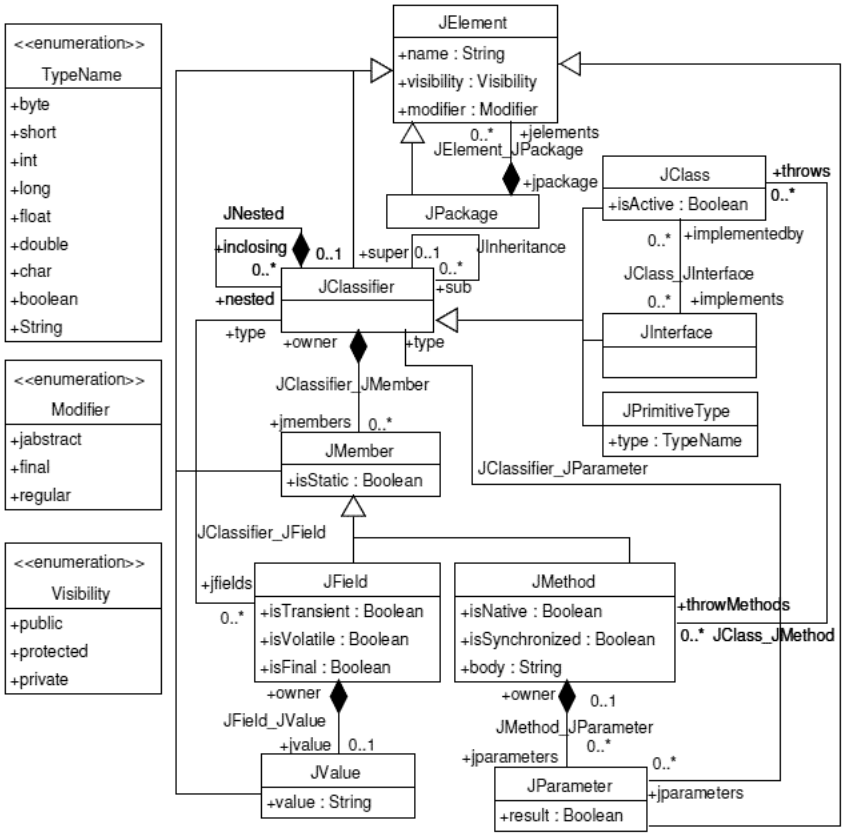

Figura 5.12: Un meta-modelo de Java (fragmento).

web.xml se utiliza para desplegar un servicio Web en el contenedor de las aplicaciones Tomcat. El archivo config.xml se utiliza para generar los stubs (para el lado del cliente) a partir del documento WSDL que describe el servicio Web deseado.

5.7. La plataforma .NET

La plataforma .NET se compone igualmente de diferentes tecnologías, como el lenguaje C#, el framework para los servicios Web y WebForms. Entre estas tecnologías, nos interesaríamos en el lenguaje C# y el soporte de los servicios Web con WCF. La Figura 5.14 ilustra un meta-modelo de C# obtenido a partir de su gramática [61].

Este meta-modelo de C# está constituido de los siguientes elementos:

- **Element**: elemento raíz.
- **Namespace**: contenedor para otros elementos (Class, Interface, etc).
- **Classifier**: generalización de los miembros, clases, interfaces y estructuras. Contiene un conjunto de miembros que pueden formar parte de: Field, Method y Property.
- **Class**: especialización de Classifier, realiza la Interface.

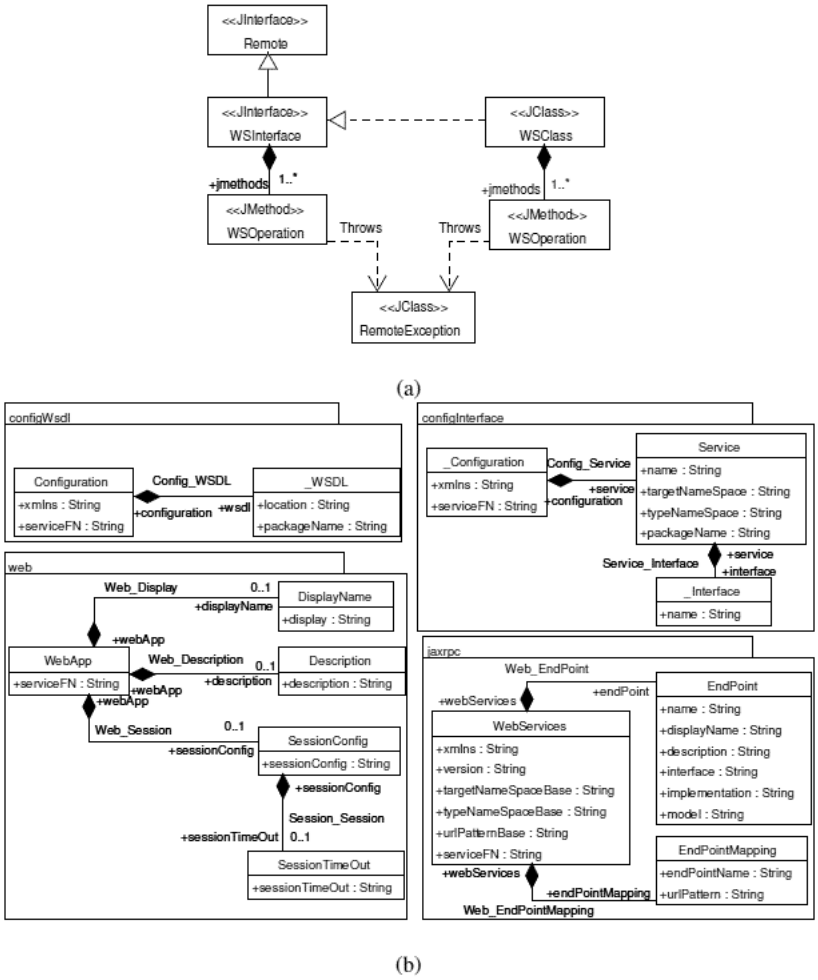

Figura 5.13: (a) un template y (b) un meta-modelo para JWSDP (fragmentos).

- **Interface**: otra especialización de Classifier, que contiene solamente las firmas de los métodos (sin el cuerpo).
- **TypedElement**: información declarativo que puede ser vinculado a las entidades de programas (como Class y Methods) y recuperado durante la ejecución.
- **DataType**: composición de un tipo que puede ser un ValueType o un Reference-Type.
- **Operation**: posee una firma de operaciones compuestas del tipo de retorno, del identificador y de los parámetros.
- **Property**: presenta las características de un Field y de un Method. Contiene un método get y/o set.
- **Tag**: tipo de retorno de un método.
- **Parameter**: argumentos de una operación.

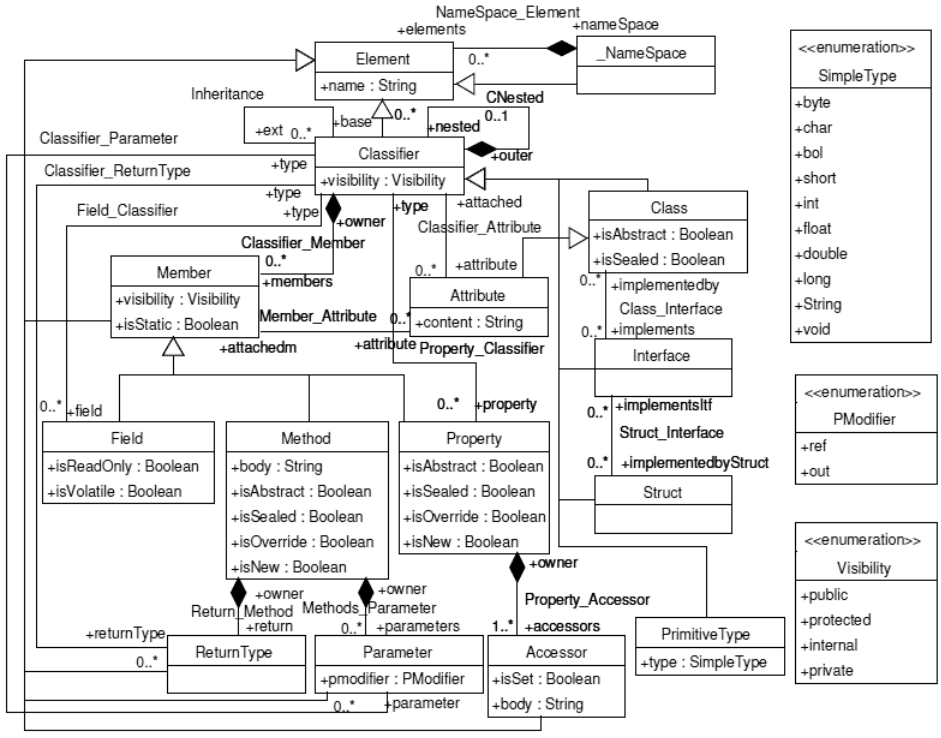

Figura 5.14: Un meta-modelo de C# (fragmento).

La creación de un servicio Web en .NET con C# se hace a través de la utilización los siguientes elementos :

- **ServiceContract**:atributo que se utiliza en las interfaces que definen el modelo del servicio Web..
- **OperationContract**:atributo que se utiliza para definir cada metodo del servicio.
- **DataContract**: atributo que una clase sea definido como un tipo de dato que sera enviado a traves de los servicios.
- **DataMember**: este atributo se utiliza para definir los miembros de los las clases de tipo DataContract.

La innovación de los servicios Web en C# con el framework de WCF, es que nos permite hostear nuestros servicios web en cualquier cosa y en cualquier protocolo por lo cual no es necesario utilizar un servidor Web (en el caso de Microsoft IIS) y utilizar diferentes protocolos. En este caso utilizaremos los servicios Web WSDL con WCF y C# como lenguaje de programación.

5.8. La plataforma PHP

La plataforma de PHP por si solo no cuenta con la tecnología necesaria para crear servicios Web, sin embargo existen muchas herramientas que nos permiten realizarlos.

Para este trabajo de tesis la herramienta elegida fue NUSOAP por contar con estándares para la creación y consumo de servicios Web de manera eficiente.

Para utilizar NUSOAP antes se debe definir un meta-modelo del lenguaje PHP, el cual nos permitira ver la estructura del mismo y como utilizarlo para la creación de modelos. La Figura 5.15 muestra el meta-modelo de PHP.

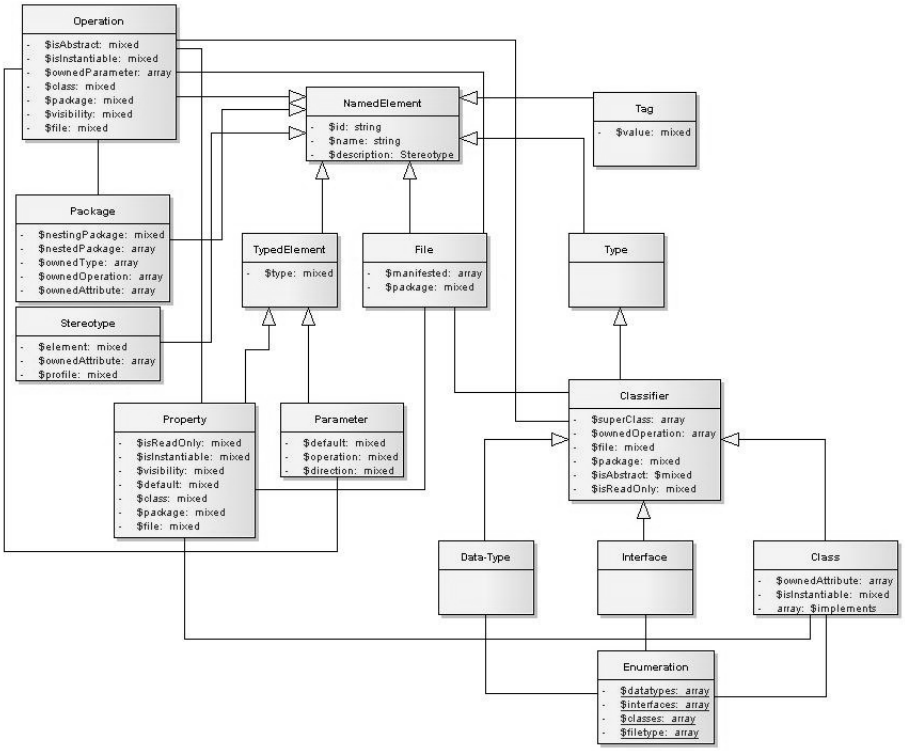

Figura 5.15: Metamodelo de PHP.

Este meta-modelo de PHP está constituido de los siguientes elementos:

- **NamedElement**: elemento raíz.
- **Package**: contenedor para otros elementos (Class, Interface, etc).
- **Type**: generalización de los miembros, clases, interfaces y estructuras. Contiene un conjunto de miembros que pueden formar parte de: Field, Method y Property.
- **Interface**: otra especialización de Classifier, que contiene solamente las firmas de los métodos (sin el cuerpo).
- **Struct**: similar a Class, pero no tiene herencia.
- **Attribute**: información declarativo que puede ser vinculado a las entidades de programas (como Class y Methods) y recuperado durante la ejecución, un atributo se utiliza para proporcionar la información adicional a un Classifier (por ejemplo, Class y Methods).

- **Member**: elemento de base para Fields y Methods.
- **Field**: composición de un tipo que puede ser un ValueType o un ReferenceType.
- **Method**: posee una firma de operaciones compuestas del tipo de retorno, del identificador y de los parámetros.
- **Property**: presenta las características de un Field y de un Method. Contiene un método get y/o set.
- **ReturnType**: tipo de retorno de un método.
- **Parameter**: argumentos de una operación.
- **Accessor**: puede igualmente ser un método get o set de una propiedad.

NUSOAP necesita crear un archivo PHP por cada servicio Web, en estos archivos se definen los metodos del servicio Web, los metodos de los servicios, y la creación de tipos complejos(complexType).Como PHP no es un lenguaje tipado la verificación de los datos en servicios Web debe ser más cuidadosas.

Capítulo 6

Aplicación MDA con los servicios Web

6.1. Los servicios Web y MDA

En este capítulo se hablara sobre las plataformas PSM que se trabajan en el enfoque MDA con los servicios Web. Las plataformas objetivo que se manejaron en este trabajo a partir de las plataformas independientes de software son meta-modelos de Servicios Web, y modelos en Java, .NET (C#) y PHP.

Para el manejo de datos en las diferentes tecnologías se utilizó un ORM (Object Relational Mapping) perteneciente a cada una de las tecnologías, en cada una de las secciones de este capítulo se hablara más sobre estos Frameworks y herramientas.

Cabe mencionar que dependiendo la tecnología se utilizaron diferentes framework que dan herramientas ORM (Object Relational Mapping) para el manejo de las bases de datos. A continuación se mostrara el enfoque MDA aplicado a las diferentes tecnologías.

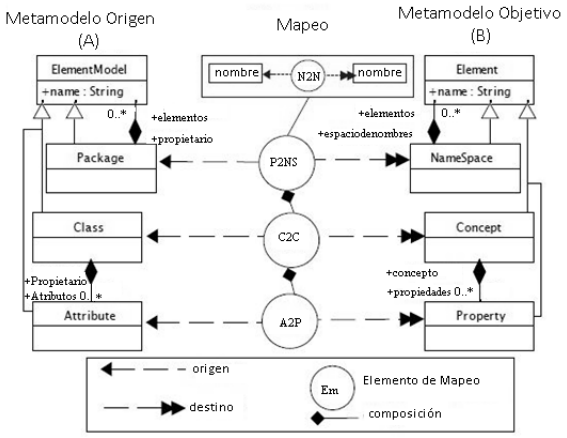

Figura 6.1: Ejemplo de transformaciones realizadas entre modelos.

En la Figura 6.1 se muestra un ejemplo básico de la correspondencia que existe entre meta-modelos en las transformaciones entre diferentes tecnologías.

6.2. UML y los servicios Web

UML es la principal herramienta para modelado, por lo cual se utilizó para realizar un meta-modelo PSM. UML es un lenguaje de modelado con diferentes tipos de diagramas por lo cual es el más utilizado para modelar sistemas y arquitecturas.

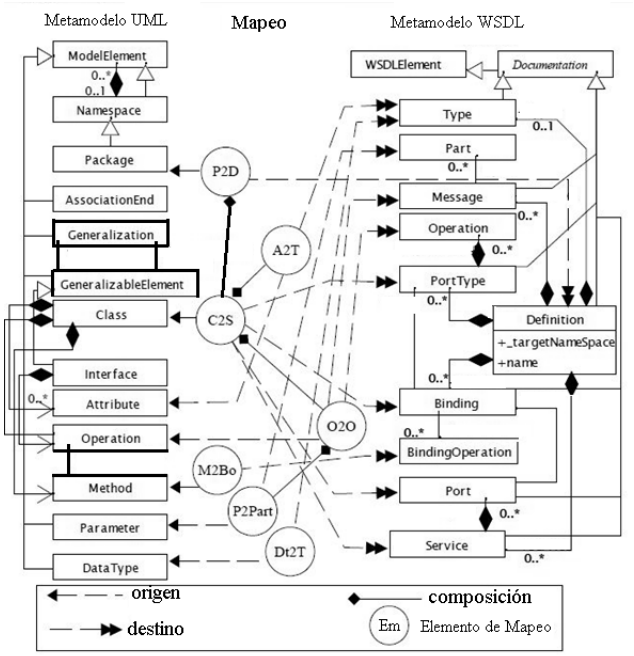

Figura 6.2: Transformación de modelos entre UML y WSDL

Los servicios web son descritos con WSDL (Web Services Description Language), UML es el lenguaje de modelado elegido por lo cual se necesita definir una manera de utilizar estas dos tecnologías. Se muestra la transformación realizada entre estos para posteriormente realizar una transformación para cada una de las plataformas objetivo. En la Figura 6.2 se muestra la transformación entre UML y WSDL.

La transformación realizada entre UML y WSDL en la Figura 6.2 necesita también especificar la correspondencia que existe entre los diferentes tipos de datos. En la Figura 6.3 se muestra la transformación que se debe de realizar para cada tipo de dato entre WSDL y UML.

El cuadro numero muestra una relación entre los tipos de datos existentes entre WSDL y UML. Es importante decir que existen más tipos de datos en WSDL que se pueden consultar para ser utilizados [112].

El siguiente diagrama de clases es un meta-modelo PSM el cual está utilizando la tecnología de los servicios Web como se puede ver en la Figura 6.4 el cual se transformara a las distintas plataformas.

En la siguiente sección se hablara sobre cómo aplicar el enfoque MDA con el meta-modelo expuestos, con las diferentes plataformas objetivo: Java, .NET y PHP.

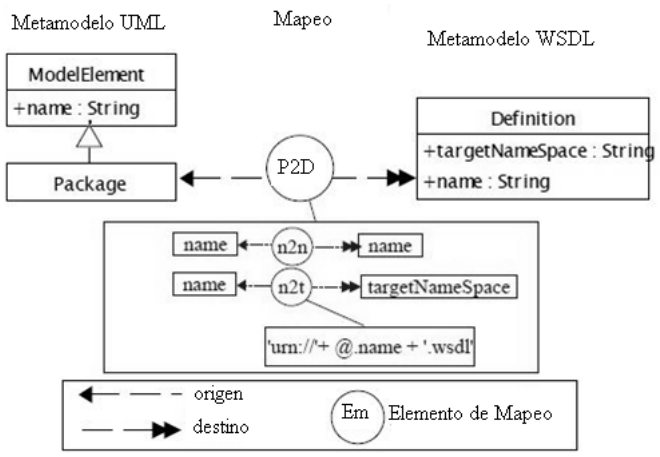

Figura 6.3: Transformación entre los tipos de datos entre UML y WSDL

UML (DataType)	**WSDL (PrimitiveType)**
Float	xsd:float
Byte	xsd:byte
Double	xsd:double
Integer	xsd:int
Long	xsd:long
Short	xsd:short
String	xsd:string
Boolean	xsd:boolean

Cuadro 6.1: Tipos de datos compatibles entre UML y WSDL

6.3. Aplicación en Java

Una de las tecnologías especifica de la plataforma (PSM) seleccionadas fue Java que actualmente es licenciada por Oracle. Esta tecnología lleva años dedicada a realizar software orientado a servicios como CORBA y RMI. Actualmente cuenta con herramientas para desarrollar servicios Web estables y que siguen en plena evolución.

Para la realización de los modelos en Java se utilizó la biblioteca JWSDP (Java Web Services Developer Pack), esta biblioteca facilita el desarrollo de servicios Web, el manejo de datos y de los servicios haciendo casi transparente la utilización de los mismos bajo Java.

Para poder transformar un meta-modelo a un modelo tiene que quedar claro que deben existir reglas de correspondencia. Como en este caso se utilizara UML como lenguaje de modelado, es importante mostrar la correspondencia que existe entre UML y Java. En la Figura6.5 se muestra una transformación que se hace entre UML y Java.

Para terminar la transformación es necesario aclarar la compatibilidad que existe entre los diferentes tipos de datos entre UML y Java esto con el fin de ayudar a la creación del modelo de plataforma en Java. En el Cuadro 6.2 se muestra la compatibilidad existente entre Java y UML. Es importante agregar que se utilizara el diagrama de UML con WSDL para realizar el de UML con Java. Por esta razón se puede ver que existen tipos

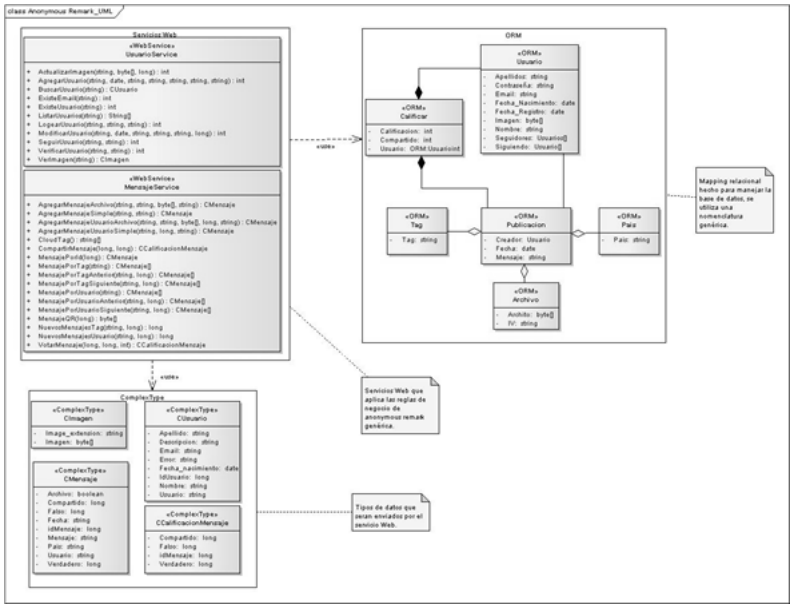

Figura 6.4: Modelo de clases UML de AnonymousRemark

de datos compatibles entre XML y Java [75]; el Cuadro A.1 del apéndice da forma a esta compatibilidad.

UML (DataType)	**Java (PrimitiveType)**
Float	float
Byte	byte
Double	double
Integer	int
Long	long
Short	short
String	String
Boolean	boolean

Cuadro 6.2: Tipos de datos compatibles entre UML y Java

Como JWSDP será el Framework que se encargara de realizar los servicios Web en Java, se mostrara la correspondencia que tiene con UML para la creación del modelo especifico de la plataforma para esta tecnología. La Figura 6.6 muestra la transformación realizada entre UML y JWSDP.

Habiendo mostrado la relación que existe entre estas distintas tecnologías pasamos a mostrar el modelo de la red social realizado en UML y como este se adapta a la tecnología Java y este utiliza el meta-modelo realizado en la sección anterior, Figura 6.4. Se muestra el diagrama del modelo dependiente de la plataforma PSM. La Figura 6.7, muestra igualmente el diagrama de clases en java, el cual genera el servicio Web y la estructura de funciones necesarias para la red con el lenguaje Java.

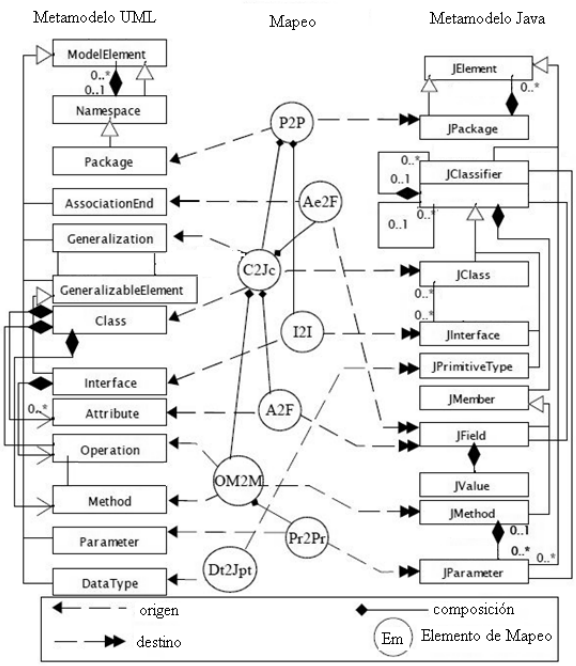

Figura 6.5: Modelo de transformación entre UML y Java

En el siguiente Cuadro 6.3 se muestra un fragmento del servicio Web MensajeService realizado en Java con JWSDP como ejemplo de la creación de servicios Web para esta plataforma.

```
@WebService(serviceName = "MensajeServices")
public class MensajeService {
    /*Web service operation
     */
    @WebMethod(operationName = "MensajePorTag")
    public CMensaje MensajePorTag(@WebParam(name = "tag") String
        Tag)
                {
        CMensaje u= new CMensaje();
       //Implementacion de codigo.
        return u;
                }}
```

En el Cuadro 6.3 se muestra una implementación de la forma de serializar objetos para ser enviados por el servicio Web en Java con JWSDP.

```
public class CImagen implements Serializable {
    public byte[] imagen;
    public String imagen_extension;
}
```

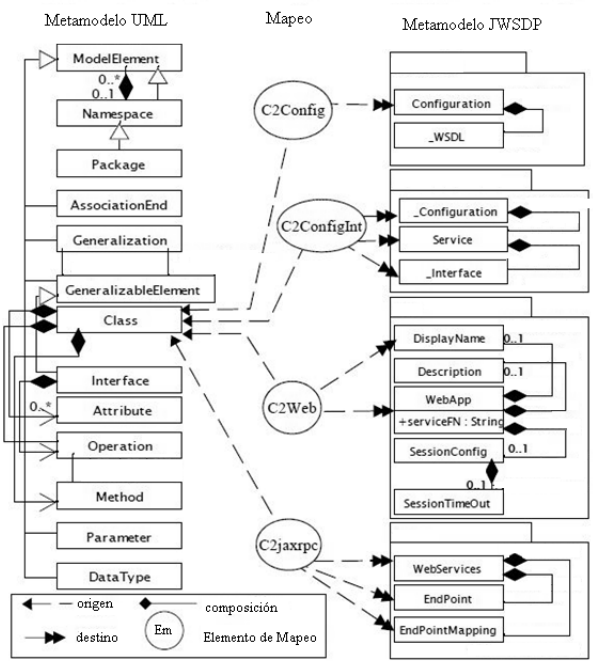

Figura 6.6: Modelo de transformación entre UML y JWSDP

6.4. Aplicación en .NET

.NET y Visual Studio son un conjunto de tecnologías de Microsoft, las cuales permiten desarrollar aplicaciones robustas en diferentes lenguajes de programación de una manera rápida y sencilla. .NET maneja principalmente 3 lenguajes de programación Visual C++, Visual Basic y Visual C#, nosotros nos enfocaremos en C# por ser un lenguaje orientado a objetos y ser el más utilizado por Microsoft en cuestiones Web.

.NET cuenta con una alta gama de herramientas que permiten desarrollar sistemas y apliaciones orientados a servicios. El framework creado por Microsoft para esta tendencia de desarrollo de aplicaciones tiene como nombre WCF (Windows Communication Foundation), el cual permite programar de manera rápida, fácil y robusta, servicios We;, estos pueden tener diferentes formatos, protocolos y manejo de WorkFlows para los procesos de negocio.

En este caso WCF no será tomado como un meta-modelo porque este mismo se acopla de manera transparente a los servicios Web, por lo cual no es necesario; sin embargo si el usuario quiere definir permisos con esta tecnología es recomendable la utilización de Interfaces (contratos), con la única diferencia que los servicios tendrán marcas para especificar las operaciones. La Figura 6.8 muestra una transformación de meta-modelo entre C# y UML.

Al igual que con Java, es importante ver los tipos de datos que son compatibles entre UML y C# para que se complemente el meta-modelo mostrado en la Figura 6.8, estas compatibilidades se muestran en el cuadro 6.3. En el apéndice A.2 se muestra un cuadro de referencia entre datos XML y .NET más completa [62].

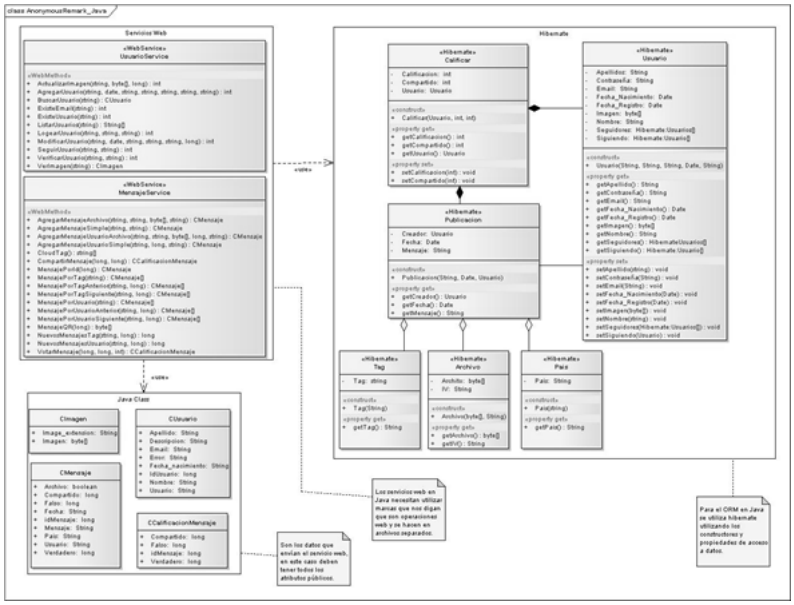

Figura 6.7: Modelo en Java con JWSDP de la red social AnonymousRemark

UML (DataType)	C# (PrimitiveType)
Float	float
Byte	Byte
Double	Double
Integer	int
Long	long
Short	short
String	String
Boolean	boolean

Cuadro 6.3: Tipos de datos compatibles entre UML y C#

Considerando el Cuadro 6.3 podemos mostrar la implementación del diagrama de clases y a partir de este generar automáticamente su WSDL. Es importante mencionar que se utilizaran Interfaces por la seguridad que debe manejar la aplicación y para el manejo de datos se utiliza el Entity Framework el cual es un ORM que nos permite manejar los datos. La Figura 6.9 muestra el modelo PSM realizado en C# para la realización de la red social AnonymousRemark.

Los servicios Web en .Net son creados con Interfaces que heredan a clases para implementar los métodos y con etiquetas entre corchetes que permiten especificar la operación o característica con la cual contara un método o propiedad. En el cuadro 6.4 se muestra una implementación de un método del servicio Web MensajeService.

La forma de serializar los datos que serán enviados en .Net es a través de los DataContract en el cuadro se muestra la estructura para realizarlo. Un DataContract es una etiqueta que se agrega a la clase que se desea enviar, y los DataMember son los miembros de esa clase que podrán enviarse.

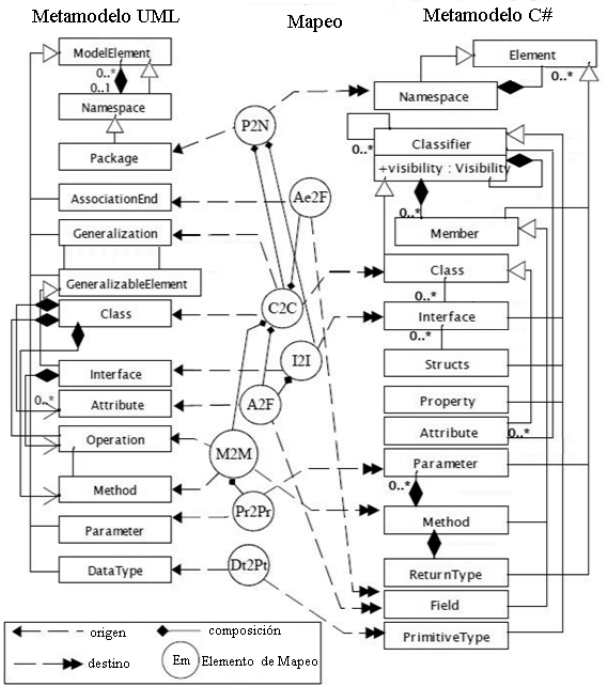

Figura 6.8: Transformación entre UML y C#

```
[ServiceContract]
public interface IMensajeService
{    [OperationContract]
    List<CMensaje> MensajePorTag(string tag);}
public class MensajeService: IMensajeService
{
    public CMensaje[] MensajePorTag(string tag)
        {
        CMensaje[] u;
        return u;}}
```

```
[DataContract]
public class CEstado
{    byte[] _imagen;
    String _imagen_extension
        [DataMember]
        public byte[] Imagen
        { get; set;}
        [DataMember]
        public String Imagen_extension
        { get; set;}
}
```

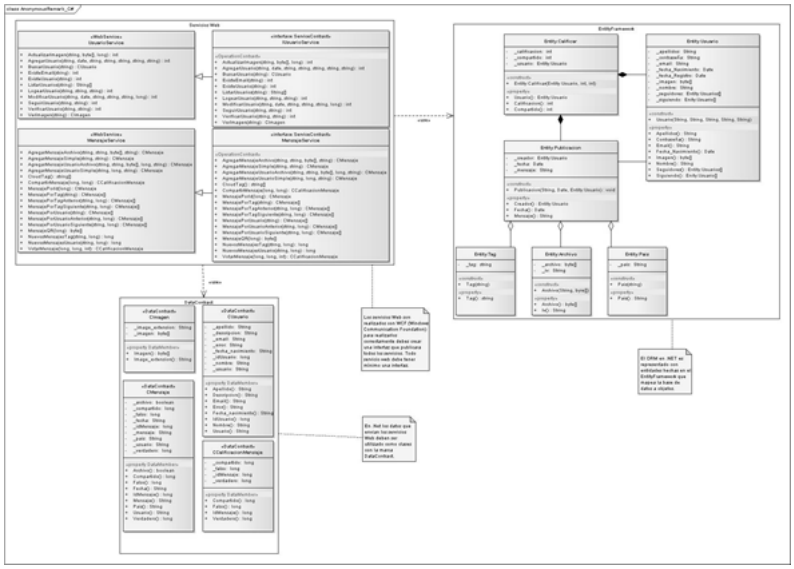

Figura 6.9: Modelo en C# de la red social AnonymousRemark

6.5. Aplicación en PHP

PHP es el lenguaje más utilizado para el desarrollo de aplicaciones Web, este cuenta con una innumerable cantidad de herramientas, Frameworks y librerías que le han permitido a los desarrolladores la capacidad de desarrollar aplicaciones de manera rápida y eficiente. Por esta misma razón y dado que el lenguaje es software libre, no cuenta con un estándar para la realización de servicios Web, por lo cual se eligió el Framework NUSOAP por ser el más completo y que cumple con los estándares de los servicios Web, trabajando de una forma casi transparente.

NUSOAP es una herramienta muy poderosa que no trabaja directamente con clases por lo cual la creación de los servicios se hace en un archivo PHP en el cual se crea el Servicio Web, los datos que se van a utilizar y la seguridad que se desea implementar. Es importante mencionar que en este enfoque NUSOAP especifica de una manera transparente el WSDL por lo que se utilizara exactamente el mismo meta-modelo de WSDL y UML mostrado en la Figura 6.4.

Sin embargo, el lenguaje como tal no es tipado por lo cual es necesario hacer validaciones con funciones que nos otorga el lenguaje. En el siguiente cuadro se muestran los valores en UML y la función con la cual son validados los tipos de datos.

Tomando en cuenta la definición del servicio Web, el documento WSDL, y los tipos de datos de intercambio se realizan en archivos extra por el Framwork de NUSOAP, se debe agregar una capa extra en el diagrama de clases, donde se realizan los objetos con la lógica de la red, y otra donde se definen los servicios y los datos. La Figura 6.10 muestra el diagrama de clases del servicio Web realizado en PHP con NUSOAP y el Framework ActiveRecord para el manejo de los datos.

A continuación se muestra la manera de crear un servicio Web en PHP con NUSOAP. La creación del servicio Web con NUSOAP se realiza ejecutando los siguientes pasos.

UML (DataType)	PHP (Funciones de Validación)
Float	is_float
Byte	is_string
Double	is_double
Integer	is_int
Long	is_long
Short	is_int
String	is_string
Boolean	is_bool

Cuadro 6.4: Tipos de datos compatibles entre UML y PHP

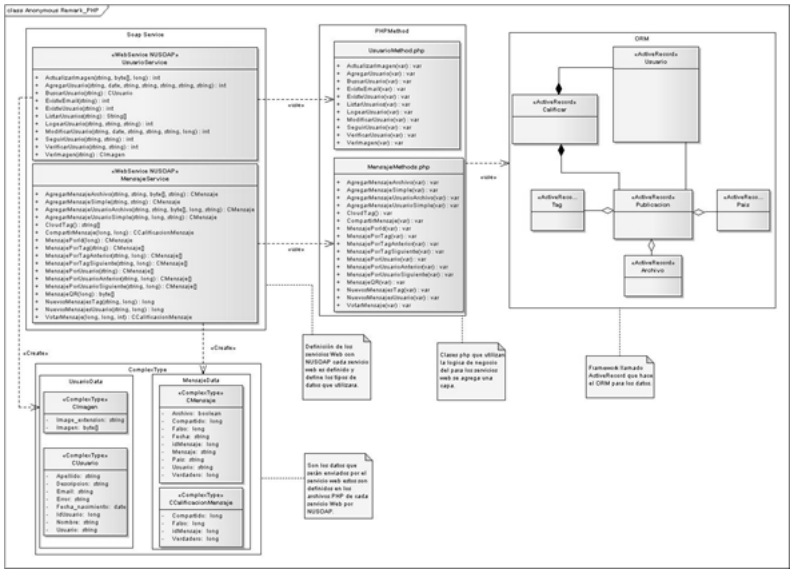

Figura 6.10: Modelo en PHP de la red social AnonymousRemark

1. Se importa la librería de NUSOAP y se crea un objeto de tipo soap_server y se inicializan sus datos.
2. Se crean las funciones que se desean con la lógica de negocios que se requiere para cada uno de los servicios Web.
3. Se registra en el objeto de tipo soap_server las funciones anteriormente definidas que ayudaran con el servicio Web, su nombre con el cual serán consumidas y la información que le devolverán al consumidor del mismo.
4. Finalmente se publica el servicio Web para que este a la escucha de alguna petición.

En el cuadro 6.5 se muestra el código en PHP para realizar un servicio Web con NUSOAP

```
require_once '../lib/nusoap/nusoap.php';
$server = new soap_server;
$ns = 'nombre_de_mi_namespace';
$server->configureWSDL('Mensaje', $ns);
$server->wsdl->schemaTargetNamespace = $ns;
```

```
$server->decode_utf8 = false;
$server->soap_defencoding = 'UTF-8';
function MensajePorTag($tag) {
    $mensaje1= new MensajeOperaciones();
    $mensajes=$mensaje1->MensajePorTag(array('tag'=>$tag));
    return $mensajes;
}
$server->register('MensajePorTag', array(
    'tag' => 'xsd:string'
        ), array('return' => 'tns:CMensajes'), $ns);
        $server->wsdl->addComplexType(
    'CMensajes',
    'complexType',
    'array',
    '',
    'SOAP-ENC:Array',
    array(),
        array(
            array(
                'ref'=>'SOAP-ENC:arrayType',
                'wsdl:arrayType'=>'tns:CMensaje[]'
            )
        ),
    'tns:CMensaje'
    );
    if (!isset($HTTP_RAW_POST_DATA))
    $HTTP_RAW_POST_DATA = file_get_contents('php://input');
$server->service($HTTP_RAW_POST_DATA);
```

Al serializar los datos en PHP con NUSOAP, podremos utilizar los tipos de datos definidos en XML-Schema, y crear un tipo de dato complejo al WSDL. En el Cuadro 6.5 se muestra la manera de serializar los datos.

```
$server->wsdl->addComplexType(
        'CMensaje', 'complexType', // typeClass: complexType,
            simpleType, attribute
        'struct', // phpType: struct, array
        'all', // compositor: all, sequence, choice
        '', array(
    'idMensaje' => array('name' => 'idMensaje', 'type' => 'xsd:long
        '),
    'Usuario' => array('name' => 'Usuario', 'type' => 'xsd:string')
        ,
    'Archivo' => array('name' => 'Archivo', 'type' => 'xsd:boolean'
        ),
    'Fecha' => array('name'=>'Fecha', 'type'=>'xsd:string'),
    'Mensaje' => array('name' => 'Mensaje', 'type' => 'xsd:string')
        ,
    'Verdadero' => array('name' => 'Verdadero', 'type' => 'xsd:long
        '),
    'Falso' => array('name' => 'Falso', 'type' => 'xsd:long'),
```

```
13      'Compartido' => array('name' => 'Compartido', 'type' => 'xsd:
            long'),
14      'Pais' => array('name' => 'Pais', 'type' => 'xsd:string')
15          )
16  );
```

Capítulo 7

Resultados

En este capítulo se hablara sobre los resultados obtenidos al implementar los servicios Web con el enfoque MDA en la creación de la red social AnonymousRemark y los resultados sobre la implementación de la misma ayudándonos de los servicios Web.

Se mostró que el desarrollo con el enfoque MDA es muy potente porque permite entender el problema desde diferentes niveles y crear una estructura sólida de software, ya que proporciona las herramientas necesarias para realizar la ingeniera de un sistema que está a punto de crearse o en su defecto por su naturaleza de modelos, generar la ingeniería inversa de un sistema existente. Por otro lado el utilizar los servicios Web en este desarrollo permitió de una manera transparente crear la capa de la vista debido a que toda la lógica necesaria para esta red social ya se encontraba en los servicios Web.

MDA es un enfoque que permite desarrollar un sistema de software por el hecho de dividir el trabajo y la estructura del mismo en etapas bien definidas, las cuales están especificadas por los modelos PIM (Platform-Independent Model), las cuales describen las características del software que no dependen de ninguna plataforma como la lógica de negocios, las necesidades y requerimientos de la misma, para después realizar o buscar los meta-modelos que ayuden a cumplir las necesidades del software objetivo final o en su defecto, desarrollarlos. En el siguiente diagrama se muestra el proceso realizado para la creación de la red AnonymousRemark con la utilización de los servicios Web y MDA.

Se logró el entendimiento correcto de los servicios Web, que son un conjunto de tecnologías que permiten comunicar software de diferentes plataformas. Las tecnologías necesarias para realizar esta tarea son WSDL, UDDI, SOAP y XML. La Figura 7.2 muestra la tarea que ejecuta cada una de las tecnologías para integrar los servicios Web.

Una vez entendido el concepto de MDA y la tecnología de los servicios Web se realizó la tarea de integrarlos para desarrollar la red social, en la Figura 7.3 se muestran las tareas que se realizaron en cada una de las etapas de esta metodología.

Se generaron diagramas de casos de uso, diagramas de secuencia, diagramas de composición de servicios, meta-modelos, modelos y lenguajes objetivos, todo estos se muestran a grandes rasgos en el la Figura 7.4.

Al realizar estas tareas se generaron los servicios Web con sus propios WSDL con cada tecnología los cuales fueron consumidos por una capa final realizada en PHP. El apéndice B.1 muestra el WSDL del servicio Web de Usuario y en el apéndice B.2 muestra

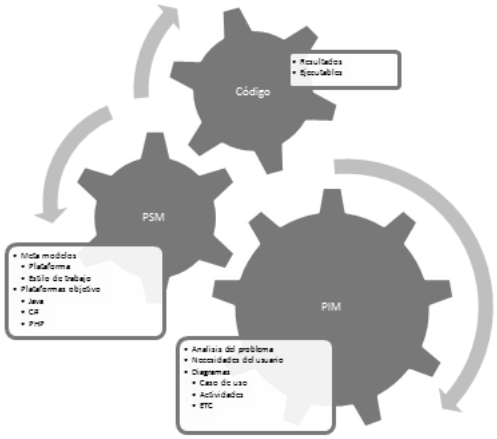

Figura 7.1: Metodología de MDA.

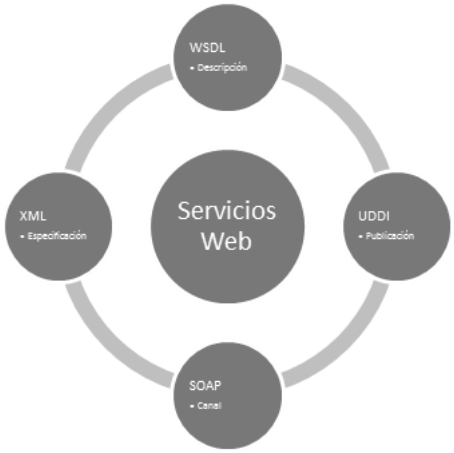

Figura 7.2: Estructura de los servicios Web.

el WSDL del servicio Web de Mensjas.

7.1. Red social AnonymousRemark

La red social AnonymousRemark actualmente se encuentra en línea con la dirección `http://anonymousremark.com` la cual puede ser accedida por cualquier persona que cuente con acceso a internet. Para describir la usabilidad de la red social se realizaron diagramas de navegación que muestran cómo interactúan los diferentes tipos de usuarios. En la Figura 7.5 se muestra el diagrama de navegación del usuario anónimo y la Figura 7.5 se muestra el diagrama de navegación del usuario registrado.

En la Figura 7.7 se muestra la página de inicio de nuestra red, en esta se pueden ver el menú principal en el cual se observa buscar, usuario, id, registrarse y acceder. En el cuerpo se observa la zona para crear una publicación, las marcas más sobresalientes de las publicaciones y en la parte de abajo las publicaciones más actuales.

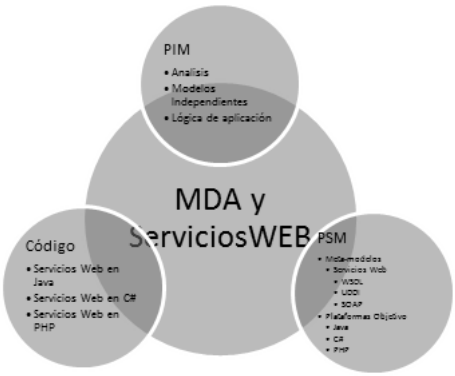

Figura 7.3: Integración de MDA y los servicios Web.

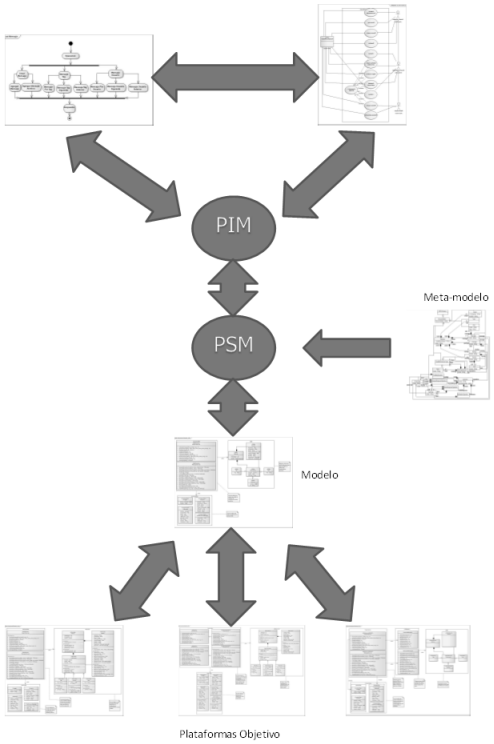

Figura 7.4: Implementación de MDA y los servicios Web para la red social Anonymous-Remark.

Para que un usuario anónimo se pueda registrar en la red debe seguir el menú registrarse que se muestra en la Figura 7.7 y a continuación introducir información básica sobre el usuario que se desea registrar. Este módulo contiene un mecanismo de validación conocido como captcha el cual garantiza que los datos fueron ingresados por un humano con lo cual acepta los términos y condiciones de la red. En la Figura 7.8 se muestra la página de registro de la red.

Una vez que una persona se ha registrado puede acceder a la red, para ello este debe seguir el enlace de acceso que se encuentra en el menú principal para que lo lleve a esta

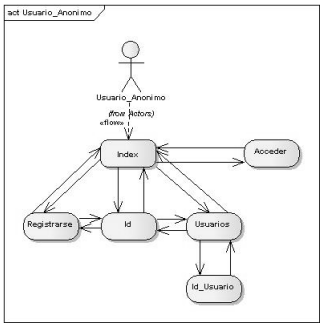

Figura 7.5: Diagrama de navegación del usuario anonimo.

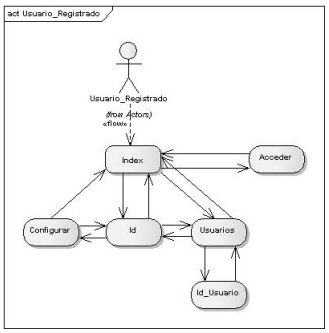

Figura 7.6: Diagrama de navegación del usuario registrado.

sección, una vez en esta sección de la red, el usuario debe introducir su nombre de usuario y su contraseña, si la contraseña y el usuario son correctos el usuario accederá, si son incorrectas se mostrara un mensaje de error. En la Figura 7.9 se muestra la introducción de un usuario y contraseña a la red.

Para realizar una publicación en la red un usuario registrado o no registrado debe introducir la información que desea compartir, teniendo un máximo de 200 caracteres disponibles. Una vez introducido el texto de la publicación el usuario debe validar el captcha (al igual que con el registro para validar que es un ser humano) y presionar el botón de publicar para que se realice. En la Figura 7.10 se muestra una publicación a ser realizada.

Al crear una publicación esta será compartida inmediatamente, los usuarios que están registrados pueden calificar las publicaciones y compartirlas en su cuenta personal. En la Figura 7.11 se muestra la publicación realizada en la Figura 7.10.

Un usuario registrado es capaz de modificar sus datos personales como lo son, su nombre, correo electrónico, email, contraseña entre otros. En la Figura se muestra el panel de configuración de la información de usuario.

Figura 7.7: Página principal de la red social.

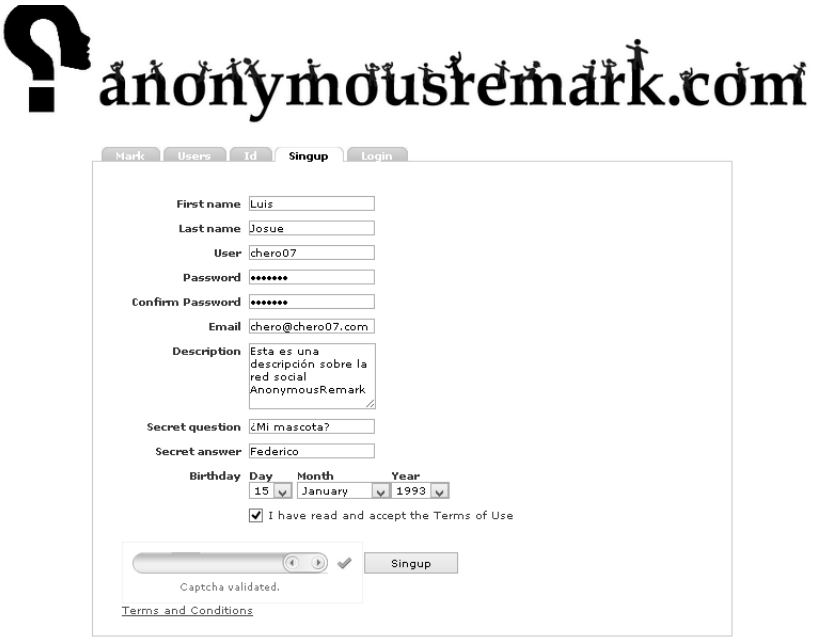

Figura 7.8: Página de registro de la red social.

Figura 7.9: Página de acceso a la red social.

Figura 7.10: Datos para realizar una publicación en la red social.

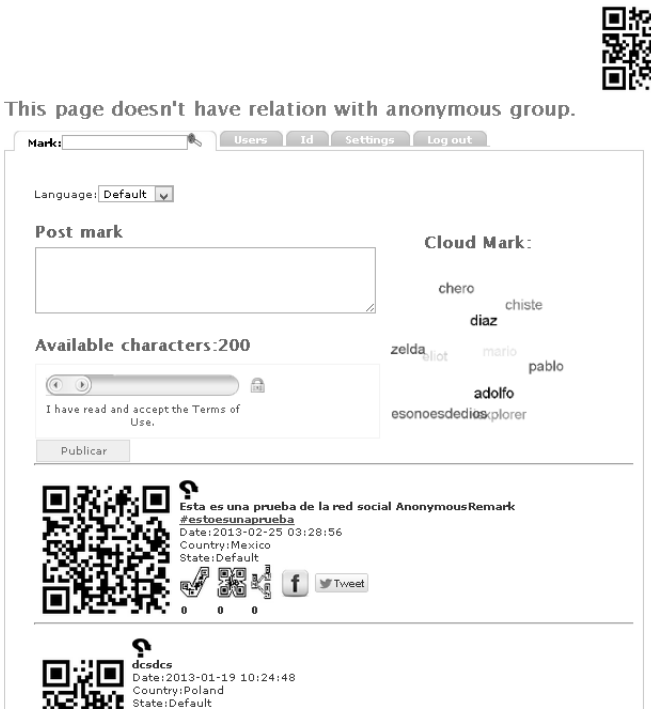

Figura 7.11: Publicación de la Figura 7.10 realizada exitosamente.

Figura 7.12: Configuración de los datos personales de un usuario registrado.

Capítulo 8

Conclusiones y trabajo futuro

Al realizar el estudio del enfoque MDA y los servicios Web para el desarrollo de la red social AnonymousRemark, lo que ayudo a entender el problema al que se enfrentaría al desarrollar la red social, entender el modelo de negocios la estructura y los diferentes tipos de usuario que utilizaran la red social, la implementación de los servicios Web los cuales se desarrollaron en distintas tecnologías objetivo (Java, C#, PHP) y que fueron consumidas de una manera fácil por una vista en HTML.

Si bien MDA es una herramienta que permite realizar un diseño e implementación de un software de una manera rápida, y sencilla, en algunos casos no es tan intuitivo cuando las herramientas a ser utilizadas no son del todo conocidas o no hay compatibilidades entre diferentes plataformas. Entre las ventajas que existen de este enfoque se encuentran que la línea de aprendizaje no es tan larga, permite la implementación de la misma en diferentes áreas de un ciclo de desarrollo de software y permite diferentes plataformas objetivo (C#, Java, Etc). Entre las desventajas que existen para este tipo de desarrollo se encuentran que para usuarios inexpertos en desarrollo de software puede ser complicado el nivel de los diagramas, se debe de conocer bien el modelo y los requerimientos del sistema para realizar correctamente los modelos, se deben conocer bien las plataformas objetivo para realizar correctamente las transformaciones entre las diferentes plataformas.

En cuanto a los servicios Web se puede tomar como un acierto debido a que esto permitirá en trabajos futuros crear diferentes plataformas objetivo como son móviles o de escritorio para que la red social sea más accesible. Aunque actualmente se realizaron servicios Web que se orquestan entre ellos. Es importante mencionar que en estos momentos no se realizaron coreografías de servicios, por lo cual se buscara realizarlo con servicios Web de Facebook y Twitter por ser estas las redes sociales más usadas actualmente.

En la red social se desarrolló un sistema de marcado que permite a los usuarios marcar las publicaciones de manera que estas se relacionen con algún concepto, en trabajos futuros se desarrollara un buscador semántico, es decir que concuerden en ideas o en marcas similares de manera que al realizar una búsqueda con una marca especifica esta de información extra sobre lo que se estaba buscando.

Bibliografía

[1] *Web Service Composition as Planning.*, Junio 2003. [49]

[2] *Decentralized Orchestration of CompositeWeb Services.*, 2004. [49]

[3] Serge ABITEBOUL. Distributed information management with xml and web services. In *7th International Conference Fundamental Approaches to Software Engineering (FASE 2004)*, 2004. [49]

[4] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications.* Data-Centric Systems and Applications. Springer, 2010. [1, 6, 7, 25]

[5] Gustavo Alonso. Myths around web services. In *Bulletin of the Technical Committee on Data Engineering*, pages 3–9. IEEE Computer Society, December 2002. [26]

[6] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1.* IBM, 2003. [24, 49, 88]

[7] Luis E. Anido-Rifón, Manuel Caeiro, Judith S. Rodríguez, and Juan M. Santos. Applying mda concepts to develop a domain corba facility for e-learning. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, UML '02, pages 321–335, London, UK, UK, 2002. Springer-Verlag. [47]

[8] Bharat KUMAR et Arnaud SAHUGUET Aniruddha GOKHALE. Reinventing the wheel corba vs. web services. In *The Eleventh International World Wide Web Conference (WWW2002).*, 2002. [17]

[9] Philip A. BERNSTEIN. *Transaction Processing Monitors. Comm. ACM.* 1990. [11]

[10] Philip A. BERNSTEIN. Applying model management to classical meta data problems. In *Proceedings of the 2003 CIDR*, page 209220, 2003. [37, 43, 46, 54, 58]

[11] Behzad Bordbar, Gareth Howells, Michael Evans, and Athanasios Staikopoulos. Model transformation from owl-s to bpel via sitra. In *Proceedings of the 3rd European conference on Model driven architecture-foundations and applications*, pages 43–58, Berlin, Heidelberg, 2007. Springer-Verlag. [69, 79]

[12] Behzad Bordbar and Athanasios Staikopoulos. On behavioural model transformation in web services. In *ER (Workshops)*, pages 667–678, 2004. [2, 65, 66]

[13] Marlon DUMAS et Quan Z. SHENG Boualem BENATALLAH. *The SELF-SERV Environment for Web Services Composition. IEEE Internet Computing.* Enero 2003. [71]

[14] Mario Bravetti, Roberto Lucchi, Gianluigi Zavattaro, and Roberto Gorrieri. Web services for e-commerce: guaranteeing security access and quality of service. In *in Proc. of the 19th ACM Symposium on Applied Computing (SAC04), special track on E-Commerce Technologies, ACM Press*, pages 800–806, 2004. [49]

[15] Louis Felipe Cabera, George Copeland, Max Feingold, Tom Freund, Jim Johnson, Chris Kaler, Johannes Klein, David Langworthy, Anthony Nadalin, David Orchard, Ian Robinson, Tony Storey, and Satish Thatte. Web Services Atomic Transaction (WS-AtomicTransaction). Technical report, Noviembre 2004. [49]

[16] MICROSOFT. COM. Component object model technologies. Website, 2013. `http://www.microsoft.com/com/default.mspx`. [9]

[17] MIDDLEWARE COMPANY. Model driven development for j2ee utilizing a model driven architecture (mda) approach. Website, 2013. `http://www.theserverside.com/`. [42]

[18] Steve COOK. *Domain-Specific Modeling and Model Driven Architecture. MDA Journal.* Enero 2004. [46, 47, 79]

[19] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web services on demand: Wsla-driven automated management. *IBM Syst. J.*, 43(1):136–158, January 2004. [49]

[20] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin. Business process coordination: State of the art, trends, and open issues. In *in Proceedings of the 27th Very Large Databases Conference (VLDB 2001)*, pages 3–13, 2001. [11]

[21] Jean BÉZIVIN y Frédéric JOUAULT Denivaldo LOPES, Slimane HAMMOUDI. Mapping specification in mda: From theory to practice. In *First International Conference INTEROP-ESA 2005 Interoperability of Enterprise Software and Applications*, Febrero 2005. [2, 81]

[22] David FRANKEL et John PARODI. *Using Model-Driven ArchitectureTM to Develop Web Services. Rapport technique, IONA Technologies PLC.* Abril 2004. [2, 46, 64, 65, 66]

[23] Jean BZIVIN et Nicolas PLOQUIN. *Tooling the MDA Framework: a new Software Maintenance and Evolution Scheme Proposal. Journal of Object-Oriented Programming (JOOP).* Diciembre 2001. [38, 42]

[24] Krzysztof CZARNECKI et Simon HELSEN. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA03 Workshop on Generative Techniques in the Context of MDA.*, Octubre 2004. [2, 43, 60]

[25] Lisa CRISPIN et Tip HOUSE. *Testing Extreme Programming.* Addison-Wesley, 1ra edición. [35]

[26] The Eclipse Foundation. Eclipse modeling framework project (emf). Website, 2013. `http://www.eclipse.org/modeling/emf/`. [33]

[27] Ed MERKS Raymond ELLERSICK et Timothy J. GROSE Frank BUDINSKY, David STEINBERG. *Eclipse Modeling Framework: A Developers Guide. Addison-Wesley Pub Co, 1st édition.* Agosto 2003. [33, 80]

[28] David Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing.* John Wiley & Sons, Inc., New York, NY, USA, 2002. [6]

[29] Doug STEIN Art HARKIN y ET AL Geoof BULLEN, Ash NANGIA. *Open Management Interface (OMI) Specification version 1.0. OASIS.* `http://xml.coverpages.org/OMISpecification-10rev1-OASIS.pdf`. [49]

[30] Anna Gerber, Michael J. Lawley, Kerry Raymond, Jim Steel, and Andrew Wood. Transformation : the missing link of mda. In A. Corrandini, E. Hartmut, H. Kreowski, and G. Rozenberg, editors, *1st International Conference on Graph Transformation*, pages 90–105, Barcelona, Spain, 2002. Springer. [41]

[31] Douglas C. Schmidt Aniruddha Gokhale, Ran Natarajan, Eep Neema, Ted Bapty, Jeff Parsons, Jeff Gray, Andrey Nechypurenko, and Nanbor Wang. Cosmic: An mda generative tool for distributed real-time and embdedded component middleware and applications. In *In Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*. ACM, 2002. [47]

[32] H T GORANSON. *Semantic distance and enterprise integration.* Octubre 2004. [57]

[33] Sridhar IYENGAR James RUMBAUGH y Bran SELIC Grady BOOCH, Alan BROWN. *An mda manifest. MDA Journal.* Mayo 2004. [57, 58]

[34] Roy Gronmo, David Skogan, Ida Solheim, and Jon Oldevik. Model-driven web services development. In *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 42–45, 2004. [2, 67, 69]

[35] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference - Volume 17*, ADC '03, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc. [49]

[36] Jan Hendrik Hausmann and Stuart Kent. Visualizing model mappings in uml. In *SOFTVIS*, pages 169–178, 2003. [2, 53, 54, 55]

[37] D. Hollingsworth. Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition, 1995. [11]

[38] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. Qos aggregation for web service composition using workflow patterns. In *EDOC*, pages 149–159, 2004. [49]

[39] Denivaldo LOPES y Frdric JOUAULT Jean BZIVIN, Slimane HAMMOUDI. Applying mda approach for web service platform. In *8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004).*, page 5870, Septiembre 2004. [1, 38, 39, 46]

[40] Frdric JOUAULT et Patrick VALDURIEZ Jean BZIVIN. First experiments with a modelweaver. In *OOPSLA 2004 - Workshop on Best Practices for Model Driven Software Development.*, 2004. [44]

[41] Frdric JOUAULT Gilles PITETTE y Jamal Eddine ROUGUI Jean BZIVIN, Grgoire DUP. First experiments with the atl model transformation language: Transforming xslt into xquery. In *2nd OOPSLA Workhop on Generative Techniques in the context of Model Driven Architecture.*, Octubre 2003. [43, 60, 63]

[42] Marten van SINDEREN y Luis Ferreira PIRES Joâo Paulo ALMEIDA, Remco DIJKMAN. On the notion of abstract platform in mda development. In *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC 2004)*, page 253263, Septiembre 2004. [38, 77]

[43] Ghica van EMDE BOAS y E.D. WILLINK Jorn BETTIN. Generative model transformer: An open source mda tool initiative. In *Companion of the 18th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003)*, page 294295, 2003. [44]

[44] Matjaz B. Juric, Bostjan Kezmah, Marjan Hericko, Ivan Rozman, and Ivan Vezocnik. Java rmi, rmi tunneling and web services comparison and performance analysis. *SIGPLAN Not.*, 39(5):58–65, May 2004. [17]

[45] Fabio CASATI y Farouk TOUMANI Karim BAINA, Boualem BENATALLAH. Model-driven web service development. In *Procs of 16th International Conference on Advanced Information Systems Engineering - CAiSE 2004, Lecture Notes in Computer Science.*, Junio 2004. [2, 71, 74, 75]

[46] Olaf Kath, Andrei Blazarenas, Marc Born, Klaus-Peter Eckert, Motohisa Funabashi, and Chiaki Hirai. Towards executable models: Transforming edoc behavior models to corba and bpel. In *8th International Enterprise Distributed Object Computing Conference (EDOC 2004), 20-24 September 2004, Monterey, California, USA, Proceedings*, pages 267–274. IEEE Computer Society, 2004. [2, 70, 71, 73, 74]

[47] Anneke G. Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. [35, 79]

[48] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological spaces: An initial appraisal. In *International Symposium on Distributed Objects and Applications, DOA 2002*, 2002. [42]

[49] Sébastien LETÉLIÉ. Les services web asynchrones, Diciembre 2002. [11]

[50] C.J. Leune, W.J.A.M. van den Heuvel, and M. Papazoglou. Exploring a multi-faceted framework for soc: How to develop secure web-service interactions? (an extended abstract). Open access publications from tilburg university, Tilburg University, 2004. [49]

[51] Kees Leune. Specification and querying of security constraints in the efsoc framework, 2004. [49]

[52] Frank Leymann. Web services flow language (wsfl 1.0), Mayo 2001. [24, 25]

[53] Benchaphon Limthanmaphon and Yanchun Zhang. Web service composition transaction management. In *Proceedings of the 15th Australasian database conference - Volume 27*, ADC '04, pages 171–179. Australian Computer Society, Inc., 2004. [49]

[54] Jayant Madhavan, Philip A. Bernstein, Pedro Domingos, and Alon Y. Halevy. Representing and reasoning about mappings between domain models. In *Eighteenth national conference on Artificial intelligence*, pages 80–86. American Association for Artificial Intelligence, 2002. [37, 43]

[55] Daniel J. Mandell and Sheila A. McIlraith. Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, pages 227–241. Springer, 2003. [49, 66]

[56] Jean marie Favre. Towards a basic theory to model model driven engineering. In *In Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*, 2004. [32, 37, 47, 52]

[57] Frank Marschall and Peter Braun. Model transformations for the mda with botl. Technical report, University of Twente, 2003. [60]

[58] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services. Website, 2004. `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/`. [49]

[59] Joanne Martin, Ali Arsanjani, Peri Tarr, and Brent Hailpern. Web services: Promises and compromises. *Queue*, 1(1):48–58, March 2003. [64]

[60] Stephen J. Mellor, Kendall Scott, and Dirk Weise. *MDA distilled: principles of model-driven architecture*. Addison-Wesley, 2004. [45]

[61] MICROSOFT. C# language specification 4.0. Website, 2013. `http://www.microsoft.com/en-us/download/details.aspx?id=7029`. [91]

[62] MSDN. Xml schema data type mapping. Website, Enero 2013. `http://msdn.microsoft.com/en-us/library/aa976860(v=vs.71).aspx`. [102]

[63] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. pages 77–88, 2002. [25]

[64] OASIS. Security assertions markup language version 2.0. oasis. Website, 2013. `http://goo.gl/k8KRS`. [49]

[65] OASIS. Web service security: Soap message security 1.0. Website, 2013. `http://goo.gl/jVJks`. [27, 49]

[66] OMG. Trading object service specification version 1.0, Mayo 2000. [1, 36, 38, 39, 41, 42, 77]

[67] OMG. The common object request broker architecture: Core specification, Diciembre 2002. [9]

[68] OMG. Meta object facility (mof) specification - version 1.4, Abril 2002. [33, 80, 90]

[69] OMG. Request for proposal: Mof 2.0 query/views/transformations rfp, Octubre 2002. [38, 43, 47, 58]

[70] OMG. Uml profile for enterprise distributed object computing specification, Febrero 2002. [30, 89]

[71] OMG. Mda guide version 1.0.1, Junio 2003. [1, 37, 43, 44]

[72] OMG. Unified modeling langage: Superstructure version 2.0 final adopted specification, Junio 2003. [32, 33]

[73] OMG. Xml metadata interchange (xmi) specification, version 2.0, Mayo 2003. [34]

[74] OMG. Uml v 2.0, 2007. [33]

[75] Oracle. Binding xml schemas. Website, Enero 2013. `http://docs.oracle.com/javase/tutorial/jaxb/intro/bind.html`. [100]

[76] Oracle. Java remote method invocation (java rmi),. Website, Enero 2013. `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html`. [9]

[77] Oracle. Java web services developer pack 2.0. Website, Enero 2013. `http://www.oracle.com/technetwork/java/webservicespack-jsp-140788.html`. [90]

[78] Mike P. Papazoglou. Service -oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, WISE '03, pages 3–, Washington, DC, USA, 2003. IEEE Computer Society. [49]

[79] Mike P. Papazoglou. Web services and business transactions. *World Wide Web*, 6(1):49–91, 2003. [49]

[80] Abhijit Patil, Swapna Oundhakar, Amit Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW2004)*, New York, USA, May 2004. LSDIS Lab, University of Georgia. [25]

[81] Octavian Patrascoiu. Mapping edoc to web services using yatl. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*, EDOC '04, pages 286–297, Washington, DC, USA, 2004. IEEE Computer Society. [2, 67, 68, 69]

[82] Octavian Patrascoiu. Yatl: Yet another transformation language. In *University of Twente, the Nederlands*, pages 83–90, 2004. [43, 60]

[83] Marten van SINDEREN y Jos Gonalves Pereira FILHO Patrcia Dockhorn COSTA, Lus Ferreira PIRES. Towards a services platform for mobile context-aware applications. In *Proc. of the First International Workshop on Ubiquitous Computing (IWUC 2004).*, 2004. [49]

[84] M. Peltier. Techniques de transformation de modèles basées sur la méta-modélisation, 2003. [43]

[85] T. Pilioura, A. Tsalgatidou, and S. Hadjiefthymiades. Scenarios of using web services in m-commerce. *SIGecom Exch.*, 3(4):28–36, December 2002. [49]

[86] Rachel A. Pottinger and y al. Merging models based on given correspondences. In *IN VLDB*, pages 826–873, 2003. [2, 55, 56]

[87] JAVA COMMUNITY PROCESS. The $java^{TM}$ metadata interface (jmi) specification. Website, 2013. `http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html`. [61]

[88] J. Mark Pullen, Ryan Brunton, Don Brutzman, David Drake, Michael Hieb, Katherine L. Morse, and Andreas Tolk. Using web services to integrate heterogeneous simulations in a grid environment. *FUTURE GENERATION COMPUTER SYSTEMS*, 21(1):97–106, 2005. [49]

[89] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, December 2001. [55]

[90] Richard E. Schantz and Douglas C. Schmidt. Research advances in middleware for distributed systems: State of the art, 2002. [11]

[91] Giovanni DELLA-LIBERA Siddharth BAJAJ. Web services federation language (ws-federation) version 1.0, Julio 2003. [63]

[92] Oliver SIMS. Enterprise mda or how enterprise systems will be built. Website, Septiembre 2004. `http://www.bptrends.com/search.cfm?keyword=MDA+journal&gogo=1&go.x=68&go.y=4`. [46, 47, 58]

[93] SINTEF. Uml model transformation tool (umt). Website, Enero 2013. `http://sourceforge.net/projects/umt-qvt/`. [2, 68, 70]

[94] David Skogan, Roy Gronmo, and Ida Solheim. Web service composition in uml. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*, EDOC '04, pages 47–57, Washington, DC, USA, 2004. IEEE Computer Society. [2, 68, 71]

[95] Monika Solanki, Antonio Cau, and Hussein Zedan. Augmenting semantic web service descriptions with compositional specification. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 544–552, New York, NY, USA, 2004. ACM. [25]

[96] Jie MENG Raja KRITHIVASAN y Arun JAGATHEESAN Sumi HELAL, Stanley SU. *The Internet Enterprise.* Second IEEE/IPSJ Symposium on Applications and the Internet -SAINT 02, Enero 2002. [49]

[97] Satish THATTE. *XLANG - Web Services for Business Process Design*. Microsoft Corporation, Diciembre 2001. [24, 25]

[98] Andreas Tolk. Avoiding another green elephant - a proposal for the next generation hla based on the model driven architecture. *CoRR*, abs/1011.6671, 2010. [47]

[99] Andreas Tolk and James A. Muguira. M&s within the model driven architecture. In *INTERSERVICE/INDUSTRY TRAINING, SIMULATION, AND EDUCATION CONFERENCE*, 2004. [47]

[100] Jana KOEHLER et Rainer HAUSER Tracy GARDNER, Catherine GRIFFIN. A review of omg mof 2.0 query / views / transformations submissions and recommendations towards the final standard, Julio 2003. [59]

[101] Carnegie Mellon UNIVERSITY. Middleware, cmu/sei-97-hb-001. rapport technique, software engineering institute - carnegie mellon university, 1997. [1, 11, 12]

[102] Carnegie Mellon UNIVERSITY. Remote procedure call, cmu/sei-97-hb-001. rapport technique, software engineering institute - carnegie mellon university, 1997. [11]

[103] Yannis Velegrakis, Renée J. Miller, and Lucian Popa. Mapping adaptation under evolving schemas. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 584–595. VLDB Endowment, 2003. [56]

[104] W3C. Extensible markup language (xml) 1.0. Website, Enero 2013. `http://www.w3.org/TR/1998/REC-xml-19980210`. [19]

[105] W3C. Soap. Website, Enero 2013. `http://www.w3.org/TR/soap/`. [22]

[106] W3C. Web services architecture requirements. Website, Enero 2013. `http://www.w3.org/TR/wsa-reqs/`. [14]

[107] W3C. Web services architecture (wsa). Website, Enero 2013. `http://www.w3.org/TR/2003/WD-ws-arch-20030514/`. [11, 13, 22, 27]

[108] W3C. Web services description language (wsdl) 1.1. Website, Enero 2013. `http://www.w3.org/TR/wsdl`. [83]

[109] W3C. Web services description language (wsdl) version 2.0. Website, Enero 2013. `http://www.w3.org/TR/wsdl20/`. [20, 66]

[110] W3C. Web services glossary. Website, Enero 2013. `http://www.w3.org/TR/ws-gloss/`. [22]

[111] W3C. Xml path language. Website, Enero 2013. `http://www.w3.org/TR/xpath/`. [20]

[112] W3C. Xml schema part0. Website, Enero 2013. `http://www.w3.org/TR/xmlschema-0/`. [20, 98]

[113] Don WELLS. Extreme programming: A gentle introduction. extreme programming. Website, Enero 2013. `http://www.extremeprogramming.org/`. [1, 36]

[114] Andrew Williams, Anand Padmanabhan, and M.B. Blake. Local consensus ontologies for b2b-oriented service composition. In *In AAMAS*, pages 647–654. Press, 2003. [49]

[115] Edward D. Willink. E.d.willink umlx: A graphical transformation language for mda umlx: A graphical transformation language for mda, 2003. [43]

[116] Behzad BORDBAR y Athanasios STAIKOPOULOS. On behavioural model transformation inweb services. In *Proceeding of the 23rd International Conference on Conceptual Modeling (ER2004) and eCOMO 2004.*, Noviembre 2004. [2, 4, 69, 72, 73]

[117] Guy CAPLAT y Jean Louis SOURROUILLE. Model mapping in mda. In *Workshop in Software Model Engineering (WISME2002).*, 2002. [1, 50, 51]

[118] Guy CAPLAT y Jean Louis SOURROUILLE. Considerations about model mapping. In *Workshop in Software Model Engineering.*, Octubre 2004. [1, 52, 53]

[119] Christine PARENT y Stefano SPACCAPIETRA. Intégration de bases de données: Panorama des problémes et des approches, 1996. [46]

Appendices

Apéndice A

Apéndice Figuras

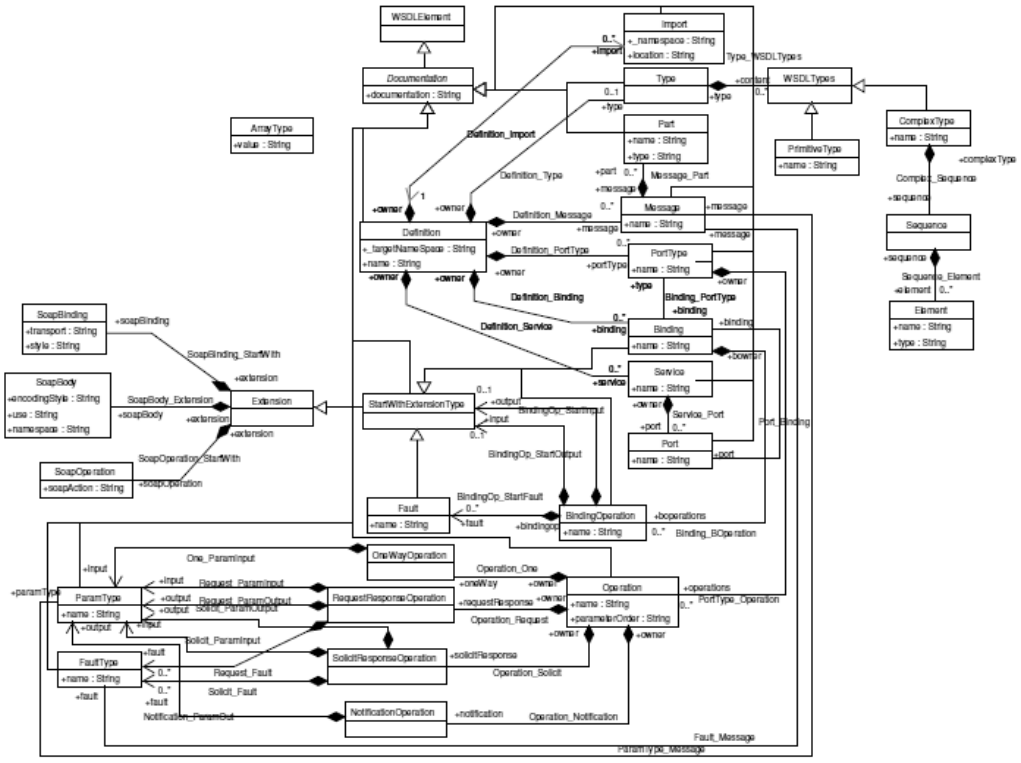

Figura A.1: Meta-modelo completo de WSDL.

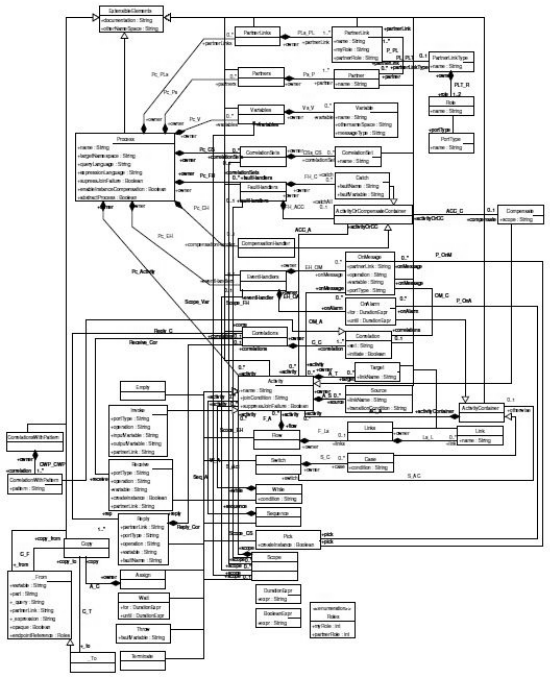

Figura A.2: Meta-modelo completo de BPEL4WS.

XML Schema Type	**Java Data Type**
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd.long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedByte	short
xsd:anySimpleType	java.lang.Object
xsd:duration	javax.xml.datatype.Duration

Cuadro A.1: Tipos compatibles XML Schema y Java

XML Schema Type	**.NET Framework type**
xsd:anyURI	System.Uri
xsd:base64Binary	System.Byte[]
xsd:Boolean	System.Boolean
xsd:Byte	System.SByte
xsd:Date	System.DateTime
xsd:dateTime	System.DateTime
xsd:decimal	System.Decimal
xsd:Double	System.Double
xsd:duration	System.TimeSpan
xsd:ENTITIES	System.String[]
xsd:ENTITY	System.String
xsd:Float	System.Single
xsd:gDay	System.DateTime
xsd:gMonthDay	System.DateTime
xsd:gYear	System.DateTime
xsd:gYearMonth	System.DateTime
xsd:hexBinary	System.Byte[]
xsd:ID	System.String
xsd:IDREF	System.String
xsd:IDREFS	System.String[]
xsd:int	System.Int32
xsd:integer	System.Decimal
xsd:language	System.String
xsd:long	System.Int64
xsd:month	System.DateTime
xsd:Name	System.String
xsd:NCName	System.String
xsd:negativeInteger	System.Decimal
xsd:NMTOKEN	System.String
xsd:NMTOKENS	System.String[]
xsd:nonNegativeInteger	System.Decimal
xsd:nonPositiveInteger	System.Decimal
xsd:normalizedString	System.String
xsd:NOTATION	System.String
xsd:positiveInteger	System.Decimal
xsd:QName	System.Xml.XmlQualifiedName
xsd:short	System.Int16
xsd:string	System.String
xsd:time	System.DateTime
xsd:timePeriod	System.DateTime
xsd:token	System.String
xsd:unsignedByte	System.Byte
xsd:unsignedInt	System.UInt32
xsd:unsignedLong	System.UInt64
xsd:unsignedShort	System.UInt16

Cuadro A.2: Tipos compatibles XML Schema y .NET Framework

Apéndice B

Apéndice WSDL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
    envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP
    -ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="
    service" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://
    schemas.xmlsoap.org/wsdl/" targetNamespace="service">
<types>
<xsd:schema targetNamespace="service">
 <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"
     />
 <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
 <xsd:complexType name="CUsuario">
  <xsd:all>
   <xsd:element name="nombres" type="xsd:string"/>
   <xsd:element name="apellidos" type="xsd:string"/>
   <xsd:element name="email" type="xsd:string"/>
   <xsd:element name="descripcion" type="xsd:string"/>
   <xsd:element name="usuario" type="xsd:string"/>
   <xsd:element name="fecha_nacimiento" type="xsd:date"/>
   <xsd:element name="idUsuario" type="xsd:long"/>
   <xsd:element name="error" type="xsd:string"/>
  </xsd:all>
 </xsd:complexType>
 <xsd:complexType name="CImagen">
  <xsd:all>
   <xsd:element name="imagen" type="xsd:hexBinary"/>
   <xsd:element name="imagen_extension" type="xsd:string"/>
  </xsd:all>
 </xsd:complexType>
 <xsd:complexType name="CListaUsuarios">
  <xsd:complexContent>
   <xsd:restriction base="SOAP-ENC:Array">
```

```
28  <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string
        []"/>
29     </xsd:restriction>
30    </xsd:complexContent>
31   </xsd:complexType>
32  </xsd:schema>
33  </types>
34  <message name="AgregarUsuarioRequest">
35    <part name="nombres" type="xsd:string" />
36    <part name="apellidos" type="xsd:string" />
37    <part name="email" type="xsd:string" />
38    <part name="password" type="xsd:string" />
39    <part name="usuario" type="xsd:string" />
40    <part name="fecha_nacimiento" type="xsd:date" />
41    <part name="descripcion" type="xsd:string" /></message>
42  <message name="AgregarUsuarioResponse">
43    <part name="return" type="xsd:int" /></message>
44  <message name="LogearUsuarioRequest">
45    <part name="usuario" type="xsd:string" />
46    <part name="password" type="xsd:string" />
47    <part name="huella" type="xsd:string" /></message>
48  <message name="LogearUsuarioResponse">
49    <part name="return" type="xsd:int" /></message>
50  <message name="BuscarUsuarioRequest">
51    <part name="usuario" type="xsd:string" /></message>
52  <message name="BuscarUsuarioResponse">
53    <part name="return" type="tns:CUsuario" /></message>
54  <message name="SeguirUsuarioRequest">
55    <part name="usuario1" type="xsd:string" />
56    <part name="usuario2" type="xsd:string" /></message>
57  <message name="SeguirUsuarioResponse">
58    <part name="return" type="xsd:int" /></message>
59  <message name="ModificarUsuarioRequest">
60    <part name="idUsuario" type="xsd:long" />
61    <part name="nombres" type="xsd:string" />
62    <part name="apellidos" type="xsd:string" />
63    <part name="email" type="xsd:string" />
64    <part name="fecha_nacimiento" type="xsd:date" />
65    <part name="descripcion" type="xsd:string" /></message>
66  <message name="ModificarUsuarioResponse">
67    <part name="return" type="xsd:int" /></message>
68  <message name="ModificarPasswordRequest">
69    <part name="idUsuario" type="xsd:long" />
70    <part name="passwordAnterior" type="xsd:string" />
71    <part name="passwordNuevo" type="xsd:string" /></message>
72  <message name="ModificarPasswordResponse">
73    <part name="return" type="xsd:int" /></message>
74  <message name="ActualizarImagenRequest">
75    <part name="idUsuario" type="xsd:long" />
76    <part name="imagen" type="xsd:hexBinary" />
77    <part name="imagen_extension" type="xsd:string" /></message>
```

```
78  <message name="ActualizarImagenResponse">
79    <part name="return" type="xsd:int" /></message>
80  <message name="VerImagenRequest">
81    <part name="usuario" type="xsd:string" /></message>
82  <message name="VerImagenResponse">
83    <part name="return" type="tns:CImagen" /></message>
84  <message name="ListarUsuariosRequest">
85    <part name="usuario" type="xsd:string" /></message>
86  <message name="ListarUsuariosResponse">
87    <part name="return" type="tns:CListaUsuarios" /></message>
88  <message name="VerificarUsuarioRequest">
89    <part name="usuario" type="xsd:string" />
90    <part name="uuid" type="xsd:string" /></message>
91  <message name="VerificarUsuarioResponse">
92    <part name="return" type="xsd:int" /></message>
93  <message name="ExisteUsuarioRequest">
94    <part name="usuario" type="xsd:string" /></message>
95  <message name="ExisteUsuarioResponse">
96    <part name="return" type="xsd:int" /></message>
97  <message name="ExisteEmailRequest">
98    <part name="email" type="xsd:string" /></message>
99  <message name="ExisteEmailResponse">
100   <part name="return" type="xsd:int" /></message>
101 <portType name="UsuarioPortType">
102   <operation name="AgregarUsuario">
103 <input message="tns:AgregarUsuarioRequest"/><output message="
      tns:AgregarUsuarioResponse"/>
104   </operation>
105   <operation name="LogearUsuario">
106 <input message="tns:LogearUsuarioRequest"/>
107 <output message="tns:LogearUsuarioResponse"/>
108   </operation>
109   <operation name="BuscarUsuario">
110 <input message="tns:BuscarUsuarioRequest"/>
111 <output message="tns:BuscarUsuarioResponse"/>
112   </operation>
113   <operation name="SeguirUsuario">
114 <input message="tns:SeguirUsuarioRequest"/>
115 <output message="tns:SeguirUsuarioResponse"/>
116   </operation>
117   <operation name="ModificarUsuario">
118 <input message="tns:ModificarUsuarioRequest"/>
119 <output message="tns:ModificarUsuarioResponse"/>
120   </operation>
121   <operation name="ModificarPassword">
122 <input message="tns:ModificarPasswordRequest"/>
123 <output message="tns:ModificarPasswordResponse"/>
124   </operation>
125   <operation name="ActualizarImagen">
126 <input message="tns:ActualizarImagenRequest"/>
127 <output message="tns:ActualizarImagenResponse"/>
```

```
128   </operation>
129   <operation name="VerImagen">
130 <input message="tns:VerImagenRequest"/>
131 <output message="tns:VerImagenResponse"/>
132   </operation>
133   <operation name="ListarUsuarios">
134 <input message="tns:ListarUsuariosRequest"/>
135 <output message="tns:ListarUsuariosResponse"/>
136   </operation>
137   <operation name="VerificarUsuario">
138 <input message="tns:VerificarUsuarioRequest"/>
139 <output message="tns:VerificarUsuarioResponse"/>
140   </operation>
141   <operation name="ExisteUsuario">
142 <input message="tns:ExisteUsuarioRequest"/>
143 <output message="tns:ExisteUsuarioResponse"/>
144   </operation>
145   <operation name="ExisteEmail">
146 <input message="tns:ExisteEmailRequest"/>
147 <output message="tns:ExisteEmailResponse"/>
148   </operation>
149 </portType>
150 <binding name="UsuarioBinding" type="tns:UsuarioPortType">
151   <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/
        soap/http"/>
152   <operation name="AgregarUsuario">
153 <soap:operation soapAction="http://anonymousremark.com/Service/
      Usuario.php/AgregarUsuario" style="rpc"/>
154 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
155 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
156   </operation>
157   <operation name="LogearUsuario">
158 <soap:operation soapAction="http://anonymousremark.com/Service/
      Usuario.php/LogearUsuario" style="rpc"/>
159 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
160 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
161   </operation>
162   <operation name="BuscarUsuario">
163 <soap:operation soapAction="http://anonymousremark.com/Service/
      Usuario.php/BuscarUsuario" style="rpc"/>
164 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
165 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
166   </operation>
167   <operation name="SeguirUsuario">
```

```
168 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/SeguirUsuario" style="rpc"/>
169 <input><soap:body use="encoded" namespace="service" encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"/></input>
170 <output><soap:body use="encoded" namespace="service" encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/></output>
171   </operation>
172   <operation name="ModificarUsuario">
173 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/ModificarUsuario" style="rpc"/>
174 <input><soap:body use="encoded" namespace="service" encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"/></input>
175 <output><soap:body use="encoded" namespace="service" encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/></output>
176   </operation>
177   <operation name="ModificarPassword">
178 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/ModificarPassword" style="rpc"/>
179 <input><soap:body use="encoded" namespace="service" encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"/></input>
180 <output><soap:body use="encoded" namespace="service" encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/></output>
181   </operation>
182   <operation name="ActualizarImagen">
183 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/ActualizarImagen" style="rpc"/>
184 <input><soap:body use="encoded" namespace="service" encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"/></input>
185 <output><soap:body use="encoded" namespace="service" encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/></output>
186   </operation>
187   <operation name="VerImagen">
188 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/VerImagen" style="rpc"/>
189 <input><soap:body use="encoded" namespace="service" encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"/></input>
190 <output><soap:body use="encoded" namespace="service" encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/></output>
191   </operation>
192   <operation name="ListarUsuarios">
193 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/ListarUsuarios" style="rpc"/>
194 <input><soap:body use="encoded" namespace="service" encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"/></input>
195 <output><soap:body use="encoded" namespace="service" encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/"/></output>
196   </operation>
197   <operation name="VerificarUsuario">
198 <soap:operation soapAction="http://anonymousremark.com/Service/
        Usuario.php/VerificarUsuario" style="rpc"/>
```

```
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="ExisteUsuario">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Usuario.php/ExisteUsuario" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="ExisteEmail">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Usuario.php/ExisteEmail" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
</binding>
<service name="Usuario">
  <port name="UsuarioPort" binding="tns:UsuarioBinding">
<soap:address location="http://anonymousremark.com/Service/Usuario.
    php"/>
  </port>
</service>
</definitions>
```

Figura B.1: WSDL del servicio Web de Usuarios.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
    envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP
    -ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="
    service" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns="http://
    schemas.xmlsoap.org/wsdl/" targetNamespace="service">
<types>
<xsd:schema targetNamespace="service"
 <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"
     />
 <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
 <xsd:complexType name="CMensaje">
  <xsd:all>
   <xsd:element name="idMensaje" type="xsd:long"/>
   <xsd:element name="Usuario" type="xsd:string"/>
```

```
11      <xsd:element name="Archivo" type="xsd:boolean"/>
12      <xsd:element name="Fecha" type="xsd:string"/>
13      <xsd:element name="Mensaje" type="xsd:string"/>
14      <xsd:element name="Verdadero" type="xsd:long"/>
15      <xsd:element name="Falso" type="xsd:long"/>
16      <xsd:element name="Compartido" type="xsd:long"/>
17      <xsd:element name="Pais" type="xsd:string"/>
18     </xsd:all>
19    </xsd:complexType>
20    <xsd:complexType name="CCalificacionMensaje">
21     <xsd:all>
22      <xsd:element name="idMensaje" type="xsd:long"/>
23      <xsd:element name="Verdadero" type="xsd:long"/>
24      <xsd:element name="Falso" type="xsd:long"/>
25      <xsd:element name="Compartido" type="xsd:long"/>
26     </xsd:all>
27    </xsd:complexType>
28    <xsd:complexType name="CMensajes">
29     <xsd:complexContent>
30      <xsd:restriction base="SOAP-ENC:Array">
31  <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="
        tns:CMensaje[]"/>
32      </xsd:restriction>
33     </xsd:complexContent>
34    </xsd:complexType>
35    <xsd:complexType name="CCloudTag">
36     <xsd:complexContent>
37      <xsd:restriction base="SOAP-ENC:Array">
38  <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="xsd:string
        []"/>
39      </xsd:restriction>
40     </xsd:complexContent>
41    </xsd:complexType>
42  </xsd:schema>
43  </types>
44  <message name="AgregarMensajeSimpleRequest">
45    <part name="mensaje" type="xsd:string" />
46    <part name="pais" type="xsd:string" /></message>
47  <message name="AgregarMensajeSimpleResponse">
48    <part name="return" type="tns:CMensaje" /></message>
49  <message name="AgregarMensajeUsuarioSimpleRequest">
50    <part name="mensaje" type="xsd:string" />
51    <part name="usuario" type="xsd:long" />
52    <part name="pais" type="xsd:string" /></message>
53  <message name="AgregarMensajeUsuarioSimpleResponse">
54    <part name="return" type="tns:CMensaje" /></message>
55  <message name="AgregarMensajeArchivoRequest">
56    <part name="mensaje" type="xsd:string" />
57    <part name="archivo" type="xsd:hexBinary" />
58    <part name="pais" type="xsd:string" /></message>
59  <message name="AgregarMensajeArchivoResponse">
```

```
60    <part name="return" type="tns:CMensaje" /></message>
61  <message name="AgregarMensajeUsuarioArchivoRequest">
62    <part name="mensaje" type="xsd:string" />
63    <part name="usuario" type="xsd:long" />
64    <part name="archivo" type="xsd:hexBinary" />
65    <part name="pais" type="xsd:string" /></message>
66  <message name="AgregarMensajeUsuarioArchivoResponse">
67    <part name="return" type="tns:CMensaje" /></message>
68  <message name="MensajePorTagRequest">
69    <part name="tag" type="xsd:string" /></message>
70  <message name="MensajePorTagResponse">
71    <part name="return" type="tns:CMensajes" /></message>
72  <message name="MensajePorTagSiguienteRequest">
73    <part name="tag" type="xsd:string" />
74    <part name="idPrimero" type="xsd:long" /></message>
75  <message name="MensajePorTagSiguienteResponse">
76    <part name="return" type="tns:CMensajes" /></message>
77  <message name="MensajePorTagAnteriorRequest">
78    <part name="tag" type="xsd:string" />
79    <part name="idAnterior" type="xsd:long" /></message>
80  <message name="MensajePorTagAnteriorResponse">
81    <part name="return" type="tns:CMensajes" /></message>
82  <message name="MensajePorUsuarioRequest">
83    <part name="usuario" type="xsd:string" /></message>
84  <message name="MensajePorUsuarioResponse">
85    <part name="return" type="tns:CMensajes" /></message>
86  <message name="MensajePorUsuarioSiguienteRequest">
87    <part name="usuario" type="xsd:string" />
88    <part name="idPrimero" type="xsd:long" /></message>
89  <message name="MensajePorUsuarioSiguienteResponse">
90    <part name="return" type="tns:CMensajes" /></message>
91  <message name="MensajePorUsuarioAnteriorRequest">
92    <part name="usuario" type="xsd:string" />
93    <part name="idAnterior" type="xsd:long" /></message>
94  <message name="MensajePorUsuarioAnteriorResponse">
95    <part name="return" type="tns:CMensajes" /></message>
96  <message name="MensajePorIdRequest">
97    <part name="idMensaje" type="xsd:long" /></message>
98  <message name="MensajePorIdResponse">
99    <part name="return" type="tns:CMensaje" /></message>
100 <message name="MensajeQRRequest">
101   <part name="idMensaje" type="xsd:long" /></message>
102 <message name="MensajeQRResponse">
103   <part name="return" type="xsd:hexBinary" /></message>
104 <message name="CompartirMensajeRequest">
105   <part name="idUsuario" type="xsd:long" />
106   <part name="idMensaje" type="xsd:long" /></message>
107 <message name="CompartirMensajeResponse">
108   <part name="return" type="tns:CCalificacionMensaje" /></message>
109 <message name="VotarMensajeRequest">
110   <part name="idUsuario" type="xsd:long" />
```

```
111   <part name="idMensaje" type="xsd:long" />
112   <part name="idVoto" type="xsd:int" /></message>
113 <message name="VotarMensajeResponse">
114   <part name="return" type="tns:CCalificacionMensaje" /></message>
115 <message name="CloudTagRequest"></message>
116 <message name="CloudTagResponse">
117   <part name="return" type="tns:CCloudTag" /></message>
118 <message name="NuevosMensajesTagRequest">
119   <part name="tag" type="xsd:string" />
120   <part name="idPrimero" type="xsd:long" /></message>
121 <message name="NuevosMensajesTagResponse">
122   <part name="return" type="xsd:long" /></message>
123 <message name="NuevosMensajesUsuarioRequest">
124   <part name="usuario" type="xsd:string" />
125   <part name="idPrimero" type="xsd:long" /></message>
126 <message name="NuevosMensajesUsuarioResponse">
127   <part name="return" type="xsd:long" /></message>
128 <portType name="MensajePortType">
129   <operation name="AgregarMensajeSimple">
130 <input message="tns:AgregarMensajeSimpleRequest"/>
131 <output message="tns:AgregarMensajeSimpleResponse"/>
132   </operation>
133   <operation name="AgregarMensajeUsuarioSimple">
134 <input message="tns:AgregarMensajeUsuarioSimpleRequest"/>
135 <output message="tns:AgregarMensajeUsuarioSimpleResponse"/>
136   </operation>
137   <operation name="AgregarMensajeArchivo">
138 <input message="tns:AgregarMensajeArchivoRequest"/>
139 <output message="tns:AgregarMensajeArchivoResponse"/>
140   </operation>
141   <operation name="AgregarMensajeUsuarioArchivo">
142 <input message="tns:AgregarMensajeUsuarioArchivoRequest"/>
143 <output message="tns:AgregarMensajeUsuarioArchivoResponse"/>
144   </operation>
145   <operation name="MensajePorTag">
146 <input message="tns:MensajePorTagRequest"/>
147 <output message="tns:MensajePorTagResponse"/>
148   </operation>
149   <operation name="MensajePorTagSiguiente">
150 <input message="tns:MensajePorTagSiguienteRequest"/>
151 <output message="tns:MensajePorTagSiguienteResponse"/>
152   </operation>
153   <operation name="MensajePorTagAnterior">
154 <input message="tns:MensajePorTagAnteriorRequest"/>
155 <output message="tns:MensajePorTagAnteriorResponse"/>
156   </operation>
157   <operation name="MensajePorUsuario">
158 <input message="tns:MensajePorUsuarioRequest"/>
159 <output message="tns:MensajePorUsuarioResponse"/>
160   </operation>
161   <operation name="MensajePorUsuarioSiguiente">
```

```
<input message="tns:MensajePorUsuarioSiguienteRequest"/>
<output message="tns:MensajePorUsuarioSiguienteResponse"/>
  </operation>
  <operation name="MensajePorUsuarioAnterior">
<input message="tns:MensajePorUsuarioAnteriorRequest"/>
<output message="tns:MensajePorUsuarioAnteriorResponse"/>
  </operation>
  <operation name="MensajePorId">
<input message="tns:MensajePorIdRequest"/>
<output message="tns:MensajePorIdResponse"/>
  </operation>
  <operation name="MensajeQR">
<input message="tns:MensajeQRRequest"/>
<output message="tns:MensajeQRResponse"/>
  </operation>
  <operation name="CompartirMensaje">
<input message="tns:CompartirMensajeRequest"/>
<output message="tns:CompartirMensajeResponse"/>
  </operation>
  <operation name="VotarMensaje">
<input message="tns:VotarMensajeRequest"/>
<output message="tns:VotarMensajeResponse"/>
  </operation>
  <operation name="CloudTag">
<input message="tns:CloudTagRequest"/>
<output message="tns:CloudTagResponse"/>
  </operation>
  <operation name="NuevosMensajesTag">
<input message="tns:NuevosMensajesTagRequest"/>
<output message="tns:NuevosMensajesTagResponse"/>
  </operation>
  <operation name="NuevosMensajesUsuario">
<input message="tns:NuevosMensajesUsuarioRequest"/>
<output message="tns:NuevosMensajesUsuarioResponse"/>
  </operation>
</portType>
<binding name="MensajeBinding" type="tns:MensajePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/
      soap/http"/>
  <operation name="AgregarMensajeSimple">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/AgregarMensajeSimple" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="AgregarMensajeUsuarioSimple">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/AgregarMensajeUsuarioSimple" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
```

```
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="AgregarMensajeArchivo">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/AgregarMensajeArchivo" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="AgregarMensajeUsuarioArchivo">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/AgregarMensajeUsuarioArchivo" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="MensajePorTag">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/MensajePorTag" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="MensajePorTagSiguiente">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/MensajePorTagSiguiente" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="MensajePorTagAnterior">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/MensajePorTagAnterior" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="MensajePorUsuario">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/MensajePorUsuario" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
```

```
239   </operation>
240   <operation name="MensajePorUsuarioSiguiente">
241 <soap:operation soapAction="http://anonymousremark.com/Service/
      Mensaje.php/MensajePorUsuarioSiguiente" style="rpc"/>
242 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
243 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
244   </operation>
245   <operation name="MensajePorUsuarioAnterior">
246 <soap:operation soapAction="http://anonymousremark.com/Service/
      Mensaje.php/MensajePorUsuarioAnterior" style="rpc"/>
247 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
248 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
249   </operation>
250   <operation name="MensajePorId">
251 <soap:operation soapAction="http://anonymousremark.com/Service/
      Mensaje.php/MensajePorId" style="rpc"/>
252 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
253 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
254   </operation>
255   <operation name="MensajeQR">
256 <soap:operation soapAction="http://anonymousremark.com/Service/
      Mensaje.php/MensajeQR" style="rpc"/>
257 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
258 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
259   </operation>
260   <operation name="CompartirMensaje">
261 <soap:operation soapAction="http://anonymousremark.com/Service/
      Mensaje.php/CompartirMensaje" style="rpc"/>
262 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
263 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
264   </operation>
265   <operation name="VotarMensaje">
266 <soap:operation soapAction="http://anonymousremark.com/Service/
      Mensaje.php/VotarMensaje" style="rpc"/>
267 <input><soap:body use="encoded" namespace="service" encodingStyle="
      http://schemas.xmlsoap.org/soap/encoding/"/></input>
268 <output><soap:body use="encoded" namespace="service" encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/"/></output>
269   </operation>
270   <operation name="CloudTag">
271 <soap:operation soapAction="http://anonymousremark.com/Service/
```

```
    Mensaje.php/CloudTag" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="NuevosMensajesTag">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/NuevosMensajesTag" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
  <operation name="NuevosMensajesUsuario">
<soap:operation soapAction="http://anonymousremark.com/Service/
    Mensaje.php/NuevosMensajesUsuario" style="rpc"/>
<input><soap:body use="encoded" namespace="service" encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/"/></input>
<output><soap:body use="encoded" namespace="service" encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"/></output>
  </operation>
</binding>
<service name="Mensaje">
  <port name="MensajePort" binding="tns:MensajeBinding">
<soap:address location="http://anonymousremark.com/Service/Mensaje.
    php"/>
  </port>
</service>
</definitions>
```

Figura B.2: WSDL del servicio Web de Mensajes.