Universidad Nacional del Comahue
Facultad de Informática

Tesis de Licenciado en Ciencias de la Computación

Evaluación de Servicios Web mediante un Metamodelo de Contratos de servicios, basado en el estándar SoaML

Lucas Nahuel Cavaliere

Directores:

Dr. Andrés Flores **Lic. Alan De Renzis**

Neuquén Argentina

Abril 2018

Prefacio

Esta tesis es presentada como parte de los requisitos finales para optar al grado Académico de *Licenciado en Ciencias de la Computación*, otorgado por la Universidad Nacional del Comahue, y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otras. La misma es el resultado de la investigación llevada a cabo en la *Facultad de Informática* en el periodo comprendido entre Noviembre de 2016 y Abril de 2018, bajo la dirección del Dr. Andrés Flores y del Lic. Alan De Renzis.

Lucas Nahuel Cavaliere
Facultad de Informática
Universidad Nacional del Comahue
Neuquén, 30 de Mayo de 2018

UNIVERSIDAD NACIONAL DEL COMAHUE

Facultad de Informática

La presente tesis ha sido aprobada el día ....., mereciendo la calificación de .....

Resumen

Una práctica común para el desarrollo de software es reusar funcionalidad provista por terceras partes, lo cual no sólo ayuda a reducir los costos, sino también a enfocar el proceso de desarrollo en la funcionalidad principal del sistema. En esta tesis se profundizó sobre un Método de Selección de Servicios Web que tiene como objetivo asistir a los ingenieros de software en la construcción de Aplicaciones Orientadas a Servicios, proponiendo una extensión del proceso de Selección y de su herramienta de soporte. En particular, se propusieron mejoras sobre el procedimiento de análisis de interfaces (validación contractual) completamente basado en la especificación funcional (descripta en WSDL versiones 1.1 y 2.0) de los Servicios Web, que permite extraer toda la información disponible en dicha especificación sin requerir de un marcado semántico adicional. En la actualidad, es necesario contar con una especificación de contratos de Servicios Web, que sea independiente de cualquier tecnología de implementación, y que cubra convenientemente la existencia de servicios heterogéneos. En consecuencia, se ha decidido como primera etapa del trabajo, desarrollar un Metamodelo utilizando como base un conjunto de estándares para descripción de contratos/responsabilidades de Servicios Web heterogéneos. Se tomaron como base distintos estándares de descripción de servicios, tal como el Perfil UML SoaML, y el lenguaje WSDL versión 2.0 ambos estándares de OMG. Al contar con un Metamodelo donde se describen contratos ofrecidos por Servicios Web, resultaba necesario que la evaluación de servicios candidatos se realizara en función de instanciaciones del Metamodelo. Para ello, se diseñó e implementó un componente de software en la plataforma Java que contiene el Metamodelo permitiendo la exploración de la información de contratos de servicios sobre las instancias de este último. La segunda etapa fue integrar el componente del Metamodelo en la Herramienta de Evaluación de Servicios Web adaptando las estrategias de evaluación subyacentes, tanto a nivel estructural o de tipos de datos en los mensajes de entrada/salida, como a nivel semántico o de nombres en los identificadores de operaciones y mensajes. Esta adaptación se realizó de acuerdo a los elementos del Metamodelo definido y su correspondencia mediante interfaces Java. Como tercera etapa se creó un Conversor de descripciones WSDL hacia instanciaciones del Metamodelo, el cual explora los elementos que forman un documento WSDL y produce instanciaciones correspondientes a los elementos que componen el Metamodelo.

Finalmente se realizó una evaluación experimental a partir de la cual comprobamos que la utilización de la Herramienta de Evaluación de Servicios Web mejora la visibilidad de los servicios relevantes – estas mejoras se expresan en términos de ganancias en Precisión. Considerando que la selección de servicios candidatos se realiza luego de algún proceso de descubrimiento, incrementar la visibilidad de los candidatos más adecuados facilita en gran medida el desarrollo de Aplicaciones Orientadas a Servicios.

Abstract

A common practice for software development is to reuse functionality provided by third parties, which not only helps to reduce costs, but also to focus the development process on the main functionality of the system. This thesis is focused in a Web Services Selection Method that aims to assist software engineers building Service Oriented Applications. We propose an extension to improve the Selection process and its support tool. Improvements mainly concern the interface analysis procedure (contractual validation) completely based on the functional specification (described in WSDL versions 1.1 and 2.0) of Web Services. Such functional specification allows extracting all the information available without the need of an additional semantic markup. Nowadays, the existence of heterogeneous services makes it necessary to count with a specification of Web Services contracts, conveniently featured as independent of any implementation technology. Consequently, as a first step was decided to develop a Metamodel under the basis of a set of standards to describe contracts/responsibilities of heterogeneous Web Services. Different service description standards, such as the SoaML UML Profile, and the WSDL version 2.0 language – both OMG standards – were applied to define the Metamodel. As a consequence, the evaluation of candidate services should be now under Metamodel instantiations. Thereby, the Metamodel was designed as a software component with a Java implementation. This component allows to explore the information of service contracts upon the Metamodel's instances. The second step was to integrate the Metamodel component into the Web Services Evaluation Tool adapting the underlying evaluation strategies. This involves the structural level with data types in the input/output messages; and the semantic level with names from identifiers of operations and messages. This adaptation was made according to the elements of the defined Metamodel and their correspondence to those from Java interfaces. As a third step, a Converter component was built from WSDL descriptions towards instantiations of the Metamodel. The converter explores elements comprising a WSDL document and produces instantiations of corresponding elements from within the Metamodel.
Finally, an experimental evaluation was carried out, from which we verified that the use of the Web Services Evaluation Tool improves the visibility of the relevant services. These improvements are expressed in terms of Precision gains. Considering that the selection of candidate services is made after a discovery process, increasing the visibility of the most suitable candidates greatly facilitates the development of Service-Oriented Applications.

Agradecimientos

Agradezco en especial a mis padres Rosa Panetta y Pascual Cavaliere y a mi pareja Yesica Riquelme que siempre insistieron y se empeñaron a que terminara la carrera de Licenciado en Ciencias de la Computación, por todo el apoyo incondicional y comprensión a la hora de suspender actividades por temas de estudio. Sumado a las horas de ausencia que tuvieron que tolerar tanto para la producción de la tesis, como para la carrera universitaria en si misma. A todos mis amigos y compañeros que me acompañaron durante los distintos momentos universitarios, con los cuales compartimos gratos momentos.

Al Dr. Andrés Flores por permitirme realizar este trabajo bajo su dirección, y por estar siempre atento y disponible en cada momento requerido aportando visiones estructurales claras y correcciones a la altura esperada.

Al Lic. Alan De Renzis, por su gran dedicación y arduo esfuerzo tanto en tiempos de correción, como de guia en el desarrollo de mi tesis. Por la asistencia técnica, respuesta fuera de horarios laborales y excelente predisposición por sobre todas las cosas.

Al Instituto PROBIEN-CONICET por apoyar la finalización de mis estudios. Especialmente al Dr. Germán Mazza, que con convicción en mi persona, ha impulsado el crecimiento de mi carrera hacia una especialización destacable.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Motivación

1.1. Introducción

El paradigma de Computación Orientada a Servicios (SOC[1]) se percibe como un campo interdisciplinar para el estudio, diseño, e implementación de sistemas orientados a servicios, y actúa como un paraguas que cubre todos los aspectos de computación utilizados – especificación y diseño orientado a servicios, Arquitectura Orientada a Servicios (SOA[2]), Servicios Web, etc. – siendo actualmente una de las áreas de investigación más activas en el ámbito de la informática. La provisión de servicios y la innovación de los mismos están basadas sobre todo en las tecnologías de la información [49].

Una práctica común para el desarrollo de software es reusar funcionalidad provista por terceras partes, lo cual no sólo ayuda a reducir los costos, sino también a enfocar el proceso de desarrollo en la funcionalidad principal del sistema. En este contexto, el crecimiento de la Web habilita a los desarrolladores a ofrecer software no sólo en forma de bibliotecas, sino como servicios – componentes software que se pueden invocar de forma dinámica. En particular, la tecnología de Servicios Web es uno de los principales motivos para la adopción del paradigma SOC [4].

El sector de los servicios representa el mayor porcentaje de la economía en los países desarrollados, y es innegable que una gran parte de la innovación en servicios se basa en la tecnología informática, que ha contribuido con aportes concretos que configuran la base tecnológica para la implementación de servicios: SOA, Servicios Web o Computación en la Nube (*Cloud Computing*). Estos conceptos obligan a revisar los paradigmas de desarrollo de software, del mismo modo que en el ámbito de la gestión de proyectos se está evolucionando de una lógica basada en el producto a una basada en el servicio [54], obligando a reformular conceptos como marketing, innovación, sostenibilidad, ciclo de vida, etc., de producto a servicio. En el ámbito de la informática también es necesario revisar todo lo existente en ingeniería de software para el desarrollo de sistemas de información clásicos, y adaptarlo para el desarrollo de sistemas orientados a servicios [49].

El paradigma SOC está siendo ampliamente aceptado como medio para abordar la actualización y automatización de un proceso de negocio abierto y colaborativo, principalmente porque permite exponer las competencias de una organización de manera estándar, programática

[1] SOC: *Service-Oriented Computing*
[2] SOA: Service Oriented Architecture

y bajo pautas de celeridad y reducción de costos [42].

En general, las Aplicaciones Orientadas a Servicios se basan en la tecnología de Servicios Web: programas con una interfaz bien definida que puede ser localizada, publicada e invocada utilizando la infraestructura estándar de la Web [4]. El paradigma SOC se presenta como una evolución de la construcción tradicional de sistemas desde cero hacia la reutilización masiva de software, generando un proceso de desarrollo basado en el descubrimiento y combinación de piezas de software provistas por terceras partes [50]. Su principal objetivo es el desarrollo de aplicaciones distribuidas en ambientes heterogéneos, donde los sistemas se construyen ensamblando o componiendo funcionalidad existente, denominada servicio. Estos servicios se publican a través de una red y es posible su acceso mediante protocolos específicos [21]. La infraestructura tecnológica de soporte al paradigma SOC está provisto por SOA, donde se define la interacción entre clientes o consumidores y los proveedores de servicios a través del intercambio de mensajes. Una entidad de descubrimiento puede ser vista como un registro o directorio de servicios. Permite que un proveedor publique sus servicios y que luego un consumidor pueda realizar un descubrimiento de tales servicios.

El paradigma SOC provee ventajas muy claras, dado que genera un bajo grado de acoplamiento entre consumidor/proveedor de un determinado servicio y además promueve fuertemente la reusabilidad de componentes software. Sin embargo, se produce un incremento de esfuerzo en dos etapas de un proyecto de desarrollo de software: implementación y mantenimiento [33]. En primer lugar, la búsqueda de servicios publicados en un registro requiere invertir mucho tiempo, en particular considerando el registro UDDI[3] (Universal Description, Discovery, and Integration) para la tecnología de Servicios Web [13]. Esto impacta directamente en los costos de la fase de implementación, ya que el paradigma SOC reemplaza el desarrollo de piezas específicas por el descubrimiento y contratación de las mismas. En segundo lugar, al introducir servicios externos a una aplicación, en general se produce un efecto colateral donde la lógica del negocio queda "contaminada" con aspectos no funcionales, tales como localización, comunicación de datos sobre la red, etc. Esto no es un atributo de calidad deseable, dado que produce sistemas difíciles de entender, mantener y extender [21]. Además, los frameworks actuales para invocar servicios, por ejemplo WSIF[4], producen código fuente subordinado a un determinado proveedor de servicios [20]. En consecuencia, ante los cambios en las interfaces de los servicios externos o su reemplazo por nuevos proveedores, se requiere reconstruir el código para efectuar la invocación de los mismos, lo cual propaga cambios por las partes "contaminadas" de la aplicación [39]. Esta situación, por lo tanto, genera un alto impacto sobre los costos en la etapa de mantenimiento de software.

Si bien existen esfuerzos actuales en la identificación de servicios [26], que poseen mecanismos semi-automáticos para facilitar la tarea de un desarrollador, en general proveen resultados parciales compuestos de conjuntos de servicios candidatos, donde aún deben efectuarse tareas manuales de análisis para realizar la selección definitiva del servicio candidato más adecuado. Tales conjuntos de servicios candidatos pueden variar tanto en la interfaz esperada como en el comportamiento que se requiere dentro de una aplicación de destino. En general, la búsqueda de servicios se resuelve mediante un conjunto específico de palabras clave, considerando que

[3]UDDI: *http://uddi.xml.org/*

[4]WSIF: Web Services Invocation Framework, *http://ws.apache.org/wsif/*

las mismas fueron utilizadas para categorizar y ubicar los servicios dentro de un registro. Sin embargo, esto significa que se está confiando en que el criterio de publicación y categorización del Servicio Web será el mismo (o similar) al utilizado por el desarrollador (o consumidor) que busca dicho servicio. Por otra parte, el conjunto de servicios candidatos puede asumir un tamaño considerable, a partir de lo cual la tarea de selección de un servicio adecuado demanda un esfuerzo desproporcionado, que afecta seriamente los costos de la fase de desarrollo de una aplicación y por lo tanto incrementa los costos totales.

La motivación de esta tesis se basa en que, por un lado, cada proveedor de servicios utiliza diferentes versiones de WSDL para especificar las interfaces de sus servicios, las cuales presentan una gran diferencia en estructuras y elementos de marcado [11, 12]. Por otro lado, al experimentar con diferentes herramientas – tal como EasyWSDL[5], JWSDL[6], WODEN[7], y SOA Membrane[8] entre otras – hemos detectado que debido a las variaciones en versiones de especificación, en algunos casos resulta imposible explorar los documentos WSDL para un proceso de evaluación de Servicios Web. En general estos procesos requieren que se deriven estructuras intermedias en el lenguaje de desarrollo del proceso de evaluación (por ejemplo, en la plataforma Java). Sin embargo, tales estructuras intermedias (tal como, interfaces Java), se derivan en forma parcial, errónea, o directamente se frustra la derivación. Por lo tanto, las diferencias en versiones de WSDLs producen importantes inconvenientes en su tratamiento generalizado y limitan la evaluación y consumo de un gran número de servicios.

A estos desafíos, se le agrega una necesidad creciente en el paradigma SOC, que conduce al desarrollo y utilización de servicios heterogéneos. Además del modelo de servicios con descripciones WSDL, ha surgido en los últimos años otro modelo que posee una descripción distintiva: los servicios RESTful (Representational State Transfer) [23, 26]. Los servicios RESTful proveen una alternativa ligera y económica debido a su escalabilidad y la simplicidad para ser publicado y consumido [45]. Las interfaces de servicios RESTful se describen con métodos sencillos del protocolo HTTP [9], y en algunas propuestas se utiliza WADL [10] [32]. Por lo tanto, es necesario contar con una especificación de contratos de Servicios Web, que sea independiente de cualquier tecnología de implementación, y que cubra convenientemente la existencia de servicios heterogéneos. Esto permitiría ampliar la oferta de servicios accesibles a un proceso de evaluación, evitando las limitaciones tanto en versiones de WSDL como de cualquier otro formato de descripción en servicios heterogéneos, permitiendo una amplia gama de servicios que podrían ser integrados, compuestos y consumidos para satisfacer los requerimientos de las Aplicaciones Orientadas a Servicios.

1.1.1. Solución propuesta

En esta tesis se profundiza la investigación sobre el enfoque que ha sido definido en [8, 14, 19, 28], cuyo objetivo es la mejora del desarrollo de Aplicaciones Orientadas a Servicios; asistiendo a los ingenieros de software por medio de una serie de procesos semi-automáticos

[5] `http://easywsdl.ow2.org/easywsdl-features.html`
[6] `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`
[7] `http://ws.apache.org/woden/`
[8] `https://www.membrane-soa.org/soa-model/`
[9] HTTP: Hyper Text Transfer Protocol
[10] WADL: Web Application Description Language

para la Selección de Servicios Web, que posee soporte de una herramienta desarrollada en la plataforma Java. En este sentido, el trabajo de esta tesis propone una extensión del proceso de selección y de su herramienta de soporte, enfocando en la especificación de contratos de Servicios Web para lograr una independencia respecto de la tecnología y que se pudieran abarcar servicios heterogéneos. Para ello se ha decidido desarrollar un Metamodelo utilizando como base un conjunto de estándares para descripción de contratos/responsabilidades de Servicios Web heterogéneos. La propuesta consiste de tres actividades: *desarrollo e implementación del Metamodelo para Descripción de Contratos de Servicios Web, integración a la herramienta de evaluación de Servicios Web y construcción de un Conversor de descripciones WSDL* [11] *hacia instanciaciones del metamodelo.*

- *Desarrollo e implementación del Metamodelo para Descripción de Contratos de Servicios Web:* establecer una base estructurada, que permita definir la interrelación entre los elementos involucrados en la descripción del conjunto de responsabilidades (operaciones y mensajes de entrada/salida simples/complejos) de los contratos de servicios. Para ello, se tomar como base distintos estándares de descripción de servicios, tales como el Perfil UML SoaML[12], y el lenguaje WSDL versión 2.0 [13] – ambos son estándares de OMG[14]. Al contar con un metamodelo donde se describen contratos ofrecidos por Servicios Web, resulta necesario que la evaluación de servicios candidatos se realice en función de instanciaciones del metamodelo. Para que esto sea posible, es necesario desarrollar un componente de software que contenga el metamodelo, y que permita la exploración de la información de contratos de servicios sobre las instancias del metamodelo mismo.

- *Integración a la herramienta de evaluación de Servicios Web:* en el nuevo enfoque, tanto los requerimientos funcionales por parte de desarrolladores de Aplicaciones Orientadas a Servicios, como los propios servicios candidatos, son representados y comparados mediante instanciaciones del metamodelo desarrollado. Para realizar la integración del nuevo componente de software es necesario adaptar las estrategias de evaluación subyacentes, que incluyen dos niveles: estructural o de tipos de datos en los mensajes de entrada/salida; y semántica o de nombres en los identificadores de operaciones y mensajes [8, 28]. Esta adaptación se realiza de acuerdo a los elementos del metamodelo definido y su correspondencia respecto a la descripción sencilla de contratos de servicios, mediante interfaces Java, utilizadas en el enfoque anterior.

- *Construcción de un Conversor de descripciones WSDL hacia instanciaciones del metamodelo:* como primera aproximación hacia servicios heterogéneos se considera la conversión de Servicios Web descriptos en el lenguaje WSDL. Así el conversor recibe como entrada un documento WSDL (generalmente por medio de la URL) y devuelve como salida una instanciación del metamodelo.

Objetivos

[11] WSDL: Web Service Description Language
[12] SoaML: Service oriented architecture Modeling Language `http://www.omg.org/spec/SoaML/About-SoaML/`
[13] WSDL versión 2.0 `https://www.w3.org/TR/2007/REC-wsdl20-20070626/`
[14] OMG: Object Management Group `http://www.omg.org/`

En función de lo anteriormente expuesto, se puede entonces enunciar el objetivo general de esta tesis, de la siguiente manera:

> *"Evaluación de Servicios Web, mediante un Metamodelo de Contratos de servicios, basado en el estándar SoaML"*

Para la consecución de este objetivo general, se propusieron los siguientes objetivos específicos:

1. Desarrollar un Metamodelo para especificación de Servicios Web basado en el estándar OMG SoaML, con su implementación en la plataforma Java.
2. Modificar la herramienta para evaluación de Servicios Web integrando el Metamodelo de Servicios Web.
3. Desarrollar un Componente Conversor de descripciones WSDL hacia instanciaciones del Metamodelo propuesto.

En la Sección 1.2 se introducen algunos conceptos relacionados a Aplicaciones Orientadas a Servicios y el paradigma SOC, y la tecnología de Servicios Web. En la Sección 1.3 se presentan los estándares que han sido considerados como base para la descripción de servicios en SOA. Finalmente en la Sección 1.4 se describe la organización del resto de los capítulos de esta tesis.

1.2. Aplicaciones Orientadas a Servicios

En el paradigma SOC, un *servicio* se considera un contenedor de capacidades para un propósito común que define un contexto funcional distintivo. Tales capacidades se expresan de acuerdo con un contrato de servicio [22], y se encuentran encapsuladas como funciones autónomas que interactúan a través de una interfaz bien definida. Una definición de servicio debe incluir un identificador (ID), la interfaz (que describe los medios para comunicarse con el entorno del servicio) y su comportamiento operacional (un conjunto de operaciones a ser ejecutadas de acuerdo con alguna estructura interna de control) [37].

Desde una perspectiva de negocios, una Aplicación Orientada a Servicios implica una solución de cara a un negocio, la cual consume servicios de uno o más proveedores y los integra en un proceso de negocios [51]. Desde una perspectiva arquitectónica, puede ser vista como una aplicación basada en componentes, la cual es creada a partir de ensamblar dos tipos de componentes: *internos*, que son embebidos dentro de la aplicación, y *externos*, que se encuentran estática ó dinámicamente vinculados al servicio – ambos exponiendo una clara interfaz de sus capacidades funcionales [39]. Cuando se construye una nueva aplicación, el ingeniero de software debe tomar la decisión de proveer una implementación para algún componente de la aplicación, ó bien, utilizar una implementación ya existente. Esto se denomina *tercerización*, es decir llenar el espacio que deja una funcionalidad faltante, con la implementación de un servicio existente, desarrollado por terceras partes [42].

El paradigma SOC reemplaza el desarrollo de un componente software dado con una combinación de distintas actividades: *descubrimiento* de servicios, *selección* de servicios e *integración* de servicios en aplicaciones [33, 4].

SOA es una manera de describir y entender a las organizaciones, comunidades y sistemas que buscan agilidad, escalabilidad e interoperabilidad. El enfoque SOA es simple: personas, organizaciones y sistemas proveen servicios unos a los otros. Estos servicios permiten acceder a funcionalidades sin la necesidad de conocer la implementación de las mismas, asegurando el encapsulamiento de dichas funcionalidades. SOA es un paradigma arquitectural para definir como las personas, sistemas y organizaciones proveen y usan servicios para alcanzar resultados.

La arquitectura básica de un sistema orientado a servicios (SOA) incluye componentes capaces de: 1) intercambiar mensajes, 2) describir los servicios, 3) publicar y descubrir las descripciones de los servicios [52].

En SOA se define la interacción entre componentes software como un intercambio de mensajes entre *solicitantes* y *proveedores* de servicios, mediante un *agente de servicios*, como se muestra en la Figura 1.1 [21, 29]. Un componente *solicitante* realiza la búsqueda de un servicio en el registro que provee un *agente de servicios* de acuerdo a sus necesidades y solicita la ejecución del mismo. Un componente *proveedor* es responsable de publicar la descripción de un servicio en el registro de un *agente de servicios*, así como aceptar y ejecutar las solicitudes de dichos servicios. Un componente puede asumir tanto el rol de proveedor como de solicitante de servicios. Un *agente de servicios* es un componente en el cual el servicio es publicado, puede ser descubierto y/o ser ser visto como un registro o directorio de servicios.

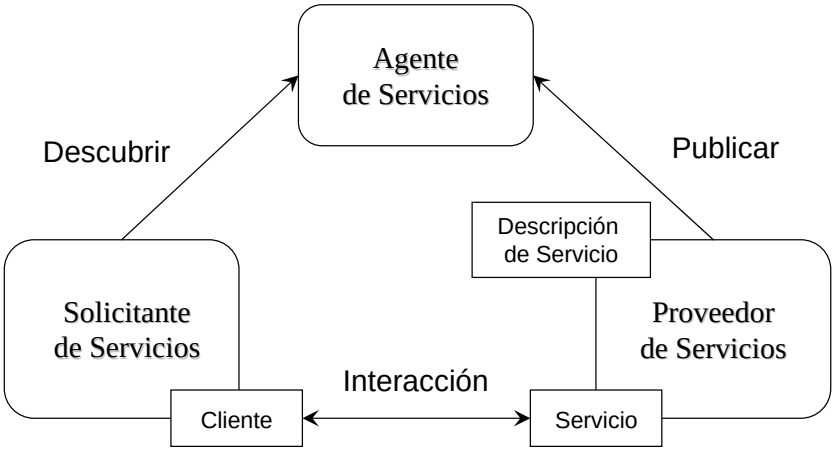

Figura 1.1: Arquitectura básica de un sistema orientado a servicios

1.2.1. Tecnología de Servicios Web

En su mayoría, la industria del software ha adoptado el paradigma SOC utilizando la tecnología de Servicios Web, donde el concepto de servicio se implementa mediante una interfaz especificada en WSDL y un identificador dado por un URI [37, 42]. En el mismo sentido, la W3C[15] define que "*un servicio Web es un sistema software (identificado por un URI), diseñado para soportar la interacción máquina-a-maquina sobre una red interoperable. Tiene una interfaz descripta en un formato procesable por máquina (específicamente WSDL), y otros sistemas interactúan con el Servicio Web de la manera que prescribe su descripción, utilizando (por lo general) mensajes SOAP transmitidos mediante HTTP con una serialización XML, junto con*

[15]W3C: *World Wide Web Consortium*

otros estándares de la Web" [5]. La arquitectura de Servicios Web [57] consta de una serie de protocolos de acceso, que si bien se encuentran en constante evolución, se los puede agrupar actualmente en cuatro capas principales, como se observa en la Figura 1.2.

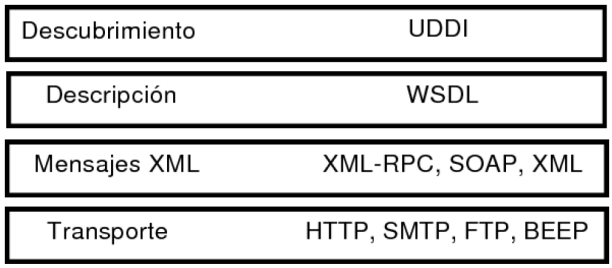

Figura 1.2: Arquitectura de Protocolos de Servicios Web

La capa inferior, denominada capa de *Transporte*, es la responsable de transportar los mensajes entre los componentes software, y actualmente incluye los protocolos HTTP (*Hyper Text Transfer Protocol*), SMTP (*Simple Mail Transfer Protocol*), y FTP (*File Transfer Protocol*), entre otros. La capa de *Mensajes* es la responsable de codificar los mensajes en un único formato XML (*eXtensible Markup Language*), para lograr un entendimiento común. Esta capa incluye protocolos como XML-RPC (*XML-Remote Procedure Call*) y SOAP (*Simple Object Access Protocol*). La capa de *Descripción* de servicios es la responsable de describir la interfaz pública de un Servicio Web específico, que actualmente se logra a través del protocolo WSDL. Por último, la capa de *Descubrimiento* de servicios es la responsable de centralizar servicios y proveer una interfaz para la búsqueda y publicación de servicios. Actualmente, el descubrimiento de servicios se encuentra materializado por UDDI. En la Figura 1.3 se muestra la implementación de la arquitectura SOA, utilizando las tecnologías de Servicios Web, UDDI, WSDL y SOAP.

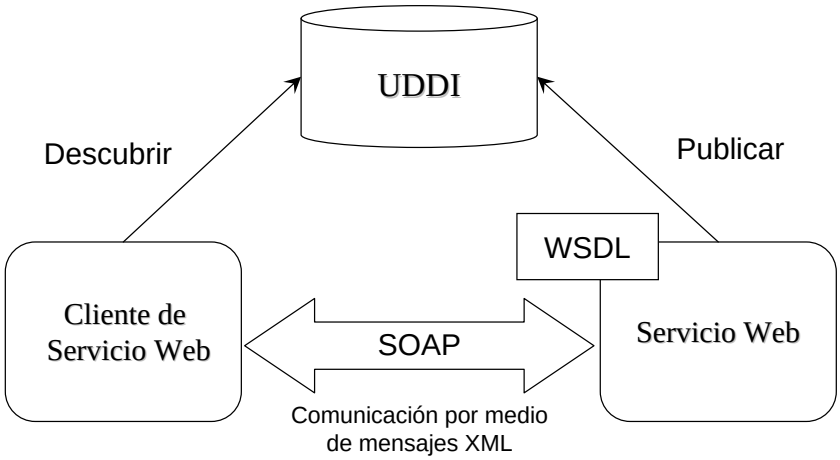

Figura 1.3: Infraestructura estándar de un Sistema orientado a Servicios

Servicios RESTFul

En los últimos años, los servicios RESTful – REpresentational State Transfer – [23] aparecieron como una alternativa ligera y rentable para los servicios basados en SOAP. Los servicios RESTful livianos están diseñados para facilitar el consumo, la composición y la creación de servicios dirigidos por la comunidad (denominados *mashups*). Los servicios REST

ofrecen una alternativa simple, liviana y escalable a los servicios basados en SOAP. REST utiliza los métodos intrínsecos básicos integrados HTTP (PUT, POST, GET y DELETE) que aplican su semántica intencionada para acceder a cualquier recurso referenciable mediante un identificador unificado (URI) [23], por ejemplo, un recurso puede ser cualquier pieza de datos de la Web, como un documento, un tweet o un pronóstico del tiempo. Además, los servicios RESTful exhiben cuatro propiedades:

1. Los recursos representan una abstracción para las aplicaciones de estado del servidor, es decir, un elemento puede ser una referencia de hipertexto.

2. Cada recurso se puede enviar usando su URI.

3. Todos los recursos comparten una interfaz uniforme – métodos HTTP – para interactuar con aplicaciones cliente.

4. La interacción con un recurso es sin estado (*stateless*)

La primer fortaleza de los servicios RESTful es que son más simples que los servicios SOAP porque REST aprovecha los estándares Web conocidos, como son HTTP de la capa de aplicación o transporte y XML de la capa de sesión (Figura 1.2), además cuenta con el URI y la única infraestructura necesaria es la Web. Los clientes y servidores HTTP están disponibles para todos los principales lenguajes de programación y sistemas operativos/plataformas de hardware. Esto lleva a la segunda fortaleza de los servicios RESTful, ligereza, donde los servicios se pueden construir con herramientas mínimas y poco costosas de adquirir [46]. Los servicios son fáciles de consumir y son interesantes para los servicios de la comunidad [36], dado el creciente entorno informático generalizado, donde los dispositivos móviles con diferentes capacidades pueden actuar como clientes e incluso servidores o hosts para Servicios Web. Las nociones de simplicidad y ligereza hacen que sea escalable, ya que un servicio RESTful puede escalar para mantener un gran número de clientes, gracias al soporte integrado para el almacenamiento de caché y el manejo de carga – load balance – de REST. El hecho de no tener que almacenar el estado entre las solicitudes permite que el servidor libere rápidamente los recursos de la empresa y simplifique la implementación porque el mismo no tiene que ceder con el uso de recursos "conversacionales" a través de solicitudes [23]. Además, los servidores sin estado permiten al usuario del servicio (humano o máquina) manipular directamente el estado de la aplicación a través del hipervínculo, lo que se conoce como el principio de HATEOAS (Hypermedia As The Engine of the Application State). La Web, como la arquitectura RESTful por excelencia, potencia la escalabilidad de REST.

En relación con WADL, el objetivo del mismo es definir contratos, que especifican cómo es la comunicación entre los socios de negocios. Si se crease una aplicación hogareña desde cero, no sería necesario definir contratos mediante WADL; pero el enfoque que se espera en el presente trabajo es para desarrollos de software extensos tipo empresariales o gubernamentales, donde sí es necesario contratos que definan lo más estrictamente posible la manera en que los mensajes serán eviados/recibidos. Además de definir contratos, mediante los documentos WADL se permite generar código, testeo y documentación con herramientas de software ya disponibles como lo sería un generador de cliente REST en base a un documento WADL [16].

[16] `https://www.npmjs.com/package/rest-client-generator`

1.3. Descripción de Servicios en SOA

Este trabajo posee como eje principal la necesidad de describir contratos de servicios heterogéneos, para lo cual se propuso desarrollar un Metamodelo, utilizando como base un conjunto de estándares para descripción de contratos/responsabilidades de Aplicaciones orientadas a Servicios. Los estándares que fueron considerados para tal fin pertenecen a OMG: WSDL, WADL y SoaML. En la Sección 1.3.1 se detallan los aspectos principales de cada uno, los cuales fueron considerados como pilares para el desarrollo del metamodelo propuesto. Luego, en la Sección 1.3.2 se detallan las herramientas evaluadas para ser utilizadas potencialmente por el componente Conversor del Metamodelo. En principio, resultó necesario satisfacer la necesidad de parsear un documento WSDL, para lo cual se analizaron distintas herramientas, tales como *JWSDL*, *WODEN*, *EasyWSDL* y *SOA Membrane*. De cada una se explican las características principales.

1.3.1. Estándares para descripción de servicios en SOA

Los estándares que fueron considerados para la descripción de contratos/responsabilidades de Aplicaciones orientadas a Servicios pertenecen a OMG: WSDL, WADL y SoaML, los cuales se explican a continuación.

WSDL

WSDL[17] es un lenguaje basado en XML utilizado para describir la funcionalidad que proporciona un Servicio Web. Un documento WSDL proporciona una descripción de la interfaz de un Servicio Web entendible por la máquina, indicando cómo se debe invocar al servicio, qué parámetros espera, y qué estructuras de datos retorna. Un documento WSDL define servicios como una colección de puertos de la red. En el WSDL la definición abstracta de los puertos y mensajes son separados de su red concreta o ligadura (*binding*) al formato. Esto permite el reuso de definiciones abstractas. Los mensajes son descripciones abstractas de los datos que serán intercambiados en los distintos tipos de puertos que existen. Estos últimos son colecciones abstractas de operaciones. Al usar un lenguaje de programación, se puede crear la parte concreta implementando la parte abstracta. De esta manera, los WSDL tendrán dos componentes principales:

- Información de ligadura sobre el protocolo a utilizar
- La dirección en donde localizar el servicio

La versión actual del estándar WSDL es la 2.0[18], donde se cambió el significado de *Definition,* y se plantearon cambios de nomenclatura y estructura del archivo XML que contiene la descripción del servicio. En la Figura 1.4 se muestra la estructura que siguen los archivos WSDL en las versiones 1.1 y 2.0, en donde se observa el cambio de nomenclatura para la versión 2.0. Además, en WSDL 2.0 puede usarse para servicios REST de ser requeridos/necesarios.

[17] Web Services Description Language. `https://www.w3.org/TR/wsdl`
[18] `https://www.w3.org/TR/wsdl20/`

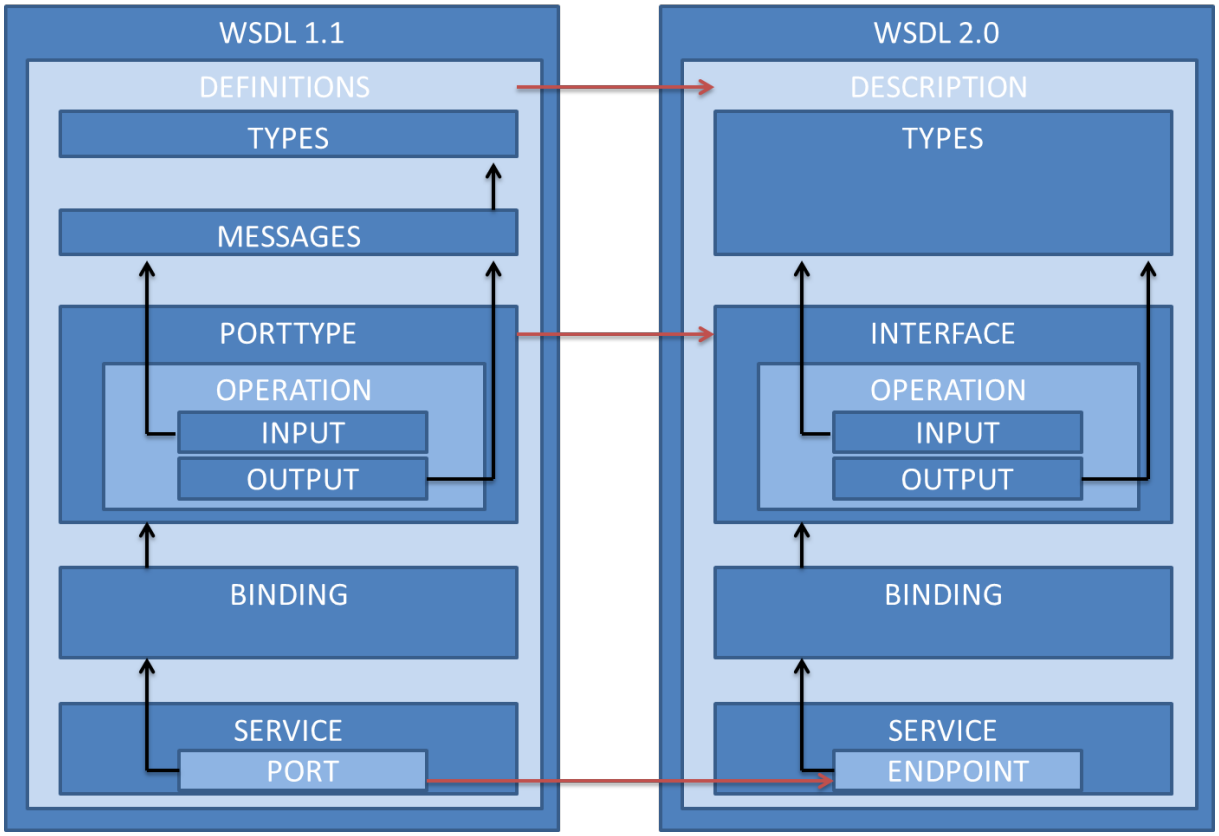

Figura 1.4: Diferencia estructural entre WSDL 1.1 y WSDL 2.0

WSDL Versión 1.1

En concreto, un documento WSDL versión 1.1 usa los siguientes elementos para la definición de los servicios de red:

- Types: un contenedor para la definición de tipos que posteriormente se utiliza en el intercambio de mensajes usando algún sistema de tipos. Podemos definir dichos tipos directamente dentro de este elemento, o importar la definición de un archivo de esquema (XSD). La definición de tipos puede verse, por ejemplo, como las definiciones de clases Java, con variables que pueden ser de tipo primitivo o referencias a otras clases u objetos. Los tipos primitivos se definen en los espacios de nombres del Schema (namespaces) e incluyen tipos simples tales como string, int, double, etc.

- Message: una definición abstracta de tipos de datos que van a a ser comunicados. Un mensaje consiste en partes lógicas, cada una asociada con una definición encuadrada en un sistema de tipos. Es necesario definir los mensajes de entrada y salida para cada operación que ofrezca el servicio.

- Operation: una descripción abstracta de las acciones que soporta el servicio.

- Port Type: colecciones abstractas de operaciones soportadas por más de un punto de acceso (*endpoint*). Cada punto de acceso indica una localización específica para acceder a un Servicio Web usando un protocolo y formato de datos específico. Un punto de acceso

es una entidad o recurso referenciable al que se puede enviar mensajes. Una referencia a un punto de acceso debe proporcionar toda la información necesaria para direccionar un punto de acceso. Cada operación refiere a mensajes de entrada y mensajes de salida, utilizando para ello los mensajes definidos en el apartado anterior.

- Binding: especifica el protocolo de red concreto y el formato de los datos para las operaciones y mensajes definidos en un PortType en particular. Un portType puede tener múltiples bindings asociados. El formato de datos utilizado para los mensajes de las operaciones del portType puede ser orientado al documento u orientado a RPC (Remote Procedure Call)[19]. Si es orientado al documento tanto el mensaje de entrada como el de salida contendrán un documento XML. Si es orientado a RPC el mensaje de entrada contendrá el método invocado y sus parámetros, y el de salida el resultado de invocar dicho método, siguiendo una estructura más restrictiva.

- Port: un punto de acceso definido como una combinación de ligadura y dirección de red. Dicha dirección de red es la dirección (URL) donde el servicio actúa, y por lo tanto, será la dirección a la que las aplicaciones deberán conectarse para acceder al servicio.

- Service: una colección de punto de acceso relacionados.

WSDL 2.0

Un documento WSDL versión 2.0 usa los siguientes elementos para la definición de los servicios de red:

- Interface: describe una secuencia de mensajes que un servicio envía y/o recibe. Esto lo hace agrupando los mensajes relacionados en las operaciones.

- Operation: es una secuencia de mensajes de entrada y salida, y una Interface es un conjunto de operaciones.

- Interface Fault: provee un claro mecanismo para nombrar y describir un conjunto de faltas que una interfaz puede generar. Esto permite a las operaciones identificar de manera sencilla las faltas individuales que puedan generar por el nombre. Este mecanismo permite la identificación de la misma falta a través de múltiples operaciones y referenciados por la ligadura así como también reducir la duplicación de descripciones de una falta individual.

- Binding: describe concretamente el formato de mensajes de transmisión de protocolos que pueden ser usados para definir el punto final. Es decir definir la implementación necesaria para acceder al servicio.

- Type: define el contenido de mensajes y faltas, que se basan en un modelo de dato específico, y se expresan usando un lenguaje de esquema en particular. Aunque una variedad de modelos de datos pueden ser expresados mediante extensiones WSDL 2.0, esta especificación sólo define restricciones basadas en esquema XML.

[19] https://technet.microsoft.com/en-us/library/cc787851(v=ws.10).aspx

- Service: describe un conjunto de puntos de acceso en el cual se desarrolla una implementación del servicio. Estos endpoints son lugares alternativos donde el servicio es provisto. Los servicios son llamados constructores y pueden ser referenciados por su nombre que debe ser único.

WADL

Web Application Description Language (WADL)[20] es una descripción XML entendible por la computadora, utilizada en aplicaciones Web basadas en HTTP -- con un uso más orientado hacia los servicios REST. WADL es independiente de la plataforma y del lenguaje de programación que se quiera usar, y su objetivo es promover la reutilización de aplicaciones más allá del uso básico en un navegador Web. WADL permite modelar los recursos proporcionados por un servicio y las relaciones entre ellos, y está definido para simplificar la reutilización de Servicios Web basados en la arquitectura HTTP existente de la Web. WADL es el equivalente de REST al lenguaje WSDL de SOAP y su aspecto distintivo es que los Servicios Web se describen mediante un conjunto de elementos *resource* (recursos). El Listado de código 1.1 muestra un ejemplo de la estructura de un documento WADL, donde se pueden desatacar los siguientes elementos:

- un elemento *resources* que actúa como contenedor de los recursos que provee el servicio.
- elementos *resource* que representan a cada uno de los recursos del servicio y contienen su descripción
- elementos *param* (parámetros) que describen las entradas
- elementos *method* que describen la *request* y *response* del recurso
- *request*: especifica cómo representar la entrada, qué tipos son requeridos y las cabeceras HTTP específicas que son requeridas
- *response* (respuesta) describe la representación de la respuesta del servicio, así como cualquier información de fallos, para hacer frente a errores.

Listado de código 1.1: Ejemplo de Estructura WADL

```
<resources base="...">
  <resource path="/">
    <method name="GET">
      <request>
        <param name="page" style="query" type="xs:int"/>
      </request>
      <response>
        <representation mediaType="app/json"/>
      </response>
```

[20] https://www.w3.org/Submission/wadl/

```
    </method>
  </resource>
</resources>
```

SoaML

SoaML (Service-oriented architecture Modelling Language)[21] es un Perfil UML (Unified Modelling Language) de reciente desarrollo, que provee una manera estándar para definir la arquitectura y modelado de soluciones SOA. El Perfil SoaML permite crear un modelo de servicios como una derivación de un modelo de proceso de negocio. Un analista de negocio puede crear un modelo de proceso de negocio como forma de comunicar requisitos a un equipo de Tecnologías de la Información (TI) y puede utilizar este modelo para comprobar la forma en que los requisitos de negocio a nivel genérico pueden convertirse en un modelo más detallado orientado a TI. Cuando se manipula un modelo de SoaML y sus artefactos relacionados, se pueden explorar y examinar los elementos siguientes:

- Servicios candidatos (conocidos como posibilidades en SoaML), que ayudan a priorizar los servicios que deben diseñarse e implementarse.
- Interfaces para servicios, que incluyen sus operaciones y las reglas, expectativas o restricciones relacionadas.
- Estructuras de datos, que se pasan como parámetros y se devuelven como resultados.
- Servicios atómicos, que muestran cómo deben ensamblarse los servicios para formar servicios compuestos.

El Perfil UML SoaML se enfoca en los conceptos básicos de SOA y en el modelado de servicios, incluyendo sus capacidades funcionales, aquellas que deberían proveer los consumidores de servicios, los protocolos o reglas para el uso de los servicios, y la información a intercambiar entre consumidores y proveedores de servicios. Además permite especificar cómo tales capacidades funcionales de servicios (requeridas/provistas) son consistentes con los protocolos de interacción de los mismos. Un modelo de Diagrama de Modelado SoaML es representado en la Figura 1.5. En la misma se muestra un modelo de negocios relacionado a la industria de barcos cargueros, en la cual hay organizadores (*dealer*), estados del translado (*Ship Status*), los encargados de la exportación (*shipper*), entre otros agentes.

Cada servicio definido en la arquitectura posee información específica de sí mismo que suele ser representada en el modelo de negocio. A través de un contrato de servicio (*ServiceContract*) se puede definir tanto la coreografía del servicio como su contrato de invocación e interfaces – como se muestra en la Figura 1.6. Se puede apreciar que el recuadro "Opt" dentro de la coreografía representa que el componente *Quote* es opcional, a diferencia de *Order* que no lo es. Las líneas existentes en la coreografía corresponden a los diferentes tipos de mensajes que participan de la invocación del servicio (Figura 1.7), que pueden ser representados a través de un modelo de clases.

[21] http://www.omg.org/spec/SoaML/1.0.1/PDF

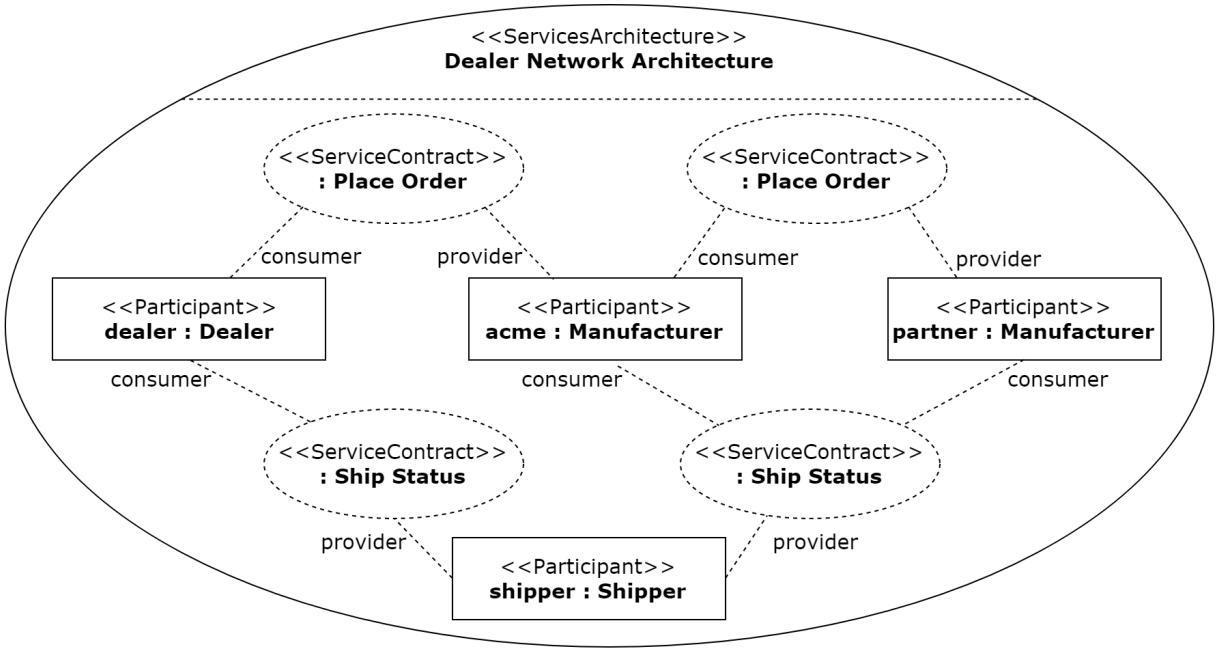

Figura 1.5: Diagrama de Modelo SoaML para industria de transporte marítimo

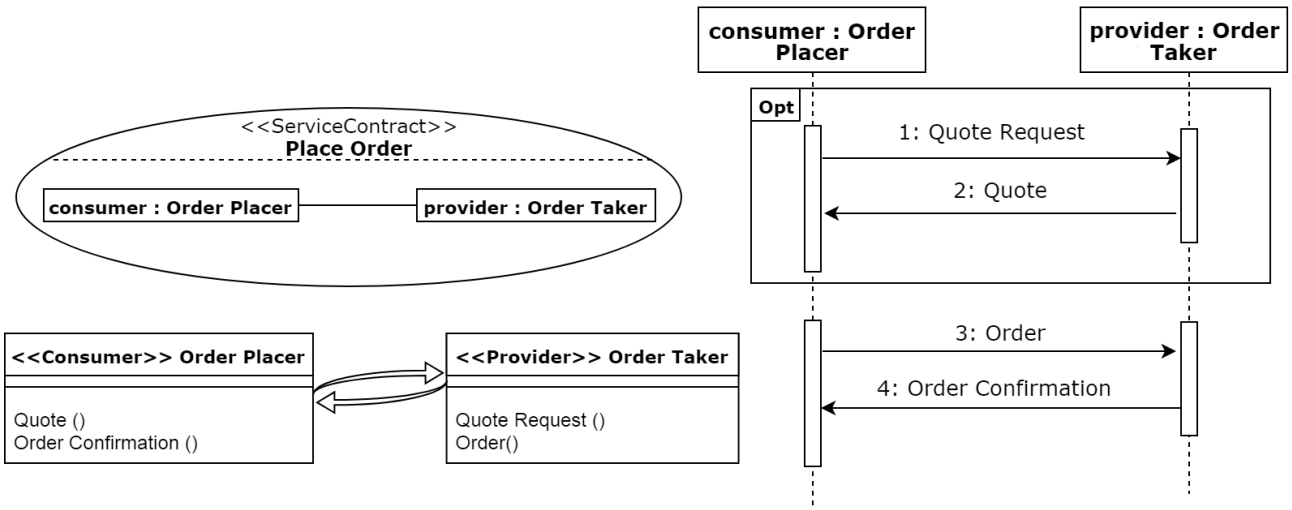

Figura 1.6: Esquema de coreografías y contratos de invocación

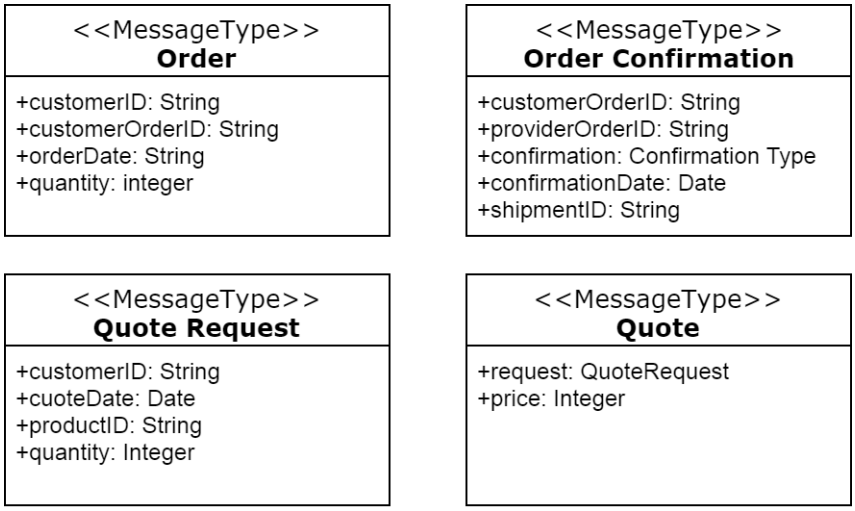

Figura 1.7: Modelo de Diagrama SoaML

1.3.2. Herramientas para Procesamiento de Descripciones de Servicios en SOA

En la presente subsección se detallan las herramientas evaluadas para ser utilizadas por el módulo de software Conversor del Metamodelo para parsear un documento WSDL. En la actualidad existen varias herramientas para manipular/crear documentos WSDL, cuyas características difieren en funcionalidad, alcance, y versión de WSDL soportada, entre otras características.

JWSDL

Soporta WSDL versión 1.1, cumpliendo las bases de W3C del 15 de marzo 2001[22]. Su funcionalidad principal es permitir la lectura, escritura y modificación de documentos WSDL. Esta herramienta está desarrollada en lenguaje Java y especifica a las APIs la manera de agregar elementos para su extensibilidad funcional en cuanto a lectura, escritura y modificación. JWSDL no valida los documentos más allá de la validez sintáctica. Para que esto se cumpla, debe estar correcta la especificación del esquema WSDL. Algunas implementaciones pueden soportar la lectura no ordenada, sin errores ni excepciones[23].

SOA Membrane

SOA Membrane[24] es una API de Java que se utiliza documentos WSDL 2.0 y XML. Tiene varias funciones, entre las más importantes se encuentran las de crear, parsear, manipular y/o comparar dos documentos de la versión previamente descrita. Inicialmente, entre sus funciones se encuentra la comparación y manipulación de los elementos del documento WSDL.

EasyWSDL

[22] http://www.w3.org/TR/2001/NOTE-wsdl-20010315
[23] http://wsdl4j.sourceforge.net/downloads/JSR110_proposed_final_draft.pdf
[24] https://www.membrane-soa.org/soa-model/

EasyWSDL[25] permite a los usuarios leer, escribir, editar y crear documentos WSDL 1.1, WSDL 2.0 y esquemas XML (siguiendo las recomendaciones de la W3C) en el lenguaje Java. Provee los bindings mas usados por las dos versiones del WSDL, al igual que todos los métodos necesarios para retornarlos. Una característica que lo identifica es que permite no usar conversiones de objetos, estrategia que lo diferencia de la herramienta WODEN.

WODEN

WODEN[26] es una herramienta que provee una API Java, que permite la lectura, manipulación creación y escritura de documentos WSDL, soportando versiones 1.1 y 2.0. Adicionalmente, posee un componente validador que permite una validación estructural y semántica de los WSDL. El proyecto se encuentra en desarrollo, y todavía no está en su etapa madura de producción.

DOM

Document Object Model[27] es una plataforma y un lenguaje neutral que permite a los programas y scripts acceder dinámicamente y modificar el contenido, estructura y estilo de un documento. Proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos. DOM define el estándar de acceso y manipulación de documentos. Conforme a DOM, el contenido del documento se representa como una estructura de árbol conformado de nodos, interconectados y relacionados de acuerdo con una jerarquía. DOM, define como nodo padre (tambien llamado inicial o principal) al tag *document* (en WSDL versión 2.0). *Document,* es el nodo que encapsula todas las estructuras definidas dentro de un documento XML. A su vez, cada nodo puede tener uno o varios hijos hasta llegar a los nodos hoja (puntos terminales que no poseen más descendientes).

WORDNET

WordNet[28] es una base de datos léxico-semántica del idioma inglés. Sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos cognitivos (synsets), cada una expresando un concepto distinto. Estos conjuntos están vinculados entre sí por medio de relaciones semánticas y léxicas. El árbol resultante de las palabras y los conceptos relacionados por su semántica queda a disposición para ser utilizado. Esta base de datos se encuentra presente en todo el trabajo realizado, por ser de condición base para realizar comparaciones. La principal relación que se tiene en cuenta es la sinonimia, donde dos palabras son sinónimos si denotan el mismo significado. Además, se pueden establecer relaciones jerárquicas (del tipo todo-partes o es-un) como la hiponimia y la hiperonimia. La estructura de WordNet hace

[25] `http://easywsdl.ow2.org/easywsdl-features.html`
[26] `http://ws.apache.org/woden/`
[27] `https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html`
[28] `http://wordnet.princeton.edu`

que sea una herramienta útil para la lingüística computacional y procesamiento del lenguaje natural.

1.4. Organización de la Tesis

A continuación se describe en forma sintética el contenido del resto de los capítulos que comprenden la estructura de esta Tesis:

Capítulo 2 Se presenta la descripción general del enfoque completo para asistir a un ingeniero de software en la construcción de Aplicaciones Orientadas a Servicios. Este enfoque consiste de un Método de *Selección* de Servicios Web, que posee una herramienta de soporte semi-automático.

Capítulo 3 Se presenta el Metamodelo para Descripción de Contratos de Servicios Web, y cómo el mismo fue integrado a la herramienta de evaluación de servicios. Además, se detalla el dieseño e implementación del componente conversor de descripciones WSDL de Servicios Web para derivar instanciaciones del metamodelo propuesto.

Capítulo 4 Se presenta una evaluación experimental de la herramienta de evaluación de Servicios Web, en función de las extensiones desarrolladas.

Capítulo 5 Finalmente, se presentan las conclusiones de todo el trabajo de tesis, las inferencias que surgen de la evaluación experimental, y la posibilidad efectiva de utilización de la herramienta de evaluación de Servicios Web. Se identifican futuras líneas de acción basadas en los resultados obtenidos.

Capítulo 2

Desarrollo de Aplicaciones Orientadas a Servicios

2.1. Introducción

En el capítulo previo se describió la motivación principal de esta tesis que consiste en la creación e instanciación de un metamodelo para comparación de Servicios Web, considerado una mejora de suma importancia para el Proceso de Selección de Servicios Web. Este último se basa actualmente en especificaciones sencillas derivadas de las descripciones WSDL de servicios; para evaluar el nivel de compatibilidad y también estimar el esfuerzo de adaptación de un servicio candidato para su integración en una aplicación en desarrollo específica. Las especificaciones actuales de servicios son interfaces Java, las cuales no permiten considerar la amplia heterogeniedad tecnológica de Servicios Web, es decir, WSDL, WADL, REST, entre otros.

La evaluación de interfaces es un tema muy estudiado en el paradigma de Desarrollo de Software Basado en Componentes (CSBD[1]), y gran parte del trabajo alcanzado en este área ha sido adaptado para la evaluación de interfaces de servicios. Un ejemplo es el Algoritmo de Stroulia-Wang analizado en [53, 55, 56]. En el presente Capítulo se describe el enfoque para facilitar el desarrollo de Aplicaciones Orientadas a Servicios, que se constituye como un *Proceso de Selección de Servicios Web* [8, 14, 19] y posee una herramienta como soporte semi-automático.

2.2. Método de Selección de Servicios Web

Durante el desarrollo de una Aplicación Orientada a Servicios, un ingeniero de software puede decidir la implementación de algunas partes específicas de un sistema en la forma de componentes *in-house*. Sin embargo, para otras piezas del sistema se podría optar por la adquisición de componentes de terceras partes, que a su vez podrían ser resueltos por medio de Servicios Web. Luego, una Aplicación Orientada a Servicios generalmente posee funcionalidades que no están implementadas de manera local. Esto se ilustra en la Figura 2.1, que presenta un proceso posible de desarrollo en SOA, basado en el enfoque EasySOC [47]que incluye el descubrimiento y la selección de Servicios Web candidatos. Para este proceso se requiere que los diseñadores especifiquen la (potencial) interfaz (en el lenguaje Java) del componente de la aplicación que se desea tercerizar, que denominaremos I_R (Interfaz Requerida), agregando opcionalmente alguna anotación en la forma de comentarios Javadoc[2] – herramienta mediante

[1] Component-based Software Development

[2] Javadoc: http://www.oracle.com/technetwork/articles/java/index-137868.html

la cual se genera documentación a partir de comentarios en el código fuente Java.
Luego en el enfoque EasySOC se extrae información relevante del servicio requerido a partir del código fuente de la aplicación, incluyendo la mencionada interfaz I_R y los comentarios de dicho código, y para ello utiliza ciertas técnicas de minería de textos, para la generación inicial de una consulta (*query*) compuesta de términos relevantes [17] – Paso 1.1 de la Figura 2.1. Esta consulta permite buscar en descripciones de servicios WSDL en un registro de descubrimiento de servicios (repositorio local) mediante el motor de búsqueda de EasySOC [16] – Paso 1.2.

Luego de la búsqueda en el registro, se obtiene un conjunto de documentos candidatos WSDL versión 1.1, cuyas operaciones se asemejan en signatura a las que contiene la I_R, los cuales son usados de base por el Paso 1.3 para convertir dichas operaciones en interfaces Java, que individualmente se denominan I_S (Interfaz Servicio). Este proceso puede ser realizado de manera manual o bien con una herramienta externa (por ejemplo AXIS[3]). De esta manera las operaciones ya transformadas al lenguaje Java, representan a las operaciones que se encuentran en los documentos asociados (en formato WSDL versión 1.1). Es importante destacar que para generar las interfaces Java, es necesario generar todos los archivos correspondientes a los servicios requeridos y candidatos, más los archivos de Tipos Complejos de datos que se encuentran en los WSDL utilizados en los servicios, para luego compilarlos, obteniendo los archivos .class necesarios para hacer el análisis (este proceso se hace para cada WSDL involucrado). El conjunto de interfaces Java (I_S) derivadas de los WSDL de servicios candidatos, son evaluadas en el Método de Selección de Servicios Web – Paso 2, Figura 2.1.

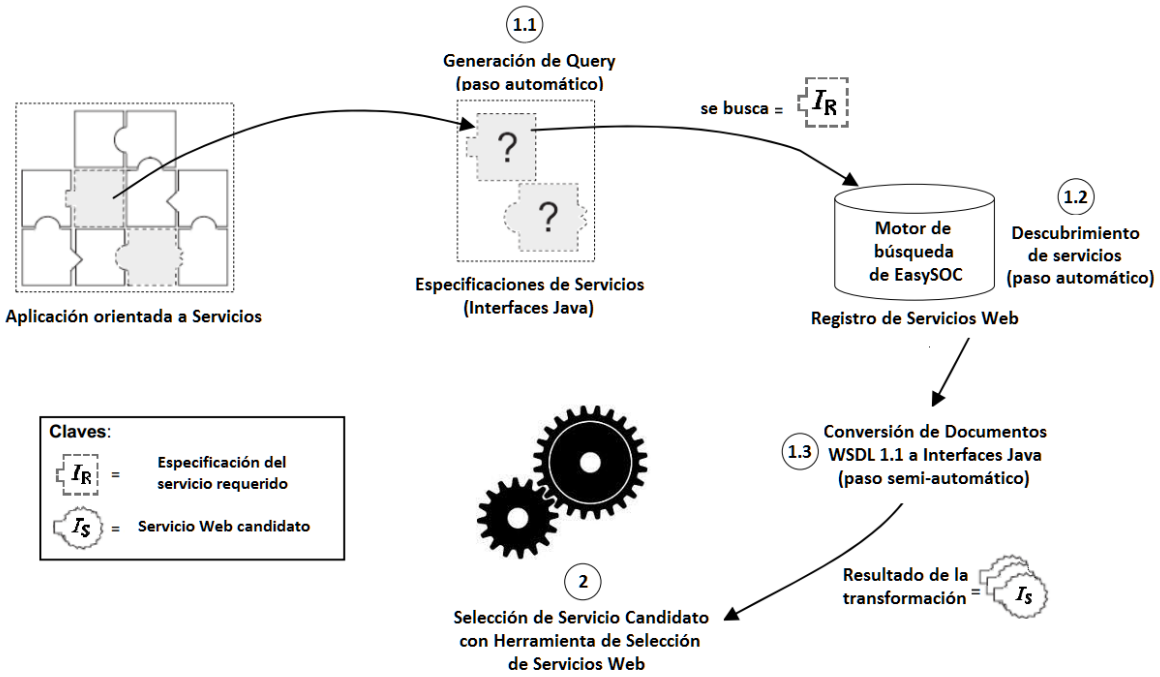

Figura 2.1: Proceso de Descubrimiento y Selección de Servicios Web

Cuando más de un Servicio Web es recuperado del registro de descubrimiento, el ingeniero de software necesita evaluar cuál sería el servicio candidato más apropiado – Paso 2 de la Figura 2.1 – y para eso se vale de la asistencia semi-automática del Método de Selección de Servicios Web,

[3]AXIS: http://axis.apache.org/axis/java/

que consiste de tres procedimientos según se muestra en la Figura 2.2, los cuales se describen a continuación:

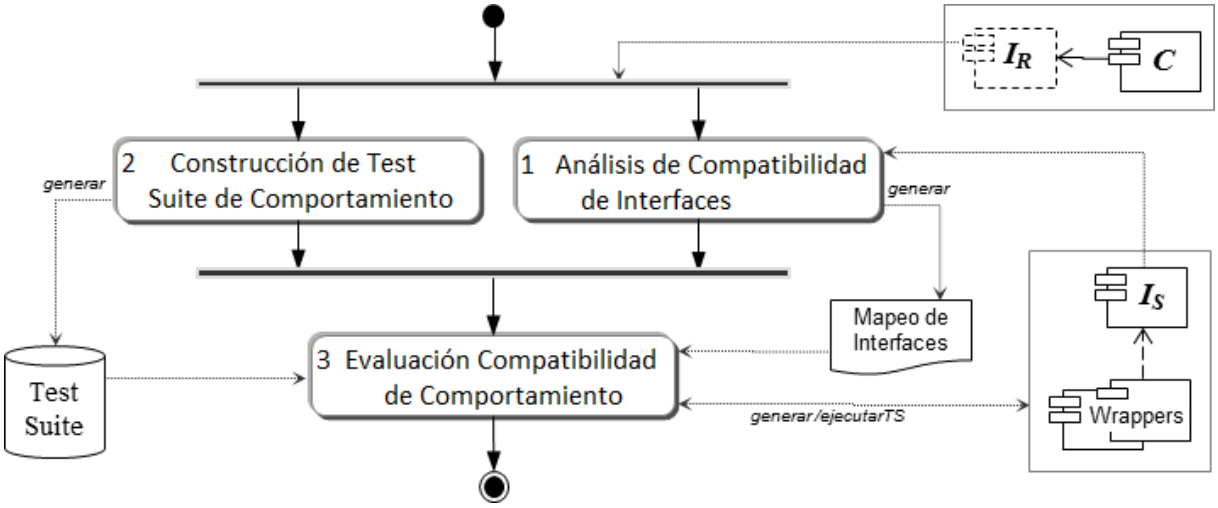

Figura 2.2: Método de Selección de Servicios Web

Nuevamente la interfaz Java I_R (que está en dependencia de un componente C de la aplicación cliente) provee la especificación de requerimiento funcional y servirá de entrada a sus tres procedimientos.

El procedimiento de *Análisis de Compatibilidad de Interfaces* (Paso 1 de la Figura 2.2) se realiza a nivel estructural y semántico, por medio de un esquema exhaustivo, para evaluar la interfaz requerida (I_R) y la interfaz provista por los servicios candidatos (I_S), para luego caracterizar los elementos de signatura de las operaciones (retorno, nombre, parámetros, excepciones) de acuerdo a distintos niveles de compatibilidad. Dicho análisis es explicado en detalle en la Sección 2.3.

El procedimiento de *Construcción de Test Suite de Comportamiento* (Paso 2 de la Figura 2.2) se aplica al identificar que un requerimiento funcional de una aplicación (en la forma de una interfaz requerida I_R) podría ser cubierto por un potencial Servicio Web candidato. Para ello, se construye un conjunto de casos de test o Test Suite (TS), que se ha denominado *TS de Comportamiento*, para ser ejercitado sobre un conjunto de servicios candidatos previamente descubiertos. El objetivo es cumplir la métrica de testing *Facilidad de Observación* [25, 34] que identifica el comportamiento operacional de un componente, al analizar las transformaciones funcionales de datos (input/output) que realiza un componente. Desde ello, se podría exponer una compatibilidad potencial de un servicio candidato, como se ha analizado en [9, 24, 2].

El procedimiento de *Evaluación de Compatibilidad de Comportamiento* (Paso 3 de la Figura 2.2) evalúa el comportamiento requerido de los Servicios Web candidatos mediante la ejecución del *TS de Comportamiento*. Para esta evaluación, se efectúa un procesamiento de los mapeos de interfaces obtenidos del Paso 1, para generar un conjunto de *wrappers* (o adaptadores) que permiten la ejecución del TS contra los servicios candidatos (a través de la interfaz provista I_S) y así poder evaluar el grado de compatibilidad de comportamiento alcanzado. Cada *wrapper* representa una posible correspondencia de las operaciones de la interfaz requerida I_R con respecto a las operaciones de la interfaz I_S de un servicio candidato. Las invocaciones remotas al servicio S se resuelven mediante un proxy (P_S). Luego de la ejecución del TS, al menos uno de los wrappers

deberá pasar exitosamente los tests para confirmar una compatibilidad de comportamiento, y además este *wrapper* exitoso permitirá que un componente C pueda invocar en forma segura al servicio candidato S (a través de P_S), una vez integrado en la aplicación cliente.

Se enfatiza que de los tres procedimientos mencionados, en la sección siguiente se explica en particular el procedimiento de Análisis de Compatibilidad de Interfaces, ya que resulta ser el enfoque central del trabajo de esta tesis. Para más información dirigirse a las tesis que se enuncian a continuación:

- El procedimiento de *Evaluación de Compatibilidad de Comportamiento* puede ser analizado en detalle en la tesis *Test Suite basado en Matching de Interfaces para Evaluación de Comportamiento de Servicios Web* [3]

- Detalles sobre Evaluación Estructural y Semántica de Servicios Web en la tesis *Extensión a la Evaluación Estructural y Semántica de Servicios Web Orientada a la Adaptabilidad* [8]

2.3. Análisis de Compatibilidad de Interfaces

El procedimiento completo para el análisis semántico estructural de compatibilidad de interfaces se resume en la Figura 2.3. Como paso inicial se necesita la especificación sencilla de la funcionalidad requerida en forma de una interfaz requerida (I_R). El procedimiento de compatibilidad de interfaces evalúa la interfaz requerida I_R y la interfaz candidata I_S provista por un servicio candidato S. A través de un análisis estructural y semántico, los elementos de signatura de las operaciones (tipo de retorno, nombre operación, parámetros, excepciones) se caracterizan de acuerdo a distintos niveles de compatibilidad. A continuación se introducen los pasos principales del Análisis de Compatibilidad de Interfaces.

El primer paso que se realiza es la extracción de elementos de signatura: por cada una de las interfaces bajo análisis, se extraen tipos y nombres de la operación, parámetros, retornos y excepciones para ser comparados contra los de la interfaz requerida. Luego de recuperados cada uno de los elementos de signatura, se procede a realizar la extracción de información estructural y semántica.

Respecto a los nombres de operaciones, se generan las listas de términos respectivas para los nombres de operación de $op_R \in I_R$ y de $op_S \in I_S$ que se estén comparando, se eliminan términos sin significado (stop words) y finalmente se identifica la raíz o lema (stem) de los términos significativos – en un proceso denominado stemming. De manera similar, se procesan los identificadores y se extraen los tipos de cada parámetro incorporando además el análisis exhaustivo de parámetros de tipo complejo.

Para el análisis estructural se comparan los tipos de datos de los elementos de signatura de las operaciones $op_R \in I_R$ contra las operaciones $op_S \in I_S$. Para la comparación entre dos tipos de datos entre sí se utilizan las nociones de equivalencia de tipos de datos y subtipificación, definidas en la Sección 2.4.1. Para el análisis semántico de parámetros, se comparan los nombres de los parámetros y, en caso que se trate de parámetros complejos, también se comparan los nombres de los campos que lo componen. Análogamente, se aplica análisis semántico de retorno de tipo complejo. No se realiza análisis semántico para las excepciones, ya que en el contexto de

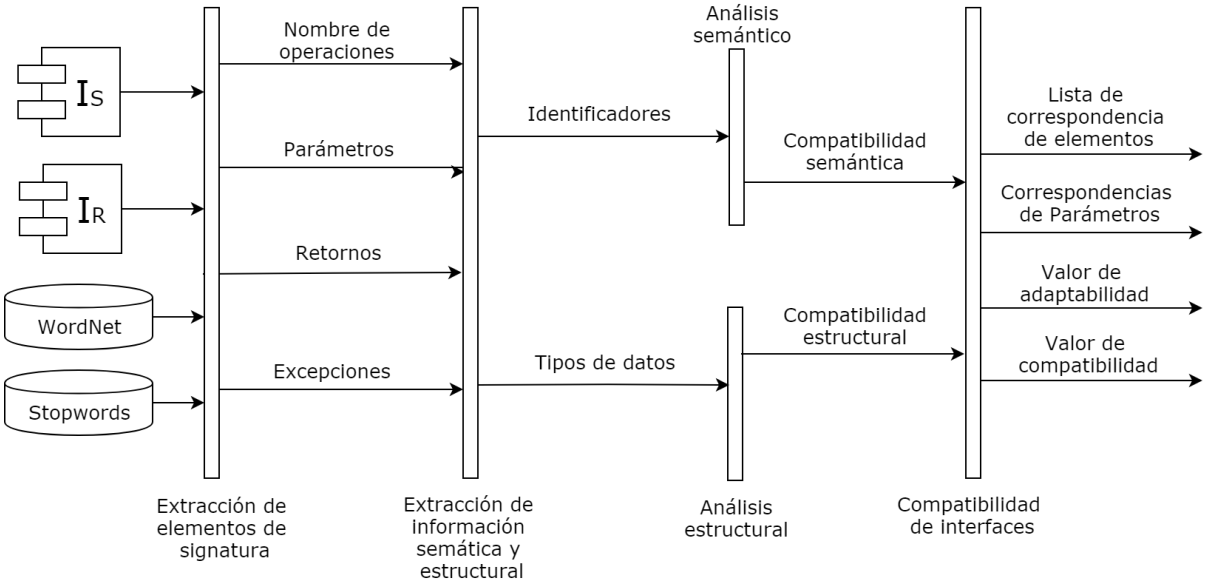

Figura 2.3: Análisis de Compatibilidad de Interfaces

Servicios Web la definición de excepciones (faults de acuerdo a la terminología para WSDL) no es una práctica común [27]. Por ello el análisis semántico de excepciones no afecta de manera crítica la compatiblidad de interfaces.

Luego de las etapas de análisis, se realiza el cálculo de los valores de compatibilidad estructural y semántica entre las operaciones, que serán utilizados para determinar el valor de compatibilidad final entre la interfaz requerida y la candidata. El resultado principal del análisis de compatibilidad de interfaces es una lista de mapeos de interfaces identificando correspondencias entre las operaciones de la interfaz requerida I_R y las operaciones de la interfaz I_S de un Servicio Web candidato S. Además, se obtiene una lista de sugerencias de correspondencias de parámetros que permite distinguir, para cada par de operaciones compatibles ($op_R \in I_R$, $op_S \in I_S$) en la lista de mapeos de interfaces, qué parámetro de la interfaz candidata le corresponde a cada parámetro de la interfaz requerida para que la compatibilidad de las operaciones sea la mayor. Finalmente se obtienen un valor de compatibilidad (relativo a aspectos funcionales) y un valor de adaptabilidad que refleja el esfuerzo que se requiere para adaptar e integrar la interfaz candidata a la aplicación cliente. Los aspectos más relevantes sobre los procedimientos de evaluación estructural y semántica, serán detallados en la Sección 2.4.3.

2.4. Equivalencia Estructural y Semántica

Sea I_R la interfaz de cierta funcionalidad requerida para una aplicación cliente, e I_S la interfaz de un Servicio Web candidato S. Para cada par de operaciones (op_R, op_S), las probables equivalencias entre esas operaciones se basan, en primera instancia, en las condiciones estructurales y semánticas para cada elemento de signatura. Nótese que dichos elementos se nombran de acuerdo a la terminología Java, en vez de usar la convención WSDL para interfaces de Servicios Web. Esto se debe a que las evaluaciones se realizan sobre interfaces Java previamente derivadas de especificaciones WSDL. Los aspectos principales en cuanto a las condiciones estructurales son detallados a continuación.

Tipo en op_R		**Tipo en op_S**
char	$<_1$	string
byte	$<_1$	short, int, long, float, double, string
short	$<_1$	int, long, float, double, string
int	$<_1$	long, float, double, string
long	$<_1$	float, double, string
float	$<_1$	double, string
double	$<_1$	string

Tabla 2.1: Equivalencia de Subtipos

2.4.1. Equivalencia de Tipos de Datos

La equivalencia de los tipos de datos ha sido establecida en función de la relación de subsumción conocida como *subtipificación* (se escribe <:) [59, 30] para tipos primitivos de datos, de acuerdo a la relación *directa* de subtipos (se escribe $<_1$) del lenguaje Java [30], que se muestra en la Tabla 2.1. Para más información dirigirse a la Tesis [8].

Así para las signaturas de las operaciones se consideran relaciones de subtipos como se muestra en la Tabla 2.1, donde los tipos en op_S deben al menos tener tanta precisión como los tipos en op_R. Por ejemplo si la operación op_R incluye un tipo `int`, una operación correspondiente op_S no puede tener una precisión menor como `short` o `byte` (entre los tipos numéricos). Sin embargo, se da un caso especial con el tipo `String`, que se considera un tipo *comodín* dado que es usado comúnmente en la práctica para alojar virtualmente cualquier tipo de dato [43].

2.4.2. Evaluación de Identificadores

El algoritmo para la Evaluación de Identificadores compara el nombre de una operación op_R con el nombre de una operación op_S. Los pasos principales para la comparación son:

- *Separación de Términos*: Los identificadores se restringen normalmente a secuencias de una o más letras en código ASCII, caracteres numéricos y guiones bajos ('_') o medios ('-'). El paso de *Separación de Términos* analiza los identificadores reconociendo términos potenciales (secuencias y/o cambios de mayúsculas a minúsculas y visceversa, caracteres especiales). Luego WordNet es utilizado para analizar todos los potenciales términos y determinar la separación de palabras más adecuada. Cuando un identificador no sigue estrictamente las convenciones de nombrado, algunas suposiciones son realizadas, por ejemplo una secuencia de mayúsculas podría representar un acrónimo, al que se le buscará una definición. Sea el caso de la palabra *SQLLogin*, que no cumple con las convenciones de nombrado de Java. El análisis preliminar de términos representativos diría que hay dos secuencias factibles, *SQLL* y *ogin*. De acuerdo con el algoritmo de *Separación de Términos* las secuencias previamente nombradas serían analizadas de esta manera con WordNet. Como no son palabras válidas, el algoritmo analiza la última mayúscula junto con la secuencia de minúsculas $L + ogin = Login$. Como esta palabra si existe en el diccionario WordNet, *SQL* es considerada un acrónimo para su análisis (que actualmente significa Structured Query Language).

- *Eliminación de Stop Words:* es el nombre que reciben las palabras sin significado tales como artículos, pronombres, preposiciones, etc. Estas palabras son filtradas antes o después del procesamiento de datos (texto) en lenguaje natural [7], por no ayudar a determinar la semántica de la operación, siendo irrelevantes para determinar si un conjunto de términos es similar a otro. El paso de *Eliminación de Stop Words* toma una lista de términos y elimina cada ocurrencia de una palabra perteneciente a la lista de stop words. En [19], se determinó que la lista de stop words debía ser extendida agregando el alfabeto completo (mayúsculas y minúsculas) y también los dígitos. La razón es que el vector resultante del algoritmo de separación de términos puede incluir letras y/o dígitos como términos individuales. Ejemplos comunes del uso de dígitos en los identificadores son: 4 (four) y 2 (two), para referirse a las palabras for y to respectivamente, por tener una pronunciación parecida en el idioma inglés.

- *Stemming:* es el proceso de reducir un término a su forma base, raíz o *stem.* Inicialmente, se consideró la utilización de un stemmer estándar (frecuentemente usado en Recuperación de Información) como el Algoritmo de Porter [58], pero dichos stemmers pueden generar datos semánticamente incorrectos, siendo los principales errores el *overstemming* y *understemming. Overstemming* es un error en el que dos palabras derivan en la misma raíz (*stem*), pero no deberían ya que son semánticamente diferentes (falso positivo). *Understemming* es un error en el que dos palabras deben derivar en la misma raíz, pero derivan en raíces distintas (falso negativo). El algoritmo de *Stemming* utilizado recibe como entrada una lista de términos, y para cada término en la lista, se verifica si pertenece al diccionario de WordNet. Si eso ocurre, se agrega su stem a la lista resultante; en caso contrario, se agrega el término original que puede ser un acrónimo o abreviatura.

- *Comparación semántica de listas de terminos*: luego de generar las listas de términos de op_R y op_S (con sus respectivos acrónimos y *stems*) se extrae la información para la comparación semántica, que incluye: (1) Términos idénticos (exactos) entre op_R y op_S; (2) Sinónimos de los términos de op_R que pertenezcan a la lista de términos de op_S; (3) Hiperónimos (padres) de los términos de op_R que pertenezcan a op_S; y (4) Hipónimos (hijos) de los términos de op_R que pertenezcan a la lista de op_S. Para evaluar sinónimos, hiperónimos e hipónimos, se consideraron los siguientes aspectos. En primer lugar, se utilizó un nivel único de hipo/hiperonimia para evitar alterar demasiado el significado de las palabras. Por ejemplo, *house* es un hipónimo de primer nivel de construcción (por lo tanto, esa relación es considerada por el algoritmo). Segundo, se considera la sinonimia total, donde dos términos son sinónimos si son intercambiables en el mismo contexto sin afectar la semántica. Por ejemplo, *land (tierra)* y *ground (suelo)* son sinónimos totales ya que son semánticamente intercambiables.

- *Cálculo de compatibilidad de nombres:* en este punto, con la información obtenida se puede calcular la compatibilidad de nombres de acuerdo a la Fórmula 2.1, donde *terms* es el total de términos entre op_R y op_S (terms = #({termsOpR} $\bigcup$ {termsOpS})), *exact* es el total de términos idénticos, y *sin*, *hipo* e *hiper* son los totales de sinónimos, hipónimos e hiperónimos respectivamente. Luego, este valor es discretizado para corresponderse con las condiciones para nombres de operación. A los hiperónimos e hipónimos se le da la mitad

del peso de los términos exactos y sinónimos, ya que los primeros implican una semántica ligeramente diferente entre los términos del segundo.

$$compNombres = \frac{exact + sin + 0.5 * (hipo + hiper)}{terms - sin} \tag{2.1}$$

2.4.3. Evaluación de Parámetros

Para la comparación de parámetros, la cantidad de parámetros no constituye un factor de incompatibilidad cuando $op_R \in I_R$ tiene menos parámetros que una operación $op_S \in I_S$. Ambas operaciones aún son potencialmente compatibles si los campos de algún parámetro complejo resultan compatibles con los parámetros simples en su contraparte. El algoritmo de análisis de compatibilidad de parámetros entre las dos operaciones se basa en calcular tres matrices y realizar dos cálculos que maximizan los resultados obtenidos:

- La matriz de compatibilidad de tipos (T): se utilizan las nociones de equivalencias de tipos de datos y subtipificación detalladas en la Tabla 2.1, para evaluar tipos de parámetros. El objetivo de la matriz T es almacenar la relación entre todos los pares de tipos de parámetros en op_R y op_S.

- La matriz de compatibilidad de nombres (N): la matriz N contiene los valores de compatibilidad entre el nombre de cada parámetro de op_R y cada parámetro de op_S. La lógica subyacente a este paso es similar a la matriz de tipos. La celda N_{ij} de la matriz de nombres N contiene el valor de compatibilidad entre el nombre del i-ésimo parámetro de op_R y el j-ésimo parámetro de op_S.

- La matriz de compatibilidad (C): la matriz C se genera a partir de las matrices T y N. El objetivo de esta matriz es almacenar el valor de compatibilidad entre todos los pares de parámetros de op_R y op_S, considerando aspectos estructurales (resumidos en la matriz T) y semánticos (resumidos en la matriz N). La celda C_{ij} de la matriz C contiene el producto entre T_{ij} y N_{ij} – luego: $C_{ij} = T_{ij} * N_{ij}$.

- *Maximización de mapeo de parámetros:* Luego de calcular la matriz C, se debe seleccionar el mejor mapeo de parámetros entre todas las posibles combinaciones de pares de parámetros – es decir, la asignación de parámetros que maximiza la compatibilidad entre op_R y op_S. Cada posible asignación de parámetros consiste en tomar cada fila de la matriz C y elegir una columna, con la condición de no utilizar cada columna más de una vez (no asignar más de una vez cada parámetro). Luego, esto asegura un mapeo uno-a-uno entre los parámetros de op_R y op_S. El mapeo con el mayor valor total (sumando cada celda elegida) será el más compatible. Dicho mapeo se obtiene aplicando el método Hungarian [35], que modela el problema de la asignación como una matriz de costos de $n * m$ – en este caso n y m son los números de parámetros de op_R y op_S respectivamente.

- C*álculo de Compatibilidad de Parámetros:* se calcula un valor mediante la ya calculada matriz de compatibilidad C, analizando la misma en busca de la correspondencia que maximice el valor de compatibilidad de acuerdo a la Fórmula 2.2.

$$compParam = \frac{\sum C_{ij}}{\#(parCamposOps)} - penalizacion * \#(paramNoUsadosOps) \quad (2.2)$$

Donde C es la matriz de compatibilidad que maximiza los valores de compatibilidad, y *penalizacion* es un valor que depende de si los parámetros de op_R y op_S son de tipos simples o complejos, y de un valor configurable que penaliza los parámetros no usados. Esto es, si la op_R consta de un parámetro complejo y la op_S es una lista de parámetros simples o viceversa entonces la penalización tiene un coste mayor que si ambas interfaces tuvieran una lista de parámetros simples o un único parámetro complejo. Además, podrían penalizarse los parámetros extra a través del valor. Por último, este valor es discretizado para corresponderse con las condiciones para parámetros.

Parámetros complejos

El algoritmo de análisis de compatibilidad de parámetros para tipos de datos complejos entre op_R y op_S consiste en determinar tres valores:

1. Valor de compatibilidad de los campos (basado en nombres y tipos) que componen los tipos de datos complejos de los parámetros de op_R y op_S.

2. Valor de compatibilidad de los nombres de los tipos de datos complejos de los parámetros de op_R y op_S.

3. Valor final de compatibilidad entre los tipos de datos complejos de los parámetros de op_R y op_S en base a los valores de compatibilidad obtenidos en los puntos (1) y (2).

Compatibilidad de campos

Para poder realizar una comparación entre parámetros complejos, es necesario conocer la estructura interna de cada uno de ellos y realizar una comparación entre los nombres y tipos de cada uno de los campos que lo componen. Si la cantidad de campos del parámetro complejo de la op_R es menor a la cantidad de campos del parámetro complejo de la op_S entonces se puede proceder con la comparación entre ellos.

En caso contrario no se puede avanzar con la comparación, puesto que quedarían campos del tipo complejo en op_R que no se podrían instanciar con ningún campo en op_S. El algoritmo para el análisis de compatibilidad de los campos es análogo al que se presenta en la Sección 2.4.4 para el análisis de nombres de los tipos de retorno.

Compatibilidad de nombres de parámetros

Se compara directamente parámetro contra parámetro o campo contra campo en caso de que se traten tipos complejos. En ambas situaciones también se tiene en cuenta que la cantidad de parámetros (o campos del tipo complejo) de la op_R sea menor o igual a la cantidad de parámetros (o campos del tipo complejo) de la op_S.

Una vez determinado el caso que corresponda, la comparación entre los nombres propiamente dichos se realiza de acuerdo al algoritmo de comparación de identificadores presentado en la Sección 2.4.2.

Comparación de parámetros simples contra complejos

Cuando se comparan dos operaciones donde una presenta una lista de parámetros de tipo primitivo y la otra un único parámetro complejo, el algoritmo de análisis de compatibilidad aplana [6] el parámetro de tipo complejo y compara los campos con los parámetros primitivos. El aplanado (*flatten*) es un operador de mutación de interfaces que permite desencapsular los tipos de datos complejos o registros en una lista de sus tipos componentes. Por lo tanto, el análisis de compatibilidad se reduce al mismo caso que cuando se comparan operaciones con listas de parámetros primitivos, sin tener en cuenta el nombre del tipo complejo.

Valor final de compatibilidad de parámetros

El valor de compatibilidad de parámetros de op_R y op_S se realiza teniendo en cuenta el valor obtenido con la compatibilidad de nombres y de tipos de todos los parámetros de las operaciones de acuerdo a la siguiente Fórmula:

$$compParamFinal = compNombres * (1 + compTipos) \tag{2.3}$$

Generación de la Sugerencia de Correspondencias de Parámetros

Una vez obtenidos los mapeos de parámetros, de acuerdo al mejor mapeo encontrado para cada operación, se genera una sugerencia de correspondencias de parámetros. Esto es, se almacena aquella correspondencia que maximiza la compatibilidad entre un par de operaciones (op_R , op_S). Luego, este procedimiento debe ser extendido para contemplar las mejoras realizadas al análisis de compatibilidad de interfaces. Partiendo de una matriz de compatibilidad C ya calculada, cada mapeo posible consiste en tomar cada fila (parámetro/campo de op_R) y elegir una columna (un parámetro/campo de op_S) para ese elemento. Luego, para cada posible asignación de parámetros entre op_R y op_S, se calcula el valor de compatibilidad que posee esta asignación. Este es el valor que corresponde al mapeo dentro de la matriz de compatibilidad de parámetros. El resultado final de cada posible combinación de parámetros es la suma de todos los pesos de los mapeos que componen la misma. Luego, la asignación que posea el mayor valor, será la más compatible.

2.4.4. Evaluación del Retorno

Si la operación requerida $op_R \in I_R$ y la operación del servicio $op_S \in I_S$ tienen tipos de retorno complejos, se realiza un análisis exhaustivo teniendo en cuenta la parte semántica de cada retorno. Implicando un análisis estructural y semántico del identificador y los campos que componen los tipos complejos. El algoritmo de evaluación de retornos complejos compara los

tipos de retorno de una operación de la interfaz requerida $op_R \in I_R$ y una operación de la interfaz del servicio candidato $op_S \in I_S$ y realiza distintas evaluaciones:

1. Compatibilidad de campos comparando los tipos y nombres de cada uno (todos contra todos) y maximizando el valor de compatibilidad.
2. Compatibilidad de los nombres del tipo complejo de retorno en op_R y op_S.
3. Valor de compatibilidad final entre los tipos de retorno complejo de op_R y op_S en base a las determinaciones obtenidas en los puntos anteriores.

El análisis del tipo de retorno primero verifica si el tipo de retorno de op_R y op_S es idéntico. En tal caso, la compatibilidad del tipo de retorno es exacta. Si los tipos de retornos son diferentes, se analiza si el tipo de retorno de op_R es subtipo del tipo de retorno de op_S – de acuerdo a los criterios de subtipificación para el lenguaje Java (Tabla 2.1). Si se cumple alguno de esos criterios entonces la compatibilidad del tipo de retorno es equivalente. Si ninguna de las condiciones anteriores se cumple, entonces al menos uno de los retornos es complejo y se requiere un análisis exhaustivo.

Comparación de tipos de retorno complejos

Para poder realizar una comparación entre tipos de retorno complejo, en un principio es necesario conocer la estructura interna de cada uno de ellos y realizar una comparación del tipo "todos contra todos" entre los nombres y tipos de cada uno de los campos que lo componen. La *compatibilidad de nombres para tipos de retornos complejos* se evalua utilizando el algoritmo de evaluación de identificadores presentado en la Sección 2.4.2. Si la cantidad de campos en op_R es menor o igual a la cantidad de campos en op_S entonces se puede proceder con la comparación entre ellos. En caso contrario se consideran incompatibles, ya que quedarían campos requeridos en op_R que no se podrían corresponder con ninguno de la operación candidata op_S.

Comparación de retorno simple contra retorno complejo

Para realizar el análisis de un tipo de retorno simple respecto a uno complejo, es necesario aplanar este último, trabajando directamente con los campos que lo componen. Luego se compara directamente el tipo de dato del retorno simple contra todos los tipos de datos de los campos del complejo, seleccionando aquel que maximice el valor de compatibilidad entre los retornos. Nótese que el análisis en este caso es puramente estructural, ya que el tipo de retorno simple no contiene información semántica que pueda ser analizada.

Valor final de compatibilidad de retorno

El valor final de compatibilidad entre los tipos de retorno complejo para op_R y op_S se calcula de acuerdo a la Fórmula 3.2 teniendo en cuenta el valor obtenido comparando campos y el valor de compatibilidad de nombres de los tipos propiamente dichos.

$$compRetComplejo = \frac{compCampos * (1 + compNombres)}{2} \tag{2.4}$$

Donde *compCampos* representa el valor de compatibilidad estructural y semántica entre los campos del tipo complejo, y *compNombres* representa el valor de compatibilidad semántica de los nombres de los tipos de retorno. A este último valor se le suma 1 ya que se pretende que no se genere una incompatibilidad sólo por la ausencia de correspondencia semántica entre nombres. Luego, se divide el valor final por 2 para normalizar el valor de retorno complejo.

Distancia de Adaptabilidad

Considerando las condiciones semántico-estructurales para compatibilidad de interfaces, es posible definir un valor numérico que refleje el esfuerzo de adaptación requerido para integrar un servicio candidato en una Aplicación Orientada a Servicios. Dicho valor se denomina *Distancia de Adaptabilidad* y se calcula de acuerdo a la Fórmula 2.5, donde N es el tamaño de la interfaz I_R y $AdapMap$ es el mejor valor entre los valores de equivalencia $adaptValue(op_R, op_S)$.

$$DistAdap(I_R, I_S) = \frac{\sum_{i=1}^{N}(Min(AdapMap(op_{Ri}, I_S)))}{N} + 1 \tag{2.5}$$

El valor de adaptabilidad ($adapValue$) entre una operación op_R y una operación op_S potencialmente compatibles se calcula de acuerdo a la Fórmula 2.6, donde R, E son los valores de equivalencia estructural para Retorno y Excepciones, y $compNombre$ y $compParam$ son los valores de equivalencia semántico-estructural para Nombres y Parámetros.

$$adapValue(op_R, op_S) = R + E - (compNombre + compParam) \tag{2.6}$$

Cuanto mayor sea el esfuerzo de adaptación necesario para integrar un servicio candidato, mayor será el valor de la *distancia de adaptabilidad.* En el caso en que todas las operaciones requeridas tengan una equivalencia *exacta,* la distancia de adaptabilidad entre I_R e I_S es cero. Aunque esto puede parecer una correspondencia perfecta entre las interfaces, inicialmente significa que I_R está incluida en I_S, y que la correspondencia entre ambas requiere de un esfuerzo de adaptación nulo o trivial – por ejemplo, conversiones (*casting*) entre tipos compatibles sin pérdida de precisión.

La fórmula de *distancia de adaptabilidad* incorpora la información estructural y semántica extraída de las interfaces en un valor numérico preciso, que refleja adecuadamente el esfuerzo de adaptación e integración del servicio candidato. Esto hace que el Método de *Selección* de Servicios pueda ser considerado como *orientado a la adaptabilidad.*

Capítulo 3

Metamodelo para Descripción y Evaluación de Contratos de Servicios Web

3.1. Introducción

Como se mencionó en la Sección 1.1.1 (Capítulo 1), el objetivo principal de este trabajo es mejorar el Método de Selección de Servicios Web presentado en la Sección 2.2 (Capítulo 2). En este contexto, uno de los desafíos más importantes a la hora de evaluar Servicios Web, es contar con una especificación de contratos de servicios que sea lo suficientemente descriptiva en cuanto a las funcionalidades que ofrece cada Servicio Web. Para lidiar con este desafío, se propuso definir y desarrollar una especificación de contratos de Servicios Web, que sea independiente de cualquier tecnología de implementación. Algunas soluciones para problemas similares [26] se basan en estándares existentes como WSDL [18], o bien proponen especificaciones ad-hoc [41, 10].

Con el fin de satisfacer el objetivo mencionado, la Herramienta de Evaluación de Servicios Web que da soporte al Método de Selección de Servicios presentado en la Sección 2.2 (Capítulo 2) fue ajustada y extendida, incluyendo dos nuevos componentes: un Metamodelo de descripciones de Servicios Web heterogéneos y un nuevo componente Conversor de documentos WSDL hacia instanciaciones de dicho metamodelo.

El resto del capítulo se distribuye de la siguiente manera: en la Sección 3.2 se presenta el Metamodelo para Descripción de Servicios Web desde dos perspectivas: general y específica. En la Sección 3.3 se detalla el Componente Conversor de WSDLs para instanciar el Metamodelo de Servicios Web. Finalmente, en la Sección 3.4 se presenta cómo se ha modificado el proceso de descubrimiento y selección de servicios, mediante la integración del metamodelo y el componente conversor a la Herramienta de Evaluación de Servicios Web.

3.2. Metamodelo para descripción de Servicios Web

Como se mencionó en la Sección 2.2 (Capítulo 2), anteriormente el Método de Selección de Servicios Web tomaba como entrada interfaces Java generadas a partir de documentos WSDL. Cuando estas interfaces se generan, por ejemplo, mediante la utilización de herramientas como EasyWSDL[1], se pueden observar diversos inconvenientes y limitaciones. Como se explicó en el Capítulo 1, existen diversas versiones de WSDL que los proveedores de servicios utilizan y esto dificulta generalizar los procedimientos para parsear y explorar documentos WSDL. Al

[1] http://easywsdl.com/

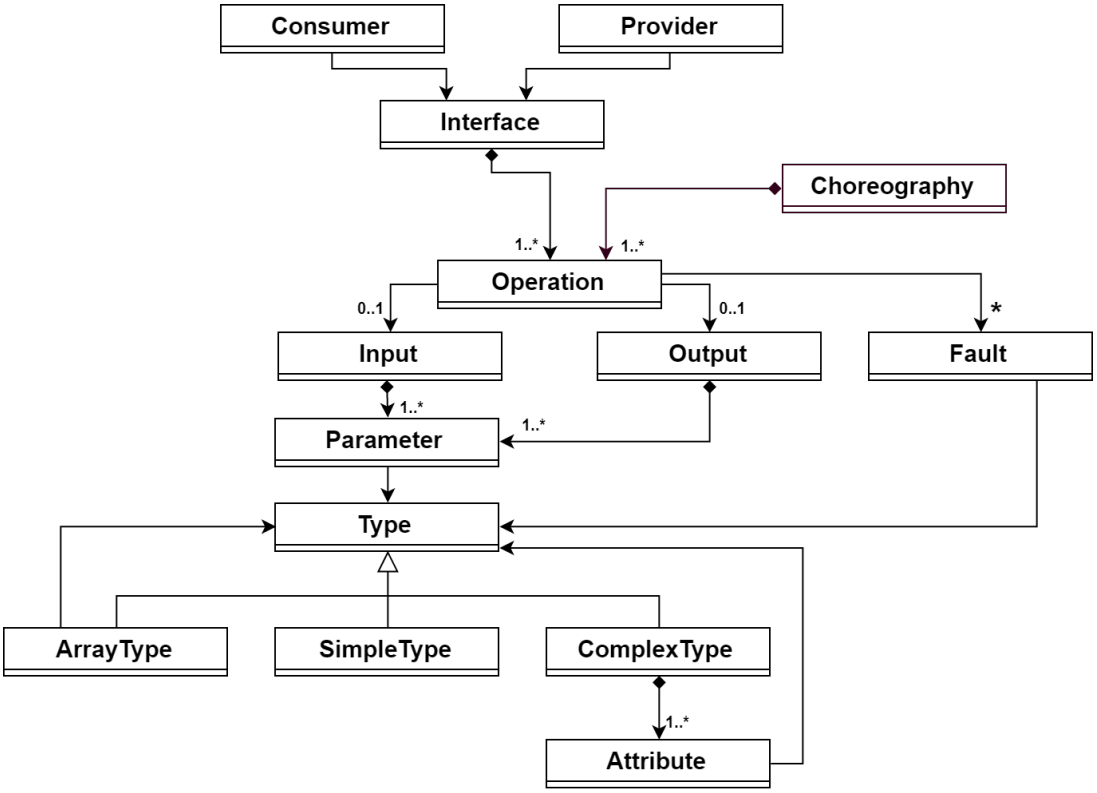

Figura 3.1: Diagrama de clases del Metamodelo propuesto

experimentar con diferentes herramientas hemos detectado que debido a tales diferencias, en algunos casos resulta imposible derivar interfaces Java, y en otros casos se derivan interfaces erróneas. Además, al considerar el surgimiento de otro tipo de Servicios Web como los servicios RESTful, que presentan interfaces distintivas mediante HTTP o WADL, se incrementa el desafío en el tratamiento generalizado de interfaces de servicios. Por lo tanto, es necesario contar con una especificación de contratos de Servicios Web, que sea independiente de cualquier tecnología de implementación. Por este motivo, utilizando como base los estándares mencionados en la Sección 1.3.1, SoaML, WSDL y WADL, se ha definido e implementado un Metamodelo para Descripción de Contratos de Servicios Web. La Figura 3.1 muestra el diagrama de clases del metamodelo definido, que se explicará en la Sección 3.2.1.

Como se mencionó previamente, el metamodelo surge de la conjunción de estándares y criterios, a partir de los cuales se han definido las clases involucradas. En la Figura 3.2 se aprecia la distribución de las clases en base al origen de cada una. Del estándar SoaML provienen las clases *Consumer*, *Provider*, *Interface* y *Choreography*. Del estándar WSDL 2.0 se definieron las clases *Operation*, *Input*, *Output*, *Fault*, *SimpleType*, *ComplexType* e *Interface*. Se destaca que *Interface* se encuentra en ambos estándares, convirtiéndose en el nexo entre las mismas. Para denotar la semántica esperada que permitirá realizar evaluaciones exhaustivas de servicios, se han creado clases adicionales, como son *Parameter*, *Type*, *ArrayType* y *Attribute*.

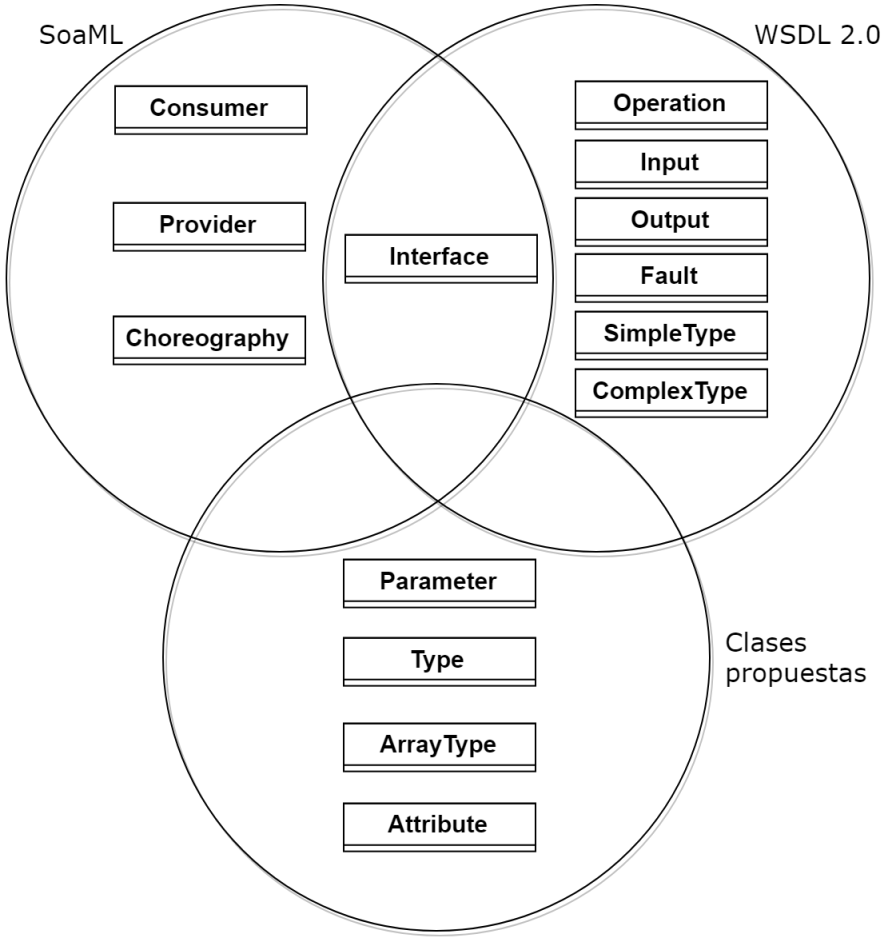

Figura 3.2: Diagrama de Venn para categorización de clases de acuerdo a los estándares utilizados

3.2.1. Visión Detallada

En esta sección se describe cada *clase* que compone al diagrama de clases de la Figura 3.1. El mismo comienza por dos clases principales, que son las desencadenantes del comienzo del proceso de descubrimiento y selección de servicios, la clase *Consumer*, y la clase *Provider*. La clase *Consumer* (Figura 3.3) reemplaza la especificación de I_R de la Sección 2.2 por una interfaz – que denominamos I_C (Interfaz a Consumir) – que representa las funcionalidades que necesita satisfacer cuando se consume el Servicio Web adecuado. Por otro lado, la clase *Provider* (Figura 3.4) reemplaza a la especificación I_S (de la Sección 2.2) por la interfaz que denominamos I_P (Interfaz Provista), que representa al proveedor del servicio cuyas responsabilidades incluyen conocer el nombre del servicio (por ejemplo, servicio de alquiler de coches), y eventualmente podría contener datos propios del proveedor que se consideren relevantes. De esta forma, las especificaciones de la interfaz requerida (I_R) y de la interfaz del servicio (I_S), fueron reemplazadas por instancias de las clases *Consumer* y *Provider* respectivamente, para adherir a la convención de nombrado del estándar SoaML.

La clase *Choreography* (Figura 3.5) denota los pasos que se deben respetar a la hora de representar secuencias válidas de llamadas a operaciones de un servicio en particular. Durante el desarrollo de este trabajo se tomará como soporte a la explicación, un Servicio Web perteneciente al dominio de alquiler de automóviles (RentACar). Por ejemplo, una secuencia válida en dicho

dominio sería llamar primero a una operación para comprobar si un auto está disponible para luego invocar a la operación para alquilar el mismo. Al momento de evaluar las responsabilidades del servicio, este aspecto secuencial resulta irrelevante. Sin embargo creemos que tiene mucho potencial a la hora de evaluar el comportamiento del servicio, puesto que la lógica que representa permite agregar claridad en el comportamiento esperado de la secuencialidad de llamadas a funciones específicas. Complementario a la clase *Choreography* se debería proveer un diagrama auxiliar (por ejemplo, diagrama de secuencia UML) para denotar la lógica del orden de llamados para el uso de una función específica.

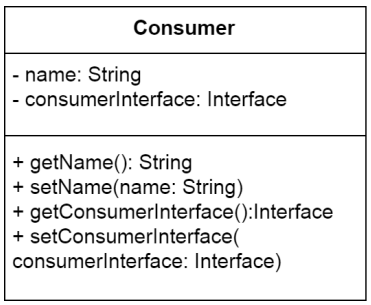

Figura 3.3: Diagrama de clase Consumer

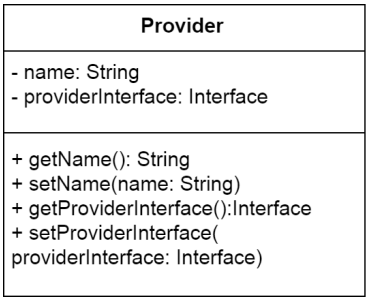

Figura 3.4: Diagrama de clase Provider

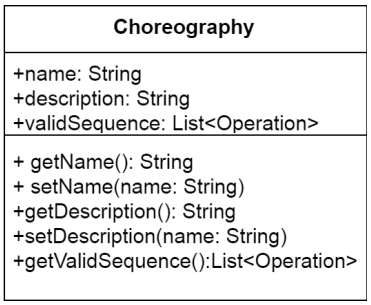

Figura 3.5: Diagrama de clase Choreography

La clase *Interface* (Figura 3.6) es la responsable de encapsular las operaciones que ofrece cada servicio, y cada instancia de *Interface* está intimamente vinculada con *Consumer* o *Provider* dependiendo con quién se relacione explícitamente. El procedimiento de Análisis de Compatibilidad de Interfaces compara la instancia Interface del *Consumer* (lo esperado por el consumidor) con las instancias de cada Interface del servicio candidato *Provider*. A su vez, *Interface* está compuesta por una lista de al menos una operación (*Operation*, Figura 3.7).
La clase *Operation* representa a cada una de las funciones operacionales provistas/solicitadas por cada proveedor/consumidor de un servicio. Cada operación está compuesta por su nombre (un identificador que describe semánticamente la función de la operación), puede o no tener una entrada (*Input*, Figura 3.8) y puede o no tener una salida (*Output*, Figura 3.9). A su vez, podría o no tener una o más excepciones o fallas (*Faults*, Figura 3.10).
La clase *Input* (Figura 3.8) representa los datos que la operación espera como entrada y está compuesto por un nombre y una lista de al menos un parámetro (*Parameter*, Figura 3.11) que actúa como entrada de la operación.
Inversamente, la clase *Output* (Figura 3.9) representa la salida/retorno de una operación. En el metamodelo propuesto, la salida de una operación (*Output*) está compuesta por un nombre, que puede ser cualquier tipo de identificador que represente a la salida de la operación y un conjunto de parámetros que determina cada uno de los datos que son retornados por la operación del servicio. En la versión anterior de la Herramienta de Evaluación de Servicios Web, utilizando interfaces Java, las operaciones cuentan simplemente con un tipo de retorno, ya sea un tipo simple o un tipo complejo, y en caso de que la operación retorne distintos elementos, este deberá encapsular cada uno de estos, siendo que no necesariamente en conjunto conformen una entidad única. Por ejemplo, utilizando la clase *Fee* (pago de tarifa) como salida de la operación, al agregarle un nombre de retorno, podremos dar información semántica del tipo de retorno que estamos brindando, si se instancia el nombre del parámetro con "*carFee*" estamos diciendo que

el pago será de autos.
En el contexto de los Servicios Web, la definición de excepciones representadas por la clase *Fault* no se han convertido en una práctica común [48] y de hecho, la mayoría de los WSDL no incluyen el detalle del manejo de excepciones.

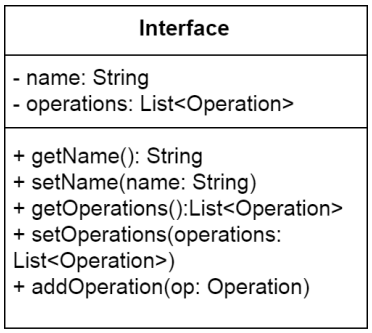

Figura 3.6: Diagrama de clase Interface

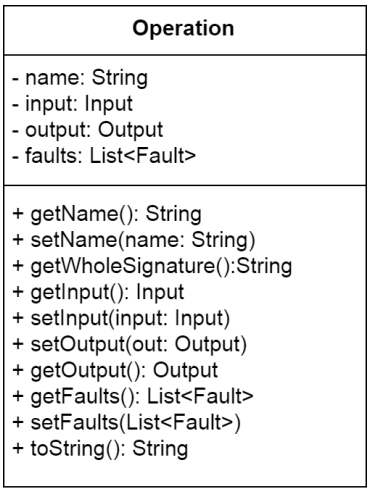

Figura 3.7: Diagrama de clase Operation

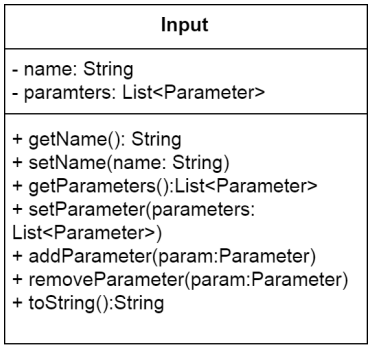

Figura 3.8: Diagrama de clase Input

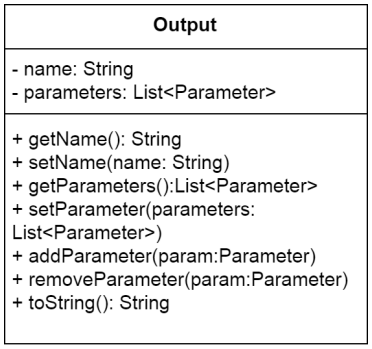

Figura 3.9: Diagrama de clase Output

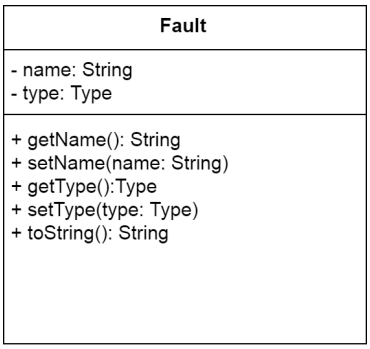

Figura 3.10: Diagrama de clase Fault

Para los tipos de datos se definió una clase abstracta llamada *Type* (Figura 3.12), la cual es la superclase de todos los posibles tipos de datos. La clase *Type* tiene tres subclases concretas:

- *SimpleType* (Figura 3.13) representa todos los posibles tipos simples o primitivos como un valor numérico entero secuencial. La clase cuenta con las siguientes propiedades públicas y estáticas: *STRING, INTEGER, BOOLEAN, LONG, SHORT, DECIMAL, FLOAT, DOUBLE, BYTE, DATE_TIME, DATE, BASE64_BINARY, HEX_BINARY, BASE64, ANY_TYPE, NORMALIZEDSTRING, ANY_URI.* Por convención, todas las variables se encuentran en mayúsculas por ser constantes. El atributo privado *type* de la clase descrita sólo puede tomar el valor de alguna de dichas constantes. Por ejemplo para crear una instancia de un *SimpleType* del tipo *short* hay que escribir la sentencia en el lenguaje Java correspondiente a *SimpleType shortType = new SimpleType (SimpleType.SHORT);* y de esta manera se evita que el programador tenga que saber cuál número entero representa la variable *SHORT,* lo que reduce posibles errores por parte del desarrollador.

- La clase *ComplexType* (Figura 3.15) representa los tipos complejos utilizados en la operación de servicio. Los tipos complejos están compuestos por atributos (*Attribute*, Figura 3.16), formados por un nombre (por ejemplo DNI, nombre, posición, etc) y un tipo que representa propiamente al tipo de atributo, puede ser de tipo simple, complejo o array. Es importante mencionar que la implementación propuesta permite el anidamiento de tipos complejos de distintos tipos, no exigiendo que sean todos de la misma clase.

- *ArrayType* (Figura 3.16) representa las colecciones tales como listas y arrays. Compuesto por el nombre del tipo propiamente dicho y el tipo de dato de los elementos contenidos. Si bien, a primera vista, *ArrayType* podría haberse representado con la clase *ComplexType,* cabe destacar que la clase *ArrayType* presenta un fuerte significado semántico y estructural, ya que indica que en su interior habrá una colección de objetos del mismo tipo.

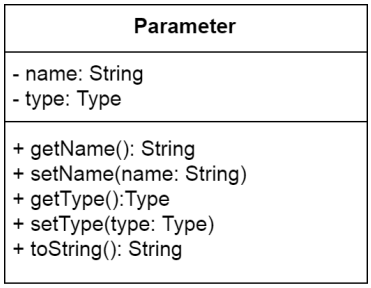

Figura 3.11: Diagrama de clase Parameter

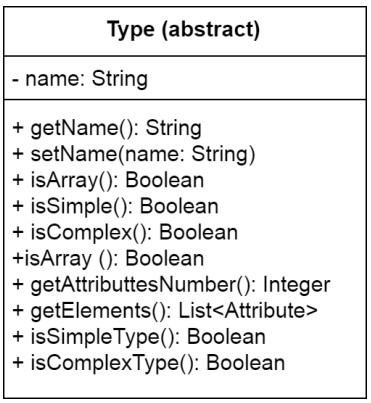

Figura 3.12: Diagrama de clase Type

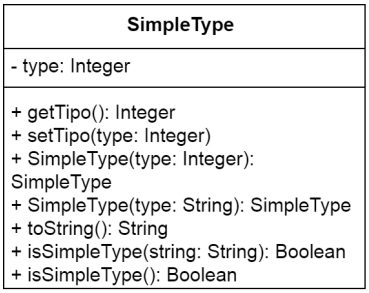

Figura 3.13: Diagrama de clase SimpleType

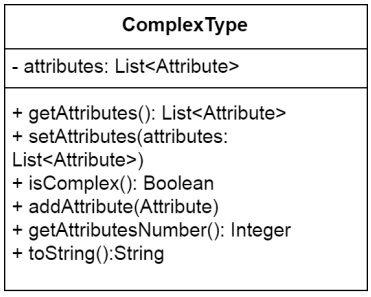

Figura 3.14: Diagrama de clase ComplexType

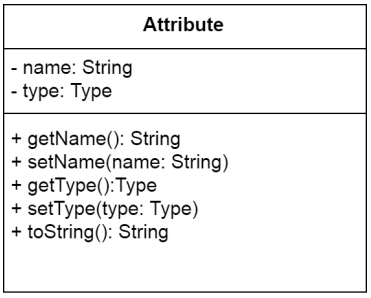

Figura 3.15: Diagrama de clase Attribute

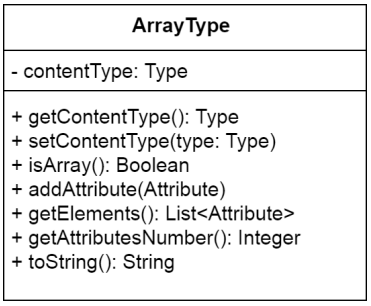

Figura 3.16: Diagrama de clase ArrayType

3.3. Conversor de WSDLs para instanciar el Metamodelo de Servicios Web

En la Sección 3.3.1 se analiza la relación existente entre el estándar WSDL y el metamodelo propuesto, usando como soporte a la explicación un caso de estudio del dominio de alquileres de autos (RentACar). Esta relación será analizada en profundidad, detallando la funcionalidad de cada operación WSDL y la forma en la que se relacionan con el metamodelo. Finalmente se muestra un diagrama de objetos instanciados de acuerdo al caso de estudio.

En la Sección 3.3.2, se explica como se construyó el componente Conversor del metamodelo propuesto, detallando las estructuras de datos analizadas y el proceso de análisis que se realiza para poder instanciar el metamodelo a partir de una descripción WSDL.

3.3.1. Relación entre WSDL y Metamodelo

En esta subsección se detalla la correlación entre el Metamodelo de Servicios Web y el estándar WSDL 2.0, para lo cual se utilizará un caso de estudio sencillo como soporte a la explicación.

Caso de estudio Durante el desarrollo de este trabajo se tomará como soporte a la explicación, un Servicio Web perteneciente al dominio de alquiler de automóviles (RentACar). El diagrama de clases UML para este dominio es presentado en la Figura 3.17. En el dominio de alquiler de autos, el servicio está compuesto por 3 operaciones encapsuladas en la interfaz *RentACar*: *getReservation*, *getCarFee* y *getAvailableCars*. La operación *getReservation* es utilizada para buscar reservas a partir de un número de identificador, retornando un string con la reserva correspondiente. Para *getCarFee*, dada una serie de comodidades que se desean que el vehículo a alquilar posea, retorna el costo del alquiler de un vehículo con estas características, a su vez *getCarFee* conoce las clases *Fee* y *CarSupplements*. Finalmente, *getAvailableCars* es una operación que retorna la lista de vehículos disponibles entre un rango de fechas dado. En el Listado de Código 3.1 se muestra un fragmento del documento WSDL 2.0 que describe el servicio *RentACar*.

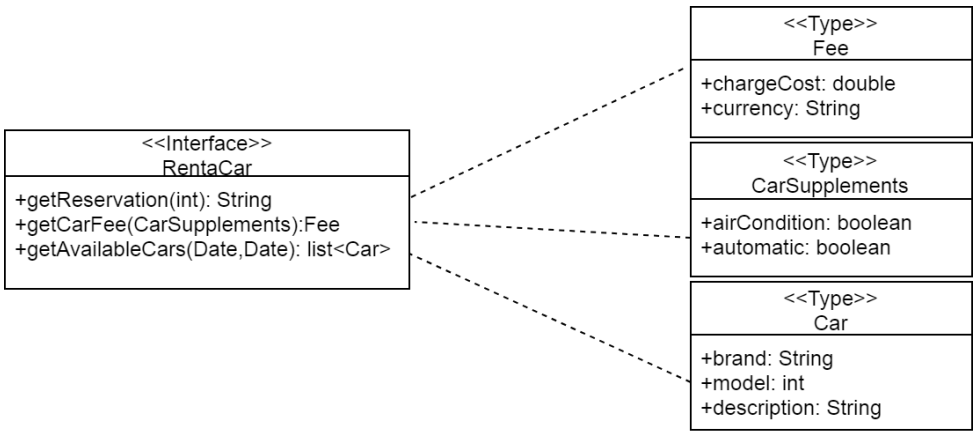

Figura 3.17: Visión esquemática UML del Caso de estudio

Listado de código 3.1: Documento WSDL 2.0 RentACar

```
1<description>
2  <types>
3     <xsd:schema attributeFormDefault="unqualified" xmlns:tns=""
elementFormDefault="unqualified" targetNamespace="" xmlns:xsd="">
4        <xsd:element name="getReservationInput">
                ...
10        </xsd:element>
```

```
11      <xsd:element name="getReservationOutput">
          ...
17      </xsd:element>
18      <xsd:element name="getCarFeeInput">
           ...
31          </xsd:element>
32      <xsd:element name="getCarFeeOutput">
          ...
45      </xsd:element>
46        <xsd:element name="getAvailableCarsInput">
47         <xsd:complexType>
48          <xsd:sequence maxOccurs="1" minOccurs="1">
49                <xsd:element minOccurs="1" maxOccurs="1" name="initDate"
type="xsd:date"/>
50            <xsd:element minOccurs="1" maxOccurs="1" name="endDate"
type="xsd:date"/>
51              </xsd:sequence>
52          </xsd:complexType>
53        </xsd:element>
54      <xsd:element name="getAvailableCarsOutput">
55          <xsd:complexType>
56            <xsd:sequence maxOccurs="unbounded" minOccurs="0">
57                  <xsd:element name="car">
58                    <xsd:sequence maxOccurs="1" minOccurs="1">
59                        <xsd:element minOccurs="1" maxOccurs="1"
name="brand" type="xsd:string"/>
60                        <xsd:element minOccurs="1" maxOccurs="1"
name="model" type="xsd:int"/>
61                        <xsd:element minOccurs="1" maxOccurs="1"
name="description" type="xsd:string"/>
62                    </xsd:sequence>
63                </xsd:element>
64            </xsd:sequence>
65          </xsd:complexType>
66      </xsd:element>
67      </xsd:schema>
68  </types>
69  <interface name="RentACar">
70    <operation name="getReservation">
71     <input messageLabel="In" element="tns:getReservationInput"/>
72      <output messageLabel="Out" element="tns:getReservationOutput"/>
73    </operation>
74      <operation name="getCarFee">
```

```
75      <input messageLabel="In" element="tns:getCarFeeInput"/>
76       <output messageLabel="Out" element="tns:getCarFeeOutput"/>
77     </operation>
78     <operation name="getAvailableCars">
79      <input messageLabel="In" element="tns:getAvailableCarsInput"/>
80      <output messageLabel="Out" element="tns:getAvailableCarsOutput"/>
81     </operation>
82  </interface>
83</description>
```

WSDL 2.0 vs. Metamodelo

En el fragmento de WSDL 2.0 – Listado de Código 3.1, línea 78 a 81 –, se define para la interfaz (*interface*) del servicio, la operación (*operation*) *getAvailableCars*, indicando la entrada y salida de la misma utilizando los tags *input* y *output* respectivamente. Cada entrada y cada salida se representa como un elemento (*element*) del documento WSDL. Los elementos (*elements*) que conforman las entradas y salidas de las operaciones se detallan dentro del apartado *types* del documento. Entre las líneas 2 y 68, se aprecia como se encuentra conformada la sección *types.* A su vez, la operación *getAvailableCars* se detalla desde la línea 46 hasta la 66 inclusive.

Por un lado, *getAvailableCarsInput* – línea 46 – es un tipo complejo que encapsula la entrada de dicha operación. La misma está compuesta por dos elementos (*element*), *initDate* y *endDate*, ambos del tipo fecha (*simple date*). Por otro lado, *getAvailableCarsOutput* – línea 54 – es un tipo complejo que encapsula la salida de la operación, compuesta por una colección de autos (*car*), donde cada auto es definido por tres elementos (*element*), *brand, model y description.* Nótese que dicho tipo de dato es una colección de *car* ya que la propiedad *maxOccurs* dentro del tag *sequence* tiene el valor *unbounded*, lo que significa que la máxima ocurrencia de objetos para este tipo de dato es ilimitada.

En la Figura 3.18 se muesta el diagrama de objetos que representa la salida (instancia del metamodelo) generada por el componente Conversor para el documento WSDL utilizado como ejemplo. Es importante destacar que la estructura de datos utilizada generará una sola vez los tipos de datos, y si fuera necesario realizar otra instancia de la misma clase, se fija la dirección de memoria mediante un puntero a aquella que fue creada por primera vez. Esto quiere decir que sólo existirá una instancia por cada tipo definido dentro del documento. Por ejemplo sólo va a existir una instancia que representa al tipo simple *string*, y cada elemento relacionado con este tendrá una referencia a dicha instancia.

En el diagrama de objetos de la Figura 3.18 se puede observar una instancia de la clase *Interface*, llamada *RentACar* la cual agrupa las operaciones que ofrece el servicio. Recordamos que este servicio cuenta con tres operaciones, *getReservation, getCarFee* y *getAvailableCars*, cada una asociada a su respectiva entrada y salida (*input/output*). A su vez cada entrada/salida está relacionada con los parámetros que las componen. A fines de mostrar resumidamente como quedaría el grafo, solo se detalla la función getAvailableCars. Podemos destacar que para la operación *getAvailableCars* se define como entrada el objeto llamado *getAvailableCarInput*

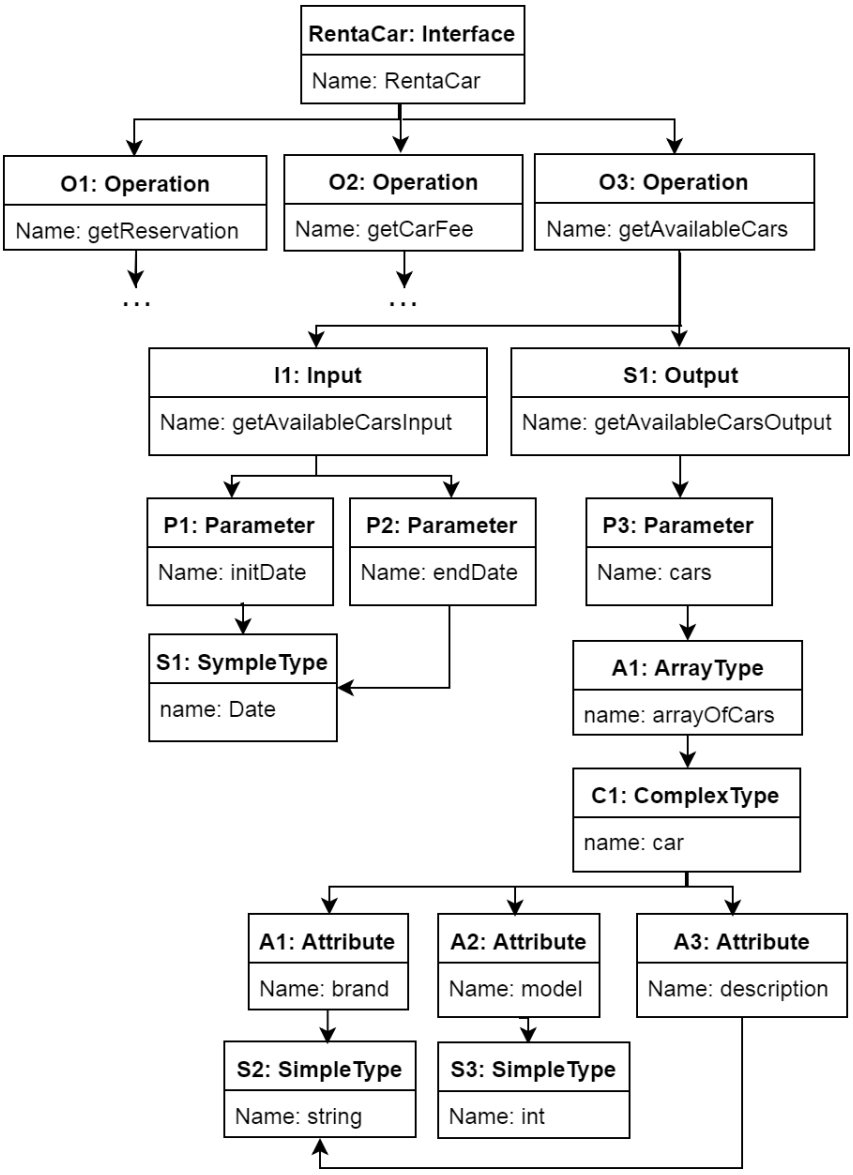

Figura 3.18: Ejemplo de metamodelo instanciado para el caso de estudio

de la clase *Input*. Dicha entrada se encuentra conformada por dos parámentros asociados al tipo simple *date*, que hacen referencia a la fecha de inicio y fin respectivamente. La salida de esta operación está compuesta por un parámetro llamado *cars*, que representa a los autos disponibles entre las fechas ingresadas como entrada de la operación. El parámetro *cars*, es asociado a una instancia de la clase *ArrayType* llamada de la misma manera, que a su vez posee como *contentType* una instancia de la clase *ComplexType* llamada *Car* (que representa a cada automovil individualmente). En resumen, la salida de esta operación es un arreglo (*ArrayType*) de automoviles (*Car*). De cada auto (*Car*) conoce su modelo, marca y una breve descripción.

3.3.2. Construcción del Conversor

En los párrafos siguientes se detallan los pasos principales para la construcción del Conversor. Primeramente fue necesario analizar las herramientas que se encuentran disponibles que podrían ser potencialmente útiles a la hora de implementar el módulo de software Conversor del

Metamodelo. Luego se explica en detalle el proceso de conversión de documentos WSDL 2.0 a instancias del metamodelo junto con los algoritmos involucrados para realizar dichas operaciones.

Selección de herramientas para construcción de módulo de software Conversor del Metamodelo Tomando como base la explicación detallada en la Sección 1.3.2, donde se analizaron las herramientas evaluadas para ser utilizadas potencialmente por el módulo de software Conversor del Metamodelo, en la presente sección se pretende destacar ventajas y desventajas de dichas herramientas. En principio, resultó necesario satisfacer la necesidad de parsear un documento WSDL. Para ello se analizaron las herramientas *JWSDL*, *WODEN*, *EasyWSDL, DOM* y *SOA Membrane*. Luego de este análisis, WODEN y DOM resultaron ser los candidatos más apropiados para la construcción del módulo de software Conversor.

Motivos principales por los cuales no se utilizaron las siguientes herramientas

- **JWSDL:** posee un inconveniente fundamental (éste figura en la documentación y se pudo comprobar en la práctica) por el cual no se utilizó esta herramienta para manipular documentos WSDL. Cualquier hijo de <types> (dentro del documento) es tratado como elemento de una extensión que no se encuentra implementada entre sus funciones. Debido a esto resulta imposible poder analizar las estructuras de los tipos complejos, y por consiguiente las entradas y salidas de las operaciones dentro de un documento WSDL. En otras palabras, no se utilizó ya que no era posible analizar los datos definidos en el *tag types* de los documentos. Además no soporta documentos WSDL 2.0, y la intención es concentrarse en las tecnologías más actuales que dispone el mercado.

- **SOA Membrane**: cuando se intentó acceder a los atributos, se dificultó el acceso a los tipos complejos de datos. Esto es porque no se pudo profundizar en su contenido con el detalle necesario para el componente de software que se pretendía desarrollar.

- **EasyWSDL:** en la teoría soporta versiones de WSDL 1.1 y 2.0, pero al intentar cargar el conjunto de documentos de la plataforma Mashape, un porcentaje no pudo ser utilizado porque no eran sintácticamente válidos según el metamodelo propuesto por esta herramienta. Debido a esto, el proceso de análisis para los documentos afectados no pudo continuar. Además, el alcance de la herramienta es extenso, coordinar con los mensajes de entrada y salida para la creación y/o manipulación de datos lleva un proceso de aprendizaje muy grande. Por estas razones se decidió no utilizar esta herramienta.

Herramientas utilizadas en el Conversor

Para la creación del módulo de software Conversor, se utilizaron dos herramientas que fueron de principal relevancia a la hora de poder concretar la creación de dicho módulo. WODEN y DOM permitieron, gracias a sus diferentes funcionalidades, abarcar los distintos aspectos que se detallan a continuación.

- **WODEN:** utilizada para validar que los documentos WSDL utilizados como entrada al Conversor del metamodelo estuvieran bien formados. Esto quiere decir que con WODEN se validó que los documentos utilicen las etiquetas (tags) definidas por la W3C para

documentos WSDL, que haya correspondencia entre mensajes, la definición de los mismos, etc. De esta manera, a la hora de testear la Herramienta de Evaluación de Servicios Web, se trabajó con documentos bien formados desde un principio. Además esta herramienta permitió poder trabajar con documentos WSDL versión 1.1. Básicamente si se necesita trabajar con documentos WSDL 1.1, se utiliza una función que convierte documentos de la versión WSDL 1.1 a la versión de WSDL 2.0 para luego si procesarlo como entrada al componente Conversor del metamodelo. Como un ejemplo, en el Capítulo 4, se pudo experimentar con un conjunto de 1146 archivos WSDL versión 1.1 de Servicios Web reales que provienen de la plataforma Mashape[2].

- **DOM:** acorde con DOM, cualquier elemento perteneciente al proyecto de un documento XML es un nodo. De esta manera se pudo acceder a todos los detalles específicos dentro de un documento WSDL sin ningún tipo de restricción. Esta librería es una de las herramientas más usadas y populares a la hora de manipular documentos XML. En el presente trabajo se hizo énfasis en las estructuras del documento relevantes para el módulo de software Conversor del Metamodelo. Por esta razón, se prestó especial atención a la sección *types* e *interface* que son parte de los elementos que conforman un WSDL 2.0 bien formado. Para esto resultó especialmente útil contar con una herramienta que permitiera recorrer y analizar un documento WSDL de forma confiable y robusta, sin nigun tipo de restricción, es decir, permitiendo el acceso a la totalidad del documento. DOM cumple con las características necesarias para llevar adelante el objetivo de construir un instanciador automático. Además, DOM nos permite gracias a una de sus funciones de navegación en el código fuente, analizar si el documento está bien formado, es decir que no tenga errores sintácticos.

Conversión de documentos WSDL 2.0 a instancias del metamodelo

El Conversor desarrollado recibe como entrada un documento WSDL (generalmente por medio de la URL) y devuelve como salida una instancia del metamodelo propuesto, tratando al documento WSDL como un XML. Tratarlo de tal manera nos permite que el mismo sea parseado utilizando la herramienta DOM previamente mencionada. Los elementos del XML son vistos como nodos, y cada nodo se encuentra en un determinado nivel en la estructura de arbol, siendo N_0 el nivel inicial, y cada vez que se desciende un nivel, el subíndice de N aumenta en 1 unidad. El nodo *document*, es el nodo padre y se encuentra en el nivel N_0. En el nivel N_1 se encuentran los nodos *interface* y *types.* En *types* se definen todos los tipos de datos existentes en el documento. En *interface* se encuentran detalladas las operaciones (*operation*) que ofrece el servicio (N_2). En el nivel N_3 tenemos los mensajes de entrada (*input*) y de salida (*output*).

Siguiendo nuestro ejemplo de base del dominio RentACar, la Figura 3.19 muestra un esquema de como quedaría la estructura con los distintos niveles que conforman un documento WSDL 2.0 al parsearla con la herramienta DOM. Cabe destacar que en esta figura, solo se detalla la operación getAvailableCars, con el objetivo de dar un ejemplo gráfico de fácil comprensión.

[2] `http://mashape.com`

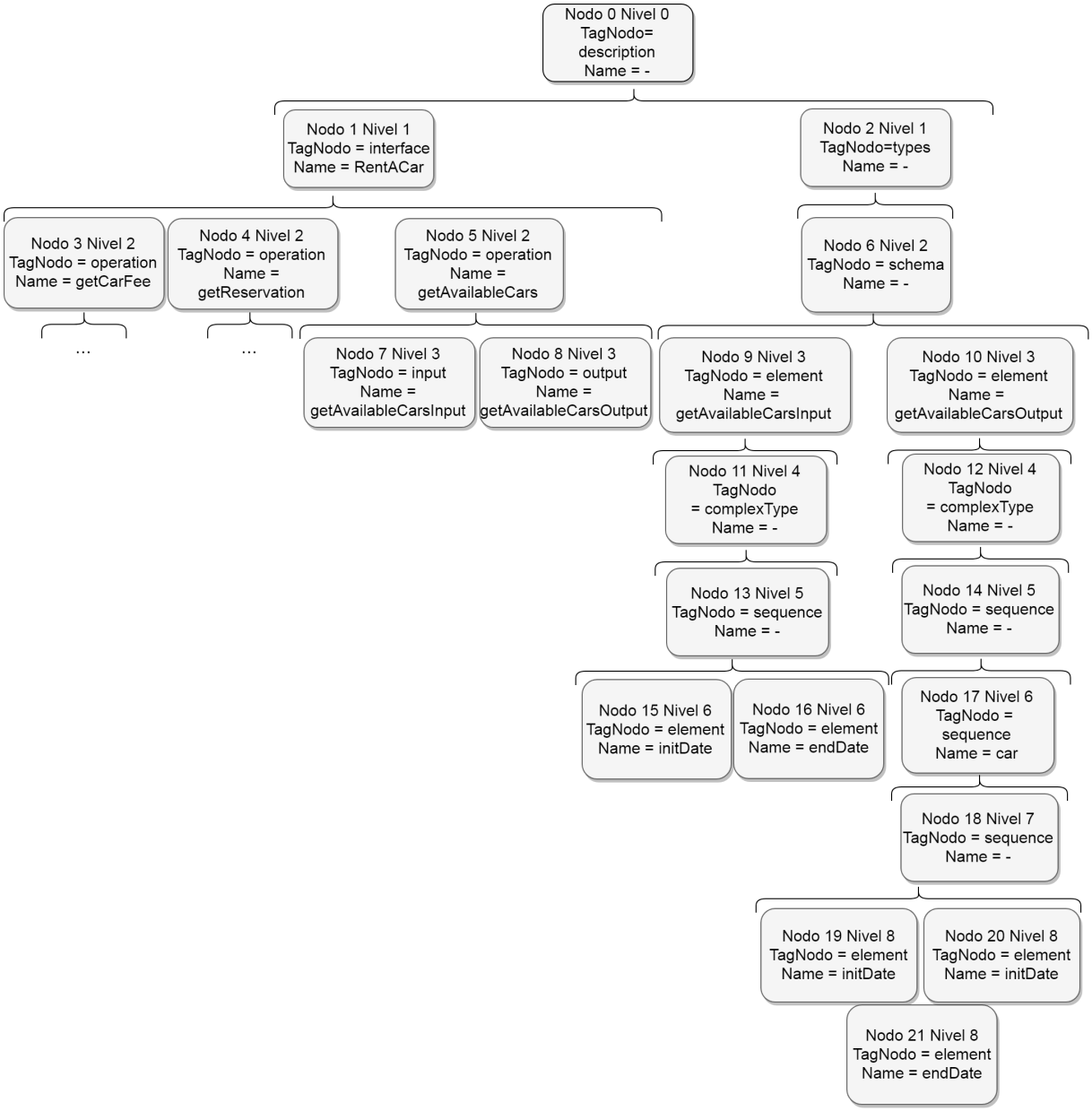

Figura 3.19: Ejemplo de objetos que se analizan del documento WSDL parseado con la herramienta DOM

A continuación se detalla el pseudocódigo del módulo de software de conversión de documentos WSDL 2.0 a instancias del metamodelo propuesto. En el Algoritmo 1 la función propuesta recibe como entrada el documento WSDL – línea 1. El primer paso es tratarlo como un documento XML, en cuanto a las propiedades del mismo – línea 2. Inicialmente obtenemos el primer nodo del árbol (la raíz del documento) siendo el *nodo cero* de la Figura *3.19* y luego sus hijos directos. Para el metamodelo propuesto, en este nivel estructural sólo nos interesan los nodos que contienen a *types* e *interface*. Si el nodo es *types,* se instancian los tipos definidos en esta clase. Los detalles de este procedimiento son presentados en el Algoritmo 3.2. Si el nodo es *interface*, al obtener sus descendientes directos obtendremos las *operations* –O1, O2 y O3 de la Figura 3.18. Los hijos de *operation* pueden ser *input* y/u *output*. En caso de ser el primero se crea la estructura de *input* y se setea la misma al metamodelo – *I1: Input* de la Figura 3.18. En caso de ser la segunda opción, se crea la estructura *output* – *S1: Output* – y se setea al

metamodelo. Una vez terminado de recorrer los hijos de la última *operation* se agregan a la interfaz del metamodelo. Este procedimiento finaliza cuando se termina de recorrer todas las operaciones existentes. Finalmente es retornada una instancia del metamodelo generado a partir del documento WSDL utilizado como entrada.

Algoritmo 1 Convertir WSDL a Metamodelo

```
1 function convertir_Wsdl_A_Metamodelo (DocumentoWSDL documentoWSDL){
2  documentoXML = DOM.leerDocumento(documentoWSDL);
3  Elemento raiz = documentoXML.obtenerNodoRaiz();
4  Lista<Nodo> nodos = raiz.obtenerNodosHijos();
5  SoaMLInterface interface = new SoaMLInterface();
6  for(Nodo nodo: nodos){
7   if(nodo.getTagNodo()=='types'){
8        List<Type> metamodelTypes =
     'instanciar todos los tipos definidos en types'
9   }
10  if(nodo.getTagNodo()=='interface'){
11   Interface metamodelInteface = 'crear interface SoaMl'
12   List<Nodo> wsdlOperations = nodo.obtenerNodosHijos()
13   for(Nodo wsdlOperations: wsdlOperation){
14    Operation metamodelOperation = 'instanciar operacion metamodelo'
15    List<Nodo> wsdlInOuts = wsdlOperation.obtenerNodosHijos();
16    for(Nodo inOut: wsldInOuts){
17     if(inOut.getTagNodo=='input'){
18      Input entrada= 'creo input según metamodelo';
19      metamodelOperation.setInput (entrada);
20     }
21     else if(inOut.getTagNodo=='output'){
22      Output salida= 'creo output según metamodelo';
23      metamodelOperation.setOutput(salida);
24     }
25    }
26    metamodelInterface.agregarOperation(metamodelOperation);
27   }
28  }
29 }
30 return metamodelInterface;
31}
```

Para evaluar los tipos de datos nos basamos principalmente en los Algoritmos 3.2, 3.3, 3.4, 3.5, donde en el primero se encuentra la lógica principal, y los restantes describen las funciones auxiliares. Los mismos cumplen la función de discriminar la estructura sintáctica de los programas involucrados. Dado un nodo como entrada a la función del primero, el análisis retorna si se tiene tipos de datos simples, complejos o arreglos.

En los dos párrafos siguientes se explicará brevemente el funcionamiento del Algoritmo 3.2. En la línea 3, se verifica si el nodo es de tipo simple, en cuyo caso se crea el objeto de tipo simple con sus respectivos datos. Luego, se analiza en la línea 5, si estamos ante la presencia de un tipo de dato arreglo, para lo cual consultamos sobre un atributo llamado *unbounded*. En

caso de no ser *unbounded*, se retorna el tipo simple que se creó en la línea 4. Caso contrario, nos encontramos ante un tipo de dato arreglo (*ArrayType*), el cual contendrá al tipo simple previamente mencionado, siendo una colección de tipos simples.

Ahora bien, si tenemos un tipo de dato complejo – *ComplexType* (línea 13) –, debemos indagar en su estructura interna, puesto que puede estar compuesto por más tipos de datos complejos, arreglos o tipos simples. Mediante llamados recursivos se construirá la estructura del tipo de dato complejo. Una vez que creamos el tipo de dato complejo, resta preguntar si el mismo se encuentra contenido en una estructura de arreglo, lo que nos llevará en caso afirmativo a crear un arreglo asociado al tipo complejo recientemente creado y retornarlo; caso contrario retornamos el tipo complejo previamente instanciado.

Para saber si estamos ante la presencia de un tipo de dato complejo se creó la función *esTipoComplejo(Nodo nodo)* (Algoritmo 3.3), en la cual se obtiene los hijos del *nodo* que entra por parámetro de la función , en caso de que por lo menos tenga uno, y el nombre del mismo contenga la palabra compuesta *ComplexType* estamos en presencia de un tipo de dato complejo. Para saber si tenemos un tipo de dato simple recurrimos a la función *esTipoSimple(Nodo nodo)* (Algoritmo 3.4). Se obtiene el atributo *type* del *nodo* y si coincide con algún tipo de dato simple de los definidos en la Sección 3.2.1 *SimpleType*, entonces efectivamente es un tipo simple. Por último tenemos la función *obtenerNombreTipoSimple(Nodo nodo)* (Algoritmo 3.5) que retorna el nombre de un tipo de dato simple.

Listado de código 3.2: Obtener tipo de dato

```
1 function Type obtenerTipoDeDato(Nodo nodo) {
2 String unArreglo = nodo.getAttribute("maxOccurs");
3 if (esTipoSimple(nodo)) {
4  SimpleType tipoSimple = new SimpleType(obtenerNombreTipoSimple(nodo));
5  if (unArreglo.equalsIgnoreCase("unbounded")) {
6   ArrayType array = new ArrayType();
7   array.setearTipoDeContenido(tipoSimple);
8   array.setearNombre("ArrayOf" + tipoSimple.obtenerNombre());
9   return array;
10 }
11  else return tipoSimple;
12 } else {
13  if (esTipoComplejo(nodo)) {
14   ComplexType metamodeloComplexType = new ComplexType();
15   metamodeloComplexType.setearNombre(nodo.getAttribute("name"));
16   NodeList nodoSecuencia = nodo.obtenerNodosHijos();
17   if (nodoSecuencia != null) {
18    ArrayList<Attribute> atributos = new ArrayList<Attribute>();
19    Node nodosDeTipoComplejo = nodoSecuencia.obtenerNodosHijos();
20    if (nodosDeTipoComplejo != null) {
21     NodeList elementosDeTiposComplejos =
       nodosDeTipoComplejo.obtenerNodosHijos();
```

```
22      for (int i = 0; i < elementosDeTiposComplejos.getLength(); i++) {
23       Attribute atributoMetamodelo = new Attribute();
24       Nodo atributoACrear = elementosDeTiposComplejos.item(i);
25       atributoMetamodelo.setearNombre(
atributoACrear.getAttribute("name"));
26       atributoMetamodelo.setearTipo(obtenerTipoDeDato
elementosDeTiposComplejos.item(i)));
27       atributos.añadir(atributoMetamodelo);
28      }
29      metamodeloComplexType.setearAtributos(atributos);
30     }
31    }
32    if (unArreglo.equalsIgnoreCase("unbounded")) {
33     ArrayType array = new ArrayType();
34     array.setearContentType(metamodeloComplexType);
35     array.setearNombre("ArrayOf"+metamodeloComplexType.getName());
36     return array; //retorno el tipo de dato arreglo
37    }
38    else return metamodeloComplexType;
//retorno el tipo de dato ComplexType
39  }
40 }
41 return null; // no es tipo simple, complejo ni arreglo
42}
```

Listado de código 3.3: Es tipo complejo

```
43 private static boolean esTipoComplejo(Nodo nodo) {
44 boolean retorno = false;
45 NodeList hijos = nodo.obtenerNodosHijos();
46 if (hijos != null) {
47  Node ComplexType = hijos.item(1);
48 if (ComplexType != null &&
       ComplexType.getNodeName().contains("complextype"))
49  retorno = true;
50 }
51return retorno;
52}
```

Listado de código 3.4: Es tipo simple

```
53 private static boolean esTipoSimple(Nodo nodo) {
54 boolean retorno = false;
55 String type = nodo.getAttribute("type");
56 if (type != null) {
```

```
57   if (type.contains(":"))
58   type = type.obtenerTipoDeDato();
59   if (SimpleType.esTipoSimple(type))
60    retorno = true;
61 }
62 return retorno;
63}
```

Listado de código 3.5: Obtener nombre tipo simple

```
64private static String obtenerNombreTipoSimple(Nodo nodo)
65{
66 String type = element.getAttribute("type");
67 if(type!=null)
68       if(type.contains(":"))
69              type=type.obtenerTipoDeDato();
70 return type;
71}
```

3.4. Integración a la Herramienta de Evaluación de Servicios Web

El objetivo principal del desarrollo del Metamodelo para descripción de Contratos de Servicios Web es contar con una especificación de la funcionalidad que ofrece cada servicio (independientemente de la tecnología en la cual esté implementado). Al contar con dicho metamodelo resulta necesario que la evaluación de servicios candidatos se realice en función de instanciaciones del metamodelo. Para ello, se desarrolló un componente de software que contiene al metamodelo. En el nuevo enfoque, tanto los requerimientos funcionales por parte de desarrolladores de aplicaciones orientadas a servicios, como los propios servicios candidatos, serán representados y comparados mediante instanciaciones del metamodelo desarrollado.

3.4.1. Proceso de selección y descubrimiento modificado

En la Figura 3.20 se presenta de manera esquemática la modificación sobre el proceso de descubrimiento y selección de Servicios Web, al considerar los dos nuevos componentes (metamodelo y conversor), que son parte de la nueva Herramienta de Evaluación de Servicios Web. El nuevo proceso consiste inicialmente en generar instancias del metamodelo propuesto a partir de los documentos WSDL correspondientes a los Servicios Web candidatos. En el Paso 1.1 se generan las instancias del metamodelo que corresponden al requerimiento funcional del servicio que se espera consumir por la aplicación – que ahora denominamos I_C (Interfaz a Consumir), en vez de I_R – a partir de la cual se generan consultas (*queries*). Luego en el Paso 1.2 se realizan las consultas en el registro de descubrimiento (tal como el de EasySOC), para obtener los documentos WSDL de los servicios del Proveedor, que pueden corresponder a versiones 1.1 y 2.0. Con estos documentos, se generan las instancias del metamodelo – que ahora denominamos I_P (Interfaz Provista), en vez de I_S – utilizando el componente Conversor para

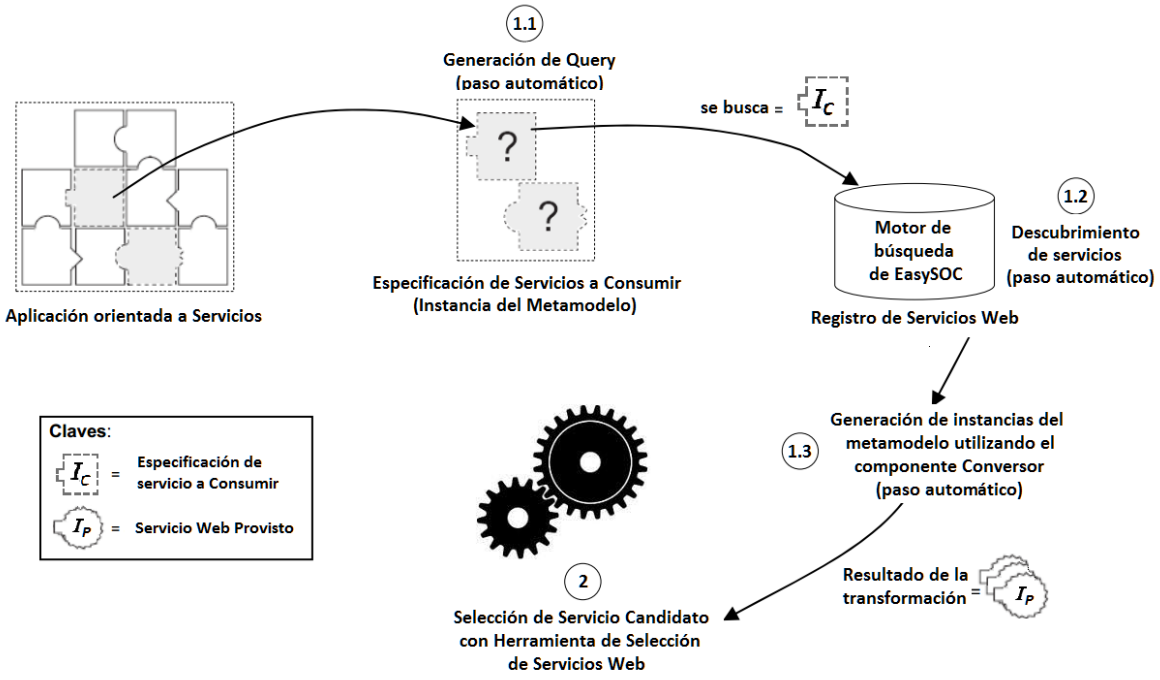

Figura 3.20: Modificación del Proceso de Descubrimiento y Selección de Servicios Web

obtener el conjunto que servirá de entrada al Paso 2, que es la selección del servicio candidato más apto.

En el procedimiento de Análisis de Compatibilidad de Interfaces, que es parte del Método de Selección de Servicios Web (Paso 2), se realizó una actualización con respecto al componente del Metamodelo. En la presente versión (Figura 3.21), se cuentan con las instancias del metamodelo, mediante el ingreso de los objetos I_P e I_C, por lo que se puede acceder a todos sus atributos en una forma directa para el proceso de obtención de datos de signatura y evaluación tanto semántica como estructural. Si bien en la versión anterior ilustrada en la Figura 2.3, desde los archivos WSDL versión 1.1 se extraían los elementos de signatura, ingresaban las clases I_S e I_R y actualmente I_P e I_C, las herramientas de soporte son las mismas: WordNet y el listado de Stop words.

Al realizar la integración del componente del metamodelo se adaptaron las estrategias subyacentes para la evaluación de servicios desde el punto de vista estructural y semántico presentadas en la Sección 2.3 (Capítulo 2). Para ello se estableció una correspondencia entre los elementos del metamodelo definido y las interfaces Java utilizadas en el enfoque anterior.

3.4.2. Ventajas sobre implementacion anterior

En las versiones anteriores de la plataforma para selección de Servicios Web [19, 8], la evaluación a nivel de contratos se realizaba analizando interfaces Java generadas a partir de documentos WSDL. Como se mencionó en la Sección 1.1, cada proveedor de servicios utiliza diferentes versiones de WSDL para describir las interfaces de sus servicios. A la hora de evaluar Servicios Web descritos mediante archivos WSDL, era necesario contar con herramientas externas, para cada servicio, generar las clases Java en archivos separados porque se utilizaba

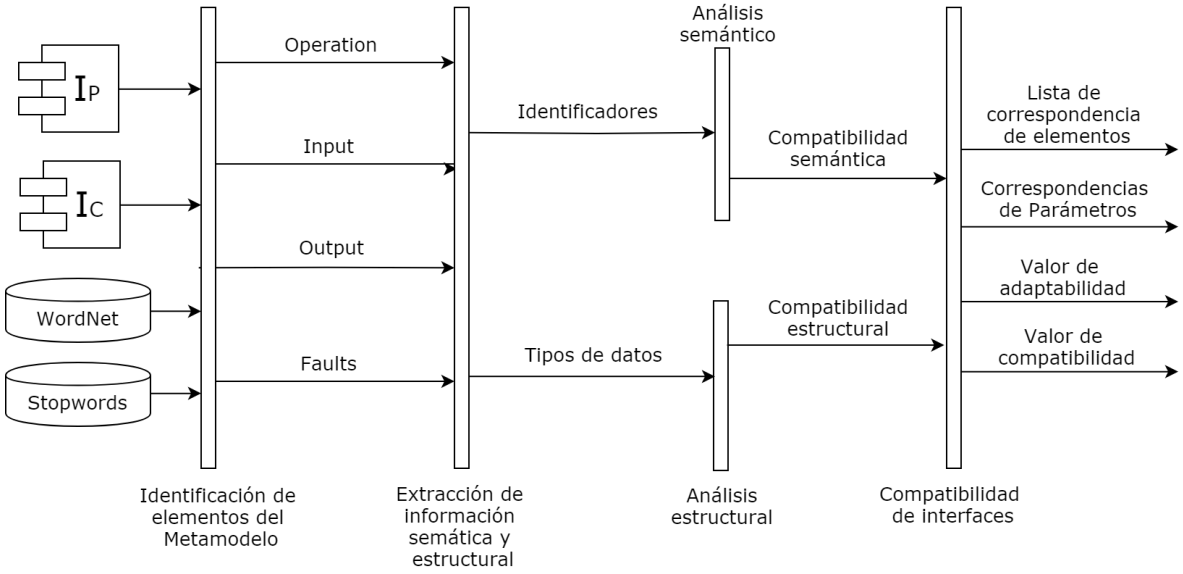

Figura 3.21: Análisis de Compatibilidad de Interfaces

la herramienta Java Reflection[3] como soporte para el análisis. Java Reflection es un mecanismo poderoso que provee el lenguaje de programación Java (que no existe en otros lenguajes, tal como Pascal, C o C++) el cual permite al programa examinarse a sí mismo (introspección), y manipular propiedades internas. Se utilizaba para acceder y explorar los archivos Java compilados (.class) y extraer la información de cada interfaz. Junto a Java Reflection, se utilizaba Paranamer[4], cuya función era permitir obtener el acceso a los nombres de los parámetros de los métodos no privados y constructores en tiempo de ejecución. Normalmente esta información se pierde en el proceso de compilación.

Con la implementación actual sólo es necesario disponer de las instanciaciones del metamodelo sin la necesidad de que existan archivos vinculados entre sí. Esa es una importante diferencia con respecto a una herramienta externa para el procesamiento de WSDL como podría ser EasyWSDL, puesto que para cada Servicio Web se generarían los siguientes archivos Java:

- Un archivo que incluye la definición de todas las operaciones, el cual representa a la clase utilizada para invocar al servicio.
- Dos archivos Java por cada operación en el servicio. Uno de los archivos representa un tipo complejo que encapsula las entradas de la operación y el otro archivo representa otro tipo de dato complejo que encapsula las salidas de la operación.
- Un archivo Java por cada tipo de dato complejo propio del dominio específico del servicio.

Además, el metamodelo propuesto ofrece mas información "semántica" correspondiente al servicio. Un claro ejemplo se ve en la representación de las salidas de una operación. En el metamodelo propuesto, la salida de una operación (output) esta compuesta por un nombre, que puede ser cualquier tipo de identificador que represente a la salida de la operación y un conjunto de parámetros que determina cada uno de los datos que son retornados por la operación del

[3]http://www.oracle.com/technetwork/articles/java/javareflection-1536171.html
[4]https://github.com/paul-hammant/paranamer

servicio. En la versión anterior de la herramienta, utilizando interfaces Java, las operaciones cuentan simplemente con un tipo de retorno, y en el caso de que la operación retorne distintos elementos, se deberá encapsular cada uno de ellos, aún cuando en conjunto no conformen una entidad única y distintiva del dominio.

Capítulo 4

Evaluación Experimental

En este experimento se compara la visibilidad de Servicios Web candidatos utilizando la nueva versión del Proceso de Selección de Servicios Web, la cual incorpora el metamodelo presentado en el Capítulo 3. Para esta evaluación se utilizó un conjunto de servicios compuesto por 1146 descripciones WSDL de Servicios Web obtenidos de la plataforma Mashape. En la Figura 4.1 se resume el proceso experimental que se encuentra conformado por tres fases: Generación de consultas, Ejecución experimental y Resultados.

Para realizar este experimento, inicialmente dos grupos de ingenieros de software seleccionaron aleatoriamente 30 descripciones de servicios (de las 1146 anteriormente mencionadas). Estas descripciones fueron consideradas como servicios relevantes en el contexto de este experimento (Fase de Generación de Consultas de la Figura 4.1 de la Sección 5.1). Las descripciones de servicios que no fueron marcadas como relevantes para ninguna consulta fueron utilizadas como "ruido" con el objetivo de incrementar la dimensión del experimento, en términos del número de descripciones WSDL. Luego, un conjunto de 42 consultas fueron generadas manualmente por los dos grupos de expertos, utilizando como base los 30 servicios relevantes (Sección 5.1).

En la Fase de Ejecución Experimental (Figura 4.1) se utilizó el conjunto de 1146 Servicios Web para poblar un registro de descubrimiento. Los resultados brindados por este registro de descubrimiento proveen una base para comparar otros enfoques de selección de servicios mas exhaustivos. De este modo, consultamos el registro de descubrimento con las 42 consultas generadas manualmente por los expertos. Luego, el registro retornó los 10 servicios mas relevantes para cada consulta, conteniendo potencialmente al servicio objetivo (comunmente denominado *gold standard*) para la consulta de acuerdo a una simple evaluación sintáctica. Sobre los resultados retornados por el registro de descubrimiento, fue ejecutado el enfoque de selección de servicios.

Finalmente, en la Fase de Resultados (Figura 4.1) se comparan los resultados en términos de visibilidad del servicio relevante entre nuestro enfoque y el registro de descubrimiento de servicios. Los resultados del experimento fueron evaluados de acuerdo a una popular métrica perteneciente al campo de Recuperación de Información – *Precision-at-n*.

El resto del Capítulo se distribuye de la siguiente manera: En la Sección 5.1 se explica cómo han sido generadas las consultas utilizando el conjunto de 1146 Servicios Web. En la Sección 4.2 se detalla cómo fue llevada a cabo la ejecución experimental. En la Sección 4.3 se explican los pasos que se llevaron a cabo para obtener los resultados esperados, incluyendo la decisión de

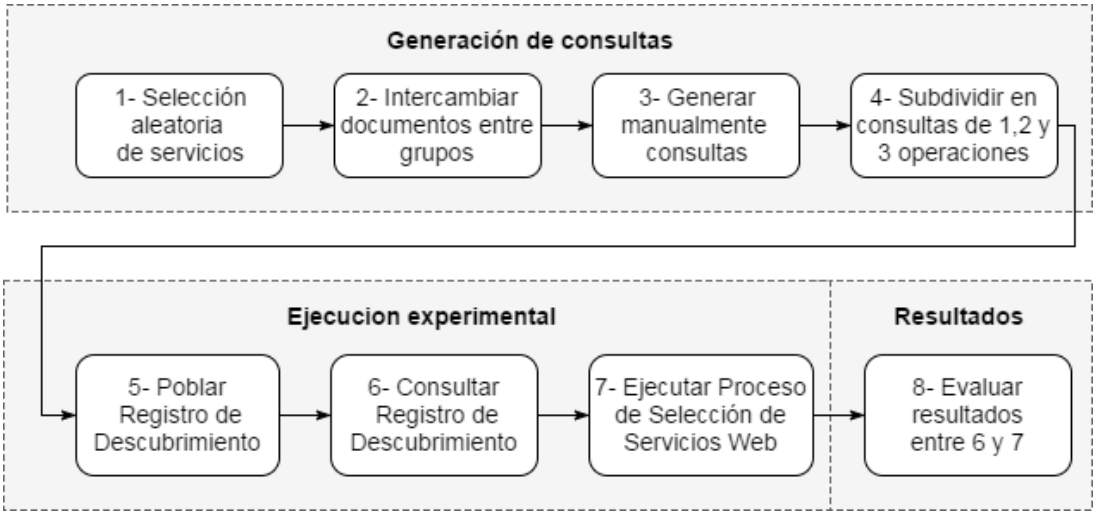

Figura 4.1: Proceso experimental

utilizar ciertas métricas para medir los resultados del experimento.

4.1. Generación de consultas

Como se mencionó anteriormente, para este experimento se generaron manualmente 42 consultas, donde cada una representa una funcionalidad esperada por una aplicación cliente (consumer). Éstas consultas no han sido automáticamente generadas por un algoritmo de mutación (a diferencia de trabajos anteriores [19, 27]), sino que fueron construidas por dos grupos de ingenieros de software (GrupoA y GrupoB). El dataset utilizado está compuesto por más de 1146 descripciones de Servicios Web en formato WSDL 2.0. Estas especificaciones fueron entregadas a los grupos de profesionales, donde cada grupo seleccionó 15 servicios como los servicios candidatos para generar las consultas (Paso 1 de la Figura 4.1). Con el objetivo de ser lo más imparcial posible, a la hora de seleccionar los candidatos se intercambiaron los documentos seleccionados por cada grupo (Paso 2).

Luego, cada grupo, reescribió sus documentos (descripciones de servicios) asignados, alterando los elementos incluidos en los mismos (Paso 3). La única condición a la hora de reescribirlos fue que la semántica tenía que ser preservada, es decir, la reescritura de la descripción no debería cambiar las funcionalidades que ofrecía el servicio ni la intencionalidad de las funciones involucradas. Los elementos alterados incluyeron identificadores (por ejemplo los nombres de las operaciones, nombres de parámetros, campos de los tipos complejos, entre otros) y los tipos de datos (incluidos en parámetros, retornos y atributos de tipos complejos). Luego de reescribir los documentos, se dividieron los mismos agrupando en nuevos documentos de una, dos y tres operaciones (Paso 4). Estos nuevos documentos conformaron las consultas utilizadas en esta evaluación experimental. De esta forma, cada consulta estuvo compuesta como máximo por tres operaciones relacionadas.

Ejemplo Utilizaremos el servicio RentACar presentado en la sección 3.2.1, Capítulo 3, para dar un ejemplo de cómo fue el proceso de generación de consultas. Este servicio ofrece las siguientes tres operaciones:

getReservation: dado un número de identificador (*reservationCode*), retorna un String que representa la reserva correspondiente (*reservationReturn*).

getCarFee: dados ciertos accesorios con los que cuenta un vehículo (*supplements*) retorna el valor correspondiente al alquiler del mismo. Las características del vehículo son *airCondition* y *automatic*. Como salida de la operación *GetCarFee* es el costo del alquiler (*chargeCost*) y la moneda (*currency*) en que se encuentra expresado dicho monto.

getAvailableCars: dadas dos fechas (*initDate* y *endDate*), retorna los automóviles disponibles en dicho período. Cada automóvil (*Car*) esta compuesto por su marca *(brand*), modelo *(model)* y descripción *(description).*

Para explicar de una manera mas completa este proceso, añadimos 2 operaciones extra al servicio de ejemplo:

getInformationMonth: dado el mes y el año (*month* e *year* respectivamente) retorna qué autos se han alquilado en dicho período, y las fechas en las que fueron alquilados (CarsAndDates).

disableCarForRental: dado un auto (*Car*), el mismo se da de baja y retorna un valor booleano; verdadero en caso de poder darlo de baja, y falso en caso contrario (response).

El siguiente paso es modificar la sintaxis, pero sin cambiar la semántica. Para ello, los ingenieros utilizaron sinónimos y palabras relacionadas a las incluidas en identificadores como nombres de operaciones, campos de los tipos complejos, etc. Además, estaba permitido modificar las estructuras de datos, agregando atributos que resultasen necesarios o eliminando los que ellos consideraron prescindibles.

La Tabla 4.1 detalla un ejemplo de modificación de identificadores (Id) para este ejemplo.

Las operaciones modificadas quedarían con la siguiente estructura:

getReservationNumber (*reservationNumber*) : *(reservationResponse)*

getCarWage (airCondition, automatic) : (*chargeCost*, *currency*)

getAvailableCars (*initialDate*, *finalDate*) : (*listOfCar*).

getInformationMonth (*month, year*) : (arrayOfCarsAndDates).

disableCarForRental (*Car*) : (iCouldDeleteTheCar).

Luego el servicio modificado consiste de cinco operaciones, por lo tanto podemos desglosar el mismo en dos servicios de 3 y 2 operaciones respectivamente. Por ejemplo, el primer servicio puede incluir las operaciones getReservationNumber, getCarWage y getAvailableCars, mientras que el segundo, getInformationMonth y disableCarForRental.

Finalmente, para cada consulta generada se registró el *gold standard* (el servicio más relevante) de la misma, siendo este el Servicio Web a partir del cual fue generada cada una de ellas. Con esta metodología, podemos simular que la funcionalidad requerida por el desarrollador (consulta) se satisface con el servicio web que ofrece la misma funcionalidad, o sea del que dicha consulta fue generada.

Id original	Operación original	Rol	Modificado	Id modificado
getReservation	getReservation	Nombre de operación	Si	getReservationNumber
reservationCode	getReservation	Nombre parám. entrada	Si	*reservationNumber*
reservationReturn	getReservation	Nombre parám. salida	Si	*reservationResponse*
getCarFee	getCarFee	Nombre operación	Si	getCarWage
supplements	getCarFee	Nombre parám. entrada	Si	airCondition, automatic
chargeCost	getCarFee	Nombre parám. salida	No	-
currency	getCarFee	Nombre parám. salida	No	-
getAvailableCars	getAvailableCars	Nombre operación	No	-
initDate	getAvailableCars	Nombre parám. entrada	Si	*initialDate*
endDate	getAvailableCars	Nombre parám. entrada	Si	*finalDate*
cars	getAvailableCars	Nombre parám. salida	Si	*listOfCar*
getInformationMonth	getInformationMonth	Nombre operación	No	-
month	getInformationMonth	Nombre parám. entrada	No	-
year	getInformationMonth	Nombre parám. entrada	No	-
carsAndDates	getInformationMonth	Nombre parám. salida	Si	arrayOfCarsAndDates
disableCarForRental	disableCarForRental	Nombre operación	No	-
car	disableCarForRental	Nombre parám. entrada	No	-
response	disableCarForRental	Nombre parám. salida	Si	iCouldDeleteTheCar

Tabla 4.1: Comparativa entre identificadores

4.2. Ejecución experimental

Para la Fase de Ejecución Experimental (Figura 4.1), se utilizaron dos versiones del registro de descubrimiento de servicios EasySOC como base: la versión original basada en VSM [16] y la versión mejorada (Enhanced EasySOC) que aplica la técnica de expansión de consultas (*query expansion*) [1, 15]. EasySOC provee un soporte simple para descubrimiento de servicios basado en técnicas de matching de documentos, explotando solo aspectos sintáticos, tanto de consultas como de descripciones de servicios. Ambos registros fueron poblados con los 1146 servicios obtenidos de la plataforma Mashape (Paso 5). Las consultas sintácticas (que son las que se le hacen a los registros de descubrimiento) fueron generadas concatenando los nombres de las operaciones extraídas de las instanciaciones del metamodelo (particularmente de las consultas generadas por los expertos). Estas consultas sintácticas luego se convirtieron en la entrada para el registro de descubrimiento de servicios (Paso 6).

Luego, cada consulta generada por los expertos (documento WSDL 2.0 modificado) fue instanciada de acuerdo al metamodelo definido, utilizando el modulo Conversor presentado en la Sección 3.3.1. Por último, las consultas instanciadas fueron ejecutadas utilizando la Herramienta de Evaluación de Servicios Web (Paso 7). Finalmente, los resultados obtenidos, tanto para las consultas ejecutadas sobre los registros de descubrimiento de servicios (EasySOC y Enhanced EasySOC), como sobre la herramienta de Evaluación de Servicios Web, fueron evaluados utilizando las métricas que se presentan en la Sección 4.3.1 (Paso 8).

4.3. Resultados

Para este escenario experimental, se consultaron ambas versiones del registro EasySOC con las consultas sintácticas. Luego, se ejecutó la Herramienta de Evaluación de Servicios Web con

el metamodelo implementado en el mismo, utilizando las consultas generadas por los ingenieros. Finalmente se compararon los resultados considerando los *gold standard* para cada consulta, de acuerdo a la métrica Precisión-en-n Acumulada que se presenta a continuación.

4.3.1. Métrica Utilizada

El presente experimento fue diseñado para medir, en términos comparativos, el desempeño de la Herramienta de Evaluación de Servicios Web. Con este objetivo, se utilizó una métrica ampliamente reconocida del área de Recuperación de Información: Precisión-en-n la cual ha sido usada exitosamente en el contexto de la selección y el descubrimiento de Servicios Web [44, 31].

Dicha métrica calcula la precisión en diferentes puntos de la lista de candidatos retornados durante la selección. Formalmente, la métrica Precisión-en-n para una consulta individual se define como:

$$Precision - en - n = \frac{RetRel_n}{n} \tag{4.1}$$

Donde $RetRel_n$ es el número de servicios relevantes recuperados en las primeras n posiciones. Por ejemplo, si los 5 primeros servicios de la lista de candidatos son relevantes y los 5 siguientes son no relevantes, se tiene una precisión del 100 % en la quinta posición de la lista, pero una precisión del 50 % en la décima. En el contexto del presente experimento, la Precisión-en-n se mide para cada consulta con una ventana de $n \in [1, 10]$, luego se promedia para el total de las consultas y finalmente se obtiene el valor acumulado a medida que n tiende a 10, denominado Precisión-en-n Acumulada (*Cumulative Precisión-at-n*). Este valor permite saber, para cada posición i (con $i \in [1, 10]$) de la lista, el porcentaje de consultas para los cuales el servicio relevante aparece en las primeras n posiciones.

La Figura 4.2 detalla los valores de Precisión-en-n Acumulada correspondientes a ambas versiones de EasySOC y a la Herramienta de Evaluación de Servicios Web. Los resultados muestran que esta última incrementa el valor Precisión-en-n Acumulada entre 19 % y 34 % para las primeras tres posiciones de la lista (con n ∈[1,3]) con respecto a los resultados originales de los registros de descubrimiento.

La Tabla 4.2 resume los resultados para los registros EasySOC, Enhanced EasySOC y la Herramienta de Evaluación de Servicios Web, utilizando las primeras 4 posiciones (con $n \in [1, 4]$) de los resultados obtenidos.

En conclusión, los experimentos muestran que la utilización de la Herramienta de Evaluación de Servicios Web mejora la visibilidad de los servicios relevantes – estas mejoras se expresan en términos de ganancias en Precisión. Considerando que la selección de servicios candidatos se realiza luego de algún proceso de descubrimiento, incrementar la visibilidad de los candidatos más adecuados puede facilitar en gran medida el desarrollo de Aplicaciones Orientadas a Servicios.

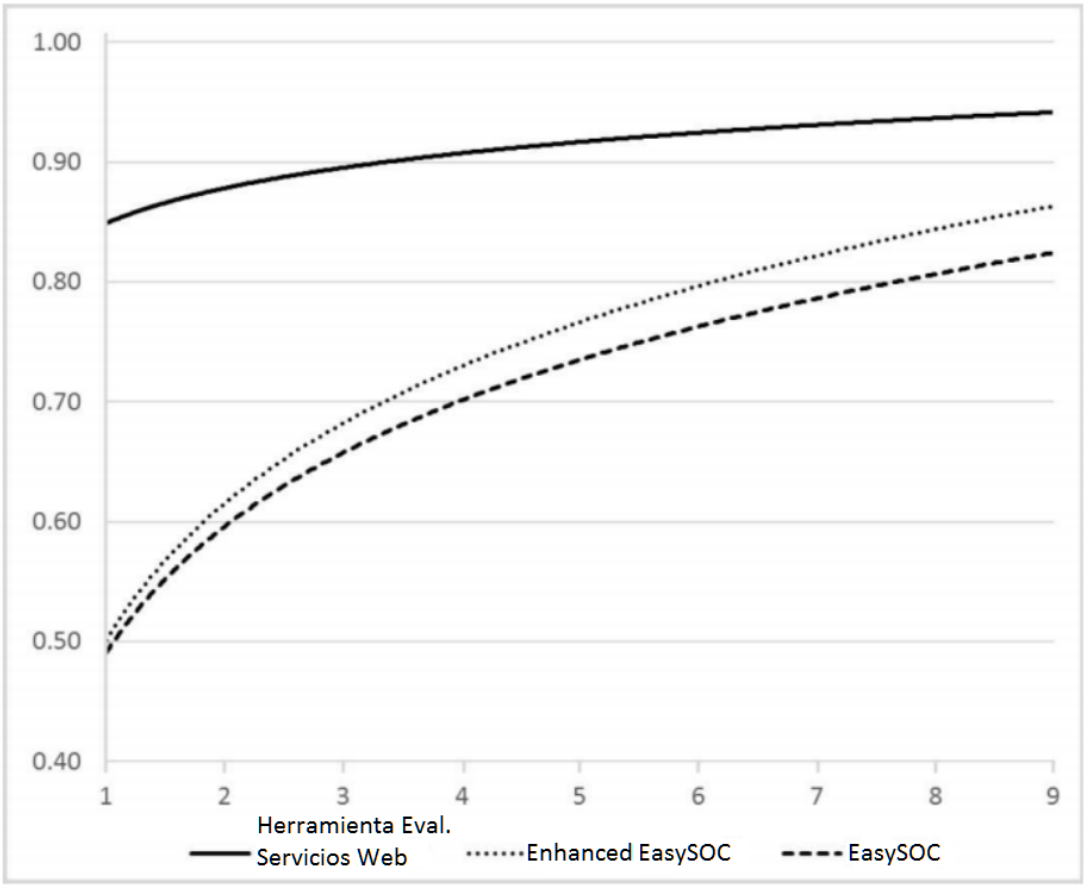

Figura 4.2: Comparativa de Presición-en-n Acumulada para diferentes herramientas evaluadas

	EasySOC	Enhanced EasySOC	Herramienta de Eval. de Serv. Web
Precisión en 1	0.49	0.50	0.85
Precisión en 2	0.58	0.61	0.87
Precisión en 3	0.65	0.67	0.89
Precisión en 4	0.70	0.73	0.91

Tabla 4.2: Precisión para los Registros de Descubrimiento y el Procedimiento de Análisis de Compatibilidad de Interfaces

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Introducción

En la presente tesis se profundizó acerca del Método de Selección de Servicios Web que tiene como objetivo asistir a los ingenieros de software en la construcción de Aplicaciones Orientadas a Servicios. En particular, se propusieron mejoras sobre el procedimiento de Análisis de Compatibilidad de Interfaces (validación contractual) completamente basado en la especificación funcional (descripta en WSDL) de los Servicios Web, que permitió extraer toda la información disponible en dicha especificación sin requerir de un marcado semántico adicional (se utilizaron los *tags* propios de un documento WSDL).

Actualmente, los servicios están siendo fuertemente considerados como la mejor elección para la integración transparente e inter-organizacional de activos de software. Sin embargo, los distintos problemas identificados evidencian la necesidad de mecanismos que soporten la selección de servicios. Los consumidores de servicios primero deben estudiar los servicios candidatos, sus funcionalidades, y las operaciones soportadas, para luego identificar la forma de invocación, la interconexión de operaciones, y los mapeos de mensajes. Sumado a esto, los documentos WSDL en general carecen de información significativa para soportar este proceso, debido a la proliferación de los Servicios Web [38] cuyos contratos se generan de manera automática, disminuyendo su expresividad y por ende la precisión de los registros de descubrimiento de servicios.

En este capítulo se analiza en la Sección 5.2 el alcance de los objetivos planteados en el Capítulo 1, en función del desarrollo del trabajo que se describe en esta tesis. Luego en la Sección 5.3 se identifican potenciales líneas futuras de acción que pueden complementar el trabajo que ha sido desarrollado.

5.2. Objetivos Alcanzados

En función de lo anteriormente expuesto, se definió el objetivo general de esta tesis, de la siguiente manera:

> *"Evaluación de Servicios Web, mediante un Metamodelo de Contratos de servicios, basado en el estándar SoaML"*

En este sentido, se propuso que la contribución materialice una extensión del proceso de selección y de su herramienta de soporte, enfocando en que las especificaciones de contratos de Servicios Web sean independientes de tecnología y que puedan abarcar servicios heterogéneos. Para ello se desarrollo un Metamodelo utilizando como base un conjunto de estándares para descripción de contratos/responsabilidades de Servicios Web. En el contexto de este objetivo general, destacan distintos objetivos específicos cuya consecución se detalla a continuación.

Para la consecución de este objetivo general, se propusieron los siguientes objetivos específicos:

Objetivo 1: Desarrollar un Metamodelo para especificación de Servicios Web basado en el estándar OMG SoaML, con su implementación en la plataforma Java Como se mencionó en el Capítulo 3, resultaba necesario contar con una especificación de contratos de Servicios Web, que sea independiente de cualquier tecnología de implementación. Por este motivo, utilizando como base los estándares SoaML, WSDL y WADL, se definió e implementó un Metamodelo para Descripción de Contratos de Servicios Web. El Metamodelo desarrollado ofrece mas información "semántica" correspondiente a un Servicio Web en comparación con las clases Java generadas en las anteriores versiones de la Herramienta de Evaluación de Servicios Web. El enfoque semántico-estructural contempla el análisis exhaustivo de tipos complejos, cuyo tratamiento no era trivial por el hecho del anidamiento de objetos. Además se planteó la estructura *ArrayType* que presenta un fuerte significado semántico y estructural, ya que indica que en su interior habrá una colección de objetos del mismo tipo. Se permitió el análisis de Tipos Complejos contra Tipos Simples puesto que se indaga dentro de sus estructuras y cantidad de parámetros de los mismos sin ser este último un condicionante taxativo a la hora de evaluar la similitud esperada.

Objetivo 2: Modificar la Herramienta de Evaluación de Servicios Web integrando el Metamodelo de Servicios Web Al contar con el Metamodelo desarrollado, resultó necesario que la evaluación de servicios candidatos se realice en función de instanciaciones del Metamodelo. Para ello, se diseño y desarrolló un componente de software que contiene al Metamodelo, el cual fue integrado a la Herramienta de Evaluación de Servicios Web. En la nueva versión de la plataforma, tanto los requerimientos funcionales por parte de desarrolladores de aplicaciones orientadas a servicios, como los propios servicios candidatos, son representados y comparados mediante instanciaciones del Metamodelo.

Para validar la Herramienta de Selección de Servicios Web con el Metamodelo integrado, se llevo a cabo una evaluación experimental. Para el experimento presentado se utilizó un conjunto de más de 1000 Servicios Web reales. Inicialmente, tres grupos de desarrolladores generaron manualmente consultas multi-operacionales. Luego, las consultas generadas se ejecutaron utilizando los registros de descubrimiento de servicios y la Herramienta de Evaluación de Servicios Web. Los resultados obtenidos siguiendo la métrica Precisión-en-n Acumulada mostraron una mejora en la precisión para los primeros resultados de la lista. De esta manera, el servicio candidato para una consulta tiene una alta probabilidad que resulte con un nivel aceptable en el subsiguiente procedimiento de Compatibilidad de Comportamiento (Capítulo 2 Sección 2.2).

Objetivo 3: Desarrollar un Componente Conversor de descripciones WSDL hacia instanciaciones del Metamodelo propuesto Se desarrolló un componente Conversor que recibe como entrada un documento WSDL (generalmente por medio de la URL) y devuelve como salida una instancia del Metamodelo. Contar con este conversor nos permitio trabajar con servicios SOAP de una manera transparente, sin necesidad de instanciar manualmente, de acuerdo al metamodelo desarrollado, cada descripcion del servicio.

El primer beneficio concreto de contar con el componente conversor lo obtuvimos a la hora de realizar la evaluación experimental, ya que para los más de 1000 servicios utilizados como dataset, fueron generadas sus correspondientes instancias del metamodelo para poder evaluar los mismos en un tiempo que se considera ínfimo en comparación a realizar la tarea manualmente.

5.3. Trabajo Futuro

En función de lo estudiado en esta tesis, se identificaron algunos aspectos para continuar la optimización del Método de Selección de Servicios Web. Estos aspectos se detallan a continuación:

- Un aspecto a tener en cuenta a la hora de seleccionar un Servicio Web candidato a integrar en una aplicación en desarrollo son las secuencias de ejecución de operaciones válidas que el mismo ofrece. Para esto, podría extenderse el procedimento de Análisis de Compatibilidad de Interfaces, incluyendo una evaluación sobre la coregrafia de los mismos. Aspecto que ahora es considerado en el nuevo metamodelo desarrollado, pero en el cual no se ha profundizado en su tratamiento, ya que sería necesario contar con un diagrama auxiliar (por ejemplo, diagrama de secuencia UML) para denotar la lógica del orden de llamados para el uso de una función específica.

- Desarrollar otros conversores, para otras descripciones de servicios como WADL [40], OpenAPI[1] o Swagger[2], le daría un valor agregado a la herramienta muy grande, ya que nos permitiría eventualmente trabajar con los servicios más utilizados en la industria.

- Sería de gran utilidad explotar la información que ofrece el metamodelo desarrollado considerando el procedimiento de Evaluación de Compatibilidad de Comportamiento de los servicios (Sección 2.2). Para esto es necesario definir nuevos esquemas y analizar cómo impacta esta información a la hora de la evaluación de comportamiento.

- Definir un proceso de generación de adaptadores (*Wrappers*) en función del Metamodelo. Cada *Wrapper* representa una posible correspondencia de las operaciones de la interfaz requerida I_R – que se denomina Interfaz a Consumir (I_C) en el Metamodelo – con respecto a las operaciones de la interfaz I_S – que se denomina Interfaz Provista (I_P) – de un servicio candidato. Esto permitiría comunicarnos con el Servicio Web candidato, de la forma más sencilla y automática posible.

[1] `https://www.openapis.org/`
[2] `https://swagger.io/`

Bibliografía

[1] Eugene Agichtein, Eric Brill, Susan Dumais, and Robert Ragno. Learning user interaction models for predicting web search result preferences. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10. ACM, 2006.

[2] R. Alexander and M. Blackburn. Component Assessment Using Specification-Based Analysis and Testing. Technical Report SPC-98095-CMC, Software Productivity Consortium, Herndon, Virginia, USA, Mayo 1999.

[3] Anabalon, D., Flores, A. y Garriga, M. Generación de Test Suite basado en Matching de Interfaces para Evaluación de Comportamiento de Servicios Web. Tesis de Grado de *Licenciatura en Ciencias de la Computación*, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina, Marzo 2015.

[4] Bichler, M. y Lin, K.J. Service-oriented computing. *Computer*, 39(3):99–101, 2006.

[5] Booth, D.; Haas, H.; McCabe, F.; Newcomer, E.; Champion, M.; Ferris, C. y Orchard, D. Web Services Architecture. W3C Working Group, February 2004. http://www.w3.org/TR/ws-arch/.

[6] T.A. Budd. *Mutation Analysis: Ideas, Example, Problems and Prospects*, chapter Computer Program Testing. North-Holand Publishing Company, 1981.

[7] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. Mining architectural responsibilities and components from textual specifications written in natural language. In SADIO, editor, *Proceedings of the XI Argentine Symposium on Software Engineering*, 2010.

[8] Castro S, Garriga M, Flores A. Extensión a la Evaluación Estructural y Semántica de Servicios Web Orientada a la Adaptabilidad. Tesis de Grado de *Licenciatura en Ciencias de la Computación*, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina, Abril 2016.

[9] Cechich, A. y Piattini, M. Early detection of COTS component functional suitability. *Information and Software Technology*, 49(2):108–121, 2007.

[10] Pat Pik Wah Chan and Michael R Lyu. Dynamic web service composition: A new approach in building reliable web service. In *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, pages 20–25. IEEE, 2008.

[11] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. *W3C recommendation*, 26:19, 2007.

[12] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, et al. Web services description language (wsdl) 1.1, 2001.

[13] OASIS Consortium. Uddi version 3.0.2. Technical Report uddi_v3, UDDI Spec Technical Committee Draft, Octubre 2004. http://www.uddi.org/pubs/uddi_v3.htm.

[14] Cors, I. y Flores, A. Evaluación de Comportamiento basado en Testing para Selección de Servicios Web. Tesis de Grado de *Licenciatura en Ciencias de la Computación*, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina, Mayo 2012.

[15] Marco Crasso, Cristian Mateos, Alejandro Zunino, and Marcelo Campo. Easysoc: Making web service outsourcing easier. *Information Sciences*, 259:452–473, 2014.

[16] Marco Crasso, Alejandro Zunino, and Marcelo Campo. Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, 71(2):144–164, 2008.

[17] Crasso, M.; Mateos, C.; Zunino, A. y Campo, M. EasySOC: Making Web Service Outsourcing Easier. *Information Sciences, Special Issue: Applications of Computational Intelligence & Machine Learning to Software Engineering*, 2010. in press.

[18] Andrea D'Ambrogio. A model-driven wsdl extension for describing the qos ofweb services. In *Web Services, 2006. ICWS'06. International Conference on*, pages 789–796. IEEE, 2006.

[19] De Renzis, A., Flores, A. y Garriga, M. Evaluación Semántico-Estructural de Servicios Web para Selección e Integración en Aplicaciones Orientadas a Servicios. Tesis de Grado de *Licenciatura en Ciencias de la Computación*, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina, Marzo 2013.

[20] Duftler, M. J.; Mukhi, N. K.; Slominski, A. y Weerawarana, S. Web services invocation framework (wsif). In *Workshop on Object Oriented Web Services, during OOPSLA'01*, Tampa, FL, USA, Octubre 2001. ACM Press.

[21] Erickson, J. y Siau, K. Web service, service-oriented computing, and service-oriented architecture: Separating hype from reality. *Journal of Database Management*, 19(3):42–54, 2008.

[22] Erl, T.; Kamarkar, A.; Walmsley, P.; Haas, H.; Yalcinalp, U.; Liu, C.; Orchard, D.; Tost, A. y Pasley, J. *Web Service Contract Design & Versioning for SOA*, volume 1. Prentice Hall, first edition, September 2008.

[23] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000.

[24] Flores, A. y Polo, M. Testing-based Process for Component Substitutability. *Software Testing, Verification and Reliability*, page 33, 2010. en prensa.

[25] Freedman, R. S. Testability of Software Components. *IEEE Transactions on Software Engineering*, 17(6):553–564, Junio 1991.

[26] Martin Garriga, Andres Flores, Alejandra Cechich, and Alejandro Zunino. Web services composition mechanisms: a review. *IETE Technical Review*, 32(5):376–383, 2015.

[27] Martin Garriga, Andres Flores, Cristian Mateos, Alejandro Zunino, and Alejandra Cechich. Service selection based on a practical interface assessment scheme. *International Journal of Web and Grid Services*, 9(4):369–393, 2013.

[28] Garriga, M., Flores, A. y Cechich, A. Evaluación Sintáctica de Compatibilidad de Interfaces para Selección de Servicios Web. Tesis de Grado de *Licenciatura en Ciencias de la Computación*, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina, Diciembre 2010.

[29] Gorton, I. *Essential Software Architecture*. Springer-Verlag, 2006.

[30] J. Gosling, B. Joy, G. Steele, and G. Bracha. *Java^{TM} Language Specification*. Sun Microsystems, Inc. Addison-Wesley, US, 3rd. edition, 2005. http://java.sun.com/docs/books/ jls/third_edition/html/j3TOC.html.

[31] Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology, 2005.

[32] Marc J Hadley. Web application description language (wadl), sun microsystems. *Inc., Mountain View, CA*, 2006.

[33] Huhns, M.N. y Singh, M.P. Service-oriented computing: key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[34] Jaffar-Ur Rehman, M.; Jabeen, F.; Bertolino, A. y Polini, A. Testing Software Components for Integration: a Survey of Issues and Techniques. *Software Testing, Verification and Reliability*, 17(2):95–133, Junio 2007.

[35] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[36] Markus Lanthaler and Christian Gütl. Towards a restful service ecosystem. In *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*, pages 209–214. IEEE, 2010.

[37] Massuthe, P.; Reisig, W. y Schmidt, K. An Operating Guideline Approach to the SOA. In *Annals Of Mathematics, Computing & Teleinformatics*, 2005.

[38] Cristian Mateos, Marco Crasso, Alejandro Zunino, and Jos? Luis Ordiales Coscia. Detecting wsdl bad practices in code–first web services. *International Journal of Web and Grid Services*, 7(4):357–387, 2011.

[39] Mateos, C.; Crasso, M.; Zunino, A. y Campo, M. Separation of Concerns in Service-Oriented Applications Based on Pervasive Design Patterns. In *25th ACM SAC'10*, pages 849–853, Sierre, Switzerland, Marzo 2010. ACM Press.

[40] Nenad Medvidovic and Richard N Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on software engineering*, 26(1):70–93, 2000.

[41] Srini Narayanan and Sheila A McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th international conference on World Wide Web*, pages 77–88. ACM, 2002.

[42] Papazoglou, M.P. y Heuvel, W.J. Service oriented architectures: Approaches, technologies and research issues. *The International Journal on Very Large Data Bases*, 16(389-4153), 2007.

[43] J. Pasley. Avoid XML Schema Wildcards For Web Service Interfaces. *IEEE Internet Computing*, 10(3):72–79, 2006.

[44] James Pasley. Avoid xml schema wildcards for web service interfaces. *IEEE Internet Computing*, 10(3):72–79, 2006.

[45] Cesare Pautasso. Restful web service composition with bpel for rest. *Data & Knowledge Engineering*, 68(9):851–866, 2009.

[46] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.

[47] J. M. Rodriguez, M. Crasso, C. Mateos, A. Zunino, and M. Campo. The easysoc project: A rich catalog of best practices for developing web service applications. In *2010 XXIX International Conference of the Chilean Computer Science Society*, pages 33–42, Nov 2010.

[48] Juan Manuel Rodriguez, Marco Crasso, Cristian Mateos, and Alejandro Zunino. Best practices for describing, consuming, and discovering web services: a comprehensive toolset. *Software: Practice and Experience*, 43(6):613–639, 2013.

[49] Marcela Ruiz, David Ameller, Sergio España, Pere Botella, Xavier Franch, and Oscar Pastor. Ingeniería de requisitos orientada a servicios: características, retos y un marco metodológico. *Jornadas de Ciencia e Ingeniería de Servicios (JCIS)*, 2011.

[50] Sivashanmugam, K.; Verma, K.; Sheth, A.P. y Miller, J. A. Adding semantics to Web Services standards. In *International Conference on Web Services*, pages 395–401, Las Vegas, NV, USA, Septiembre 2003. CSREA Press.

[51] Sprott, D. y Wilkes. L. Understanding Service-Oriented Architecture. *The Architecture Journal. MSDN Library. Microsoft Corporation*, 1:13, January 2004. http://msdn.microsoft.com/en-us/library/aa480021.aspx.

[52] Stevens, M.; McGovern, J.; Tyagi, S. y Matthew, S. *Java Web Service Architecture*. Morgan Kaufmann Publishers, 2003.

[53] Eleni Stroulia and Yiqiao Wang. Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems*, 14(04):407–437, 2005.

[54] Stephen L Vargo and Robert F Lusch. Evolving to a new dominant logic for marketing. *Journal of marketing*, 68(1):1–17, 2004.

[55] Y. Wang and E. Stroulia. Flexible interface matching for web-service discovery. In 4^{th} *International Conference on Web Information Systems Engineering (WISE)*, Roma, Italia, Diciembre 2003.

[56] Y. Wang and E. Stroulia. Semantic structure matching for assessing web-service similarity. In *First International Conference on Service Oriented Computing*, Trento, Italia, Diciembre 2003.

[57] Weerawarana, S.; Curbera, F.; Leymann, F.; Storey, T. y Ferguson, D. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.

[58] Peter Willett. The porter stemming algorithm: then and now. *Program: electronic library and information systems*, 40(3):219–223, 2006.

[59] A. M. Zaremski and J. Wing. Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology*, 6(4), Octubre 1997.