A Visualization Tool to Detect Refactoring Opportunities in SOA Applications

Guillermo Rodriguez, Alfredo Teyseyre, Álvaro Soria, Luis Berdun
ISISTAN-CONICET
UNICEN
Tandil, Bs. As., Argentina
{guillermo.rodriguez, alfredo.teyseyre, alvaro.soria, luis.berdun}@isistan.unicen.edu.ar

***Abstract*— Service-oriented computing (SOC) has been widely used by software industry for building distributed software applications that can be run in heterogeneous environments. It has also been required that these applications should be both high-quality and adaptable to market changes. However, a major problem in this type of applications is its growth; as the size and complexity of applications increase, the probability of duplicity of code increases. This problem could have a negative impact on quality attributes, such as performance, maintenance and evolution, among others. This paper presents a web tool called VizSOC to assist software developers in detecting refactoring opportunities in service-oriented applications. The tool receives WSDL (Web Service Description Language) documents, detects anti-patterns and suggests how to resolve them, and delivers a list of refactoring suggestions to start working on the refactoring process. To visualize the results in an orderly and comprehensible way, we use the Hierarchical Edge Bundles (HEB) visualization technique. The experimentation of the tool has been supported using two real-life case-studies, where we measured the amount of anti-patterns detected and the performance of clustering algorithms by using internal validity criteria. The results indicate that VizSOC is an effective aid to detect refactoring opportunities, and also allows developers to reduce effort along the detection process.**

***Index Terms*—Service-oriented applications, Web services, Unsupervised machine learning, Web service description language, Service understability, Software visualization.**

I. Introducción

Actualmente, en la industria del software es muy común encontrar sistemas implementados con el paradigma de Computación Orientada a Servicios (SOC), los cuales se construyen en base al estilo arquitectónico denominado Service-Oriented Architecture (SOA). Las soluciones SOA han sido creadas para satisfacer los objetivos de negocio, como facilidad y flexibilidad de integración con sistemas legados, alineación directa a los procesos de negocio reduciendo costos de implementación, innovación de servicios a clientes y una adaptación ágil ante cambios incluyendo reacción temprana ante la competitividad, entre otros [1]. Estas son algunas de las razones por las cuales los sistemas SOA son tan comunes a la hora de producir soluciones de software.

978-1-5386-3057-0/17/31,00 ©2017*IEEE*

En un contexto SOA, existen proveedores que ofrecen sus servicios como procedimientos remotos y los consumidores de dichos servicios que los solicitan llamándolos a través de la Web. Una manera de definir estas relaciones es a través de WSDL - Web Service Description Language [2]. WSDL es entonces, un lenguaje de descripción de interfaz de servicios Web basado en XML (Extensible Markup Language), que se utiliza para describir la funcionalidad ofrecida por un servicio Web. Una descripción WSDL de un servicio Web provee una descripción de cómo el servicio se puede llamar, qué parámetros espera y qué estructura de datos devuelve como respuesta, entre otras características. Es decir que, a través de una descripción en WSDL, es posible definir las diferentes relaciones entre servicios Web, así como también, la jerarquía de los mismos. Sin embargo, cuando una aplicación SOA posee conjuntos de datos complejos y difíciles de agrupar y/o asociar requiere generalmente una refactorización.

Las aplicaciones SOA, como cualquier otra aplicación de software, requieren mantenimiento por parte de los desarrolladores lo cual contribuye fuertemente a la evolución del sistema. Parte de este mantenimiento consiste en el refactoring (o refactorización) de código. La refactorización entonces es el proceso de reestructurar código existente de un sistema sin cambiar su comportamiento para el exterior, es decir, sin modificar la funcionalidad. La refactorización mejora los atributos de calidad no funcionales de software tales como la legibilidad del código, decrementa la complejidad y esto mejora su mantenibilidad [3]. Los documentos WSDL suelen ser extensos y requieren de una lectura muy meticulosa, ya que se basan en XML, lo cual puede ser arduo, tedioso y conducir a no distinguir posibles oportunidades de refactoring o, más preocupante aún, no detectar errores en la arquitectura y código del sistema producto de haber utilizado anti-patrones [4].

En este contexto, existen técnicas de clustering y visualización que apuntan a solucionar o aliviar la problemática mencionada anteriormente, y de esta manera, facilitar la tarea de los desarrolladores a la hora de detectar oportunidades para refactoring. Los algoritmos de clustering identifican de forma automática agrupaciones o clusters de

elementos de acuerdo a una medida de similitud entre ellos [5]. En cuanto a las estrategias de visualización de WSDL, estas no están preparadas para manejar la complejidad y la cantidad de relaciones que se dan entre los artefactos descritos que demandan visualizaciones efectivas y claras que faciliten la comprension de los desarrolladores y permitan que estos llevan a cabo sus tareas eficientemente [6]. Habiendo observado y analizado en particular servicios Web descritos en WSDL, se hizo imperante la necesidad de poder visualizar el gran caudal de información que se puede obtener o deducir de un conjunto de documentos WSDL.

En este trabajo se presenta *VizSOC*, una herramienta web basada en aprendizaje no supervisado para asistir a los desarrolladores de software en la detección de oportunidades de refactoring en aplicaciones SOA. En un trabajo previo estudiamos la factibilidad de aplicar algoritmos de clustering no supervisado sobre conjuntos de documentos WSDL [7], y en el presente trabajo extendemos la investigación mediante la construcción de una herramienta web que soporte el proceso de detección de oportunidades de refactorización. *VizSOC* analiza los documentos especificados en WSDL pertenicientes a una aplicación SOA, detecta anti-patrones presentes en dichos documentos, sugiere cómo solucionar dichos anti-patornes y aplica técnicas de clustering (K-Means, Expectation Maximization y algoritmos jerárquicos) para detectar oportunidades de refactorización, por ejemplo, reduciendo funcionalidad duplicada. La herramienta explota las ventajas de la técnica de visualización Hierarchical Edge Bundle (HEB) para facilitar a los desarrolladores la comprensión de los clusters generados por *VizSOC* [8]. De esta menena, el desarrollador con esa lista de sugerencias procede a la implementación de dichas oportunidades, esperando lograr un producto final más robusto y mejor diseñado.

Para evaluar *VizSOC* se utilizaron dos casos de estudios de aplicaciones reales, de la cuales se tuvo acceso al conjunto de documentos WSDL que describen la funcionalidad de dichas aplicaciones SOA. En primer lugar, mediante métricas de validación interna de clusters medimos la eficiencia de los diferentes algoritmos utilizados. En segundo lugar, se midió la detección de antipatrones, junto con las soluciones sugeridas. Los resultados obtenidos aseguran que *VizSOC* es una herramienta viable y útil para asistir a los desarrolladores de software en la detección de oportunidades de refactorización de aplicaciones SOA.

El resto del trabajo se organiza de la siguiente manera. La Sección II describe los trabajos relacionados con el enfoque propuesto. La Sección III presenta *VizSOC* y describe cada una de las funcionalidades provistas. La Sección IV detalla el diseño de los experimentos para evaluar *VizSOC*. La Sección V presenta los resultados obtenidos en la experimentación. Finalmente, la Sección VI analiza las conclusiones del trabajo y propone futuras líneas de trabajo.

II. Trabajos Relacionados

En los últimos años se ha prestado considerable atención en mejorar el entendimiento y descubrimiento de documentos WSDL. Por ejemplo, Rodriguez et al. han presentado un catálogo de anti-patrones de documentos WSDL para mejorar la descripción de los servicios web [9]. Mateos et al. han introducido un plug-in de Eclipse para asistir a los desarrolladores de software en obtener documentos WSDL desde el código, con la menor cantidad posible de anti-patrones [10]. Crasso et al. han descrito los errores más comunes encontrados en documentos reales de WSDL, han explicado cómo esos errores impactan en el descubrimiento de servicios y han facilitado guías para poder resolver esos errores [11]. Webster et al. han presentado un enfoque para intentar resolver el conflicto que se concreta cuando el proveedor de servicios necesita evolucionar su interfaz para responder a los nuevos requerimientos, sin que estos cambios afecten a los consumidores de servicios [12]. No obstante, estos trabajos no han considerado el hecho de que cuando los sistemas crecen, los documentos WSDL pueden evidenciar problemas de diseño, como por ejemplo, operaciones que deberían estar en el mismo documento pero están en documentos separados, o transacciones con diferentes nombre pero haciendo lo mismo, en otros problemas. En definitiva, estos enfoques se han concentrado en estudiar los documentos WSDL en forma aislada en lugar de analizarlos en conjunto en el contexto de la aplicación SOA.

En este contexto, uno de los principales problemas abordados por los desarrolladores de software al analizar un conjunto entero de documentos de WSDL es la duplicidad de código. Hoy en día, aunque es posible refactorizar manualmente, hay un amplio rango de técnicas semi-automáticas para dar soporte a este proceso; siendo la más popular clustering no supervisado. Los algoritmos de clustering apuntan a reducir la cantidad de datos agrupando datos similares en el mismo cluster [13]. Particularmente, las aplicaciones orientadas a servicio exhiben sus interfaces mediante documentos WSDL, cada uno conteniendo las operaciones que describen la funcionalidad del servicio. Estas operaciones podrían estar duplicadas entre los documentos WSDL, complejizando así la mentenibilidad de las aplicaciones. Para evitar la duplicación de código sería necesario ubicar operaciones similares en un mismo cluster y ubicar operaciones disímiles en clusters diferentes.

Numerosos trabajos han explorado técnicas de clusterización para reducir esfuerzos en el desarrollo de aplicaciones orientadas a servicios. Por ejemplo, Sabou et al. propusieron aplicar Cluster Map como una técnica de visualización para soportar análisis, comparación y búsqueda de servicios Web [14]. Kuhn et al. usaron clustering jerárquico semántico para agrupar artefactos de software que usan vocabulario similar con el propósito de mejorar la mantenibilidad del software. El clustering semántico está basado en LSI (Latent Semantic Indexing) que ubica tópicos ligüísticos en un conjunto de documentos; esta técnica es aplicada para calcular la similitud ligüística

entre artefactos de software, como por ejemplo, paquetes, clases y métodos, entre otros [15]. Liu et al. crearon un motor de búsqueda que reduce el espacio de búsqueda para el descubrimiento de servicios usando el algoritmo de hormigas "tree-traversing" [16]. Sobre esta línea de trabajo, Ma et al. apuntaron a eliminar servicios irrelevantes con respecto a la consulta de búsqueda durante el proceso de descubrimiento [17]; el enfoque propuesto utiliza K-Means y luego un enfoque probabilístico de LSA (Latent Semantic Analysis) para capturar relaciones semánticas ocultas detrás de las palabras propias de una consulta, de modo que la correspondencia con un servicio sea alcanzado a un nivel conceptual. Similar a Ma et al., Elgazzar et al. propusieron clusterizar servicios Web mediante una función de similitud obtenida antes de recuperar los servicios Web relevantenes para una consulta en el contexto de los motores de búsqueda. El enfoque aplica K-Means y la Distancia de Google Normalizada para medir la distancia entre palabras sin considerar características de las mismas [18]; sin embargo, estos enfoque no han considerado la duplicidad de código. A diferencia de nuestro trabajo, Fokaefs et al. propusieron un enfoque que usa clustering de documentos WSDL para analizar la evolución de los artefactos de software a lo largo del ciclo de vida de aplicaciones orientadas a servicios. En este caso, el enfoque primero reconoce cambios en los documentos WSDL; luego, analiza cambios que ocurren a medida que dichos documentos evolucionan. Para resolver la primera parte, el enfoque usa la distancia "Zhang-Shasha's tree-edit", mientras que para la segunda parte se usa clustering aglomerativo jerárquico [19]. En esta línea de trabajo, Dong et al. propusieron un enfoque que usa también clustering aglomerativo jerárquico usando el motor de búsqueda de servicios Web denominado Woogle. Mediante este enfoque, se supera la búsqueda tradicional basada en palabras claves explotando la estructura subyacente y semántica de los servicios Web [20]. Kumara et al. propusieron un enfoque para asistir a los desarrolladores en la búsqueda de servicios Web mediante la visualización de datos inherentes a los servicios sobre una superfice esférica. El enfoque apunta a agrupar servicios Web mediante la aplicación de un algoritmo de clusterización espacial denominado Associated Keyword Space [21]. En este contexto, la similitud es calculada mediante aprendizaje de ontologías y recuperación de información. La mayoría de los trabajos previamente mencionados han actualizado las técnicas de clusterización para optimizar los motores de búsqueda de servicios; en cambio, nuestro trabajo propone clusterizar operaciones similares de servicios Web mediante la comparacion de interfaces de servicios, como un primer paso hacia la detección de oportunidades de refactorización en aplicaciones orientadas a servicio.

III. VizSOC: La herramienta web

El objetivo de este trabajo es presentar una herramienta web para asistir a los desarrolladores de software en la detección de oportunidades de refactoring en aplicaciones SOA, analizando documentos WSDL con el fin de analizar las posibilidades de mejoras y presentarlas de manera visual que, como se mencionó anteriormente, facilite su comprensión. La Fig. 1 describe el esquema general de la herramienta denominada *VizSOC*. En primer lugar, *VizSOC* recibe como entrada los diferentes documentos WSDL (tanto del sistema completo o de un subconjunto de servicios Web). Para ello, el usuario a través de una interfaz web puede cargar los documentos.

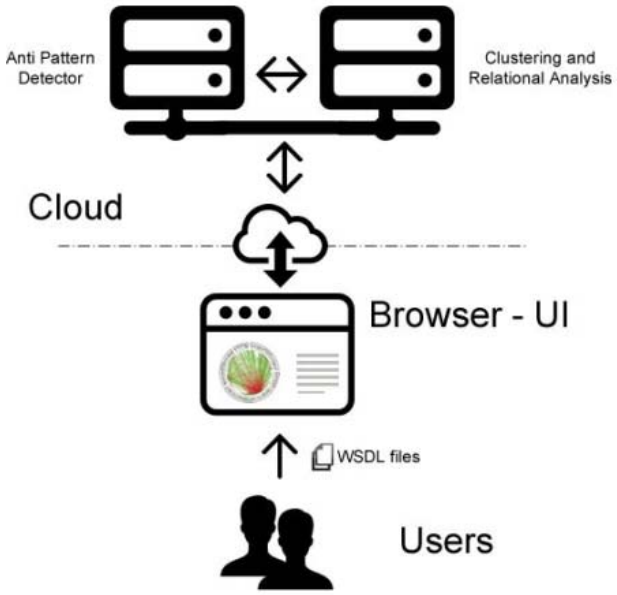

Figura 1. Esquema general de *VizSOC*

En segundo lugar, la herramienta permite hacer un análisis de detección de anti-patrones presentes en los documentos WSDL, ofreciendo también posibles soluciones. El catálogo de anti-patrones utilizado se compone de: *inapropriate or lack of documentarion, ambiguous names, redundant data models, whatever types, redudant port-types, low cohesive operation in the same port-type, empty messages, enclosed data models, and undercover fault messages with standard messages* [4]. En tercer lugar, el usuario puede seleccionar la opción de detectar oportunidades de refactorización. En este caso, *VizSOC* aplica diferentes técnicas de clustering a las aplicadas en un trabajo previo [7], y provee de distintos gráficos mostrando los datos que se sustrajeron de las relaciones y jerarquías de los diferentes servicios definidos en los documentos WSDL. En este contexto, el componente «Browser - UI» es el encargado de presentar las relaciones entre los diferentes servicios Web a través de una estrategia de visualización basada en relaciones entre las operaciones de estos servicios y la abstracción jerárquica de las mismos, permitiendo al usuario la posibilidad de generar sus propios gráficos de análisis según la vista más apropiada.

En la Fig. 2 se puede observar a la herramienta en acción. Los documentos WSDL son cargados desde la interfaz provista sobre el lado izquierdo de la herramienta.

La función «Info» permite visualizar las propiedades del servicio Web descritas en el documento WSDL (Fig. 4), mientras que la función «Stats» permite analizar los anti-patrones en los documentos WSDL seleccionados. Por ejemplo, la Fig. 3 muestra que para el documento «DatosdeDictamenes.wsdl» la herramienta detectó el anti-patrón «Redundant port-types», proveyendo la descripción del mismo junto con su posible solución.

Por un lado, en la parte superior de la herramienta, se encuentran los algoritmos de clustering que implementa *VizSOC*: algoritmo jerárquico (Hierarchical), K-Means (Kmeans) y Expectation Maximization (EM). Por otro lado, en la parte superior se encuentran campos configurables usados por la técnica de visualización. El campo «Bottom Simil» se utiliza para cuantificar la similitud mínima entre los elementos y el campo "Top Simil" se utiliza para cuantificar la similitud máxima entre dos elementos. El campo «Actual Tension» es un atributo requerido por la técnica de visualización utilizada, Hierarchical Edge Bundle. Una valor de tensión entre 0 y 1 permite controlar la atracción de la línea a sus puntos de control, que afecta a la tirantez de los arcos del modelo de visualización [22].

Para utilizar los diferentes gráficos, es requisito haber ingresado los documentos WSDL a analizar. Una vez realizada dicha tarea, se procede a seleccionar un tipo de gráfico o diagrama de la opción «Graphics» y se debe hacer click en el botón «Generate» del panel principal. Una vez generadas las estructuras, no es necesario volver a generar los gráficos, sólo basta con seleccionar el gráfico y el diagrama visible en el panel principal será actualizado con el gráfico elegido. Cabe destacar que si se desea agregar o quitar documentos WSDL del análisis ó se quiere utilizar un nuevo algoritmo de clustering, el proceso de generar las estructuras (seleccionando los documentos WSDL/algoritmo de clustering, y haciendo click en el botón «Generate») deberá ser ejecutado nuevamente. En la Fig. 2 se puede ver un conjunto de documentos WSDL que serán analizados para detectar oportunidades de refactoring. El usuario seleccionó el algoritmo de clustering jerárquico, colocó los valores 0.30, 0.80 y 0.75 en «Bottom Simil», «Top Simil» y «Actual Tension» respectivamente. El resultado obtenido se materializa en una disposición radial de la técnica de HEB, en la cual los arcos reflejan qué operaciones de los servicios Web son candidatas a ser refactorizadas. Es decir, estas operaciones tienen similitudes y podrían juntarse en un mismo documento WSDL. Por ejemplo, si se selecciona la operación *ValidarRelaciones* se puede apreciar los arcos resaltados (color verde) hacia las operaciones resaltadas en rojo, indicando que esas operaciones pertenecen al mismo cluster. Algunas de estas operaciones son las siguientes: *ObtenerEstadoCivil, ObtenerRelacionesxCuil, ObtenerDatosxApeyNom, ObtenerDatosxDocumento, ListarPersxDoc*, entre otras. Con esta información el usuario puede tomar la decisión de refactorizar la aplicación orientada a servicios.

Las siguientes sub-secciones describen distintos aspectos de la herramienta. La sub-sección III-A presenta la tecnología utilizada para desarrollar *VizSOC*. La sub-sección III-B describe la representación de un servicio Web para que sea procesable por los algoritmos de clustering. La sub-sección III-C presenta los algoritmos de clustering implementados en nuestra herramienta. Finalmente, la sub-sección III-D describe la técnica de visualización utilizada para visualizar los resultados arrojados por los algoritmos de clustering.

III-A. Tecnología Utilizada

Para desarrollar *VizSOC*, se utilizó un conjunto de tecnologías para dar soporte al patrón arquitectónico Cliente-Servidor. Como tecnología web principal se utiliza Ruby on Rails (RoR)[1] versión 2.1, ya que se basa en la arquitectura Model View Controller (MVC) y aporta considerables beneficios a la hora de realizar una aplicación web. Por ejemplo, algunos de ellos son minimización de impacto de cambio, incremento de mantenibilidad, roles de desarrollo bien separados y reuso de código [23]. Por otro lado, RoR cuenta con componentes creados por la comunidad de esta herramienta, lo cual brinda una ventaja a la hora de decidir qué plataforma utilizar.

Para la interfaz se utilizó el framework Bootstrap[2], el cual fue creado por la famosa compañía Twitter. Dicho framework facilita la maquetación de la aplicación web, no sólo agregando estilos, sino que además brinda muchas funcionalidades como la creación de tabs, menús, y en general funcionalidades básicas con Javascript. Asimismo, se utilizaron dos bibliotecas de Javascript, una llamada jQuery[3] con la cual, además de realizar funciones de interfaz de usuario, también se lleva a cabo un parsing de los documentos WSDL seleccionados por el usuario y se arman los paneles de información presentados al usuario. La segunda biblioteca responsable de la renderización de los gráficos es D3[4]; esta herramienta es útil para responder a cambios en los datos y actualizar de manera muy fácil y transparente.

Se utilizaron dos aplicaciones web escritas en Java, la primera es la mencionada en el capítulo de trabajos relacionados, «Anti-pattern detector», realizada por integrantes del ISISTAN [9]. Cabe destacar que a esta aplicación se le agregó un wrapper para que pueda ser ejecutada como una aplicación web. La segunda aplicación web también está escrita en Java, previamente desarrollada en ISISTAN, y permite aplicar técnicas de clustering a conjuntos de documentos WSDL [7]. Las dos soluciones Web integradas en la herramienta fueron acopladas utilizando SpringBoot [5]. Para el análisis y comparación de resultados a través de métricas se utilizó la herramienta R Project[6].

III-B. Representación del Servicio Web

Inicialmente, *VizSOC* recibe como entrada un conjunto de documentos WSDL pertenecientes a una aplicación

[1]http://rubyonrails.org/
[2]http://getbootstrap.com/
[3]https://jquery.com/
[4]http://d3js.org/
[5]https://projects.spring.io/spring-boot/
[6]https://www.r-project.org/

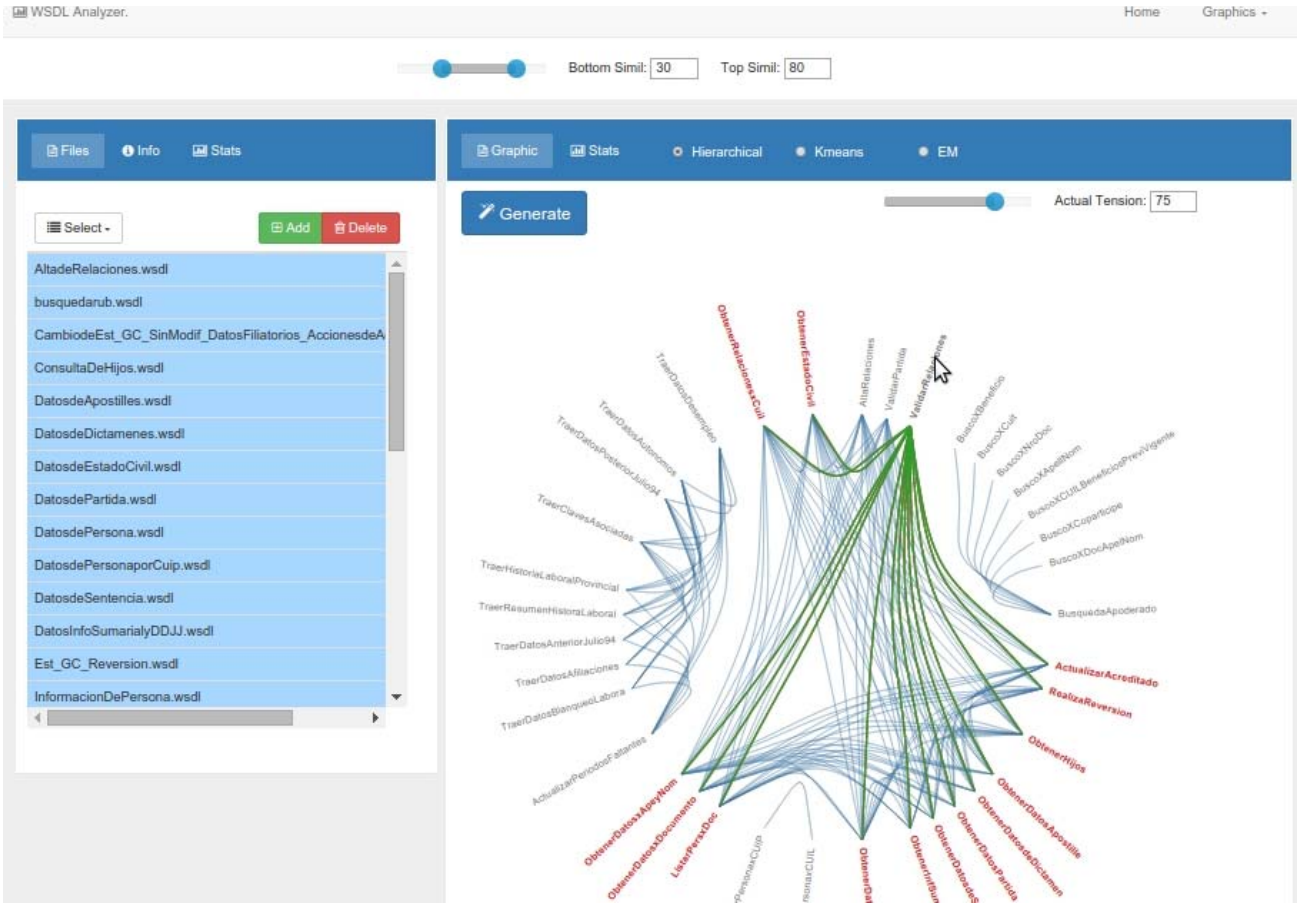

Figura 2. *VizSOC* en acción.

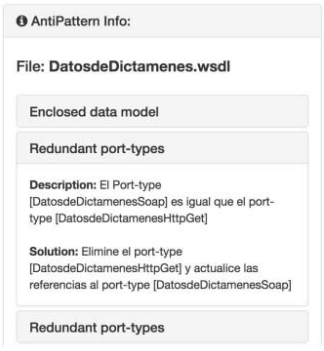

Figura 3. Detección de anti-patrón y solución sugerida.

orientada a servicios. Normalmente, un documento WSDL presenta la estructura descrita por la Fig. 5. Particularmente, *VizSOC* utiliza 3 características de esta estructura: operaciones contenidas en el «port-type», parámetros de entrada y salida, y mensajes.

Luego, un conjunto de técnicas de procesamiento de lenguaje natural son empleadas para pre-procesar el documento: división de palabras compuestas, remoción de stop-words y stemming [24]. Posteriormente, se usa la técnica de Vector Space Model (VSM) para representar las características minadas de los documentos WSDL como un vector de términos [25]. Por ejemplo, supongamos que hay 3 operaciones representadas de la siguiente manera:

O_1=(*obtenerPersonaXDoc, id, int, ApeyNom, string, fechaNac, string*)

O_2=(*obtenerPersonaXCuil, win, string, ApeyNom, string, fechaNac, string*)

O_3=(*obtenerCuilXDoc, id, int, ApeyNom, string, win, string*)

La primera operación (O_1) apunta a obtener una persona mediante su documento nacional de identidad de tipo Integer; el resultado viene dado en forma de nombre y apellido (*ApeyNom*), y fecha de nacimiento del tipo String. La segunda operación (O_2) recupera una persona dado su CUIL (Clave Única de Identificación Laboral) del tipo String; el resultado también viene dado en forma de nombre y apellido, y fecha de nacimiento del tipo String. La tercera operación (O_3) obtiene una persona mediante su CUIL del tipo String; el resultado también viene dado en forma de nombre y apellido, fecha de nacimiento y CUIL, todos del tipo String. Una vez que los vectores de cada operación son construidos, se procede a construir el vector T, el cual representa la lista de todos los términos

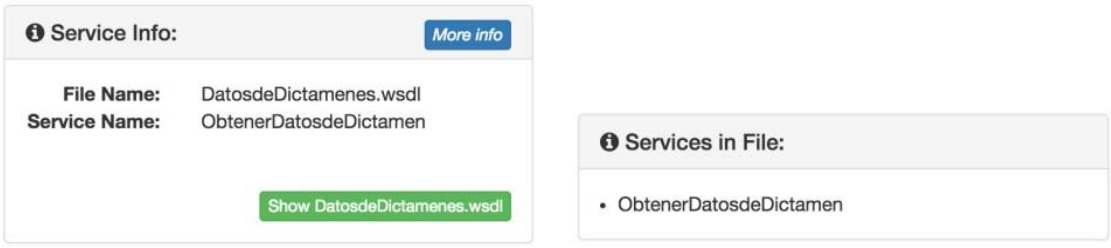

Figura 4. Información del servicio Web

```
<definitions>
    <types>
        los tipos de datos ...
    </types>
    <message>
        las definiciones del mensaje ...
    </message>
    <portType>
        las definiciones de operación ...
    </portType>
    <binding>
        las definiciones de protocolo ...
    </binding>
    <service>
        las definiciones de servicios ...
    </service>
</definitions>
```

Figura 5. Estructura genérica de un documento WSDL.

diferentes que existen en los vectors de las operaciones. Según nuestro ejemplo, T es construido de la siguiente manera:

T=(*obtenerPersonaXDoc, id, int, ApeyNom, string, fechaNac, string, obtenerPersonaXCuil, cuil, obtenerCuilXDoc*).

Por simplicidad, en el ejemplo no se utilizaron las técnicas de pre-procesamiento de lenguaje natural previamente mencionadas. Siguiendo nuestro enfoque, cada operación es representada mediante un vector numérico cuyo tamaño es la longitud de T y cada posición es el peso de cada término en la operación. La técnica para determinar el peso consiste en asignar el número de veces que cada elemento de T aparece en O_i. Luego, cada operación es representada de la siguiente manera:

$O_1 = (1, 1, 1, 1, 2, 1, 0, 0, 0)$
$O_2 = (0, 0, 0, 1, 3, 1, 1, 1, 0)$
$O_3 = (0, 1, 1, 0, 1, 0, 0, 1, 1)$

III-C. Algoritmos de clustering

Los métodos no supervisados en machine learning son aquellos que tratan de encontrar una estructura que está oculta en los datos que no fueron previamente categorizados [26]. Por eso, elegimos clustering no supervisado ya que los servicios Web de las aplicaciones SOA no están categorizados; entonces, para consumir los datos presentes en los documentos WSDL se necesita obtener dicha estructura oculta. Un buen modelo de clustering debe identificar clusters que sean tanto compactos como separados entre sí, es decir, que tengan alta similaridad intra-cluster y baja similaridad inter-cluster. Un modelo debería descubrir todos o casi todos los patrones ocultos en los datos, entonces la calidad del modelo de clustering depende casi directamente de la medida de similitud aplicada a los elementos. Los algoritmos de clustering implementados en *VizSOC* son K-Means [27], Algoritmo Jerárquico [28] y Expectation Maximization [29].

III-C1. K-Means: Este algoritmo recibe como entrada un k número de clusters y el conjunto de datos n, siendo k el valor de centros iniciales de clusters (semillas) y luego se itera para:

- Asignar o reasignar cada objeto al cluster al cual el objeto sea más similar, basándose en el valor medio de los objetos del cluster;
- Actualizar los valores medios del cluster, los centroides propiamente dichos. Es decir, calcular el valor medio de los objetos para cada cluster;
- Cuando no hay más cambios, se considera que se ha llegado a la convergencia y es el punto de corte del algoritmo.

III-C2. Algoritmo Jerárquico: Los algoritmos jerárquicos se basan en la idea de que los objetos están más relacionados a los objetos más cercanos a éstos que a los más alejados. Las diferentes estrategias para los algoritmos jerárquicos generalmente se pueden agrupar en: Bottom Up y Top Down. Al momento de decidir cuáles grupos deberían ser combinados (para Bottom Up), o cuando un grupo debería ser dividido (para Top Down), se requiere de una medida de disimilitud entre conjuntos de observaciones.

III-C3. Expectation Maximization (EM): Este algoritmo parte con N centroides gaussianos aleatorios. Luego, se itera para cada punto p y se calcula $P(p|xi)$ como la probabilidad entre 0~1 de que p pertenezca a la distribución i. Cuando no hay más cambios, se considera que se ha llegado a la convergencia y es el punto de corte del algoritmo.

III-D. Visualización de Clusters

En numerosas técnicas de visualización, el «edge» (o borde) es una importante primitiva visual para la codificación de los datos. Por ejemplo, los edges puede codificar

datos relacionales en gráficas y datos multidimensionales en plots de coordenadas paralelas. Sin embargo, las visualizaciones a menudo sufren de desorden visual cuando el número de elementos de datos aumenta. El desorden visual causado por millones de bordes no sólo afecta a la calidad estética de la representación, sino que también hace que sea difícil la obtención de información de la visualización [8]. Recientemente, Hierarchical Edge Bundles (HEB) se ha convertido en una técnica común de reducción de desorden en la visualización de la información. En HEB, los bordes similares se deforman y se agrupan en haces, lo que proporciona una visión abstracta y ordenada de la visualización original [30]. La Fig. 6 muestra dos posibles layouts de HEB que *VizSOC* proporciona como opciones de visualización de los clusters de servicios. En la figura podemos ver los nodos intermedios rotulados del 2 al 18 en cada layout, indicando los clusters obtenidos. Cada borde desde un nodo rotulado a una operación indica la pertenencia de dicha operación al cluster indicado por ese nodo. Este enfoque permite visualizar de una manera clara y comprensible los clusters de operaciones (es decir, haces), ya sea en forma radial o de árbol.

IV. Diseño del experimento

Para diseñar el experimento planteamos dos hipótesis que intentaremos corroborar con los resultados obtenidos luego de evaluar *VizSOC* con las diferentes métricas.

Hipótesis 1: Sea N la cantidad inicial de operaciones de servicios Web y M el conjunto de algoritmos de clustering, es posible encontrar un algoritmo de clustering $P \epsilon M$ de modo que agrupe las N operaciones en K operaciones con $K < N$, maximizando la distancia inter-cluster y minimizando la distancia intra-cluster.

Hipótesis 2: Sea X la cantidad inicial de documentos WSDL, Y la cantidad de anti-patrones de un catálogo C y Z la cantidad de soluciones S para esos anti-patrones, es posible encontrar al menos un anti-patron $y \epsilon Y$ en un documento $x \epsilon X$ con una solución $z \epsilon Z$ tal que la aplicación de z mejore la descripción de x.

Las siguientes sub-secciones describen las diferentes partes que componen el diseño del experimento. La sub-sección IV-A detalla los 2 casos de estudio utilizados en la evaluación de *VizSOC*, mientras que la sub-sección IV-B describe las métricas utilizadas en el experimento.

IV-A. Casos de estudio

Para evaluar *VizSOC* se utilizaron dos casos de estudio reales conformados por aplicaciones orientadas a servicios. El primer caso de estudio pertenece a un sistema del gobierno argentino que fue utilizado en trabajos previos [31]. El sistema cuenta con 32 documentos WSDL, los cuales a su vez se distribuyen en 37 operaciones totales con las que cuenta el sistema. La cantidad total de líneas de código de los documentos es de 5032 líneas, y un promedio de ~157 líneas por documento. Para el segundo caso de estudio, se emplearon 211 documentos WSDL, en los cuales a su vez se distribuyen las 259 operaciones totales con las que cuenta el sistema. La cantidad total de líneas de código es de 38701 líneas de código, y un promedio de ~183 líneas por documento.

IV-B. Métricas

En primer lugar, se utilizaron las métricas aplicadas comúnmente en clustering no supervisado como distancia intra-cluster y distancia inter-cluster. La distancia intra-cluster, es definida como el promedio de similitud entre los elementos de cada cluster. El objetivo es minimizar esta distancia, donde C_i representa el cluster i, K es la cantidad de clusters y m_i es la cantidad de elementos pertenecientes al cluster i.

$$Intr - Cluster = \frac{\sum_{i=1}^{k}(\sum_{x,t \epsilon C_i}(\frac{dist(x,t)}{m_i}))}{K}$$

La distancia inter-cluster se define como el promedio de similitud entre los elementos dentro de cada clúster y el resto de los elementos en el conjunto de datos. La Fig. 7 muestra gráficamente la distancia inter-cluster y la distancia intra-cluster. El objetivo es minimizar la distancia intra-cluster y maximizar la distancia inter-cluster.

$$Inter - Cluster = \frac{\sum_{i=1}^{k-1}(\sum_{j=i+1}^{k} dist(c_i,c_j))}{\sum_{i=1}^{k-1} i}$$

En segundo lugar, medimos cantidad de anti-patrones clasificados según la categoría mencionada en la sección III. Para ello, contamos con 2 desarrolladores de software de 5º año de la carrera de Ingeniería de Sistemas de la Facultad de Ciencias Exactas de la UNICEN[7] y con experiencia en el desarrollo de software orientado a servicios, que usaron *VizSOC* durante la evaluación.

V. Resultados Obtenidos

En esta sección se resume los resultados obtenidos según las métricas mencionadas en la sección anterior y verificamos las hipótesis establecidas en la sección IV. La sub-sección V-A describe los resultados obtenidos por los algoritmos de clustering, mientras que la sub-sección V-B detalla los anti-patrones detectados y la distribución según su catálogo. Finalmente, la sub-sección V-C resume los resultados obtenidos y describe limitaciones en la validación de la herramienta.

[7] http://www.exa.unicen.edu.ar/

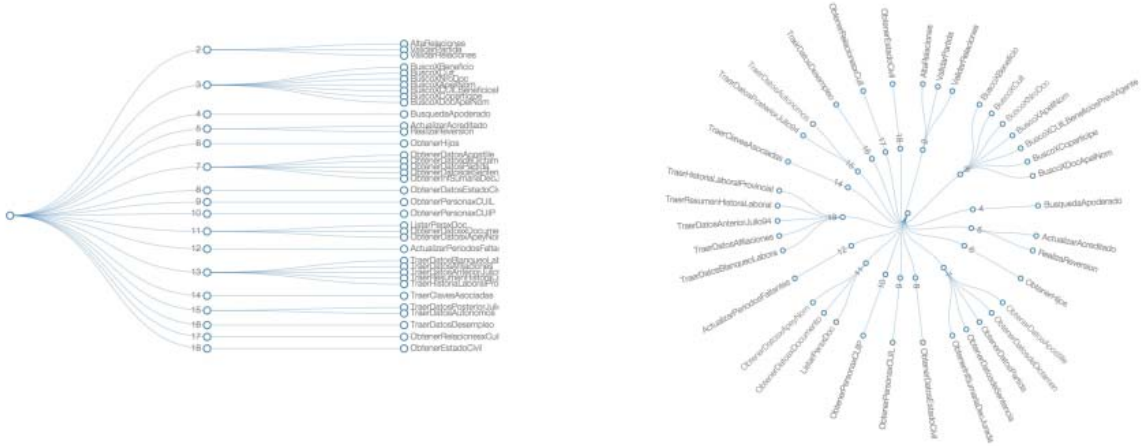

Figura 6. Layouts del tipo Tree y Radial dentro de HEB.

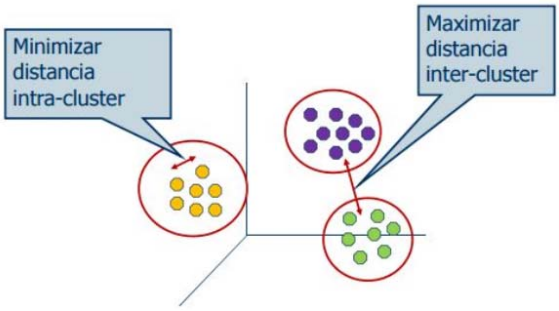

Figura 7. Diagrama de distancias inter e intra cluster.

Bottom Simil	Top Simil	#Clusters
40	60	11
30	70	13
20	80	17
1	100	18

Tabla I
CANTIDAD DE CLUSTERS CALCULADOS POR EL ALGORITMO JERÁRQUICO EN CASO DE ESTUDIO #1

V-A. Resultados de los algoritmos de clustering

Para el primer caso de estudio, analizamos la variabilidad del algoritmo jerárquico que recibe como entrada los parámetros de «Bottom Simil» y «Top Simil». La Tabla I muestra la cantidad de clusters según dichos parámetros.

Luego, calculamos la distancia inter-cluster y la distancia intra-cluster para cada uno de los algoritmos de clustering considerando la cantidad de clusters que coincidia en los 3 algoritmos (#clusters = 17). La Tabla II resume estas métricas, las cuales fueron calculadas con el software R project. Podemos concluir que el algoritmo que optimiza las distancias intra e inter cluster es el algoritmo jerárquico.

Para el segundo caso de estudio, analizamos nuevamente la variabilidad del algoritmo jerárquico que recibe como entrada los parámetros de «Bottom Simil» y «Top Simil».

Algoritmo de Clustering	Inter cluster	Intra cluster	#Clusters
Algoritmo Jerárquico	23.51	1.40	17
K-Means	8.08	5.32	17
Expectation Maximization	8.54	4.92	17

Tabla II
COMPARATIVA DE MÉTRICAS CALCULADAS PARA CASO DE ESTUDIO #1.

Bottom Simil	Top Simil	#Clusters
40	60	17
30	70	24
20	80	32

Tabla III
CANTIDAD DE CLUSTERS CALCULADOS POR EL ALGORITMO JERÁRQUICO

La Tabla III muestra la cantidad de clusters según dichos parámetros.

De igual manera, calculamos la distancia inter-cluster y la distancia intra-cluster para cada uno de los algoritmos de clustering considerando la cantidad de clusters que coincidía en los 3 algoritmos (#clusters = 32). La Tabla II resume estas métricas, las cuales fueron calculadas con el software R project. Nuevamente, podemos concluir que el algoritmo que optimiza las distancias intra e inter-cluster es el algoritmo jerárquico.

Algoritmo de Clustering	Inter cluster	Intra cluster	#Clusters
Algoritmo Jerárquico	15.01	3.13	32
K-Means	5.30	26.71	32
Expectation Maximization	5.23	29.18	32

Tabla IV
COMPARACIÓN DE MÉTRICAS CALCULADAS PARA CASO DE ESTUDIO #2.

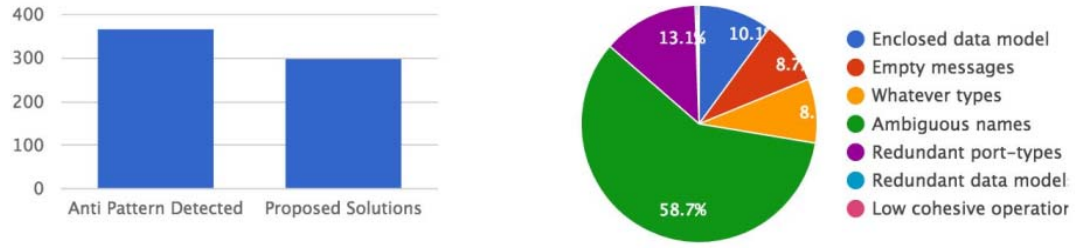

Figura 8. Anti-patrones detectados en el caso de estudio #1

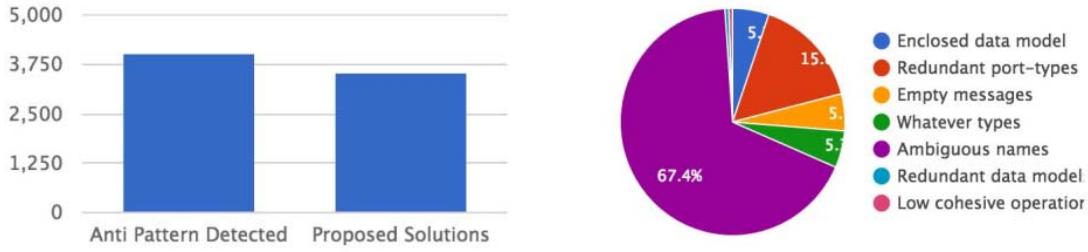

Figura 9. Anti-patrones detectados en el caso de estudio #2

V-B. Antipatterns detectados

Esta sub-sección resume el total de anti-patrones detectados y el porcentaje de detección para cada categoría del catálogo de anti-patrones. Para el primer caso de estudio, la Fig. 8 muestra que el total de anti-patrones detectados fue de 366 y el total de soluciones propuestas 300. Según cada categoría, se detectaron 37 del tipo "Enclosed data model" (10.1 %), 32 del tipo "Empty messages" (8.7 %), 32 del tipo "Whatever types" (8.7 %), 215 del tipo "Ambiguous names", 1 del tipo "Redundant data models" (0.35 %), 1 del tipo "Low cohesive operations in the same port-type" (0.35 %) y 48 del tipo "Redundant port-types" (13.1 %).

Para el segundo caso de estudio,la Fig. 9 indica que el total de anti-patrones detectados fue de 3807, para los cuales se sugirieron 3657 soluciones. Según cada categoría, se detectaron 211 del tipo "Enclosed data model" (5.47 %), 211 del tipo "Empty messages" (5.47 %), 211 del tipo "Whatever types" (5.47 %), 2708 del tipo "Ambiguous names" (67.4 %), 22 del tipo "Redundant data models" (0.32 %), 22 del tipo "Low cohesive operations in the same port-type" (0.32 %), y 633 del tipo "Redundant port-types" (15.55 %).

V-C. Análisis de los resultados obtenidos

Luego de evaluar los algoritmos de clustering, pudimos obtener que en ambos casos de estudio el mejor algoritmo de clustering es el algoritmo jerárquico. Esto no solamente corrobora la Hipótesis 1 planteada anteriormente, sino que también refuerza los resultados obtenidos es investigaciones previas [7]. Sin embargo, debemos tener en consideración algunas limitaciones suscitadas en la experimentación. En primer lugar, una limitación a trabajar es el hecho de definir el factor k de antemano, el cual debería definirse automáticamente. En segundo lugar, actualmente no contamos con un *feedback* de los usuarios para que evaluen la calidad de los clusters obtenidos, y si se aproximan a los que se hubieran obtenido manualmente. Es decir, si *VizSOC* supera o no la capacidad del usuario para encontrar agrupamientos funcionales de operaciones de servicios Web.

Por otro lado, con respecto a los anti-patrones detectados junto con sus soluciones, podemos corroborar la Hipótesis 2 planteada anteriormente. Sin embargo, debemos tener en cuenta que la solución presentada al usuario es una posible aplicación y no se materializa automática en el documento WSDL afectado, sino que el usuario es el encargado de llevar a cabo dicha reparación.

VI. Conclusiones

Este trabajo ha presentado una herramienta web denominada *VizSOC* para dar soporte a la detección de oportunidades de refactoring en aplicaciones orientadas a servicios. Asimismo, la herramienta detecta y ofrece soluciones para los anti-patrones presentes en los documentos WSDL que se analizan. La herramienta también cuenta con una amplia cantidad de opciones a la hora de seleccionar algoritmos de clusterización, los diagramas para visualizar los resultados y un visualizador de documentos WSDL, entre otras características que facilitan la tarea de los desarrolladores de software para analizar dichos documentos. Encontrar oportunidades de refactoring se vuelve una tarea compleja para el ser humano sobre todo cuando la cantidad de documentos analizar es muy grande. Para evaluar *VizSOC*, se realizaron experimentos con documentos de aplicaciones reales orientadas a servicio. El algoritmo de clustering jerárquico ha mostrado

un desempeño aceptable por optimizar las métricas de validación interna de clusters.

Nuestra herramienta posee algunas limitaciones para asociar operaciones teniendo en cuenta diferentes criterios, como por ejemplo la utilización de la semántica de las operaciones y/o servicios. Utilizar un enfoque de agrupamiento por semántica podría ser útil para refactorizar servicios Web debido a que, por ejemplo, el sistema que está siendo refactorizado puede tener mucho tiempo en construcción, lo cual contribuye a la posibilidad de que los servicios y sus operaciones no estén agrupados utilizando ningún criterio.

Como trabajo futuro, proponemos agregar a la herramienta un algoritmo que realice la refactorización de los servicios de manera automática, dando como resultado nuevos documentos WSDL teniendo en cuenta la posibilidad, por ejemplo, de interactuar con el usuario a la hora de nombrar los nuevos servicios Web o sus operaciones. Particularmente, la mejora propuesta es incluir técnicas de agrupamiento que analicen y utilicen la semántica de las operaciones, de modo de ofrecer otras oportunidades de refactorización que la herramienta actualmente no ofrece. Una alternativa sería explotar con técnicas de minería de texto los comentarios de los mensajes y documentación de las operaciones.

Agradecimientos

Agradecemos al Ing. Martin Vizzolini y al Ing. Emanuel Fernandez por su contribución en el desarrollo de *VizSOC*.

Referencias

[1] Erickson, J., Siau, K.: Web services, service-oriented computing, and service-oriented architecture: Separating hype from reality. Journal of Database management **19**(3) (2008) 42

[2] Bartolini, C., Bertolino, A., Marchetti, E., Polini, A.: Towards automated wsdl-based testing of web services. Service-Oriented Computing–ICSOC 2008 (2008) 524–529

[3] Fowler, M.: Refactoring: improving the design of existing code. Pearson Education India (1999)

[4] Rodriguez, J.M., Crasso, M., Zunino, A., Campo, M.: Improving web service descriptions for effective service discovery. Science of Computer Programming **75**(11) (2010) 1001–1021

[5] Berkhin, P.: A survey of clustering data mining techniques. In: Grouping multidimensional data. Springer (2006) 25–71

[6] Diehl, S.: Software visualization: visualizing the structure, behaviour, and evolution of software. Springer Science & Business Media (2007)

[7] Rodríguez, G., Soria, Á., Teyseyre, A., Berdun, L., Campo, M.: Unsupervised learning for detecting refactoring opportunities in service-oriented applications. In: International Conference on Database and Expert Systems Applications, Springer (2016) 335–342

[8] Holten, D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. IEEE Trans. on Vis. and Comp. Graph. **12**(5) (2006) 741–748

[9] Rodriguez, J.M., Crasso, M., Mateos, C., Zunino, A.: Best practices for describing, consuming, and discovering web services: a comprehensive toolset. Software: Practice and Experience **43**(6) (2013) 613–639

[10] Crasso, M., Mateos, C., Zunino, A., Campo, M.: Easysoc: Making web service outsourcing easier. Information Sciences **259** (2014) 452–473

[11] Crasso, M., Rodriguez, J.M., Zunino, A., Campo, M.: Revising wsdl documents: Why and how. IEEE Internet Computing **14**(5) (2010) 48

[12] Webster, D., Townend, P., Xu, J.: Interface refactoring in performance-constrained web services. In: IEEE 15th Int. Sym on Object/Component/Service-Oriented Real-Time Distributed Comput., IEEE (2012) 111–118

[13] Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. MIT press (1999)

[14] Sabou, M., Pan, J.: Towards semantically enhanced web service repositories. Web Semantics: Science, Services and Agents on the WWW. **5**(2) (2007) 142–150

[15] Kuhn, A., Ducasse, S., Gírba, T.: Semantic clustering: Identifying topics in source code. Inf. and Soft. Tech. **49**(3) (2007) 230–243

[16] Liu, W., Wong, W.: Web service clustering using text mining techniques. Int. J. of Agent-Oriented Soft. Eng. **3**(1) (2009) 6–26

[17] Ma, J., Zhang, Y., He, J.: Efficiently finding web services using a clustering semantic approach. In: Int. workshop on Context enabled source and service selection, integration and adaptation., ACM (2008) 5

[18] Elgazzar, K., Hassan, A.E., Martin, P.: Clustering wsdl documents to bootstrap the discovery of web services. In: IEEE Int. Conf. on Web Services., IEEE (2010) 147–154

[19] Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A.: An empirical study on web service evolution. In: IEEE Int. Conf. on Web Services ., IEEE (2011) 49–56

[20] Dong, X., Halevy, A., Madhavan, J., Nemes, E., Zhang, J.: Similarity search for web services. In: 30th Int. Conf. on Very large data bases., VLDB Endowment (2004) 372–383

[21] Kumara, B.T., Yaguchi, Y., Paik, I., Chen, W.: Clustering and spherical visualization of web services. In: IEEE Int. Conf. on Services Comput., IEEE (2013) 89–96

[22] Braun, L., Volke, M., Schlamp, J., von Bodisco, A., Carle, G.: Flow-inspector: a framework for visualizing network flow data using current web technologies. Computing **96**(1) (2014) 15–26

[23] Vosloo, I., Kourie, D.G.: Server-centric web frameworks: An overview. ACM Computing Surveys (CSUR) **40**(2) (2008) 4

[24] Crasso, M., Zunino, A., Campo, M.: Awsc: An approach to web service classification based on machine learning techniques. Revista Iberoamericana de Inteligencia Artificial **12**(37) (2008) 25–36

[25] Fautsch, C., Savoy, J.: Adapting the tf idf vector-space model to domain specific information retrieval. In: ACM Symposium on Applied Computing, ACM (2010) 1708–1712

[26] Marakas, G.M.: Modern data warehousing, mining, and visualization: core concepts. Prentice Hall Upper Saddle River, NJ (2003)

[27] MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: 5th Berkeley symp. on mathematical statistics and probability. Volume 1., California, USA (1967) 281–297

[28] Fisher, D.H.: Knowledge acquisition via incremental conceptual clustering. Machine learning **2**(2) (1987) 139–172

[29] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the royal statistical society. Series B (methodological) (1977) 1–38

[30] Zhou, H., Xu, P., Yuan, X., Qu, H.: Edge bundling in information visualization. Tsinghua Science and Technology **18**(2) (2013) 145–156

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