Plan de Trabajo Final

Ingeniería de Sistemas

Facultad de Ciencias Exactas - UNICEN

Una evaluación empírica de técnicas para aumentar la descubribilidad de Servicios Web Code-first

Tandil, 31 de marzo de 2014

Tema: Computación Orientada a Servicios

Alumno/s: Silvina Daiana Wesner, Diego Martín Piu

Director: Dr. Juan Manuel Rodriguez

Co-Director: Dr. Cristian Mateos Diaz

1 Introducción

La creciente complejidad de los sistemas de software hace de la reutilización de componentes una de las herramientas más valiosas para el desarrollo de nuevos sistemas de software [15]. Una de las opciones existentes para la reutilización de componentes es brindada por el paradigma SOC (Service Oriented Computing). Bajo este paradigma la funcionalidad es expuesta en componentes llamados servicios. Cada servicio cuenta con una interfaz bien definida que permite, a quienes deseen utilizarlos, invocarlos sin necesidad de conocer la implementación del servicio, ni la plataforma en la cual se ejecuta.

El paradigma SOC propone tres roles básicos [6]: el proveedor de servicios, el consumidor de servicios y el sistema de registro de servicios. Los proveedores desarrollan los servicios definiendo tanto su lógica como su interfaz y publican información acerca de éstos en un registro. Por otro lado, los consumidores que quieren reutilizar funcionalidad de terceros, buscan los servicios que necesitan en un registro de servicios y luego los invocan remotamente. Finalmente, el objetivo del registro de servicios es facilitar la vinculación entre los proveedores y los consumidores.

Generalmente, los servicios se implementan utilizando protocolos de Internet ampliamente soportados por distintas plataformas, tales como SOAP o HTTP. Cuando se utilizan estos protocolos, los servicios se conocen como *Servicios Web*. En los *Servicios Web*, las descripciones se materializan mediante la utilización del WSDL (Web Services Description Language), que es un lenguaje basado en XML. WSDL permite representar la funcionalidad de un *Servicio Web* de forma abstracta, mediante un conjunto de operaciones que contienen mensajes de entrada y salida, cada uno con un tipo de dato definido de acuerdo al estándar XSD (XML Schema). XSD es un estándar de tipado que permite definir tipos de datos que abarcan los que pueden definirse en una amplia gama de lenguajes de programación.

WSDL[1] permite a los proveedores especificar la funcionalidad de sus servicios como un conjunto de operaciones abstractas con entradas y salidas, asociando información que permita a los consumidores invocar las operaciones ofrecidas independientemente de la plataforma en la cual fueron implementadas, es decir, abstrae al consumidor de servicios de la implementación de

[1] Especificación de WSDL : `http://www.w3.org/TR/wsdl`

los mismos.

Adicionalmente, este lenguaje permite definir toda la información técnica necesaria para invocar el servicio. Esta información incluye dónde está el servicio, qué protocolos de comunicación soporta y como se sugiere más arriba, los datos de entrada/salida que se utilizan durante una invocación.

El descubrimiento de *Servicios Web* es una de las preocupaciones principales en el desarrollo de sistemas SOC ya que los proveedores quieren que los consumidores usen sus servicios y los consumidores están interesados en encontrar los servicios adecuados para construir sus sistemas. Por lo tanto, el descubrimiento es un objetivo central en el paradigma SOC ya que es la clave para el éxito del mismo [13].

A pesar de la importancia de la información contenida en los documentos WSDL, distintos trabajos [4, 15, 2, 8, 3, 7] han mostrado que estos documentos en la vida real tienden a ser de baja calidad, lo que atenta contra el descubrimiento de los servicios. Incluso, existen casos de estudio [9, 12] donde se muestra que la calidad de los documentos WSDL es un factor clave en el éxito de un sistema orientado a servicios. Estas falencias generalmente se ven reflejadas en información textual escasa, ambigua e incompleta [5, 7, 3] dentro de los documentos WSDL. Los diferentes problemas encontrados generalmente en los documentos fueron categorizados en anti-patrones en [4, 15]. Adicionalmente, estos trabajos proponen ciertas estrategias para reestructurar los documentos WSDL de manera de remover los anti-patrones.

A pesar de la existencia de este catálogo de anti-patrones y estrategias de refactorización, el problema persiste y no se encuentra resuelto aún en la práctica. Uno de los motivos es que las estrategias de refactorización requieren que el desarrollador tenga control directo sobre los documentos WSDL que describen su sistema [14]. Sin embargo, muchas veces los documentos WSDL son generados automáticamente a partir del código fuente del *Servicio Web*. A pesar de no requerir desarrolladores especializados y tener un tiempo de desarrollo menor, la generación automática de las descripciones puede resultar en documentos WSDL de mala calidad [9, 12].

Debido a esto, algunos autores [10] han apuntado su investigación a determinar si existe alguna relación entre métricas de calidad [1] de software tradicionales y la calidad de los documentos WSDL generados automáticamente. Sin embargo, estos estudios no determinan causas directas sino correlaciones entre ciertas prácticas de desarrollo y problemas de calidad en los documentos WSDL. Finalmente, en un trabajo final de carrera de grado se presentó una herramienta [5] que apunta a detectar y asistir a los desarrolladores a corregir las causas de los problemas de calidad en los documentos WSDL.

En este contexto, el objetivo de este trabajo es evaluar en profundidad esta herramienta [5] y compararlo con el enfoque basado en métricas de calidad, para determinar puntos fuertes y débiles de ambos enfoques. A continuación, en la Sección 2, se discuten las motivaciones de este trabajo. En la Sección 3 se plantean y discuten los objetivos. Finalmente, la Sección 4 presenta el cronograma de trabajo.

2 Motivación

El desarrollo de software orientado a servicios requiere proveedores que describan sus servicios y los publiquen en los registros correspondientes, donde los consumidores puedan descubrirlos. En este sentido, el primer paso para incorporar un servicio externo a una aplicación es su descubrimiento, esto es la búsqueda de servicios que cumplan con las características requeridas. Por esta razón, los servicios concisos y bien descriptos son fundamentales en el desarrollo de software con el paradigma SOC [13].

En el desarrollo de *Servicios Web* existen dos enfoques principales, llamados contract-first y code-first. En este último caso, el desarrollo del servicio comienza a partir de su código fuente. Luego, los desarrolladores pueden escribir un servicio sin tener conocimiento de los WSDL ya que su descripción es generada automáticamente por una herramienta dependiente del lenguaje. El enfoque code-first es simple, consume menos tiempo y es fácil de usar para los usuarios que no están familiarizados con los estándares de *Servicios Web* ya que los WSDL son difíciles de leer y usar para un principiante. Sin embargo este enfoque proporciona menos control sobre el contrato del servicio dado que el WSDL cambia cada vez que el código fuente es modificado. Además, debido a las falencias de las herramientas de generación automática, existe una gran posibilidad de que la calidad de la descripción de los servicios sea baja.

Por otro lado, en el enfoque contract-first, el desarrollo del servicio comienza a partir de un WSDL, que se convierte en un contrato entre el proveedor del servicio y el consumidor. Bajo este enfoque, un WSDL es definido manualmente por los desarrolladores usando estándares de *Servicios Web* y el servicio es implementando de acuerdo a ese contrato. Si bien es cierto que el enfoque contract-first es más complejo en comparación con el code-first, a largo plazo se obtienen descripciones de *Servicios Web* más concisas y más fáciles de leer debido a que los desarrolladores del servicio se enfrentan directamente con los problemas en los documentos [14, 9, 12]. A pesar que académicamente existe el consenso de que el enfoque contract-first resulta en

descripciones WSDL más descriptivas y auto-contenidas, la industria se basa principalmente en el enfoque code-first dado que es más rápido y consume menos recursos [11, 10].

En el enfoque code-first, los anti-patrones catalogados pueden ser eliminados en la etapa de desarrollo, en base a un conjunto clásico de métricas orientadas a objetos y un acabado entendimiento de cómo funcionan las herramientas de generación de WSDL. De esta manera, los servicios resultantes no sólo son más fáciles de entender y reusar por otros desarrolladores sino que también su descubrimiento mejora considerablemente [11]. Sin embargo, esta metodología sigue dependiendo de las herramientas de generación. Por ejemplo, si la herramienta de generación no considera los comentarios en el código fuente, los documentos WSDL generados no tendrán comentarios independientemente de cuán bien documentado se encuentre el código fuente [10]. Adicionalmente, como el enfoque solo detecta correlaciones, existe la posibilidad de que en algunos casos la modificación de los valores de una métrica no afecte el número de anti-patrones en los documentos WSDL.

Contrariamente a [10], la herramienta presentada en [5] también incluye un generador de documentos WSDL que intenta tomar en cuenta la mayor cantidad de información presente en el código fuente. La idea detrás de esto es que la documentación y los nombres de los métodos y parámetros se vean reflejados en los documentos WSDL. Idealmente, si el código fuente se encuentra bien documentado, el documento WSDL generado a partir de ese código debería estar bien documentado también.

3 Objetivos

El objetivo de este trabajo final es medir la mejora en la calidad, en términos de descubribilidad, de los documentos WSDL generados con el enfoque *code-first* cuando se aplican las técnicas propuestas en [5] y [10]. Básicamente se espera poder comparar ambas metodologías y determinar para qué anti--patrones funciona mejor una o la otra.

Para realizar la comparación se tomará un conjunto de implementaciones de *Servicios Web* escritas en Java, con éstas se generarán nuevos conjuntos de implementaciones aplicando las refactorizaciones propuestas en [5] y [10]. Utilizando cada uno de los conjuntos de implementaciones, es decir el original y el refactorizado, se generarán diferentes conjuntos de documentos WSDL utilizando tanto herramientas estándar, por ejemplo Axis[2] o EasyWSDL[3], como

[2] Axis: `https://axis.apache.org/axis/`
[3] EasyWSDL: `http://easywsdl.ow2.org/`

la herramienta presentada en [5]. Posteriormente, se compararán los distintos conjuntos de documentos WSDL utilizando como parámetro cuántos y qué tipo de anti-patrones afectan a los documentos WSDL. Es decir, se aplicará un análisis similar al presentado en [10, 11].

Se espera que a partir de estos resultados se pueda determinar para qué tipo de anti-patrones funciona mejor cada enfoque. Adicionalmente, se espera que los experimentos identifiquen puntos débiles en el enfoque de [5] que puedan ser mejorados en trabajos futuros. En particular se espera tener un análisis de como este enfoque se comporta a la hora de evaluar la calidad de los nombres y documentación presentes en el código fuente.

4 Cronograma de actividades

A continuación se presenta el cronograma de las actividades propuestas que conformarán el desarrollo de este trabajo, así como su duración estimada. El tiempo total de trabajo se fijó en aproximadamente nueve meses.

- Relevamiento y lectura de trabajos relacionados. 1 mes.
- Análisis y adaptación de las herramientas a utilizar. 1 mes.
- Análisis manual de las implementaciones para detectar las potenciales causas de los anti-patrones y comparación de los resultados obtenidos respecto de los arrojados por el análisis automático. 1 mes.
- Refactorización de las implementaciones para eliminar las potenciales causas de los anti-patrones encontradas. Se generarán diferentes versiones del mismo, en donde cada una de ellas carece de uno de los anti-patrones. 2 meses.
- Comparación de los documentos WSDL generados (refactorizado vs. original) para obtener las métricas de la mejora en la descubribilidad y análisis de esos resultados. 2 semanas.
- Documentación y elaboración del informe. 3 meses.
- Preparación de la defensa. 2 semanas.

Bibliografía

[1] Victor R Basili, Lionel C. Briand, and Walcélio L Melo. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22(10):751–761, 1996.

[2] Jack Beaton, Sae Young Jeong, Yingyu Xie, Jeffrey Stylos, and Brad A Myers. Usability challenges for enterprise service-oriented architecture apis. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 193–196. IEEE, 2008.

[3] M Brian Blake and Michael F Nowlan. Taming web services from the wild. *Internet Computing, IEEE*, 12(5):62–69, 2008.

[4] Marco Crasso, Juan Manuel Rodriguez, Alejandro Zunino, and Marcelo Campo. Revising wsdl documents: Why and how. *IEEE Internet Computing*, 14(5), 2010.

[5] Alejandro Matías Durante. Una herramienta para generar descripciones de servicios web evitando malas prácticas. 2012.

[6] Thomas Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.

[7] Jianchun Fan and Subbarao Kambhampati. A snapshot of public web services. *ACM SIGMOD Record*, 34(1):24–32, 2005.

[8] John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos, and Athanasios Tsakalidis. Contemporary web service discovery mechanisms. *J. Web Eng.*, 5(3):265–290, 2006.

[9] Cristian Mateos, Marco Crasso, Juan M Rodriguez, Alejandro Zunino, and Marcelo Campo. Measuring the impact of the approach to migration in the quality of web service interfaces. *Enterprise Information Systems*, (ahead-of-print):1–28, 2012.

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

[11] José Luis Ordiales. An approach to early code-first web services quality improvement. 2013.

[12] J. Rodriguez, M. Crasso, C. Mateos, A. Zunino, and M. Campo. Bottom-up and top-down cobol system migration to web services: An experience report. *Internet Computing, IEEE*, 17(2):44–51, March-April 2013.

[13] Juan Manuel Rodriguez. Malas prácticas en el desarrollo servicios web que dificultan su descubrimiento.

[14] Juan Manuel Rodriguez, Marco Crasso, and Alejandro Zunino. An approach for web service discoverability anti-patterns detection. *Journal of Web Engineering*, 12(1–2):131–158, 2013.

[15] Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo. Improving web service descriptions for effective service discovery. *Science of Computer Programming*, 75(11):1001–1021, 2010.