Universidad Nacional de Río Cuarto

Trabajo de tesis para la obtención del grado de Licenciatura en Ciencias de la Computación

# QVTSInvoker
# Automatización y Optimización de la invocación de aplicaciones externas a un Workflow, usando servicios web y transformaciones de modelos.

Autores
Jaimez Jacinto
Pereyra Orcasitas Nicolás

Director de Tesis: *Mg. Marcela Daniele*
Co-Director de Tesis: *Mg. Ariel Gonzalez*

Río Cuarto, Argentina
Septiembre 2016

# Resumen

La tecnología de Web Service (WS) permite invocar aplicaciones externas, por ejemplo desde un motor de workflow y proporcionar beneficios significativos en el Workflow Management System (WFMS). Con el gran número de WS's que proporcionan una funcionalidad similar, es importante encontrar el mejor WS que cumpla con las necesidades del usuario, incluyendo tanto sus requisitos funcionales como los no funcionales. La descripción no funcional del Servicio requiere especificar en tiempo de ejecución los atributos de calidad que pueden influir en la elección de un WS ofrecido por un proveedor. En este sentido, es fundamental el uso de métricas para evaluar las características de calidad de los WS's (QoS) con el fin de filtrar los WS's conocidos y obtener el más adecuado.

El comportamiento dinámico de los WS's en relación con el desarrollo de nuevos servicios y el constante cambio de los existentes requiere un proceso de evaluación continuo, que conduce a capturar información del WS con respecto a su calidad y rendimiento de evaluación conforme a lo solicitado por el workflow. En este trabajo se realiza un refinamiento del proceso de contrastación de WS ofrecidos por proveedores y solicitados por usuarios del workflow, se definen y usan métricas para evaluar los atributos de calidad de dichos servicios, y se propone una herramienta para automatizar este procedimiento de selección de aplicaciones externas al workflow

*Palabras clave*: Web Service o WS, workflow, características de calidad o QoS

# Agradecimientos

*Queremos agradecer a todas las personas que han hecho posible la realización de esta tesis. A nuestros directores de tesis Marcela Daniele y Ariel Gonzalez, que han permitido llevar a un final exitoso este trabajo, por su asesoría, dedicación y sus valiosos aportes guiándonos sin ser directivas y mostrando en cada momento una inmejorable disposicón ante las dudas que durante la realización del trabajo nos surgieron, aportando siempre valiosas observaciones que tutelaron esta investigación. A nuestras familias y amigos, por su incondicional apoyo durante este largo camino.*

*Además, debemos retribuir con cordial gratitud tanto a la Universidad Nacional de Río Cuarto como a la Facultad de Ciencias Exactas, Físico-Químicas y Naturales, y en especial al Departamento de Computación, por permitirnos desarrollar nuestras formaciones individuales de grado con excelente material académico, pero sobre todo humano durante todos estos años.*

# Índice de figuras

# Índice general

# Capítulo 1

# Introducción

La globalización y los cambios de paradigmas empresariales, la evolución de las tecnologías de la información y la política liberal imperante hoy en el mundo, ubican a las organizaciones en el juego de la competitividad internacional. Las iniciativas sobre calidad y la mejora continua de los procesos ya no son suficientes. Se deben considerar nuevos paradigmas empresariales que permitan a las organizaciones obtener mejoras radicales mediante el uso de nuevas y potentes herramientas que faciliten el diseño del trabajo. Las organizaciones se orientan a ser más horizontales hacia el enfoque de redes de procesos, las que deben ser diseñadas de principio a fin, empleando nuevas tecnologías. El cambio radical de procesos comprende la visualización de nuevas estrategias de trabajo, el desarrollo de la propia actividad de diseño del proceso con tecnologías de la información innovadoras y la implementación del cambio en todas sus dimensiones: la tecnológica, la humana y la organizativa, a fin de establecer los mecanismos para un constante crecimiento del valor de las organizaciones. Para lograr un adecuado cambio integral de los procesos, es conveniente confeccionar un mapa de los mismos a fin de determinar los niveles de cambios a realizar. Por lo tanto, la identificación y el modelado de negocio necesario para generar un producto o ejecutar un servicio, es de gran importancia en el desarrollo de cualquier industria. Un conocimiento claro y ordenado de los procesos de negocio facilita su optimización y adaptación. Estas características proporcionan a la organización una gran capacidad de reacción cuando el proceso es automático.

Un proceso de negocio es un conjunto de tareas relacionadas lógicamente que se ejecuta con la intención de obtener un resultado de negocio particular, el cual incluye recursos humanos así como los recursos materiales con el objetivo de producir un beneficio para la organización. El modelado de procesos de negocio permite visualizar las tareas, actividades y flujos, así como las diferentes unidades organizacionales que son afectadas por el proceso. La ingeniería de procesos de negocio se ha definido como el planteo fundamental y diseño del proceso de negocio para lograr un mejoramiento en las medidas de rendimientos tales como costo, calidad de servicio y velocidad. Estos procesos de negocio se describen y automatizan a través de los Workflow Management System (WFMS) . Los WFMS proporcionan herramientas integradas para automatizar los procesos de negocio, permitiendo a las organizaciones estandarizar y racionalizar las tareas repetitivas, y supervisar el progreso de las mismas, eliminando rutinas en las que es fácil cometer errores. La eliminación de los procesos que consumen tiempo y las costosas rutinas manuales, brinda la posibilidad de conectar al personal con la información y los procesos que necesitan para mejorar los ingresos y recortar los gastos [14].

Por otra parte, el ambiente donde las organizaciones operan es cada vez más competitivo y agresivo. En estos días, debido a la globalización de los mercados, las compañías tienen que operar "globalmente", lo cual implica la posibilidad de interoperar unas con otras. En la actualidad existe la Coalición de Administración de Workflow (Workflow Management Coalition - WfMC), fundada en 1993 y con más de 180 miembros en 25 países, la cual está abocada al avance en la tecnología de workflow y su uso en la industria con el propósito de estandarizar en esta materia. El Modelo de Referencia de Workflow, desarrollado por la WfMC, define un marco genérico para la construcción de WFMS, permitiendo la interoperabilidad entre ellos y con otras aplicaciones involucradas. Dicho modelo define cinco interfaces, que permiten a las aplicaciones del workflow la comunicación a distintos niveles. En particular, para permitir la interacción de los usuarios con el motor del workflow utiliza una lista de trabajo, que es manejada por un administrador. La Interfaz de las Aplica-

ciones de Clientes es la encargada de manejar la interacción entre el motor del workflow y el administrador de la lista de trabajo. Por otro lado, la Interfaz de las Aplicaciones Invocadas se define para la invocación de las aplicaciones externas. La WfMC ha especificado un conjunto de WAPI's (Workflow Aplication Programming Interfaces) para la administración de los Workflow. Estas WAPI's definen las funciones de las interfaces como llamadas a APIs en un lenguaje de tercera generación, obligando a conocer la información acerca de la aplicación y su invocación en tiempo de desarrollo. Si bien en muchos WFMS's se conocen a priori las aplicaciones que se desean utilizar, también existen sistemas donde diferentes aplicaciones que brindan un mismo servicio podrían ser requeridas por el motor del WFMS [12].

Los Web Services (WS) proveen esencialmente un medio estándar de comunicación entre diferentes aplicaciones de software, ofreciendo la capacidad de acceder a servicios heterogéneos de forma unificada e interoperable a través de Internet. Los WS's pueden ser registrados a través del protocolo denominado Universal Description, Discovery and Integration (UDDI) para ser localizados y usados por las aplicaciones. Este protocolo presenta algunos inconvenientes al momento de seleccionar un servicio [6]. Uno de estos inconvenientes está dado por el hecho de que no tiene en cuenta las características de calidad (QoS) de los WS's, las cuales se refieren a la habilidad del servicio de responder a las invocaciones y llevarlas a cabo en consonancia con las expectativas del proveedor y de los clientes. Diversos factores de calidad que reflejan las expectativas del cliente, como la disponibilidad, conectividad y alta respuesta, se vuelven clave para mantener un negocio competitivo y viable. Otro de los inconvenientes que presenta el protocolo UDDI es la dificultad de establecer una correspondencia entre los requerimientos del solicitante y las múltiples especificaciones de WS's existentes en la actualidad que proveen similares funcionalidades, aunque tienen diferente descripción sintáctica.

Esta tesis apunta a optimizar la selección de WS's conocidos usando técnicas de transformación de modelos y a desarrollar una herramienta que haga uso de este método, para que por ejemplo, sea utilizado para automatizar la comunicación de los WFMS con aplicaciones externas.

## 1.1. Objetivos

La presente Tesis de grado tiene como objetivo permitir un manejo transparente y automático de aplicaciones externas basadas en servicios web. Para lograrlo, se propone especificar la interfaz que permite dicha comunicación con WS, y valerse de la técnica de transformación de modelo para encontrar el WS más adecuado según los requerimientos del usuario, es decir, la construcción de un mecanismo que optimice el proceso de selección e invocación a WS's y la automatización del mismo.

La herramienta obtenida puede ser utilizada como un middleware desde la interfaz de aplicaciones externas de un Workflow, considerando que los Workflow's son una de las aplicaciones que se ven más beneficiadas con este desarrollo, no obstante, puede ser utilizada en otros contextos.

En función de ello, se propone la realización e implementación de una aplicación que, dado un listado de WS's categorizados y el tipo de la aplicación que se necesite invocar, los requerimientos del sistema y ciertas normas de calidad, localice, seleccione y ejecute el WS disponible más adecuado que cumpla con los requerimientos especificados. De esta forma, el Workflow se abstrae tanto de la ubicación específica como de la aplicación misma y su ejecución, lo cual simplifica su definición.

Concretamente se define a continuación un conjunto de metas a alcanzar:

- A fin de definir un mecanismo de comparación y selección de WS's se proporciona una especificación precisa (modelo) que represente los requerimientos tanto del Workflow como del WS.
- Definir un método cuantitativo de ponderación de los WS en función de sus QoS.
- Implementar una aplicación que, de acuerdo a los requerimientos del WFMS y un listado de WS's conocidos, proporcione el WS más adecuado que cumpla con estos requisitos. Y que a su vez, la obtención y ejecución del WS resulte transparente al motor de Workflow.

### 1.1.1. Estructura del documento

El documento se ha estructurado como se muestra a continuación:

- El capítulo 1 comprende una introducción general a los principales temas abordados por este trabajo y hace mención de los trabajos relacionados con esta investigación.
- El capítulo 2 describe las nociones preliminares y teoría general que dan la base a la problemática planteada.
- El capítulo 3 describe el método que permite la evaluación de calidad de servicio de los WS's.
- El capítulo 4 exhibe las transformaciones QVT necesarias para llevar a cabo el mecanismo de preselección de WS's.
- El capítulo 5 presenta el caso de estudio, relacionado al proceso de desarrollo de software a realizar y la invocación de aplicaciones externas durante la ejecución de las tareas propuesta en este proceso.
- El capítulo 6 presenta las conclusiones de la Tesis.
- Finalmente se presentan los anexos del trabajo.

## 1.2. Trabajos relacionados

### 1.2.1. Comunicación de los WFMS con las aplicaciones externas

La Interfaz 3 de un Workflow permite la comunicación (en la actualidad estática) entre este tipo de sistemas y el software disponible en el exterior de los mismos, no sólo a nivel invocación sino también a la hora de transferir datos en formatos entendibles por ambos componentes y la obtención e interpretación de los resultados [13]. Para llevar a cabo esta comunicación, la presente tesis se apoya en la propuesta de la maestría de Paola Martelloto [7], donde la autora propone una Especificación Funcional y otra no funcional de la Interfaz de Aplicaciones Invocadas, y de esta forma mejorar la selección de aplicaciones en tiempo de ejecución, permitiendo al motor del WFMS invocarlas dinámicamente independizándose de la ubicación exacta de las mismas. En la especificación funcional plantea definir las WAPI's de la Interfaz de Aplicaciones Invocadas con WS's que declaren la información de entrada, los datos de salida y los posibles errores en la invocación. Esto no sólo brinda mayor flexibilidad al motor del WFMS sino que también permite el requerimiento y la actualización de datos de la aplicación y otras funcionalidades importantes en tiempo de ejecución. En la especificación no funcional de la Interfaz de Aplicaciones Invocadas, propone incorporar al proceso de selección de WS's el análisis de los atributos de calidad requeridos por el solicitante. De esta forma se complementa la selección en tiempo de ejecución con la obtención del mejor WS disponible que satisfaga las necesidades del solicitante. Esta tesis propone una optimización y automatización del mecanismo presentado en [7].

### 1.2.2. Búsqueda e invocación de Web Services

Existen diversas herramientas que implementan formas de buscar e invocar WS's tanto en servidores locales como en servidores distribuidos en la web. A continuación se mencionan y describen algunas herramientas que fueron estudiadas en este trabajo:

**JUDDI Console** [1] es una herramienta web que trabaja localmente sobre un servidor de aplicaciones el cual permite al desarrollador visualizar y comprender de forma transparente cómo se manipulan los WS's en un servidor UDDI. Entre todas sus funcionalidades se destacan las siguientes:

- find_service() realiza una búsqueda sintáctica sobre los WS's alojados en el servidor local.
- get_serviceDetail(), como su nombre lo especifica, retorna el detalle de un WS determinado identificado por su clave.
- save_service() se encarga de almacenar y publicar un WS en el servidor.

En la tesis de licenciatura de Gil-Garat [8] utiliza dos plantillas genéricas, la primera es de entrada. Consiste de un conjunto de parámetros que son los encargados de transportar los requerimientos del usuario a la herramienta. Teniendo en cuenta que este usuario será en reiteradas ocasiones un WFMS, el conjunto de parámetros ha sido planteado simple y estándar. Esta plantilla está formada por los siguientes atributos: tipo de WS, el o los nombres posibles del mismo, nombres de la operación a ejecutar con sus respectivos parámetros (valores para los cuales se deseen obtener los resultados correspondientes) y las normas de calidad que el WS debe cumplir. Esta parametrización le posibilita a la herramienta acceder de manera estándar a los requerimientos del sistema siempre de una misma forma, para luego comenzar el proceso de búsqueda con los criterios y filtros correspondientes.

La salida de la herramienta depende del tipo de WS seleccionado. Para cada uno de estos tipos se define una plantilla de salida que consiste del conjunto de atributos necesarios para poder retornar el resultado obtenido de la ejecución. De esta manera, para un mismo caso, es decir un mismo tipo de WS's, no importa que aplicación provea el resultado, siempre el retorno de la herramienta será una instancia de la plantilla.

En el esquema propuesto en [5], el componente principal del motor WFMS es un agente de descubrimiento de servicios basado en los requisitos funcionales y QoS especificadas por el usuario y los requisitos mínimos establecidos por el propio motor. De este modo, el motor proporciona una clasificación de los WS's disponibles y elige el más adecuado. También es responsable de la recolección y procesamiento de información de los WS's ofrecidos por los proveedores y de mantener esta información actualizada a medida que se realizan las llamadas. Para llevar a cabo la asignación y selección final, el motor realiza los siguientes pasos

- Especificar pre- y post-condición del solicitante que se refieren a los atributos de calidad del WS;
- Especificar pre y post-condiciones de los proveedores de WS que contienen información que está disponible en el registro UDDI, refiriéndose principalmente a las características funcionales del WS.

Al realizar el proceso de contraste a nivel semántico el motor del workflow busca en el registro UDDI y obtiene una lista preliminar de los WS's que responden a las exigencias funcionales del solicitante.

Al realizar el proceso de contraste en el nivel de calidad de servicio, el motor analiza las características no funcionales considerando los QoS del solicitante. Además, evalúa las características especificadas en el registro de QoS del motor del workflow si el WS previamente se ha invocado, el motor del workflow busca en su historia y utiliza la información almacenada del mismo. Si el WS no se ha invocado antes, se analizan las características durante el proceso de verificación. Este proceso es un filtro de la lista preliminar de candidatos obtenidos previamente el cual devuelve el WS que mejor se adapte a las exigencias del solicitante. Si más de un WS cumple con las características exigidas por los requisitos, el motor del workflow aplicará unos criterios adicionales para elegir uno de ellos.

Por último, el motor del workflow invoca el WS seleccionado y almacena la ejecución en la historia de las ejecuciones anteriores.

En el paper Dynamic search and selection of WS's [10] presenta la selección de WS's compuestos y complejos de forma dinámica de acuerdo a las necesidades del usuario utilizando diferentes métodos de búsqueda y procesamiento de WSDL. La metodología utilizada para la búsqueda de los términos solicitados son búsquedas simples, busca los requisitos del usuario en un solo WS; también búsqueda horizontal y vertical la cual intenta combinar y componer distintos WS's con el objetivo de satisfacer las necesidades del usuario. La búsqueda se realiza en base al apoyo (support) y la confianza (confidence) de los términos solicitados en los resultados de búsqueda.

# Capítulo 2

# Nociones Preliminares

En este capítulo se exhiben los principales conceptos sobre las áreas de base que conforman este trabajo, estas son: Workflow y Workflow Management System (WFMS), Web Services (WS), el lenguaje WSDL, el Protocolo de Acceso a Objetos Simple (Simple Object Access Protocol - SOAP) y desarrollo y desarrollo dirigido por modelos.

## 2.1. Workflows

Uno de los problemas que se encuentra habitualmente en el desarrollo de aplicaciones para Negocios, es que las tareas o procesos que se desarrollan en el entorno laboral de las mismas quedan inmersos en el código de la aplicación que resuelve la problemática del Negocio. Está claro que la gran mayoría de los usuarios no tienen conocimiento de estas tareas, las mismas están ocultas a sus ojos y se realizan automáticamente. El hecho de realizar cambios en dichas tareas o procesos resulta muy costoso, y es muy factible que dichos cambios redunden en realizar nuevamente la aplicación.

Una buena solución al problema anterior es separar los procedimientos y asociarlos a los Workflows realizados dentro de la Negocio. El Workflow se relaciona con la automatización de los procedimientos donde los documentos, la información o tareas son pasadas entre los participantes del sistema de acuerdo a un conjunto de reglas previamente establecidas. El fin de lo anterior es llegar a culminar una meta común impuesta por la Negocio.

Se puede ver al Workflow como un conjunto de métodos y tecnologías que ofrece las facilidades para modelar y gestionar los diversos procesos que ocurren dentro de una Negocio. El Workflow es el último, de una gran línea de facilidades propuestas en respuesta de las exigencias de las organizaciones, las cuales apuntan a poder reaccionar tan rápido como sea posible ante la frenética demanda de la competición. Además, permite automatizar diferentes aspectos del flujo de la información: rutear los trabajos en la secuencia correcta, proveer acceso a datos y documentos, y manejar ciertos aspectos de la ejecución de un proceso.

La diversidad de procesos que puede haber en una organización lleva a pensar en la existencia de diferentes tipos de software de Workflow. El Workflow ofrece a un negocio la posibilidad de automatizar sus procesos, reducir costos, y mejorar servicios, obvios beneficios. Organizaciones que no hayan evaluado esta tecnología podrían encontrarse con desventajas en un futuro.

## 2.2. Workflow Management Coalition

La WfMC es una agrupación compuesta por compañías, vendedores, organizaciones de usuarios, y consultores. El objetivo de esta agrupación es ofrecer una forma de "diálogo" común a todos. De esta forma las diferentes herramientas que se implementen en esta área podrán tener cierto nivel de interoperabilidad, es decir, podrán comunicarse entre ellas para poder realizar las distintas tareas involucradas en un sistema de

Workflow.

De acuerdo a la definición de la WfMC, en un WFMS existen dos actividades bien diferenciadas. Por un lado, se tiene la definición del Workflow que implementa el proceso de negocio, lo que se denomina modelado del Workflow. Este modelado consiste en la definición de un conjunto de actividades relacionadas y un conjunto de criterios que indican el comienzo y la finalización del proceso. Además, se definen sus componentes e información adicional sobre cada actividad tal como participantes, invocación de aplicaciones, datos, etc. En el modelado participan los siguientes elementos:

- Proceso de Negocio: un conjunto de uno o más procedimientos o actividades que colectivamente realizan un objetivo o política global de una organización.
- Definición del Proceso: la representación de un proceso de negocio en una forma que soporta la manipulación automática. La definición consiste de un conjunto de actividades y sus relaciones.
- Actividades: la descripción de un trozo de trabajo que forma un paso lógico dentro de un proceso.
- Actividades Automáticas: las que soportan la automatización por computadora.
- Actividades Manuales: las que no soportan la automatización por computadora y por lo tanto quedan fuera de alcance del WFMS.

Por un lado, está la ejecución de dicho modelo. La ejecución del Workflow en un WFMS consiste en la creación, manipulación y destrucción de instancias de proceso, que representan al Workflow de acuerdo a la especificación realizada en el modelado. Cada instancia de proceso tendrá una identidad visible externamente y un estado interno que representa su progreso hacia la finalización y su estado con respecto a las actividades que lo componen. Cada vez que la ejecución de un proceso implique la invocación de una actividad, se crea una instancia de actividad que la representa. Dicha instancia se encarga de ejecutar las acciones asignadas, accediendo a los datos que sean necesarios e invocando la aplicación externa correspondiente, si así lo requiere la actividad. Los elementos que participan en la ejecución son:

- Instancias de Proceso: representan un proceso simple, incluyendo sus datos asociados.
- Instancias de Actividad: representan una actividad dentro de una instancia de proceso.
- Ítem de Trabajo: es la representación de una tarea a ser procesada por un participante del Workflow, en el contexto de una actividad dentro de una instancia de proceso. Cada tarea es un conjunto de acciones manejadas como una sola unidad. Generalmente son desempeñadas por una única persona dentro de los roles que pueden realizar dicha tarea. Las tareas surgen del análisis del Workflow, donde se define por quienes deben ser ejecutadas.
- Invocación de Aplicaciones: es la representación de las aplicaciones del Workflow que son invocadas por el WFMS para automatizar una actividad.

## 2.3. Invocación de Aplicaciones Externas a un WFMS según el Modelo de referencia de la WfMC

La WfMC ha estandarizado la automatización de los procesos de negocio. Dicho estándar define un marco genérico para la construcción de WFMS, permitiendo la interoperabilidad entre ellos y con otras aplicaciones. Entonces, el modelo de referencia de Workflow fue desarrollado desde estructuras genéricas de aplicaciones de Workflow, identificando las interfaces con estas estructuras, de forma de permitir a los productos comunicarse a distintos niveles. Todos los sistemas de Workflow contienen componentes genéricas que interactúan de forma definida. Para poder tener cierto nivel de interoperabilidad entre los diversos productos de Workflow, es necesario definir un conjunto de interfaces y formatos para el intercambio de datos entre dichas componentes [12].

En la figura 2.1 se muestran las distintas interfaces y componentes que se pueden encontrar en la arquitectura del Workflow.

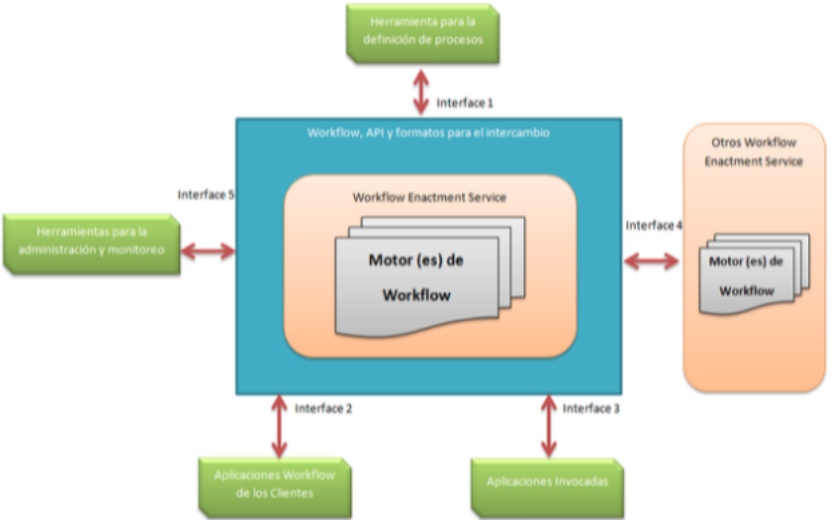


Figura 2.1: Interfaces del Modelo de Referencia.

En el modelo adoptado hay una separación entre los procesos y el control de la lógica de las actividades. Esta lógica está dentro del Workflow Enactment Service. Esta separación permite la integración de las diversas herramientas con una aplicación particular. La interacción del Enactment Service con los recursos externos se da por una de las dos interfaces siguientes:

- La interface de las Aplicaciones de los Clientes, a través de la cual el Motor de Workflow interactúa con el manejador de la Worklist, responsable de organizar el trabajo por intermedio de un recurso de usuario. Es responsabilidad del manejador del Worklist elegir y hacer progresar cada elemento de la lista de trabajo (Worklist).
- La interfaz de las Aplicaciones Invocadas, la cual le permite al motor de Workflow activar una herramienta para realizar una actividad particular. Esta interface podría ser basada en un servidor, es decir no existe la interacción con el usuario.

El Motor de Workflow (Workflow Engine) es el software que provee el control del ambiente de ejecución de una instancia de Workflow. Típicamente dicho software provee facilidades para:

- Interpretación de la definición de procesos.
- Control de las instancias de los procesos: creación, activación, terminación, etc. Navegación entre actividades.
- Soporte de interacción con el usuario.
- Pasaje de datos al usuario o a aplicaciones.
- Invocación de aplicaciones externas.

Por otra parte, las WAPI pueden ser vistas como un conjunto de llamadas API (Application Programming Interface) y funciones de intercambio de datos soportadas por el Workflow Enactment Service. Las APIs son un conjunto de llamadas a funciones de software que permiten a las aplicaciones acceder a funciones de un programa. Las WAPI permiten la interacción del Workflow Enactment Service con otros recursos y aplicaciones.

## 2.4. Web Service

El término Web Service se utiliza muy a menudo hoy en día, aunque no siempre con el mismo significado. Los conceptos y tecnologías subyacentes de los WS's son en gran medida independientes de la forma en que puedan ser interpretadas. Las definiciones existentes para el concepto de WS, van desde la muy genérica y con todo incluido a la muy específica y restrictiva.

Entonces, se puede comenzar por definir a un WS como un estándar de comunicación entre procesos y/o componentes, diseñado para ser multiplataforma y multilenguaje, es decir, no importa en qué lenguaje esté programado un WS como ser Visual Basic, C# o Java, o en qué plataforma esté corriendo, ya sea Windows, UNIX o Linux éstos serán accesibles y utilizables por otras aplicaciones desarrolladas en otras plataformas o lenguajes de programación. Es decir, diversas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los WS's para intercambiar datos tanto en redes locales como en Internet. Esta interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones, esencialmente la W3C, son los comités responsables que se encargan de la arquitectura y la reglamentación de los WS's.

Precisamente la W3C define a los WS's como "Aplicaciones de software identificadas mediante una URI (Uniform Resource Identifier), cuyas interfaces (y uso) son capaces de ser definidas, descritas y descubiertas mediante artefactos XML, y soportar interacciones directas con otras aplicaciones software usando mensajes basados en XML y protocolos basados en Internet".

En el mundo de los WFMS, tanto los usuarios como los administradores de Workflow usualmente requieren la utilización de aplicaciones externas para desarrollar sus tareas, por ejemplo, aplicaciones que le permitan utilizar servicios de e-mail, fax, realizar la administración de sus documentos, u otras aplicaciones propias del usuario como obtener fechas, climas o cálculos matemáticos provistos desde la web.

Por otra parte, los WS's pueden ser agrupados según las funcionalidades que ofrecen en: de Información de negocios (pronóstico del tiempo, calendario, noticias, chequeo de crédito de una tarjeta, subastas, cotizaciones de acciones, planificación de vuelos, etc.); de Transacciones (reservas aéreas, acuerdos para rentar un auto, orden de compra, gestión de una cadena de suministro, etc.); o de Externalización de procesos de negocio (vínculos comerciales a nivel de Workflow, etc.).

Los WS's se describen en términos de sus operaciones y de los mensajes de entrada y salida de cada una de esas operaciones que provee. Tales definiciones se expresan en el lenguaje de marcado extensible XML (Extensible Markup Language) [15], usando un lenguaje de descripción de WS's: WSDL (Web Service Description Language) [18] [19]. La noción de describir un WS independientemente de la tecnología en la cual ha sido implementado es robustamente capturada en este lenguaje. WSDL especifica claramente la ubicación del servicio junto con las operaciones que necesitan ser invocadas, los parámetros que deben pasarse, los tipos de datos de cada parámetro, y los distintos formatos de mensajes y tipos.

Así, el uso de protocolos estándares como WSDL en el contexto de los WS's es necesario sin excepciones para poder lograr la interoperabilidad en estos ambientes heterogéneos, con independencia del sistema operativo, el lenguaje de programación, etc. Otros estándares que se utilizan al hablar de WS's son:

- **Protocolo HTTP (Hiper Text Transport Protocol) [3]:** HTTP es la abreviatura de HyperText Transfer Protocol, HTTP es el protocolo utilizado por la World Wide Web. HTTP define cual es el formato de los mensajes, y cómo se transmiten los mismos. HTTP es un protocolo sin estado, se lo llama así ya que cada comando se ejecuta de forma independiente, sin ningún conocimiento de los comandos que vinieron antes que él. Esta es la razón principal por la que es difícil de implementar sitios Web que reaccionan de forma inteligente a la entrada del usuario. Esta deficiencia de HTTP está siendo abordada en una serie de nuevas tecnologías, incluyendo ActiveX, Java, JavaScript y las cookies.
- **Protocolo SOAP (Simple Object Access Protocol) [16]:** SOAP (Simple Object Access Protocol) es el protocolo base de los WS's. Este protocolo está basado en XML y no se encuentra atado a ninguna plataforma o lenguaje de programación. A su vez, también es el protocolo más aceptado por

la mayoría de las plataformas. Si bien SOAP es un protocolo, éste no es exactamente un protocolo de comunicación entre mensajes como lo es el HTTP, por ejemplo. Básicamente, SOAP son documentos XML y necesitaremos utilizar algún otro protocolo para la transmisión de estos documentos como ser el protocolo HTTP o cualquier otro protocolo de comunicación capaz de transmitir textos. En la figura 2.2 podemos ver como SOAP esa compuesto.

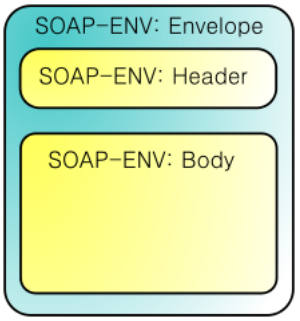


Figura 2.2: Composición del protocolo SOAP.

- **UDDI (Universal Description, Discovery and Integration) [6]:** Se utiliza para el registro y la localización de cualquier WS. Se trata de un directorio o catálogo de WS's distribuido que permite que se listen, busquen y descubran este tipo de aplicaciones de software. UDDI tiene como objetivo ser accedido por los mensajes SOAP y dar paso a documentos WSDL.

Cada uno de estos protocolos se asocian a una capa, las cuales en conjunto, definen la infraestructura de los WS's.

HTTP: Corresponde La capa de transporte utilizada para el envío de la información;

SOAP: Corresponde a la capa de mensajes.

WSDL: Corresponde a la capa de descripción.

UDDI: Corresponde a la capa de procesos que se encargan de la publicación, descubrimiento, composición, orquestación, desarrollo e integración de WS's. Al desaparecer los registos UDDI publicos, esta tesis plantea una solución usando una base conocida de WS's.

### 2.4.1. El lenguaje WSDL

WSDL (Web Service Description Language) es el lenguaje de especificación utilizado para definir los WS's. Se trata de un documento XML que se utiliza para describir los mensajes SOAP y cómo estos mensajes son intercambiados. Es decir, supongamos que creamos un WS y queremos que otras aplicaciones lo utilicen, las otras aplicaciones deberán acceder a un documento WSDL en donde podrán conocer los métodos que expone nuestro WS y cómo acceder a ellos, es decir, cuáles son los nombres de los métodos y qué tipos de parámetros espera cada uno de ellos. Entonces, en este sentido, cualquier documento WSDL permite proveer información acerca de la ubicación de cierto WS y su interfaz. Así, con esta información respecto a la descripción y especificación del WS, el usuario sabrá básicamente cómo llevar a cabo la interacción con el servicio.

Hay que tener en cuenta que todo documento WSDL separa la descripción de un servicio en dos partes [17]. Dentro de cada una de estas partes, se utilizan un número de constructores que promueven la reusabilidad de la descripción y permiten separarla de los detalles de diseño. Una de ellas es la interfaz abstracta, la cual describe las operaciones soportadas por el servicio, los parámetros de la operación y los tipos de datos abstractos. Esta descripción es completamente independiente de la dirección de red concreta, el protocolo de comunicación o las estructuras de datos concretas del servicio. La otra parte es la implementación concreta, la cual liga la interfaz abstracta a una dirección de red, protocolo y estructuras de datos

concretas. WSDL provee toda esta información a través de elementos XML.

Por un lado, en la Interfaz Abstracta los dos elementos a definir son:

1. **types:** provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar, sus operaciones los parámetros de entrada o salida de cada operación.
2. **interface:** describe la secuencia de mensajes que el servicio envía o recibe.

Y por el otro, los dos elementos a definir en la implementación concreta son:

1. **binding:** define los detalles de implementación concretos del servicio.
2. **service:** se usa para asociar un binding con el URL donde está realmente el servicio (endpoint).

Entonces cualquier documento WSDL especifica tanto la definición como una descripción del WS que representa. Pero también define los dos componentes principales de estos, es decir, sus características funcionales y sus características no funcionales. La descripción funcional se refiere a las características operacionales que definen el comportamiento general de un WS. En otras palabras, define los detalles de cómo es invocado el servicio, su ubicación, etc. Esta descripción focaliza en los detalles de la sintaxis de los mensajes y cómo configurar los protocolos de red para enviar estos mensajes. Y el otro componente, la descripción no funcional, se concentra en las características de calidad (QoS) del WS.

### 2.4.2. Características de Calidad de los Web Services

Los requerimientos de un WS no deben focalizar solamente en las propiedades funcionales del mismo, sino que también se deben concentrar en describir el ambiente en el que se desarrolla, es decir, describir las capacidades no funcionales o QoS del servicio [11]. Cada servicio puede ofrecer diversas opciones de características no funcionales basadas en los requerimientos técnicos como: disponibilidad, performance y escalabilidad, políticas de seguridad y privacidad, etc., todas las cuales deben ser descriptas.

Los elementos clave que se tienen en cuenta para soportar las QoS en el mundo de los WS's son:

- **Rendimiento o throughput:** Número de requerimientos de WS's atendidos en un período de tiempo dado.
- **Tiempo de Respuesta:** Tiempo esperado entre que se envía un requerimiento y se recibe una respuesta.
- **Latencia:** Tiempo entre que un requerimiento de un servicio arriba y el requerimiento comienza a ser atendido.
- **Tiempo de Ejecución:** Es el tiempo que necesita un WS para procesar su secuencia de actividades.
- **Tiempo de Transacción:** Representa el tiempo que transcurre mientras el WS está completando una transacción.
- **Disponibilidad:** Probabilidad de que el WS esté disponible.
- **Fiabilidad:** Habilidad de un servicio para funcionar correcta y consistentemente y proveer la misma calidad a pesar de las fallas en la red o el sistema.
- **Precisión:** Radio de error producido por un servicio.
- **Robustez:** Grado en el cual un servicio puede funcionar correctamente en presencia de entradas inválidas, incompletas o conflictivas.
- **Reputación:** Medida de la reputación otorgada al servicio por el usuario final.

Tradicionalmente, las QoS se miden por el grado en el cual las aplicaciones, sistemas, redes y otros elementos de Internet soportan la disponibilidad de los servicios a un nivel requerido de performance, en todos los accesos y condiciones de descarga. En el contexto de los WS's, las QoS se pueden ver como la capacidad de ofrecer garantía sobre un conjunto de características cuantitativas, referidas específicamente a los aspectos no funcionales de los servicios. Estas características son un requerimiento necesario para entender el comportamiento subyacente del servicio de manera tal que otras aplicaciones y servicios puedan invocarlos y ejecutarlos como parte de su proceso de negocio.

En este trabajo se propone una forma de incorporar las QoS en los WS's a partir de la necesidad de tenerlas en cuenta a la hora de optimizar el proceso de selección del WS más adecuado. Esta incorporación ofrece importantes beneficios, tanto para los usuarios como los proveedores. A los usuarios les permite expresar sus necesidades y utilizar el WS que mejor se ajusta sus necesidades, mientras que los proveedores pueden publicar mejor las capacidades de sus servicios y lucirse con las mismas.

## 2.5. Desarrollo de software dirigido por modelos

El desarrollo de software dirigido por modelos surge como respuesta a los principales problemas que actualmente tienen las compañías de desarrollo de software, por un lado, para gestionar la creciente complejidad de los sistemas que construyen y mantienen, y por otro para adaptarse a la rápida evolución de las tecnologías software.

En primer lugar, se usan modelos para representar tanto los sistemas como los propios artefactos del software. Cada modelo trata un aspecto del sistema, que puede ser especificado a un nivel más elevado de abstracción y de forma independiente de la tecnología utilizada. Este hecho permite que la evolución de las plataformas tecnológicas sea independiente del propio sistema, disminuyendo así sus dependencias.

En segundo lugar, se consigue también proteger gran parte de la inversión que se realiza en la informatización de un sistema, pues los modelos son los verdaderos artífices de su funcionamiento final y, por tanto, no será necesario empezar desde cero cada vez que se plantee un nuevo proyecto o se desee realizar algún tipo de mantenimiento sobre el producto.

En tercer lugar, en MDA los modelos constituyen la propia documentación del sistema, disminuyendo considerablemente el coste asociado y aumentando su mantenibilidad, puesto que la implementación se realiza de forma automatizada a partir de la propia documentación.

### 2.5.1. Modelos

De forma sencilla podríamos definir un modelo como una abstracción simplificada de un sistema o concepto del mundo real. Un modelo de un cierto <X>es una especificación o descripción de ese <X>desde un determinado punto de vista, expresado en un lenguaje bien definido y con un propósito determinado.

Según Bran Selic, uno de los fundadores del MDD y pionero en el uso de sus técnicas, los modelos deberían ser:

- **Adecuados:** Construidos con un propósito concreto, desde un punto de vista determinado y dirigidos a un conjunto de usuarios bien definido.
- **Abstractos:** Enfatizan los aspectos importantes para su propósito a la vez que ocultan los aspectos irrelevantes.
- **Comprensibles:** Expresados en un lenguaje fácilmente entendible para sus usuarios.
- **Precisos:** Representan fielmente al objeto o sistema modelado.
- **Predictivos:** Pueden ser usados para responder preguntas sobre el modelo e inferir conclusiones correctas.
- **Rentables:** Han de ser más fáciles y baratos de construir y estudiar que el propio sistema.

Bran Selic también señala las principales funciones que los modelos deberían tener en el ámbito de la ingeniería del software:

- **Comprender el sistema**
  - La estructura, el comportamiento y cualquier otra característica relevante de un sistema y su entorno desde un punto de vista dado.
  - Separar adecuadamente cada uno de los aspectos, describiendolos al nivel conceptual adecuado.
- **Servir de mecanismo de comunicación**
  - Con los distintos tipos de stakeholder del sistema (desarrolladores, usuarios finales, personal de soporte y mantenimiento, etc.).
  - Con las otras organizaciones (proveedores y clientes que necesitan comprender el sistema a la hora de interoperar con él).
- **Validar el sistema y su diseño**
  - Detectar errores, omisiones y anomalías en el diseño tan pronto como sea posible (cuanto antes se detecten, menos cuesta corregirlos).
  - Razonar sobre el sistema, infiriendo propiedades sobre su comportamiento (en caso de modelos ejecutables que puedan servir como prototipos).
  - Poder realizar análisis formales sobre el sistema.
- **Guiar la implementación**
  - Servir como "planos" para construir el sistema y que permitan guiar su implementación de una forma precisa y sin ambigüedades.
  - Generar, de la forma más automática posible, tanto el código final como todos los artefactos necesarios para implementar, configurar y desplegar el sistema.

### 2.5.2. Metamodelos

Un metamodelo es un modelo que especifica los conceptos de un lenguaje, las relaciones entre ellos y las reglas estructurales que restringen los posibles elementos de los modelos válidos, así como aquellas combinaciones entre elementos que respetan las reglas semánticas del dominio. De esta forma, cada modelo se escribe en el lenguaje que define su metamodelo (su lenguaje de modelado), quedando establecida la relación entre el modelo y su metamodelo por una relación de "conformidad" (y diremos que un modelo es "conforme a" un metamodelo). Esta relación se puede observar en la figura 2.3.

### 2.5.3. Terminología

La comunidad utiliza en la actualidad un conjunto de acrónimos referidos a diferentes enfoques relacionados con la ingeniería del software que usa modelos como elementos clave de sus procesos: MDD, MBE, MDA, etc. Aunque algunos de ellos ya se han mencionado en secciones anteriores, en este apartado trataremos de aclarar estos conceptos y las diferencias entre ellos.

MDD (model-driven development) es un paradigma de desarrollo de software que utiliza modelos para diseñar los sistemas a distintos niveles de abstracción, y secuencias de transformaciones de modelos para generar unos modelos a partir de otros hasta generar el código final de las aplicaciones en las plataformas destino [9].

MDA (model-driven architecture) es la propuesta concreta de la OMG para implementar MDD, usando las notaciones, mecanismos y herramientas estándares definidos por esa organización [4].

Los estándares que la OMG ofrece para realizar MDA incluyen, entre otros:

- UML (unified modeling language) como lenguaje de modelado.
- MOF (meta-object facility) como lenguaje de metamodelado.
- OCL (object constraint language) como lenguaje de restricciones y consulta de modelos.

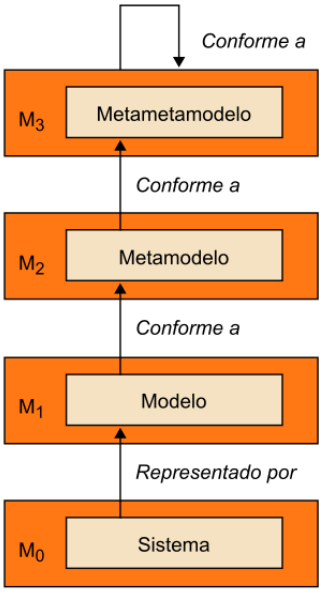


Figura 2.3: Relación modelo metamodelo.

- QVT (query-view-transformation) como lenguaje de transformación de modelos.
- XMI (XML metadata interchange) como lenguaje de intercambio de información.

MDE (model-driven engineering) es un paradigma dentro de la ingeniería del software que aboga por el uso de los modelos y las transformaciones entre ellas como piezas clave para dirigir todas las actividades relacionadas con la ingeniería del software.

MBE (model-based engineering) es un término general que describe los enfoques dentro de la ingeniería del software que usan modelos en algunos de sus procesos o actividades.

BPM (business process modeling) es una rama del model-based engineering que se centra en el modelado de los procesos de negocio de una empresa u organización, de forma independiente de las plataformas y las tecnologías utilizadas.

Architecture-driven modernization (ADM) es una propuesta de la OMG para implementar prácticas de ingeniería inversa utilizando modelos.

Model driven-interoperability (MDI) es una iniciativa para implementar mecanismos de interoperabilidad entre servicios, aplicaciones y sistemas usando modelos y técnicas de MBE.

En la figura 2.4 se puede observar la relación entre los contextos de MDE, MDD, MDA.

### 2.5.4. Transformaciones de modelos

#### La necesidad de las transformaciones de modelos

Las transformaciones de modelos proporcionan los mecanismos que permiten especificar el modo de producir modelos de salida a partir de modelos de entrada.
El concepto de transformación en el mundo de la informática no es nuevo, de hecho, las transformaciones son algo fundamental en la ingeniería del software, pudiendo verse la computación como un proceso de transformación de datos.
Entre las aplicaciones de las transformaciones de modelo en el ámbito de MDE nos encontramos:

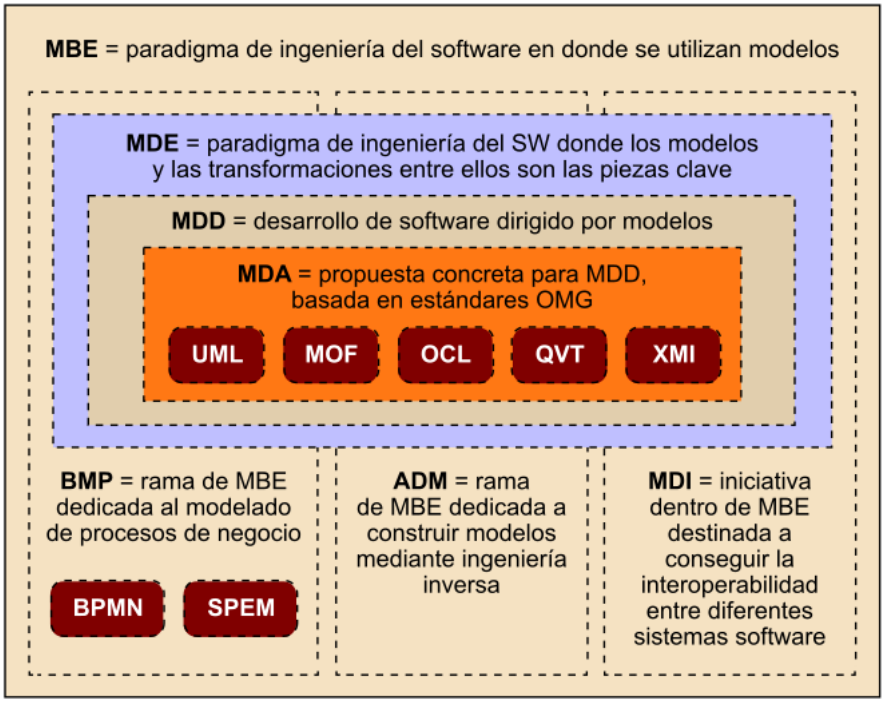


Figura 2.4: Relación entre los contextos de MDE, MDD y MDA.

- Generación de modelos a más bajo nivel, y finalmente código, a partir de modelos a alto nivel.
- Sincronización y mapping entre modelos al mismo o diferentes niveles de abstracción.
- Creación de vistas basadas en queries sobre un sistema.
- Aplicación a la ingeniería inversa para obtener modelos a más alto nivel a partir de modelos a más bajo nivel.

### Conceptos básicos

Las transformaciones de modelos pueden verse como programas que toman un modelo (o más de uno) como entrada y retornan otro modelo (o más de uno) como salida.
Generalmente, una transformación de modelos está formada por un conjunto de reglas de transformación, que son consideradas como las unidades de transformación más pequeñas. Cada una de las reglas describe cómo uno o más elementos del modelo origen son transformados en uno o más elementos del modelo destino.

La figura 2.5 demuestra como se lleva a cabo una transformación de modelo.

### Tipos de transformaciones

En MDE se describen diferentes tipos de transformaciones entre modelos, dependiendo de varios criterios:

a) Nivel de abstracción de los modelos de entrada y salida.

- Transformaciones verticales. Relacionan modelos del sistema situados en distintos niveles de abstracción y pueden aplicarse tanto en sentido descendente como en sentido ascendente, siendo estos últimos el resultado de aplicar un proceso de ingeniería inversa.
- Transformaciones horizontales. Relacionan los modelos que describen un sistema desde un nivel de abstracción similar. Se utilizan también para mantener la consistencia entre distintos modelos de un sistema, es decir, garantizan que la información modelada sobre una entidad en un modelo es

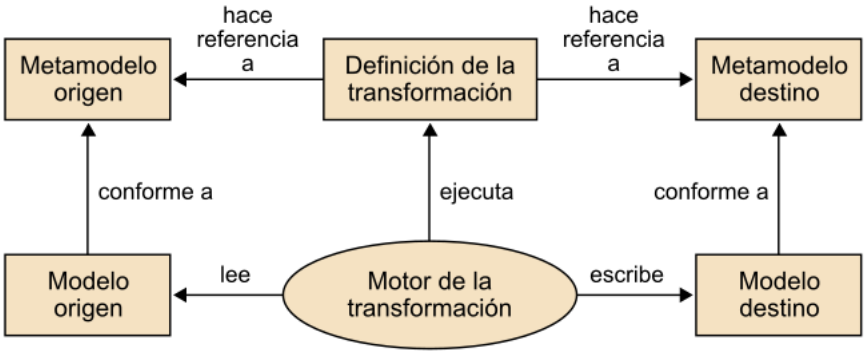


Figura 2.5: Transformación de modelo.

consistente con lo que se dice sobre dicha entidad en cualquier otra especificación situada al mismo nivel de abstracción.

b) Tipo de lenguaje que se utiliza para especificar las reglas. Se distingue entre lenguajes declarativos, imperativos o híbridos (si permiten ambos tipos de forma combinada).

c) Atendiendo a la direccionalidad, nos encontramos con:

- Transformaciones unidireccionales, las reglas se ejecutan en una sola dirección.
- Transformaciones bidireccionales. En estas, las transformaciones se pueden aplicar en ambas direcciones .

d) Dependiendo de los modelos origen y destino:

- Transformaciones exógenas. Los metamodelos origen y destino son distintos. Es el caso más común: se tiene un modelo de entrada y se obtiene el de salida a partir de él aplicando las reglas de la transformación.
- Transformaciones endógenas. Aquí, los metamodelos origen y destino son el mismo, se trata de las llamadas transformaciones in-place: las reglas de transformación se van aplicando sobre el propio modelo de entrada, de modo que este se va transformando hasta que no se puede aplicar ninguna regla más y el modelo de entrada pasa a ser el de salida.

e) Atendiendo al tipo de modelo destino:

- Transformaciones modelo a modelo (M2M), las cuales generan modelos a partir de otros modelos.
- Transformaciones modelo a texto (M2T), que generan cadenas de texto a partir de modelos. Son usadas, por ejemplo, para generar código y documentos.
- Transformaciones texto a modelo (T2M), que generan modelos a partir de cadenas de texto.

**Transformaciones de modelos en MDA**

Una transformación de modelos es el proceso de convertir un modelo de un sistema en otro modelo del mismo sistema.

En esencia, una transformación establece un conjunto de reglas que describen cómo un modelo expresado en un lenguaje origen puede ser transformado en un modelo en un lenguaje destino.

Para definir una transformación entre modelos es necesario:

1. Seleccionar el tipo de transformación y el lenguaje de definición de transformaciones a utilizar
2. Seleccionar la herramienta que nos permita implementar los modelos y las transformaciones de forma automática.

## 2.6. QVT

QVT es la propuesta del OMG para resolver el problema de la transformación de modelos. Se trata de un estándar para la definición de transformaciones sobre modelos MOF. La especificación de QVT depende de otros dos estándares de la OMG como son MOF y OCL.

El lenguaje en sí define tres diferentes lenguajes de transformación: Relations, Core y Operational.

En la dimensión de la interoperabilidad se encuentran aquellas características que permiten a una herramienta que cumple el estándar QVT interopere con otras herramientas.
Concretamente, son cuatro:

- La sintaxis ejecutable se traduce en una implementación que facilite la importación o lectura, y posterior ejecución de una sintaxis concreta que describa una transformación definida en el lenguaje dado por la dimensión del lenguaje.
- XMI ejecutable dictamina que una implementación debe facilitar la importación o lectura, y posterior ejecución de una serialización XMI de una transformación que conforma con el metamodelo de MOF del lenguaje dado por la dimensión del lenguaje.
- La sintaxis exportable establece que una implementación debe proporcionar facilidad para exportar una transformación entre modelos en la sintaxis concreta del lenguaje dado por la dimensión del lenguaje.
- XMI exportable es el nivel de interoperabilidad que debe facilitar la exportación de transformaciones entre modelos como serializaciones XMI que conformen con el metamodelo MOF del lenguaje dado por la dimensión del lenguaje.

El estándar QVT define tres abstracciones fundamentales, que se corresponden con sus siglas

- **Consultas (Queries)** Una consulta es una expresión que se evalúa sobre un modelo. Los resultados de una consulta son una o varias instancias de los tipos definidos en el modelo transformado, o en el propio lenguaje de consulta. Para la realización de consultas se utilizará un lenguaje de consultas.
- **Vistas (Views)** Una vista es un modelo obtenido en su totalidad a partir de otro modelo base. Una vista es una proyección realizada sobre un modelo, creada mediante una transformación. Una vista puede verse como el resultado de una consulta sobre un modelo, ofreciendo como resultado un punto de vista de éste, restringiéndolo de acuerdo a alguna condición impuesta.
- **Transformaciones (Transformations)** Una transformación genera un modelo a partir de otro modelo de origen. Ambos modelos podrán ser dependientes o independientes, según exista o no una relación que mantenga ambos sincronizados una vez se produzca la transformación. Las vistas son un tipo específico de transformación que toman como entrada uno o varios modelos, y lo relacionan de manera que cumplan las condiciones de la transformación, o crean uno nuevo a partir de los modelos de entrada. Si se modifica una vista, la correspondiente transformación debe ser bidireccional para reflejar los cambios en el modelo fuente. Las transformaciones se definirán utilizando un lenguaje de transformación. El lenguaje de transformación debe servir para generar vistas de un metamodelo, debe ser capaz de expresar toda la información necesaria para generar automáticamente la transformación de un modelo origen a uno destino, y debe además soportar cambios incrementales en un modelo origen que se ven reflejados en el modelo destino.

De estas abstracciones se desprenden por lo tanto dos lenguajes, de consulta y de transformación, a los que se impuso como requisito que estuvieran definidos como metamodelos MOF 2.0. Un último requisito fundamental de la propuesta QVT es relativo a los modelos manipulados: todos los modelos manipulados por los mecanismos de transformación serán además instancias de metamodelos MOF 2.0.

### 2.6.1. Arquitectura QVT

La especificación de transformaciones se propone una solución de naturaleza híbrida declarativa e imperativa.

La parte declarativa de esta especificación está estructurada en una arquitectura de dos capas:

- Un metamodelo Relations define un lenguaje declarativo para expresar relaciones entre modelos MOF. Este lenguaje soporta reconocimiento de patrones, la creación de plantillas de objetos y la creación implícita de las trazas necesarias para registrar los cambios que se producen cuando se transforman modelos.
- Un metamodelo Core y un lenguaje declarativo de menor nivel de abstracción que Relations pero con la misma potencia, definido usando extensiones mínimas de EMOF y OCL. La especificación de QVT define las reglas que permiten mapear la semántica de Relations a la de Core, y dichas reglas las define en base a transformaciones descritas utilizando a su vez Core. En este lenguaje, los objetos de traza deben definirse explícitamente y sólo soporta reconocimiento de patrones sobre un conjunto plano de variables, no sobre objetos complejos como en Relations.

### 2.6.2. Escenarios de ejecución

La semántica del lenguaje Core y el lenguaje Relaciones debe tener en cuenta los siguientes escenarios de ejecución:

- Entradas a sólo transformaciones para verificar que los modelos se relacionan de una manera específica.
- Transformaciones de una sola dirección.
- Transformaciones bidireccionales.
- La capacidad de establecer relaciones entre los modelos preexistentes.
- Actualizaciones incrementales cuando un modelo relacionado se cambia después de una ejecución inicial.
- La capacidad de crear y eliminar objetos y valores, al mismo tiempo ser capaz de especificar los objetos y los valores que no pueden ser modificados.

### 2.6.3. Metamodelos MOF

Esta especificación define tres paquetes principales, uno para cada uno de los idiomas definidos: QVTCore, QVTRelation , y QVTOperational . El paquete QVTBase define la estructura común para las transformaciones . Además, el paquete QVTRelation utiliza expresiones de plantilla de patrones definidos en el paquete QVTTemplateExp.
QVTOperational extiende QVTRelation, ya que utiliza el mismo marco de rastros definidos en ese paquete. Utiliza expresiones imperativas definidas en el paquete ImperativeOCL.
Todos QVT depende del paquete EssentialOCL de OCL 2,0, y todos los paquetes de idioma dependen de EMOF.

### 2.6.4. Lenguaje Relation

El lenguaje Relations ofrece una aproximación declarativa para la especificación de transformaciones. Dado un par de modelos candidatos para la transformación, que deberán ser instancias de metamodelos MOF, ésta quedará definida como un conjunto de restricciones que los elementos de los modelos deben satisfacer.

Estas restricciones constituyen el concepto de relación del lenguaje Relations: se definen mediante dos o más dominios y mediante una pareja de predicados when (cuándo) y where (cómo). Los dominios especifican, mediante un patrón, qué elementos de los modelos candidatos participarán en la transformación. Por su parte, la cláusula when especifica la relaciones que determinan cuándo la relación indicada debe sostenerse; y la cláusula where especifica la condición que deben satisfacer todos los elementos de modelos que participan

en la relación.

En el lenguaje Relations, una transformación entre modelos candidatos se especifica como un conjunto de relaciones que han de tener lugar para llevar a cabo la transformación. Un modelo candidato es cualquier modelo que conforme a un tipo de modelo o metamodelo, que es una especificación de los diferentes tipos que pueden tener los elementos del modelo que lo conforma. Los modelos candidatos tienen un nombre y los tipos de los elementos que pueden contener, los cuales quedan restringidos al paquete al que se hace referencia en la declaración del modelo candidato. Por ejemplo:

```
transformation uml2Rdbms (uml : SimpleUML, rdbms : SimpleRDBMS)
```

En esta declaración de transformación llamada `uml2Rdbms`, hay dos modelos candidatos: `uml` y `rdbms`. El modelo `uml` se declara referenciando el paquete `SimpleUML` como su metamodelo, y el modelo `rdbms` hace lo propio con el paquete `SimpleRDBMS`. Una `transformation` puede invocarse tanto para comprobar la consistencia de dos modelos como para transformar un modelo en otro.

### 2.6.5. El Lenguaje Operational

El lenguaje Operational permite definir transformaciones utilizando bien una aproximación imperativa o bien una aproximación híbrida, complementando las transformaciones relacionales declarativas con operaciones imperativas que implementen las relaciones.

Una transformación operacional define una transformación unidireccional expresada de forma imperativa. Se compone de una signatura que indica los modelos involucrados en la transformación y de una definición de la ejecución de la transformación. En la práctica, una transformación operacional se trata de una entidad instanciable con propiedades y operaciones.

Este lenguaje de QVT permite la definición de transformaciones mediante un lenguaje imperativo. Permite también definir mappings para complementar las relaciones desarrolladas con el lenguaje Relational (enfoque híbrido).

Entidad instanciable que posee operaciones y propiedades. Representa la definición de una transformación unidireccional expresada de manera imperativa.

# Capítulo 3

# Método para evaluar la calidad de servicio de Web Services

El número cada vez mayor de Web Service (WS) disponibles dentro de una organización y en la web, plantea un problema de la búsqueda de ellos y destaca la importante necesidad de encontrar un mejor WS que cumpla con ciertos requisitos del solicitante. Por esta razón, los requisitos de un WS no deben centrarse solamente en sus propiedades funcionales, sino también en la descripción del entorno en el que se desarrolla, es decir, la descripción de características de calidad (QoS). Cada servicio puede ofrecer varias opciones para QoS basados en normas técnicas, tales como la disponibilidad, el rendimiento y la escalabilidad, las políticas de seguridad y privacidad, etc., los cuales deben ser descritos y analizados. En este punto, es de vital importancia contar con un procedimiento para evaluar la calidad de la WS. Este procedimiento proporcionará una herramienta para estimar la calidad de los WS de una manera específica.

En consecuencia de lo expuesto hasta ahora, este capítulo propone un procedimiento para medir la calidad de la WS, desarrollado en base a la publicación [5].

## 3.1. Características de calidad de Web Services

La descripción de un WS tiene dos componentes principales: sus características funcionales y no funcionales. La descripción funcional, como mencionamos anteriormente, es una descripción sintáctica que se centra en los aspectos funcionales, detallando las características operativas que definen el comportamiento general de un WS. Un requisito importante para las aplicaciones basadas en SOA es operar de forma fiable y ofrecer un servicio consistente a una variedad de niveles. Por lo tanto, los requisitos de un WS no deben centrarse sólo en sus propiedades funcionales, sino también en la descripción del entorno que lo aloja, es decir, las capacidades no funcionales o QoS del servicio. Cada servicio puede ofrecer varias opciones para características no funcionales en base a los requisitos técnicos que resultan de la demanda de disponibilidad, el rendimiento, la escalabilidad, las políticas de seguridad, privacidad, etc., todo lo cual debe ser descrito.

La descripción no funcional define la QoS del servicio, obligan al solicitante a especificar, en tiempo de ejecución, los atributos de calidad que pueden influir en la elección de un WS ofrecido por un proveedor.

## 3.2. Marco general de selección de WS de un WFMS

La descripción y la automatización de procesos de negocio se realizan a través de los Workflow Management System (WFMS). El modelo de referencia del workflow, desarrollado por la WfMC, define un marco genérico para la construcción de WFMS, lo que permite la interoperabilidad entre ellos y otras aplicaciones implicadas. Este modelo define una interfaz para la invocación de aplicaciones externas.

Para seleccionar la aplicación más adecuada entre varias semánticamente correctas que se comportan de la misma manera pero que tienen diferentes descripciones sintácticas y QoS, se propone optimizar la selección al incorporar las QoS. El esquema propuesto para la invocación de WS presenta inicialmente una muestra de los WS disponibles en una base de datos conocida, el WS requerido en la invocación de aplicaciones externas

que corresponden al motor del workflow y la selección WS que eventualmente será invocado. En el método propuesto se consideran dos elementos importantes para que el proceso contraste entre los servicios ofrecidos y la aplicación requerida:

- Antecedentes de ejecuciones: el método propone realizar la invocación de WS, supervisar sus atributos y registros de calidad en el historial de ejecuciones.
- Las características del método: el método tiene un registro de características cualitativas para invocar una aplicación.

En la realización del proceso de contraste con el nivel de calidad de servicio, el método propuesto considera:

- Antes de invocar cualquier WS se analizan las características durante el proceso de verificación. Se aplica métricas para evaluar los atributos de calidad.
- Si el WS se ha invocado anteriormente, el método propuesto busca en la historia de las ejecuciones y observa la información almacenada del mismo.

Cuando el proceso de contraste se realiza en el nivel de calidad de servicio, es un filtro de la lista preliminar de WS candidatos y obtiene el que mejor cumpla con los requisitos del solicitante. En el esquema propuesto, el componente principal es el descubrimiento de servicios basado en los requisitos funcionales y QoS especificados por el usuario y los requisitos por el propio motor. De este modo, el motor proporciona un ranking de WS disponible y elige el más adecuado. También es responsable de la recolección y procesamiento de información de los WS disponibles en la base de datos y mantenerla actualizada a medida que se realizan las llamadas.

## 3.3. Método para seleccionar un WS teniendo en cuenta su calidad de servicio

Proponemos un método de medición para obtener un valor cuantitativo de cada servicio que permite la comparación entre ellos.

En primer lugar, se define una unidad de medida que depende de la característica que está siendo evaluado. Aquí hemos considerado:

- Unidad de tiempo: le asigna un valor en milisegundos, segundos, minutos u horas, dependiendo de las características que se evalúa.
- Porcentaje: asignado un valor numérico entre 0 y 100, que se toma de una muestra representativa, y que depende de la propiedad que está siendo evaluada.

El tiempo de respuesta (TR) se evalúa teniendo en cuenta la unidad de medida de tiempo en milisegundos. La disponibilidad (D) y la reputación (R) se mide en porcentaje.

La función de prioridad (Fp) que asigna a cada WS un peso se define por:

$$Fp = D + R - TR$$

La ecuación anterior permite que la prioridad de un WS aumente en función del incremento de su disponibilidad y reputación, y en la medida que sus tiempos de respuestas sean menores.

La evaluación del sistema se puede hacer usando técnicas cualitativas o cuantitativas. Las técnicas cualitativas se basan en el análisis de una lista de características, ventajas y desventajas, que se comparan de manera intuitiva para generar una clasificación final de los sistemas propuestos.

Los métodos cuantitativos dan indicadores cuantitativos generales que serán utilizadas para encontrar y justificar una buena decisión óptima. Generalmente, los métodos cuantitativos basados en técnicas de puntuación tienen dos indicadores para cada sistema, una puntuación de preferencia global y un indicador de costo total. El presente trabajo no tiene en cuenta el costo.

El creciente aumento de los WS's en los últimos años permite que los sistemas que los utilizan puedan realizar una selección del servicio en función de ciertos criterios. El objetivo es elegir el servicio más adecuado que satisfaga tanto los requisitos funcionales y no funcionales.

La automatización de los procesos de negocio se considera como parte de un interés principal de las organizaciones para el crecimiento, el desarrollo y la competitividad. El uso de programas como parte de Interfaz de las Aplicaciones Invocadas de los WFMS ofrece beneficios importantes, tales como la distribución transparente de las aplicaciones fuera del workflow, permitiendo que el motor invoque la aplicación sin conocer su ubicación exacta, con el importante beneficio que puede cambiar su ubicación en la red sin que suponga un cambio en su invocación. Además, los sistemas de gestión deben invocar la aplicación de workflow que es más adecuada a sus necesidades, y esto requiere la especificación de algunas restricciones que se escribirán en la memoria no funcional de una solicitud de WS.

Existe una clara necesidad de desarrollar mecanismos rápidos y eficaces que se pueden utilizar para la selección dinámica de los servicios a partir de un conjunto de proveedores de servicios. En este capítulo se presentó un método cuantitativo para medir las características no funcionales de los servicios que proporcionan la misma funcionalidad. El método puede medir y luego comparar las puntuaciones obtenidas por todos los servicios candidatos y seleccionar el WS que mejor se adapte a las necesidades del solicitante. En este capítulo, el método se aplica a la selección dinámica de WS que puede ser utilizado por un motor de workflow cuando se requiere aplicaciones externas para llevar a cabo las tareas de los procesos de negocio.

# Capítulo 4

# Mecanismo de selección de Web Services

Para permitir que la aplicación desarrollada (QVTWSinvoker) pueda seleccionar el Web Services(WS) más conveniente entre varios que semánticamente se comportan igual, se propone optimizar esta selección de servicios usando transformaciones de modelo definidos incorporando a la selección automática los atributos de calidad descriptos en el capítulo. Las transformaciones de modelo QVT permiten proveer una especificación semántica más precisa de un WS y establecer una correspondencia entre los requerimientos del usuario y los WS's disponibles, lo cual facilita su descubrimiento automático en tiempo de ejecución.

## 4.1. Preselección automática de WS's mediante tranfomaciones de modelo QVT

Las figuras 4.1 y 4.2. presentan un esquema de la propuesta de invocación de WS's con QVT. En la misma se visualizan los WS's conocidos mediante la base de datos en formato wsdl y las carecteristicas solicitadas a través de los requisitos del usuario en un ecore request, la correspondencia realizada por el software, y la preselección de los WS's a los cuales se les evaluará su calidad. Aquí se puede observar el comportamiento interno del software a la hora de preseleccionar una aplicación externa, en particular, cómo hace el programa para preelegir la aplicación, valiéndose de los WS's disponibles en la base de datos. En el esquema propuesto, el componente principal es QVTWSinvoker, quien actúa como una agente de descubrimiento y preselección de servicios, basándose en los requerimientos funcionales por el usuario. Así, QVTWSinvoker realiza una serie de transformaciones de los WS's disponibles y elige el más adecuado.

Para realizar la correspondencia y preselección QVTWSinvoker trabaja en tres etapas.

- En la primer etapa se transforman todos los WSDL conocidos a archivos XMI, cuyo contenido contiene la representación del modelo Ecore del WSDL, a través de una transformación T2M.
- En la segunda etapa se transforman todos los archivos XMI (WSDL Ecores) resultantes de la primer etapa a su respectiva representación Ecore del modelo de los requerimientos del usuario (Request Ecore) a través de una transformación M2M.
- En la tercer etapa establece la correspondencia teniendo en cuenta los requisitos del solicitante y los WS's transformados. Al solicitante, se le pide el tipo de aplicación, el nombre del método y los parámetros. Como resultado de la correspondencia, QVTWSinvoker obtiene una lista de los WS's precandidatos a invocarse

## 4.2. Modelos del solicitante y los proveedores de WS's

Para la transformación de modelos se debió utilizar dos modelos Ecore, uno que representa la solicitud del cliente, y el segundo representa un WS con formato WSDL.

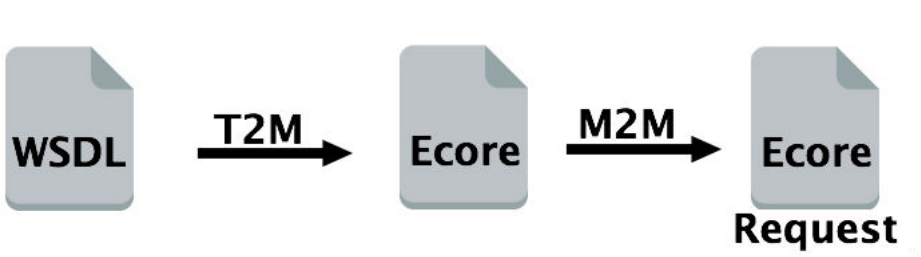


Figura 4.1: Transformación Wsdl Ecore Request.


Figura 4.2: Comparación Ecore Wsdl - Ecore Request.

### 4.2.1. Modelo de solicitud de usuario

Este modelo ecore se utiliza para representar una petición de un cliente. La representación en diagrama de UML se realiza mediante el plugin Ecore Diagram Editor SDK.

Un RequestModel tendrá asociado o no, uno o varios métodos, cada método deberá tener un nombre y posiblemente una descripción de lo que hace el método. El sistema buscará precandidatos en base al nombre del método el cual contendrá parámetros de entrada y salida, el cliente solo tendrá acceso a los parámetros de entrada. Cada parámetro deberá tener un tipo predefinido de Java y un nombre (ver figura 4.3).

Para obtener este modelo, se realiza una transformación T2M (ver código 4.1), en donde se carga y genera un modelo RequestModel con los datos obtenidos desde una interfaz gráfica en la cual el usuario carga los requerimientos que desea.

Código 4.1: Transformación T2M RequestModel

```
1   /**
2    * obtains the information from the gui, and returns a requestModel
3    *
4    * @return
5    */
6   private RequestModel getDataFromUI() {
7       //(*-)creates a request model
8       RequestModelFactory factory = RequestModelFactory.eINSTANCE;
9       RequestModel requestModel = factory.createRequestModel();
10      String methodName = deleteAccent(invokerUI.getTxtMethodName().getText());
11      methodName = methodName.replace(" ", "");//remove all blank spaces
12      requestModel.setName(methodName);
13      Method method = factory.createMethod();
14      method.setName(methodName);
15      Parameter outputParam = factory.createParameter();
16      outputParam.setName("output");
```

```
17          outputParam.setType("string");
18          method.getOutParameters().add(outputParam);
19          params = new ArrayList<>();
20          Integer numParams = Integer.valueOf(
21                  (String) invokerUI.getSpnNumberParam().getSelectedItem());
22          for (int i = 0; i < numParams; i++) {
23              /**
24               * (for all chosen parameters, it gets the type and value and
25               * casting it if the cast fails it'll return null
26               */
27              String type = ((JComboBox)(invokerUI.getPanelParamater(i).getComponent(1)))
28                      .getSelectedItem().toString();
29              String value = StringTreatment.deleteAccent(((JTextField) invokerUI
30                      .getPanelParamater(i).getComponent(3)).getText());
31              try {
32                  switch (type) {
33                      case TypesOfWsdl.typeDouble:
34                          params.add(Double.valueOf(value));
35                          break;
36                      case TypesOfWsdl.typeFloat:
37                          params.add(Float.valueOf(value));
38                          break;
39                      case TypesOfWsdl.typeInteger:
40                          params.add(Integer.valueOf(value));
41                          break;
42                      default:
43                          params.add(value);
44                  }
45              } catch (Exception e) {
46                  /**
47                   * the conversion of one type failed, returns null
48                   */
49                  System.err.println(e);
50                  return null;
51              }
52              Parameter inputParam = factory.createParameter();
53              inputParam.setName("param-" + i);
54              inputParam.setType(type);
55              method.getInParameters().add(inputParam);
56          }
57          requestModel.getMethods().add(method);
58          return requestModel;
59      }
```

Por otro lado un WS se representa mediante un modelo ecore WSDL, al cual se le ha agregando una descripción en las operaciones.

Para obtener un modelo ecore de este tipo, se realiza una transformación T2M, en donde el archivo con extensión .wsdl debe estar escrito en una versión de SOAP 1.1 o 1.2.

## 4.3. Estructura del proyecto de transformación

El proyecto consiste de 3 módulos:

- **T2MWsdl:** Este módulo realiza la transformación de texto a modelo de un wsdl.
- **Comparsion:** Este módulo contiene la comparación de modelos, ambos modelos deben ser Request-Model.

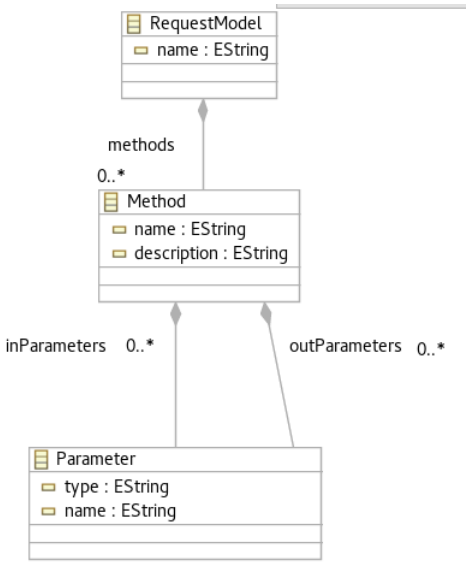


Figura 4.3: Diagrama UML de RequestModel.

- **Transformation:** Este módulo ejecuta en la transformación M2M de un wsdl, invocando la transformación QVTO definida.

Para la representación de los dos modelos ecore en Java, se utilizó el plugin de eclipse EMF (Eclipse Modeling Framework SDK) que autogenera el código Java para la representación del mismo.
Esta representación en java consiste en 3 paquetes:

- **utils:** Tiene clases auxiliares.
- **Impl:** Contiene la implementación de las interfaces.
- **Interface:** Comprende la representación de cada objeto del modelo ecore.

## 4.4. Funcionamiento de la transformación

El método consiste en 3 fases fundamentales, la primer etapa se encarga de la transformación de un wsdl en texto plano a un modelo ecore del mismo, la segunda etapa realiza la conversión M2M del modelo generado en la primer etapa a su respectiva representación en RequestModel, por último, la tercer etapa consiste en la comparación de 2 modelos RequestModel, el generado en la segunda etapa y la solicitud del usuario. Esta etapa deberá decidir si wsdl satisface la solicitud del usuario en base los criterios de comparación descritos al inicio del capítulo.

**Etapa 1: Transformación T2M wsdl**

Esta etapa realiza una transformación del wsdl en texto plano a una representación del mismo en ecore. Para ello, debemos conocer la ubicación de algún WS, por ejemplo : `www.dominio\_del\_web\_service.com/servicio\_web.asmx?WSDL`

El sistema, con el soporte de la API soa_model_distribution, realiza un análisis gramatical del WS para hacer más eficiente su uso (ver código 4.1), luego se procede a obtener los binding del WS que se encuentran definidos para trabajar con SOAP 1.1 y 1.2 y de cada binding se consiguen los PortType.

Código 4.2: Análisis gramatical del WS.

```
1    /**
```

```
     * (*+)Parsea un archivo o ruta wsdl y retorna un Definition
     *
     * @param url
     * @return
     */
    public Definition parser(String url) {
        definition = factory.createDefinition();
        WSDLParser parser = new WSDLParser();
        defs = parser.parse(url);
        if (defs.getName() == null) {
            // (*+) Si la definition no tiene nombre, le seteo el nombre de la
            // url
            definition.setQName(QName.valueOf(new File(url).getName().split("\\.")[0]));
        } else {
            definition.setQName(org.eclipse.emf.ecore.xml.type.internal.QName.valueOf(defs.getName()));
        }
        List<Binding> bindings = defs.getBindings();
        for (Binding binding : bindings) {
            // (*+)es protocolo SOAP V 1.1 O V 1.2
            if (binding.getBinding() instanceof SOAPBinding
                    || binding.getBinding() instanceof com.predic8.wsdl.soap12.SOAPBinding) {
                PortType portType = defs.getPortType(binding.getType());
                if (!existsEPortType(portType.getQName().getLocalPart())) {
                    definition.getEPortTypes().add(parserPortType(portType));
                }
            }
        }
        return definition;
    }
```

Luego a cada PortType, se le extraen las operaciones y se las analizan para obtener operaciones definidas en el modelo ecore (ver código 4.3).

Código 4.3: Extracción de operaciones de PortType.

```
    /**
     * (*+)Parsea una Operation de un archivo wsdl y retorna una Operation de un
     * modelo wsdl.ecore
     *
     * @param operation
     * @return
     */
    private tesis.wsdl_ecore.wsdl.Operation parserOperation(Operation op) {
        tesis.wsdl_ecore.wsdl.Operation operationFactory = factory.createOperation();
        operationFactory.setName(op.getName());
        operationFactory.setEInput(parserInputOperation(op.getInput()));
        operationFactory.setEOutput(parserOutputOperation(op.getOutput()));
        if (op.getDocumentation() != null)
            operationFactory.setDocumentation(op.getDocumentation().getContent());
        return operationFactory;
    }
```

Según la definición de un wsdl, cada operación consta de mensajes de entrada y mensajes de salida. Los mensajes pueden estar definidos con tipos simples, o utilizar tipos embebidos, a su vez, un tipo embebido puede contener otros tipos embebidos, por lo tanto se emplea la recursividad para "desenvolver" los parámetros y generar parámetros de tipos simples (ver código 4.4 y 4.5).

Código 4.4: Parser de messages.

```
    /**
     * (*+)Parsea un message de un archivo wsdl y retorna un Message de un
```

```
     * modelo wsdl.ecore
     *
     * @param msg
     * @return
     */
    private Message parserMessage(com.predic8.wsdl.Message msg) {
        Message msgFactory = factory.createMessage();
        msgFactory.setQName(QName.valueOf(msg.getName()));
        for (com.predic8.wsdl.Part part : msg.getParts()) {
            if (part.getElement() != null) { //(*+) es un parametro de tipo embebido
                Element element = part.getElement();
                //(*+) debo "desenvolver" el tipo embebido en varios simples
                ArrayList<Pair<String, String>> parameters = parserEmbebedType(element);
                for (Pair<String, String> param : parameters) {
                    msgFactory.getEParts().add(parserPart(param));
                }
            } else { //(*+) es un parametro de tipo simple
                msgFactory.getEParts().add(parserPart(new Pair<String, String>(part.getName(),
                        ((com.predic8.schema.BuiltInSchemaType)
                            part.getType()).getQname().getLocalPart())));
            }
        }
        return msgFactory;
    }
```

Código 4.5: Análisis y parser de tipos embebidos.

```
    /**
     * (*+)Metodo que parsea un Element, si este es de un complexType lo
     * convierte a un arraylist que contiene el nombre y tipo del parametro
     *
     * @param element
     * @return
     */
    private ArrayList<Pair<String, String>> parserEmbebedType(Element element) {
        TypeDefinition typeDefinition = element.getEmbeddedType();
        ArrayList<Pair<String, String>> ret = new ArrayList<Pair<String, String>>();
        //(*+) es un tipo complejo definido en el mismo esquema, o se encuentra en otro esquema
        if (typeDefinition != null || defs.getSchemaType(element.getType()) instanceof ComplexType)
             {
            Sequence model;
            if (typeDefinition != null) //(*+) el tipo está en el mismo esquema, obteno el modelo
                model = ((Sequence) ((ComplexType) typeDefinition).getModel());
            else//(*+) el tipo se encuentra en otro esquema, obtengo el modelo desde ese esquema
                model = ((Sequence) ((ComplexType)
                     defs.getSchemaType(element.getType())).getModel());
            for (Element elementAux : model.getElements()) {
                /*(*+) para cada parametro de este tipo, llamo recursivamente por si
                 * está definido como tipo embebido
                 */
                ret.addAll(parserEmbebedType(elementAux));
            }
        } else {
            ret.add(new Pair<String, String>(element.getQname().getLocalPart(),
                 element.getBuildInTypeName()));
        }
        return ret;
    }
```

Al terminar todo el análisis, se obtiene el mismo wsdl pero cargado en un modelo ecore, el cual es expor-

tado y almacenado en disco con una extensión .xmi para luego ser utilizado en la etapa 2.

### Etapa 2: Transformación QVTO

Esta etapa es la encargada de llevar a cabo la transformación QVTO del modelo ecore wsdl en a un modelo ecore RequestModel. El sistema comienza tomando la ruta al archivo que se generó en la etapa anterior, y una ruta de salida donde se almacenará el RequestModel.(ver código 4.6)

Código 4.6: Invocación de la transformación QVTO

```
1   /**
2    * (*+)las paths deben ser absolutas por ejemplo /home/user/desktop... o la
3    * url de un wsdl
4    *
5    * @param qvtoPath
6    * @param ginpath
7    * @param outpath
8    * @throws IOException
9    */
10  public void startTransformation(String inpath, String outpath) throws IOException {
11      // (*+) esta es la ruta de la transformacion
12      URI uriTransformation = URI.createURI("platform:/resource/transforms/wsdlToRequest.qvto");
13      ExecutionContextImpl context = new ExecutionContextImpl();
14      EList<EObject> inObjects = Utils.getFromXMI(inpath);
15      TransformationExecutor fExecutor = new TransformationExecutor(uriTransformation);
16      // (*+) para debugear la transformacion qvto
17      // context.setLog(new WriterLog(new OutputStreamWriter(System.out)));
18      // create the input extent with its initial contents
19      ModelExtent input = new BasicModelExtent(inObjects);
20      // create an empty extent to catch the output
21      ModelExtent output = new BasicModelExtent();
22      // setup the execution environment details ->
23      // configuration properties, logger, monitor object etc.
24      // run the transformation assigned to the executor with the given
25      ExecutionDiagnostic result = fExecutor.execute(context, input, output);
26      // check the result for success
27      if (result.getSeverity() == Diagnostic.OK) {
28          // the output objects got captured in the output extent
29          List<EObject> outObjects = output.getContents();
30          // let's persist them using a resource
31          ResourceSet resourceSet2 = new ResourceSetImpl();
32          resourceSet2.getResourceFactoryRegistry().getExtensionToFactoryMap().put("*", new
                XMLResourceFactoryImpl());
33          Resource outResource = resourceSet2.createResource(URI.createURI("file://" + outpath));
34          outResource.getContents().addAll(outObjects);
35          outResource.save(Collections.emptyMap());
36      } else {
37          // turn the result diagnostic into status and send it to error log
38          IStatus status = BasicDiagnostic.toIStatus(result);
39          System.err.println(status.getMessage());
40      }
41  }
```

Este método, como se puede visualizar, invoca la transformación qvto almacenada en otra carpeta dentro del mismo proyecto. La transformación toma como archivo de entrada un modelo wsdl y retorna un modelo RequestModel (ver código **??**).

Código 4.7: Perfiles y modelos de la transformación

```
1  modeltype WSDL uses wsdl('http://www.eclipse.org/wsdl/2003/WSDL');
2  modeltype REQUEST uses RequestModel('http:///request_model.ecore');
```

```
transformation WSDL_to_RequestModel(in source : WSDL, out output:REQUEST);
```

La transformación comienza buscando el objeto Definition en el wsdl, este es el que contiene toda la información necesaria para poder llevar adelante la transformación. Una vez obtenido la Definition, le aplica una transformación model2Request, obteniendo de esta forma el modelo RequestModel (ver código 4.8).

Código 4.8: Inicio de la transformación QVT.

```
main() {
    log('starting transformation...');
    source.rootObjects()[WSDL::Definition]->map modelT2model();
    log('ending transformation...');
    }

mapping Definition :: modelT2model() : RequestModel {
    init {
        result := object RequestModel{};
    }
    result.name := self.qName.toString();
    result.methods := result.methods ->union(self.ePortTypes -> map portTypeT2Methods());

}
```

Una vez dentro de Definition, el paso siguiente es, asignar al RequestModel el nombre de Definition, y obtener los portType, que contiene la definición de los métodos, por lo tanto se le aplica una transformación `portType2Methods`, la cual retorna un conjunto de `Methods` pertenecientes al modelo RequestModel (ver código 4.9).

Código 4.9: Transformación PortType a Method

```
mapping OrderedSet(PortType) :: portTypeT2Methods() : OrderedSet(Method) {
    init {
        result := OrderedSet{};
        result += result -> union(
            self -> collect(porType| porType -> map portTypeT2Method() ) -> asOrderedSet());
    }
}

mapping PortType :: portTypeT2Method(): OrderedSet(Method){
    init {
        result := OrderedSet{};
        result += result -> union(self.allInstances(Operation)-> asOrderedSet()
            ->collect(op| op -> map operationT2Method()) ->asOrderedSet());
    }
}
```

De esta manera, se continúa transformando hasta llegar a la parte más "interna", donde se encuentran los métodos. Al finalizar la transformación, se consigue la representación del wsdl en RequestModel, el cual, únicamente contiene la información necesaria, es decir, los métodos con sus respectivos parámetros de entrada y salida.

### Etapa 2: Comparación

Una vez obtenido el/los wsdl representado/s en un modelo ecore RequestModel la petición del usuario representado de la misma forma, se comparan para decidir si la petición está o no incluida en el wsdl, y en caso de estar incluido, se considerará que el wsdl resuelve la petición del usuario.

Esta comparación se realiza con comparaciones de modelos desde Java, tomando un wsdl y la petición del usuario y comparándolos por nombres de métodos, si los nombres coinciden, se continúa analizando la

cantidad de parámetros de entrada que posee cada método, en caso de ser la misma, se analizan los tipos (ver código 4.10).

Código 4.10: Transformación de un método.

```
1   /**
2    * (*+) Retorna los nombres de los métodos que machean con el wsdl y la
3    * request
4    *
5    * @param requestClient
6    * @return
7    */
8   public ArrayList<String> getMachOperations(RequestModel requestClient) {
9       this.requestModelClient = requestClient;
10      ArrayList<String> ret = new ArrayList<String>();
11      for (Method method : requestModelWSDL.getMethods()) {
12          if (method.getName().toLowerCase().equals(requestClient.getMethods().get(0)
13              .getName().toLowerCase())) {
14              if (matchInputParams(method.getInParameters())) {
15                  ret.add(method.getName());
16              }
17          }
18      }
19      return ret;
20  }
21
22  /**
23   * (*+)Retorna true si machean los parametros
24   *
25   * @param inputParams
26   * @return
27   */
28  private boolean matchInputParams(List<Parameter> inputParams) {
29      List<Parameter> paramsRequest = requestModelClient.getMethods().get(0).getInParameters();
30      boolean ret = false;
31      // (*+) si la cantidad de parametros del metodo es igual a la cantidad
32      // de
33      // (*+) parametros de la request
34      if (inputParams.size() == paramsRequest.size()) {
35          ret = true;
36          int i = 0;
37          // (*+)corroboro los tipos
38          ArrayList<Parameter> inputParamsOrdered = toArrayListOrdered(inputParams);
39          ArrayList<Parameter> paramsRequestOrdered = toArrayListOrdered(paramsRequest);
40          for (Parameter param : paramsRequestOrdered) {
41              // (*+)si los tipos son distintos, salgo
42              if (!param.getType().equals(inputParamsOrdered.get(i).getType())) {
43                  return false;
44              }
45              i++;
46          }
47      }
48      return ret;
```

La transformación QVTO es no determinista a la hora de aplicar una transformación y no respeta el orden de los parámetros, por lo tanto se procedió a ordernar los parámetros de ambos modelos de acuerdo al tipo.

Si los tipos son correctos, la comparación retorna una lista con los nombres de todos los métodos que satisfacen la petición del usuario.

## 4.5. Análisis de la correspondencia entre los modelos generados

Ahora, para decidir automáticamente si un WS ofrecido por un proveedor satisface la demanda del solicitante, es necesario comparar los ecore generados por dicho solicitante y proveedor. Se formaliza usando relaciones subecore. Si la precondición del solicitante es un subecore de la precondición del proveedor, entonces el proveedor provee toda la información necesaria para ejecutar el WS.
En la Figura 4.4 se puede observar que el modelo del WS 1 es un submodelo del modelo del solicitante.

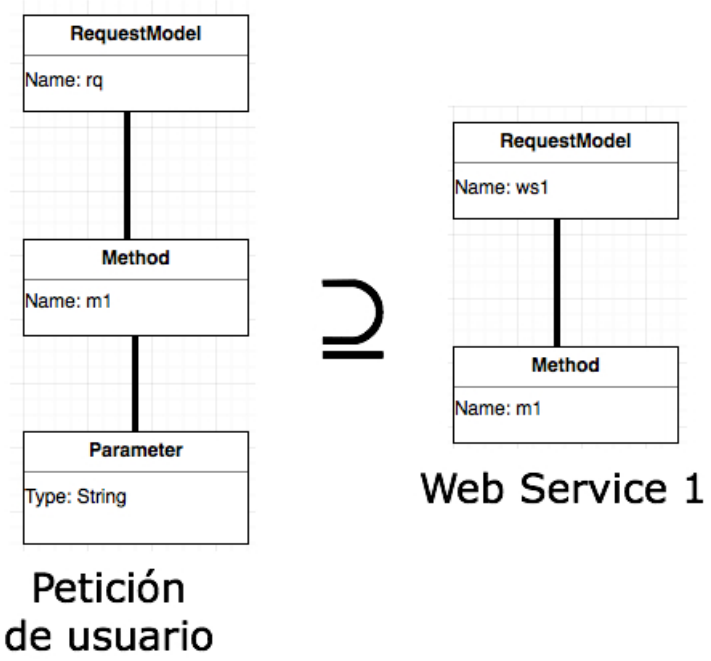


Figura 4.4: Ecores de Petición de usuario y WS 1.

Análogamente, en la Figura 4.5 se puede observar que la peticion de usuario es un submodelo del WS 2. Por lo tanto, el proveedor 2 ofrece un WS's que son compatibles con los requerimientos del solicitante, es decir, el proveedor 2 provee toda la información necesaria para ejecutar el WS.

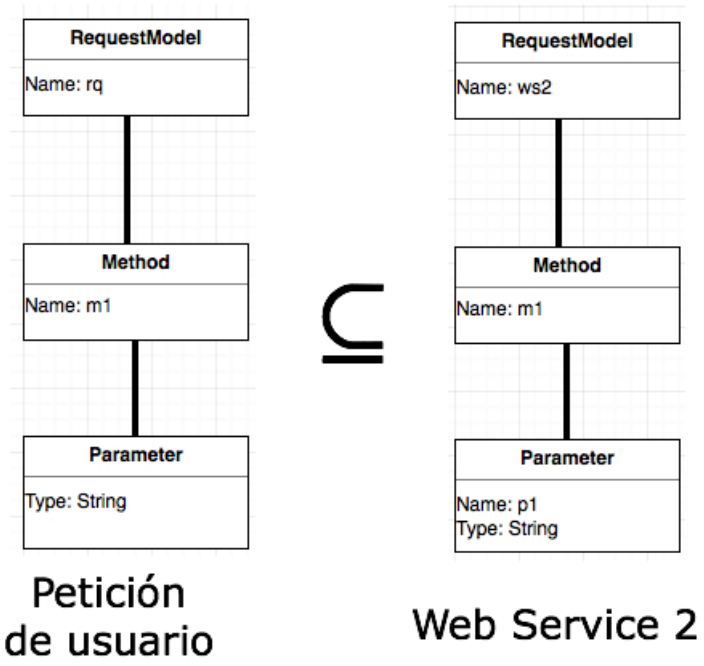


Figura 4.5: Ecores de Petición de usuario y WS 2.

# Capítulo 5

# Caso de estudio: QVTWSInvoker

En este capítulo se presenta un caso de estudio que plantea el uso de Web Services (WS) seleccionados automáticamente en la invocación de aplicaciones externas desde un Workflow, a través de su aplicación en el marco de las tareas involucradas en el proceso de desarrollo de software denominado QVTWSInvoker.

## 5.1. Descripción general de la herramienta

En virtud de lograr el cumplimiento de los objetivos enunciados en la sección 1.2 una gran cantidad de factores han sido tenidos en cuenta para finalmente llegar a una solución acorde a las expectativas.
QVTWSInvoker utiliza una plantilla genéricas de entrada. Consiste de un conjunto de parámetros que son los encargados de transportar los requerimientos del usuario a la herramienta. Y dado que este usuario será en reiteradas ocasiones un WFMS, el conjunto de parámetros ha sido planteado simple y estándar. Esta plantilla está formada por los siguientes atributos: tipo de WS, nombres y descripción de la operación a ejecutar con sus respectivos parámetros (valores para los cuales se deseen obtener los resultados correspondientes). Esta parametrización le posibilitará a la herramienta acceder de manera estándar a los requerimientos del sistema siempre de una misma forma, para luego comenzar el proceso de búsqueda con los criterios y filtros correspondientes.
La salida de la herramienta se imprime en un texto simple. Luego de la salida se le pide al usuario que especifique si está satisfecho con el resultado del WS.
Es importante destacar, que la herramienta provee al usuario las diferentes alternativas que desee en cuanto al tipo de WS y además, le da la posibilidad de proveer información para actualizar las características de calidad (QoS) que implementa dicho WS.

## 5.2. Proceso de selección de WS's

El proceso de selección de WS's con sus respectivas ejecuciones, consiste en la transformación de los requerimientos del usuario en una instancia de salida. Este proceso lo realiza QVTWSinvoker implementando los mecanismos descritos en los capítulos 4 y 5 que se describe en los siguientes puntos:

**Fase 1: Obtención de Parámetros**

En esta fase la herramienta obtiene desde la plantilla de entrada los parámetros, los números del mensaje de entrada se corresponden a estos parámetros. Ellos significan: 1 = tipo del WS; 2 = nombre de las operaciones; 3 = descripción ; 4 = cantidad y tipo de los parámetros; (Ver figura 5.1)

**Fase 2: Obtención de WS's de la base de datos**

La base de datos contiene los WS's conocidos por la herramienta. Se compone de la información los WSDL y las QoS. En esta fase la herramienta, en base al tipo ingresado por el usuario en la plantilla de

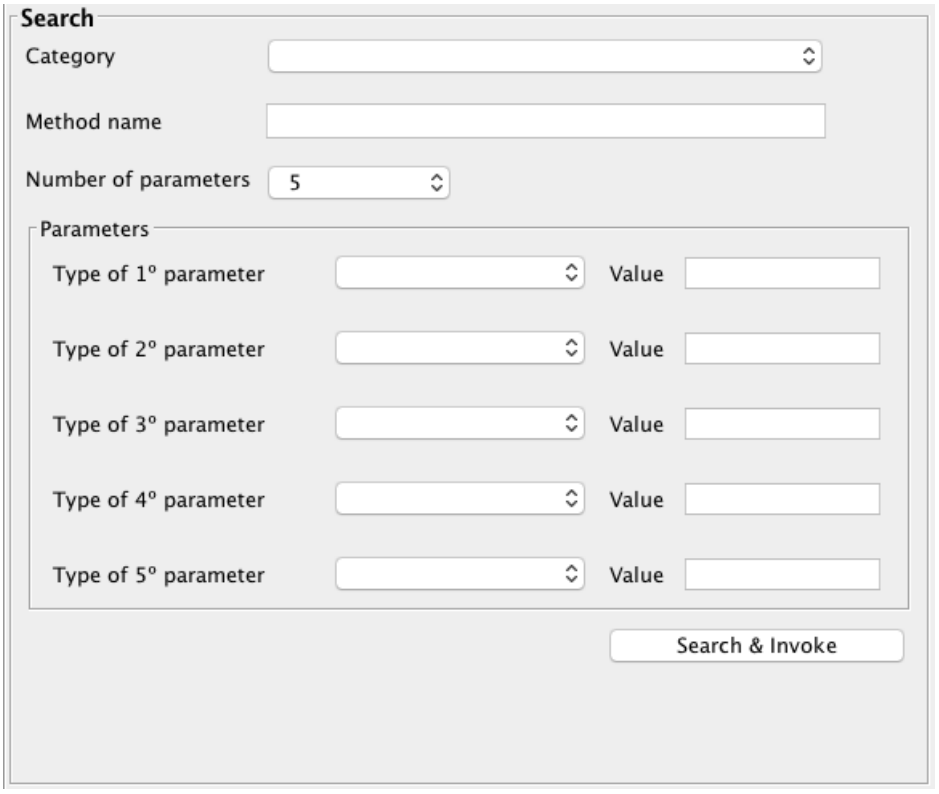


Figura 5.1: Plantilla de entrada de solicitud de usuario

entrada y las QoS , obtiene de la base de datos la lista de descripciones de los WS's ordenada por valor de ponderación de las QoS (mecanismo de calificación descrito en el capítulo 4).

Código 5.1: Método que obtiene de la base de datos la lista de descripciones

```
1   /**
2    * retirn categories order by rank
3    *
4    * @param c
5    * @return
6    */
7   public List<Wsdl> getChilds(Category c) {
8       Iterator<Wsdl> it = c.getAll(Wsdl.class).iterator();
9       WsdlComparator comparator = new WsdlComparator();
10      PriorityQueue<Wsdl> queue = new PriorityQueue<>(comparator);
11      while (it.hasNext()) {
12          queue.add(it.next());
13      }
14      LinkedList<Wsdl> list = new LinkedList<>();
15      list.addAll(queue);
16      return list;
17  }
```

Código 5.2: Método comparador para ordenar la cola

```
1   @Override
2   public int compare(Wsdl x, Wsdl y) {
3       if (x.getLong("availability") + x.getLong("reputation") - x.getLong("response") <
            y.getLong("availability") + y.getLong("reputation") - y.getLong("response")) {
```

```
            return -1;
        } else {
            return 1;
        }
    }
```

**Fase 3: Transformaciones y Filtro por nombre, descripción y parámetros de la operación**

A partir del conjunto de WS's generado en la fase previa, la herramienta procede a transformarlas siguiendo el mecanismo descripto en el cap 5 y filtrarlos teniendo en cuenta nombre, descripción y parámetros de la operación que se especificaron en la plantilla de entrada.

**Fase 4. Ejecución de WS's**

En la ejecución de WS's se selecciona el primer WS del conjunto. Una ejecución consiste en la invocación del WS con los parámetros que el usuario pasa en la plantilla de entrada. Si el resultado de esta ejecución es exitoso, la herramienta retorna la salida cuyos datos son los obtenidos de la ejecución.
Una ejecución se considera exitosa cuando se muestran los datos obtenidos de la invocación del WS. Si la ejecución no es exitosa se toma el siguiente WS y se repite esta fase.

Código 5.3: Invocación al mejor WS

```
    /**
     * invoke the best matched wsdl, if the invocation fails, call the next.
     *
     * @param methods
     */
    private void invokeWS(ArrayList<Pair<String, String>> methods) {
        if (methods.size() > 0) {
            boolean invoke = false;
            Iterator<Pair<String, String>> it = methods.iterator();
            while (it.hasNext() && !invoke) {
                Pair<String, String> pair = it.next();
                String callMethod = pair.snd();
                String url = pair.fst();
                try {
                    String result = invokeWS.obtainDataAndCallWS(url, callMethod, params);
                    invokerUI.getTxtResult().setText(result);
                    invokerUI.getWsInvoked().setText(wsdlCRUD.findByUrl(url).getString("name"));
                    invokerUI.enableResult(true);
                    invoke = true;
                } catch (Exception e2) {
                    System.err.println(e2);
                }
            }
            if (!invoke) {
                invokerUI.getTxtResult().setText("There is no coincidence with the method and
                    parameters selected");
            }
        } else {
            invokerUI.getTxtResult().setText("There is no coincidence with the method and
                parameters selected");
        }
    }
```

**Fase 5: Calificación del WS's**

Esta última fase le solicita al usuario información respecto a la calidad del servicio preguntándole si la salida ha satisfecho sus necesidades. Luego de que compute la respuesta, esta se utiliza para calcular la

reputación del servicio.

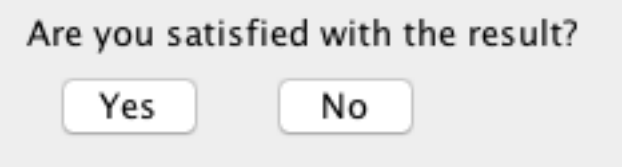


Figura 5.2: Interfaz de usuario correspondiente a la fase 5

Código 5.4: Controlador que captura la respuesta del usuario

```
1       if (e.getSource() == invokerUI.getBtnYes()) {
2           /**
3            * The invocation served to the user, calculates the new reputation.
4            */
5           Wsdl wsdl = wsdlCRUD.findByName(invokerUI.getWsInvoked().getText());
6           wsdlCRUD.editReputation(wsdl, true);
7           invokerUI.enableResult(false);
8       }
9       if (e.getSource() == invokerUI.getBtnNo()) {
10          /**
11           * The invocation not served to the user, calculates the new
12           * reputation.
13           */
14          Wsdl wsdl = wsdlCRUD.findByName(invokerUI.getWsInvoked().getText());
15          wsdlCRUD.editReputation(wsdl, false);
16          invokerUI.enableResult(false);
17      }
```

Código 5.5: Método que edita la reputación del WS en la base de datos

```
1   /**
2    * edit onyl the reputation of wsdl,
3    * if boolean param is true, reputation'll raise
4    *
5    * @param w
6    * @param b
7    * @return
8    */
9   public boolean editReputation(Wsdl w, boolean b) {
10      boolean ret = true;
11      Wsdl old = Wsdl.findById(w.getId());
12      if (old != null) {
13          Base.openTransaction();
14          if (b) {
15              w.setLong("reputation", old.getLong("reputation") + 1 / 2);
16          } else if (old.getLong("reputation") > 1) {
17              w.setLong("reputation", old.getLong("reputation") - 1 / 2);
18          } else {
19              w.setLong("reputation", 0);
20          }
21          ret &= old.set("reputation", w.get("reputation")).saveIt();
22          Base.commitTransaction();
23          return ret;
24      }
25      return !ret;
26  }
```

## 5.3. Diseño de la herramienta

### 5.3.1. Arquitectura general

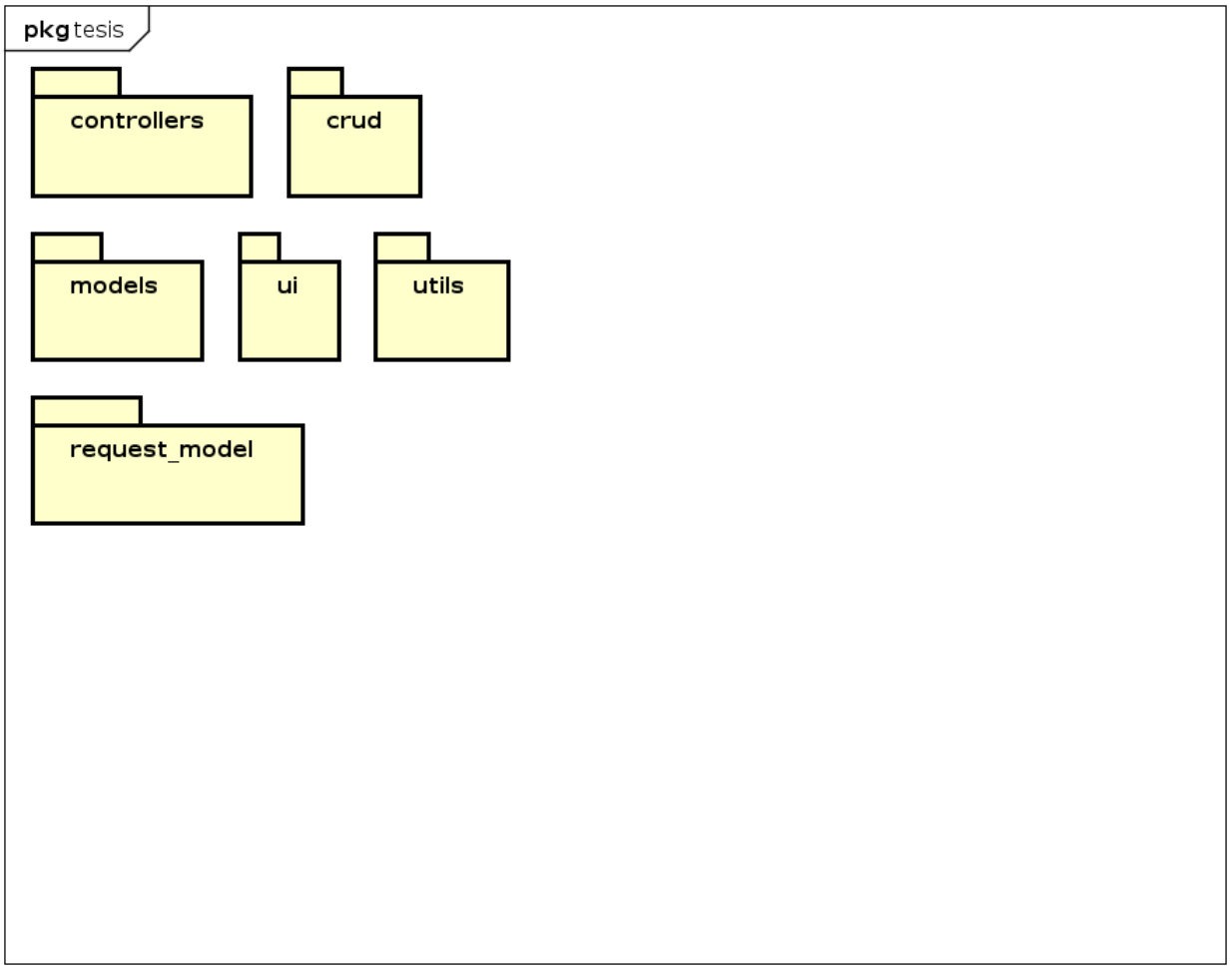


Figura 5.3: Paquetes del proyecto

La arquitectura del programa se basa en el patrón modelo–vista–controlador (MVC) [2], el cual separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. A continuación se describen los paquetes de la figura:

- **controllers:** Clases que gestionan los eventos y las comunicaciones entre los modelos y la interfaz de usuario.
- **crud:** Clases que gestionan las altas, bajas y modificaciones de los modelos.
- **models:** Cases que representan los modelos de categoría y wsdl necesarias con el uso de la librería Active JDBC.
- **ui:** Clases que implementan la interfaz de usuario.

- **utils:** Clases de utilidad en el sistema.
- **request_model:** Contiene el modelo de los requisitos del usuario.

MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado componentes para la interacción del usuario. Por ejemplo, el módulo de gestión de categorías está diseñado de la forma en que se observa en la figura 5.4.

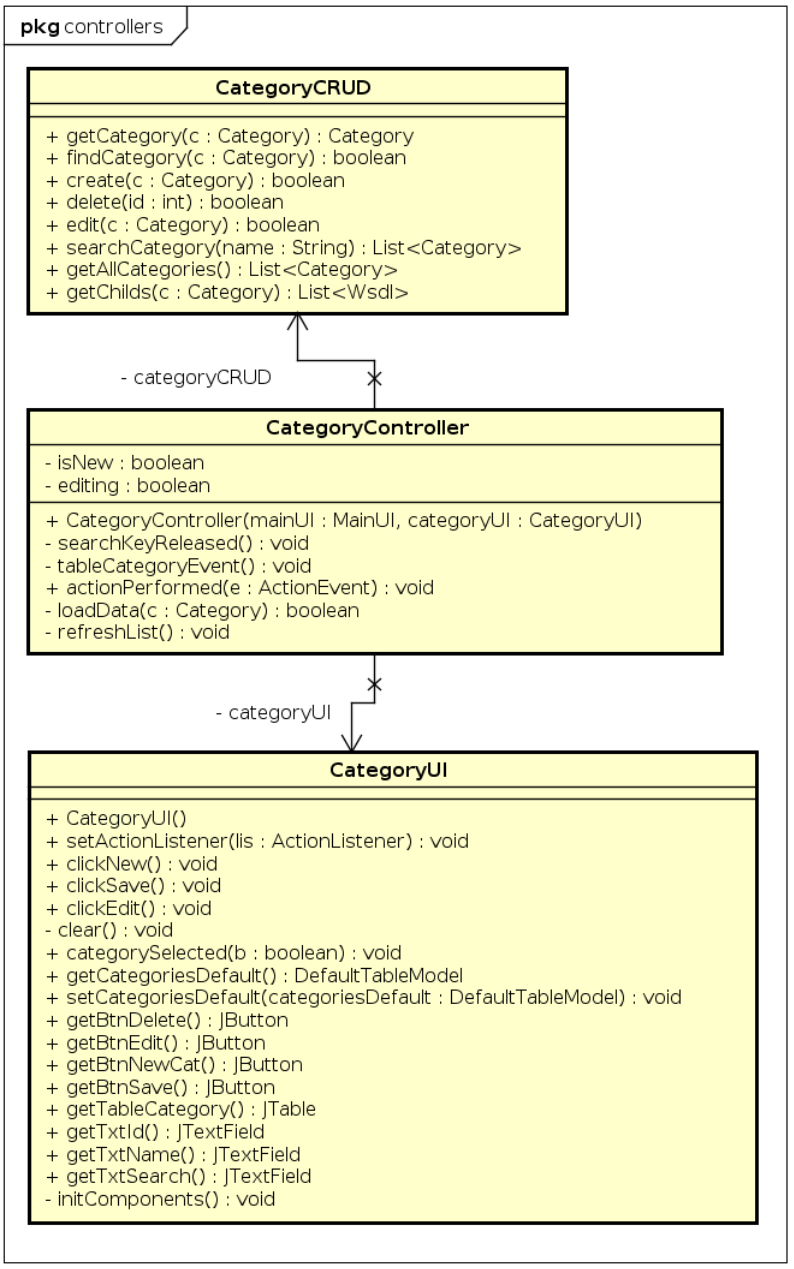


Figura 5.4: Paquete controllers ejemplo Category.

Claramente se puede observar que CategoryUI implementa la interfaz de usuario, CategoryCrud el modelo y CategoryController el controlador.

El diagrama de clase para WsdlCrud es similar al de CategoryCrud, solo difiere en los atributos.

El diagrama de clases de diseño de la herramienta QVTWSInvoker ilustrado en la figura 5.5, muestra cómo se relacionan las principales clases.

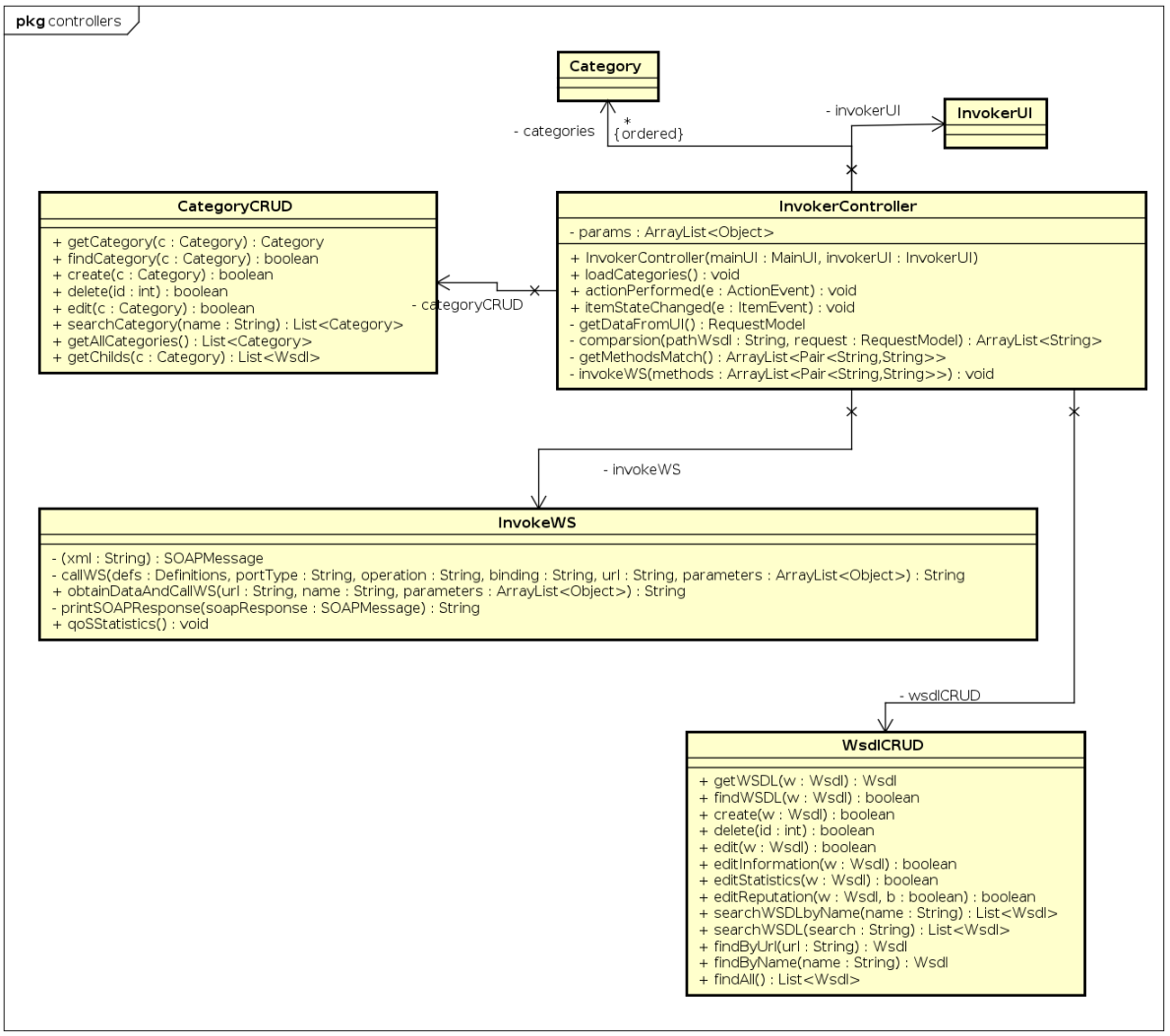


Figura 5.5: Diagrama de clases general.

En este diagrama (figura 5.5) se puede observar que las clases InvokeWS y InvokerController son las más importantes.
La clase InvokeController es la encargada de obtener la petición desde la interfaz de usuario, para luego obtener los wsdl de la categoría elegida a través del CRUD wsdlCRUD. Una vez que se tienen los wsdl y se le realiza la transformación QVT y obtención del mejor WS, se pasa a la etapa de ejecución del método del wsdl. Para ello, se utiliza la clase InvokeWS, esta fue construida con la ayuda de la librería SOA_Distribution la cual permite automatizar la invocación a un método SOAP, una vez invocado, el resultado es mostrado por la interfaz de usuario a través del controlador InvokerController.

### 5.3.2. Modelo de Persistencia

El diagrama de clases persistentes de la herramienta QVTWSInvoker se muestra en la figura 5.6. Este conjunto de clases es responsable de almacenar la información de los WS's por cada ejecución.

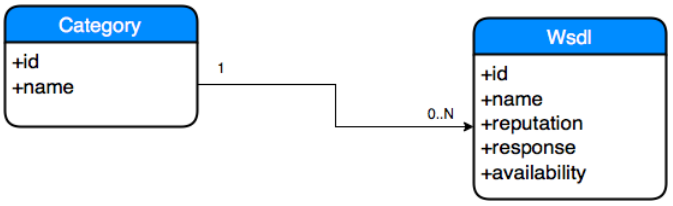


Figura 5.6: Modelo persistente de la herramienta QVTWSInvoker.

A continuación se describen las tablas de la figura:

La tabla Category representa las categorías de los WS's, tiene un identificador que es clave y un nombre, por su parte, la tabla Wsdl se encarga de mantener la información sintáctica básica de cualquier wsdl: identificador, nombre, url y categoría, además de guardar la información respecto a su calidad de servicio (reputación, tiempo de respuesta y disponibilidad).

## 5.4. Herramientas utilizadas

En esta sección se describen las diferentes herramientas y tecnologías utilizadas para el desarrollo de este proyecto:

**NetBeans y Eclipse IDE** son entornos de desarrollo integrado, que proporcionan servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador.

**Java 1.8** es un lenguaje de programación orientado a objetos que pertenece a la empresa Sun Microsystems. Entre sus características podemos nombrar: simple, distribuido, robusto, portable, interpretado, multitareas. Se utilizó como lenguaje de programación base dentro de la herramienta NetBeans.

**GlassFish** es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL.

**H2** es un sistema administrador de bases de datos relacionales programado en Java. Una de las características más importantes de H2 es que se puede integrar completamente en aplicaciones Java y acceder a la base de datos lanzando SQL directamente, sin tener que pasar por una conexión a través de sockets. Está disponible como software de código libre bajo la Licencia Pública de Mozilla o la Eclipse Public License.

**Eclipse Modeling Tools** contiene marcos de trabajos y herramientas para trabajar con modelos: Un modelador gráfico de Ecore, utilidad de generación de código Java para aplicaciones RCP y el marco de trabajo EMF, soporte para comparación de modelos, para esquemas XSD, OCL y modeladores gráficos en tiempo de ejecución.

# Capítulo 6

# Conclusiones y trabajos futuros

## 6.1. Conclusiones

En la actualidad la automatización de los procesos de negocios (Workflows) se ha convertido en un objetivo primordial para el crecimiento y desarrollo de las organizaciones. El conflicto de hoy en día a fin orden de lograr una plena automatización de los procesos, radica en la comunicación de los mismos. Y es en este sentido que la interfaz 3 de un Workflow, la de Invocación de Aplicaciones Externas, toma preponderante significado e importancia. Por ello, en función de aportar una significativa mejoría a esta comunicación, esta tesis ha llevado a cabo las siguientes propuestas expuestas con detalle a lo largo de los capítulos anteriores:

- El uso del programa no solo como parte de la Interfaz de las Aplicaciones Invocadas de los WFMS, sino como un módulo externo a cualquier sistema capaz de manipular y automatizar la selección del servicio más optimo.
- La definición y automatización de un método de búsqueda, selección, ejecución e interpretación de resultados de WS ya conocidos.
- La incorporación de modelos Ecore, la transformación de modelos QVT y la medición de las QoS para describir los aspectos no funcionales de los WS's y con ello la optimización de las invocaciones externas a un Workflow en cuanto a este tipo de aplicaciones.
- La aplicación e implementación de todo lo propuesto a un caso de estudio concreto.

Por otro lado, la utilización de WS's en el contexto de la Interfaz de Aplicaciones Invocadas, tiene el principal beneficio o ventaja de aprovechar la información disponible en la web a través de éstos, es decir, tener la posibilidad de que las aplicaciones externas al WFMS estén distribuidas en la red, en cualquier parte del mundo, y que ello resulte completamente transparente al WFMS, e incluso al motor del mismo. Así se fundamenta que tanto la búsqueda, como la selección y la ejecución de estas aplicaciones previamente conocidas, se realice de forma dinámica y automática.
Otra contribución importante de esta tesis es la optimización de la invocación (y con ello la comunicación) entre un WFMS y las aplicaciones con quienes debe interactuar. Esta optimización consiste en tener en cuenta a la hora de la selección de aplicaciones los atributos no funcionales, o QoS, de los WS's. Ello permite compartir recursos distribuidos a gran escala, lo cual puede implicar por ejemplo la integración entre Workflows. El uso de WS's plantea una forma unificada de acceder a las aplicaciones. Este acceso posibilita la interacción con servicios y/o métodos comunes favoreciendo la interoperabilidad a través de las diferentes plataformas. Existen numerosas ventajas que benefician la distribución de las aplicaciones y la interoperabilidad de los sistemas al utilizar este tipo de aplicaciones, dado que permiten la utilización de aplicaciones de componentes abiertas y auto descriptas, la composición rápida de aplicaciones distribuidas, el uso de aplicaciones modulares y desacopladas, la independencia de la plataforma subyacente y el lenguaje de programación, el acoplamiento débil, etc.

Por todas estas razones es que se convierten en un excelente recurso para permitir la distribución transparente de las aplicaciones externas a los WFMS.

## 6.2. Trabajos Futuros

A continuación se enumeran las posibles extensiones para este trabajo:

En primer lugar, se considera importante que la herramienta otorgue diferentes posibilidades en cuanto a la selección de las QoS deseadas en la aplicación que se quiera buscar. Resulta muy interesante ampliar este espectro y así permitir al usuario de la herramienta escoger entre un conjunto más amplio de QoS.

También es necesario que la herramienta sea apta para manipular intercambio de mensajes REST, debido al gran auge que están teniendo, y así dar la posibilidad de que esta comunicación no se realice únicamente a través del protocolo SOAP.

Al mismo tiempo, la herramienta necesita el uso de la plantilla de salida estándar para desligar al usuario de la responsabilidad de interpretar las diferentes sintaxis de los resultados obtenidos. Esto no solo reduciría la complejidad que implica implementar en un Workflow la interfaz de invocación de aplicaciones, sino que también logra estandarizar el resultado para cualquier tipo de WS especificado.

Una variante a lo propuesto en este trabajo, es el hecho de utilizar procesamiento del lenguaje a la hora de hacer la comparación de modelos y así poder utilizar otra información, como por ejemplo la descripción.

# Bibliografía


[1] Apache. *JUDDI. Versión 3.3.2*, último acceso Enero 2016.

[2] Ralph Johnson John M. Vlissides Erich Gamma, Richard Helm. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.

[3] IETF. *Hypertext Transfer Protocol - HTTP/1.1*, último acceso Abril 2016.

[4] J. Miller and J. Mukerji. *"Mda guide version 1.0.1," Object Management Group (OMG)*, 2003.

[5] Daniel Riesco German Montejano Narayan Debnath Marcela Daniele, Paola Martelotto. *Chapter: "Defining and using metrics to evaluate the QoS of Web Services invoked from a Workflow engine"Book: "Engineering High Quality Software: Correctness, Testing, and Verification"*. 2011. Editors: Dr. Mark Burgin (UCLA, USA) and Dr. Narayan Debnath (Winona State University, USA) (Editores), IRM Press. Idea Group Inc. (IGI).

[6] OASIS. *Universal Description, Discovery and Integration of WebServices–UDDI. Versión 3.0.2*. último acceso Abril 2016.

[7] Martelloto Paola. Definición de reglas de transformación de grafos y qos para la interfaz entre aplicaciones externas a un workflow con ws's. Tesis de maestría Universidad de San Luis, 2010.

[8] Garat M. Gil P.M. Un web service que automatiza la invocación de aplicaciones externas a un workflow optimizando la selección de las mismas en base a característica de calidad. Tesis de grado de la Licenciatura en Ciencias de la Computación, Universidad Nacional de Río Cuarto, 2010.

[9] S. J. Mellor, A. N. Clark, and T. Futagami. *"Guest editors' introduction: Model- driven development," IEEE Software, vol. 20*, 2003.

[10] Sumathi-Niranjan. *Dynamic search and selection of WS's*. 2014.

[11] W3C Working Group. *QoS for Web Services: Requirements and Possible Approaches*, último acceso Abril 2016.

[12] Workflow Management Coalition. *A Common Object Model Discussion Paper. WfMC Document WfMC-TC10-22*, último acceso Abril 2016.

[13] Workflow Management Coalition. *Programming Interface 2 and 3 Specification WfMC-TC-1009. V2.0*, último acceso Abril 2016.

[14] Workflow Management Coalition. *The Workflow Reference Model. WfMC-TC00-1003*, último acceso Abril 2016.

[15] World Wide Web Consortium. *Extensible Markup Language (XML)1.0(Fifth Edition)*, último acceso Abril 2016.

[16] World Wide Web Consortium. *SOAP Version 1.2 Part 1: Messaging Framework*, último acceso Abril 2016.

[17] World Wide Web Consortium. *WS's Architecture*, último acceso Abril 2016.

[18] World Wide Web Consortium. *WS's Description Language(WSDL) Version 2.0 Part 0: Primer*, último acceso Abril 2016.

[19] World Wide Web Consortium. *WS's Description Language(WSDL) Version 2.0 Part 1: Core Language*, último acceso Abril 2016.

# Apéndices

# Apéndice A

# WSDL

## A.1. Web Service Description Language

### A.1.1. XML: el lenguaje de los WS's

Si bien WSDL (Web Service Description Language) es en realidad el lenguaje de especificación utilizado para definir los WS's, esencialmente se trata de un documento XML que es utilizado para describir los mensajes SOAP y cómo estos mensajes son intercambiados.

#### Significado de XML

XML es el estándar de Extensible Markup Language. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados. Respecto a sus objetivos, estos son:

- XML debe ser directamente utilizable sobre Internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos XML.
- El número de características opcionales en XML debe ser absolutamente mínimo, idealmente cero.
- Los documentos XML deben ser legibles por humanos y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fácilmente creables.
- La concisión en las marcas XML es de mínima importancia.

Por otra parte, se pueden enumerar sus principales características:

1. Es una arquitectura abierta y extensible. No se necesita versiones para que puedan funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/intranet por medio de un validador de documentos.
2. Mayor consistencia, homogeneidad y amplitud de los identificadores descriptivos del documento con XML (los RDF Resource Description FrameWork), en comparación a los atributos de la etiqueta del HTML.
3. Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local o extensa.

4. Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje nos permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.

5. Gestión y manipulación de los datos desde el propio cliente web.

6. Los motores de búsqueda devolverán respuestas más adecuadas y precisas, ya que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible.

7. Se desarrollarán de manera extensible las búsquedas personalizables y subjetivas para robots y agentes inteligentes. También conllevará que los clientes web puedan ser más autónomos para desarrollar tareas que actualmente se ejecutan en el servidor.

8. Se permitirá un comportamiento más estable y actualizable de las aplicaciones web, incluyendo enlaces bidireccionales y almacenados de forma externa (El famoso epígrafe "404 file not found"desaparecerá).

9. El concepto de "hipertexto"se desarrollará ampliamente (permitirá denominación independiente de la ubicación, enlaces bidireccionales, enlaces que pueden especificarse y gestionarse desde fuera del documento, hiperenlaces múltiples, enlaces agrupados, atributos para los enlaces, etc. Creado a través del Lenguaje de enlaces extensible (XLL).

10. Exportabilidad a otros formatos de publicación (papel, web, CD-ROM, etc.). El documento maestro de la edición electrónica podría ser un documento XML que se integraría en el formato deseado de manera directa.

**Estructura del XML**

El metalenguaje XML consta de cuatro especificaciones :

1. **DTD (Document Type Definition)** Definición del tipo de documento. Es, en general, un archivo que encierra una definición formal de un tipo de documento y, a la vez, especifica la estructura lógica de cada documento. Define tanto los elementos de una página como sus atributos. El DTD del XML es opcional.

2. **XSL (eXtensible Stylesheet Language)** Define o implementa el lenguaje de estilo de los documentos escritos para XML.. Permite modificar el aspecto de un documento. Se puede lograr múltiple columnas, texto girado, orden de visualización de los datos de una tabla, múltiples tipos de letra con amplia variedad en los tamaños. Este estándar está basado en el lenguaje de semántica y especificación de estilo de documento (DSSSL, Document Style Semantics and Specification Language, ISO/IEC 10179).

3. **XLL (eXtensible Linking Language)** Define el modo de enlace entre diferentes enlaces. Se considera que es un subconjunto de HyTime (Hipermedia/Timed-based structuring Language ó Lenguaje de estructuración hipermedia/basado en el tiempo, ISO 10744) y sigue algunas especificaciones del TEI (Text Encoding Initiative o Iniciativa de codificación de texto).
   Este lenguaje de enlaces extensible tiene dos importantes componentes: Xlink y el Xpointer. Va más allá de los enlaces simples que sólo soporta el HTML. Se podrá implementar con enlaces extendidos. Jon Bosak establece los siguientes mecanismos hipertextuales que soportará esta especificación:

   - Denominación independiente de la ubicación.
   - Enlaces que pueden especificarse y gestionarse desde fuera del documento a los que se apliquen.
   - Hiperenlaces múltiples.
   - Enlaces agrupados.
   - Transclusión (el documento destino al que apunta el enlace aparece como parte integrante del documento rigen del enlace).
   - Se pueden aplicar atributos a los enlaces.

4. **XUA (XML User Agent)** Estandarización de navegadores XML.
   Todavía está en proceso de creación de borradores de trabajo. Se aplicará a los navegadores para que compartan todas las especificaciones XML

### A.1.2. WSDL y la documentación de WS's

El lenguaje de descripción de WS's (WSDL, Web Service Description Language) es un dialecto basado en XML sobre el esquema que describe un WS. Un documento WSDL proporciona la información necesaria al cliente para interaccionar con el WS.
Los documentos WSDL definen los servicios como colecciones de puntos finales de red o puertos. En WSDL, la definición abstracta de puntos finales y de mensajes se separa de la instalación concreta de red o de los enlaces del formato de datos. Esto permite la reutilización de definiciones abstractas: mensajes, que son descripciones abstractas de los datos que se están intercambiando y tipos de puertos, que son colecciones abstractas de operaciones. Las especificaciones concretas del protocolo y del formato de datos para un tipo de puerto determinado constituyen un enlace reutilizable. Un puerto se define por la asociación de una dirección de red y un enlace reutilizable; una colección de puertos define un servicio. Por esta razón, un documento WSDL utiliza los siguientes elementos en la definición de servicios de red:

- Types: contenedor de definiciones del tipo de datos que utiliza algún sistema de tipos.
- Message: definición abstracta y escrita de los datos que se están comunicando.
- Operation: descripción abstracta de una acción admitida por el servicio.
- Port Type: conjunto abstracto de operaciones admitidas por uno o más puntos finales.
- Binding: especificación del protocolo y del formato de datos para un tipo de puerto determinado.
- Port: punto final único que se define como la combinación de un enlace y una dirección de red.
- Service: colección de puntos finales relacionados.

### A.1.3. WSDL y la descripción de los WS's

WSDL separa la descripción de un servicio en dos partes. Una de ellas es la interfaz abstracta, la cual describe las operaciones soportadas por el servicio, los parámetros de la operación y los tipos de datos abstractos. La otra parte es la implementación concreta, la cual liga la interfaz abstracta a una dirección de red, protocolo y estructuras de datos concretas. WSDL provee toda esta información a través de elementos XML.
Además, la descripción de un WS tiene dos componentes principales: sus características funcionales y no funcionales. La descripción funcional detalla las características operacionales que definen el comportamiento general de un WS. La descripción no funcional se concentra en sus atributos de calidad (Quality of Service - QoS), forzando al solicitante del servicio a especificar, los atributos de calidad que pueden influir en la elección de un WS ofrecido por un proveedor.

### A.1.4. Estructura de un documento WSDL

En la primera versión de este lenguaje, WSDL 1.1, el documento que describe un WS está compuesto por un elemento raíz llamado definitions, que a su vez está compuesto por los siguientes elementos:

- **types:** provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar.
- **message:** representa una definición abstracta de los datos a ser transmitidos entre el servidor y el cliente.
- **portType:** define un conjunto de operaciones abstractas. Cada operación se refiere a un mensaje de entrada y mensajes de salida.
- **binding:** define un protocolo concreto y las especificaciones de formato de datos para las operaciones y los mensajes definidos en un portType particular.
- **port:** el cual especifica una dirección de enlace, definiendo un punto final de comunicación.
- **service:** se usa para agrupar un conjunto de puertos relacionados.

La nueva versión de este lenguaje, WSDL 2.0 [18], hereda la mayoría de los principios arquitecturales de WSDL 1.1 e incorpora, los conceptos de interface abstracta, protocolo de enlace (binding), y puntos finales de servicios (service endpoints). Algunos de los cambios incorporados en WSDL 2.0 son:
Agrega más semántica al lenguaje de descripción;
Remueve el elemento message: este se especifica dentro del elemento types, usando XML Schema Type System;
No soporta la sobrecarga de operadores;

Renombra el elemento portTypes a interfaces: La herencia de interfaces se consigue el atributo extends en el elemento interface;
Renombra los ports a endpoints;
Incorpora patrones de mensajes abstractos.

En esta versión los elementos principales que componen el documento que describe el WS, los cuales están contenidos en el elemento description, son:

- **types:** provee la definición de los tipos de datos utilizados para describir los mensajes a intercambiar.
- **interface:** describe la secuencia de mensajes que el servicio envía o recibe.
- **binding:** define un protocolo concreto y las especificaciones de formato de datos para las operaciones y los mensajes definidos en un portType particular.
- **service:** se usa para agrupar un conjunto de puertos relacionados.

La sección abstracta contiene los elementos types e interface (message y operations), mientras que la sección concreta contiene los elementos binding y service.

```
<description targetNamespace="xs:anyURI">
 <documentation />*
 [<import /> | <include />]*
 <types />?
 [<interface /> | <binding /> | <service />]*
</description>
```

Figura A.1: Descripción de un WS

**El Elemento Types**

El elemento types incluye la definición de los tipos de datos que son relevantes para el intercambio de mensajes y que se necesitarán posteriormente para la definición. Ellos son los parámetros de entrada y salida respectivamente.(ver figura A.2)

**El Elemento Interface**

El elemento interface describe la secuencia de mensajes que el servicio envía o recibe. Para ello agrupa los mensajes en operaciones. Una operación es una secuencia de mensajes de entrada y salida, y una interface es un conjunto de operaciones.(ver figura A.3)

```
<description>
 <types>
       <documentation />*
       [<xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"?/> |
        <xs:schema targetNamespace="xs:anyURI"? /> |
        Other extension elements]*
 </types>
</description>
```

Figura A.2: Elemento Types de un WS

El conjunto de operaciones disponibles en un interface incluye todas las operaciones definidas por las interfaces que ella extiende, directa o indirectamente, más las operaciones que ella define directamente. Las propiedades de una interface son:

- Nombre (name).
- Interfaces que extiende (extends).
- Componentes de falla (fault): describe las fallas que pueden ocurrir durante la
- invocación de una operación de la interface.
- Operaciones (operation): describe una operación que ocurre dentro de la interface que la soporta. Una operación es una interacción con el servicio, que consiste de un conjunto de mensajes intercambiados entre el servicio y las otras partes involucradas en la interacción.

```
<description>
 <interface
       name="xs:NCName"
       extends="list of xs:QName"?
       styleDefault="list of xs:anyURI"? >
       <documentation />*
       [<fault /> | <operation/>]*
 </interface>
</description>
```

Figura A.3: Elemento Interface de un WS

**El Elemento Binding**

El elemento Binding define los detalles de implementación concretos, que serán necesarios para acceder al servicio.(ver figura A.4)

```
<description>
 <binding
        name="xs:NCName"
        interface="xs:QName"?
        type="xs:anyURI"? >
        <documentation />*
        [<fault /> | <operation/>]*
 </binding>
</description>
```

Figura A.4: Elemento Binding de un WS

El binding debe especificar la interface a la que se aplica y definir bindings para todas las fallas definidas en la interface, que son referenciadas en las operaciones. Las propiedades de este componente son:

- nombre (name).
- interface (interface), para la cual se especifica el componente binding.
- tipo (type), que indica los detalles de binding concretos.
- falla (binding faults): describe un binding concreto para una falla particular descripta dentro de la interface.
- operaciones (binding operations): define el formato de mensaje concreto y el protocolo de interacción asociados con una operación de interface particular.

**El Elemento Service**

El elemento Service asocia un binding con el URL donde está realmente el servicio a través de un endpoint. Los endpoints son lugares alternativos que proveen el servicio. (ver figura A.5)

Las propiedades de este componente son:

- nombre (name).
- interface (interface), la interface que el servicio instancia.
- punto final (endpoint): define las particularidades de un punto final específico en el cual el servicio está disponible.

**Elemento message**

El elemento Message proporciona una abstracción común para el paso de mensajes entre el cliente y el servidor. Como puede utilizar múltiples formatos de definición de esquema en documento WSDL es necesario de disponer de un mecanismo común de identificar los mensajes. El elemento Message proporciona este nivel común de abstracción al que se hará referencia en otras partes del documento WSDL. Cada mensaje contiene uno o más elementos "Part"que describen las piezas del contenido del mensaje.

```
<description>
 <service
       name="xs:NCName"
       interface="xs:QName"? >
       <documentation />*
       <endpoint />+
 </service>
</description>
```

Figura A.5: Elemento Service de un WS

**Elemento portType**

El elemento portType contiene un conjunto de operaciones abstractas que representan los tipos de correspondencia que pueden producirse entre el cliente y el servidor. WSDL define cuatro tipos de operaciones:

- Request-response (petición-respuesta). Comunicación del tipo RPC en la que el cliente realiza una petición y el servidor envía la correspondiente respuesta.
- One-way (un-sentido). Comunicación del estilo documento en la que el cliente envía un mensaje pero no recibe una respuesta del servidor indicando el resultado del mensaje procesado.
- Solicit-response (solicitud-respuesta). La contraria a la operación petición-respuesta. El servidor envía una petición y el cliente le envía de vuelta una respuesta.
- Notification (Notificación). La contraria a la operación un-sentido el servidor envía una comunicación del estilo documento al cliente.

#### A.1.4.1. Elementos de Extensibilidad

Los elementos de extensibilidad se utilizan para representar determinadas tecnologías.