A Taxonomy-based Approach For Fault Localization In Service-Oriented Applications

E. Scott, A. Soria and M. Campo

***Abstract*— Service Oriented Computing is a relevant paradigm that allows for building software solutions as it supports the creation of dynamic and agile applications. As a result of these benefits, applications based on the paradigm are commonly used in the industry. However, the more complex software solutions become, the higher the chance of having bugs is. Identifying them in the entire solution may also be one of the most tedious, expensive and time-consuming activity within the developing process. Therefore, fault localization aims at finding the location of faults, determining the root cause of the failure and identifying the causes of abnormal behavior of a faulty program. In particular, fault localization for service oriented applications is currently a research field that has received little attention. In this context, we present a taxonomy-based approach for fault localization in service oriented applications, using information of faults in the underlying architecture of these applications, namely SOA. The results on the location of faults are not only provided to the developer but also enhanced with architectural information, that is, showing the faults and their connection with exceptions, code and architectural scenarios. The experimental results carried out indicate that developers reduce their effort of localizing faults in terms of time and lines of code.**

***Keywords*— Service-oriented, Taxonomy, Fault locatization**

I. INTRODUCCIÓN

EL PARADIGMA de la Computación Orientada a Servicios permite construir sistemas distribuidos y de alta complejidad, donde la funcionalidad usualmente es descompuesta en términos de servicios. Estos servicios pueden ser desarrollados completamente desde cero, así como también pueden resultar de la composición de otros ya existentes para satisfacer funcionalidad requerida. Esta es una de las razones que hace a las aplicaciones orientadas a servicios una atractiva propuesta para la industria. La arquitectura subyacente a estas aplicaciones es denominada Arquitectura Orientada a Servicios (SOA), conformando un estilo arquitectónico focalizado en coordinar eficientemente la interacción entre componentes dentro de ambientes cambiantes y de continua evolución [8].

Sin embargo, en un contexto distribuido como el de SOA y a medida que la complejidad de las aplicaciones aumenta, más alta es la probabilidad de observar comportamientos anormales en los sistemas [6]. Estos comportamientos anormales, que provocan desvíos en los estados correctos del sistema, se conocen como *errores*, mientras que la causa o raíz a la que se le adjudica dicho error se conoce como *falla* [1]. Identificar las fallas en una solución SOA no es una tarea sencilla por su naturaleza distribuida. Por ejemplo, los servicios habitualmente son tercerizados, lo que lleva a que no se disponga del código fuente ni se pueda controlar su disponibilidad.

Esto resulta un problema dentro de las aplicaciones orientadas a servicios, ya que los enfoques actuales para localizar fallas requieren contar con el código fuente para poder funcionar [18]. Por ello, se sostiene que la localización de fallas es una de las actividades más tediosas y complejas dentro del desarrollo, especialmente en aplicaciones orientadas a servicios. De esta manera, se hace evidente la necesidad de nuevos enfoques y herramientas que faciliten la localización de fallas especialmente en este tipo de soluciones distribuidas.

En este contexto, el conocimiento acerca de los posibles errores que pueden surgir es un factor clave para el éxito en el tratamiento de fallas [19, 2]. Contar con información sobre las potenciales fallas que generan estos errores, es crucial para localizarlas en el sistema. Por esta razón, muchos trabajos han estudiado y clasificado las posibles fallas que pueden ocurrir en SOA mediante taxonomías [3, 14]. Nuestro trabajo toma como base este tipo de taxonomías para presentar un enfoque de localización de fallas.

En nuestro enfoque, la información generada en tiempo de ejecución por el sistema bajo análisis se ubica dentro de la taxonomía. La taxonomía contiene información sobre un conjunto de posibles errores que se pueden observar en aplicaciones orientadas a servicios. Además, el enfoque utiliza el código fuente de los servicios (si se dispone de él), y su correspondencia con el escenario arquitectónico. Con esta información, el enfoque asiste al desarrollador excluyendo partes del sistema irrelevantes para el error, lo que permite al desarrollador circunscribir el análisis a las porciones del sistema que potencialmente contengan la falla.

La validación del enfoque se realizó mediante el reporte de experiencia de grupos de desarrolladores. Se formaron dos grupos de estudiantes avanzados de la Facultad de Ciencias Exactas perteneciente a la UNICEN, ambos encargados de localizar fallas en una aplicación orientada a servicios. Se registró el número de líneas de código inspeccionadas y el tiempo requerido en la localización de fallas en ambos grupos. Los resultados mostraron que la cantidad de líneas de código inspeccionadas y el tiempo necesario para llevar a cabo la tarea fueron menores en el grupo que utilizó el enfoque propuesto.

El presente artículo se organiza como sigue: en la Sección II se muestran los trabajos relacionados con la localización de fallas. La Sección III presenta el enfoque propuesto,

E. Scott, ISISTAN Research Institute (CONICET-UNICEN), Tandil, Buenos Aires, Argentina, ezequiel.scott@isistan.unicen.edu.ar

Á. Soria, ISISTAN Research Institute (CONICET-UNICEN), Tandil, Buenos Aires, Argentina, alvaro.soria@isistan.unicen.edu.ar

M. Campo, ISISTAN Research Institute (CONICET-UNICEN), Tandil, Buenos Aires, Argentina, marcelo.campo@isistan.unicen.edu.ar

explicando la heurística de localización de fallas mediante un ejemplo. Luego, en la Sección IV se explica cómo el enfoque fue evaluado. Finalmente, la Sección V presenta una conclusión acerca del enfoque.

II. TRABAJOS RELACIONADOS

Las primeras técnicas de localización de fallas fueron de naturaleza puramente intuitiva, analizando manualmente tanto el volcado de memoria como las impresiones del estado de las variables en pantalla o en archivos de *log*. En la actualidad, las técnicas no difieren considerablemente de estos métodos primitivos ya que actualmente las herramientas de *debugging* se basan en la inserción de *breakpoints* y en la inspección de los estados en esos *breakpoints*. Aunque generalmente estas herramientas resultan fáciles de utilizar, los usuarios deben desarrollar sus propias estrategias que los conduzcan a examinar sólo aquellas porciones de códigos significativas.

Técnicas más avanzadas se basan en el análisis de los estados de ejecución [7, 9, 20], es decir del conjunto de variables y sus valores en un punto particular durante la ejecución del programa. Un enfoque general utilizado este concepto, consiste en modificar los valores de algunas variables para determinar cuál de ellas es la causa de la ejecución fallida. El problema de este enfoque es el esfuerzo relativamente elevado, ya que pueden existir miles de estados en la ejecución de un programa y explorar cada uno de ellos puede ser inviable.

Los trabajos centrados en el análisis de contextos proponen enfoques basados en la clasificación y minería de datos para predecir los comportamientos del software, y así detectar fallas que son similares a otras conocidas [13]. Otros enfoques proponen la minería de reglas temporales con el fin de asistir al monitoreo, la verificación, y la comprensión del sistema [12], mientras que otros se centran en proveer un conjunto de métricas que permitan medir las propiedades de un determinado contexto proporcionando las bases para su comparación y asegurar de esta manera su calidad [10]. Por otro lado, FLABot [16, 17] asiste en la tarea de localizar fallas usando información arquitectónica y *logs* de ejecución, permitiendo realizar un análisis a un mayor nivel de abstracción.

Sin embargo, estos enfoques se basan en el análisis de código fuente suponiendo su disponibilidad, por lo que se vuelven difíciles de aplicar en determinados casos como el del desarrollo de aplicaciones orientadas a servicios. Aun así, las técnicas antes mencionadas son de las más avanzadas en el área, y es de notar que no existen trabajos que ataquen el problema de la localización de fallas en aplicaciones orientadas a servicios. Por lo tanto, este trabajo apunta a cubrir la necesidad de herramientas que permitan la localización de las fallas en este tipo de aplicaciones inherentemente complejas y distribuidas.

III. LOCALIZACIÓN DE FALLAS BASADA EN TAXONOMÍA

El componente principal en el que se basa el enfoque es en la utilización de una taxonomía de fallas específica para SOA como instrumento para localizar las causas de los errores en el sistema, ya que presenta una organización consistente y sistemática acerca de las fallas. En particular, la taxonomía resulta de la combinación de diferentes taxonomías de fallas

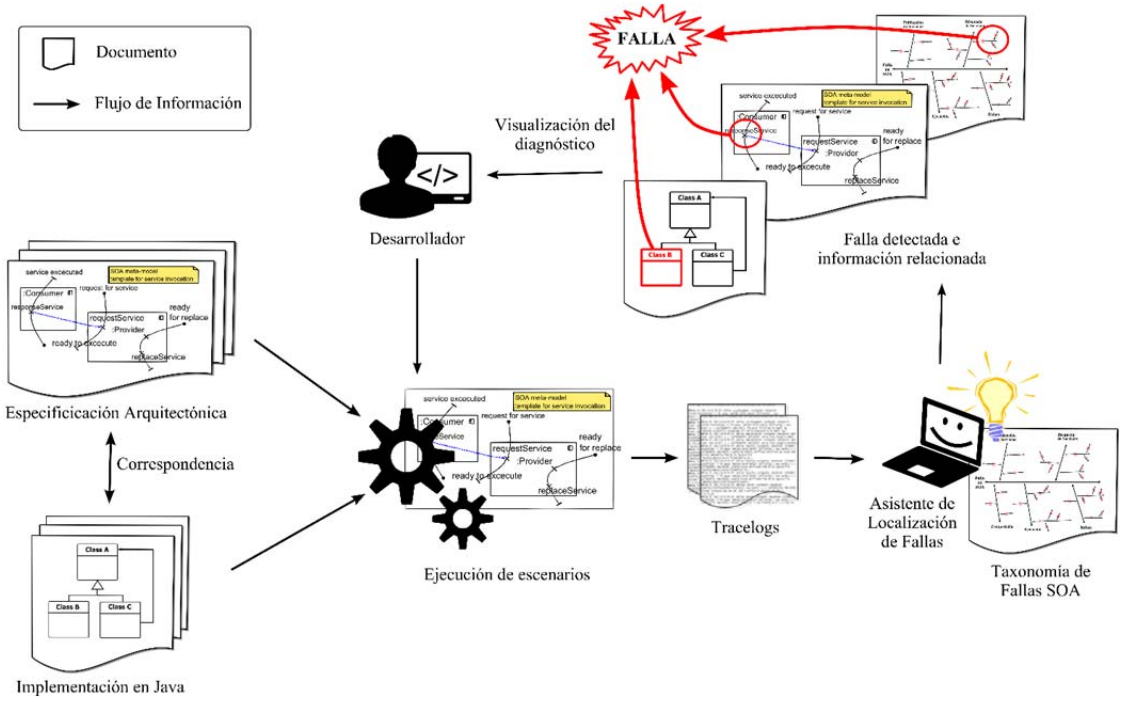

Figura 1. Modelo conceptual del enfoque.

presentadas previamente en los trabajos de [3, 5, 14].

Entonces, la información de ejecución de la aplicación que manifiesta el error se busca dentro de esta taxonomía, lo que en conjunto con el mapeo arquitectónico permite obtener información relevante acerca de la falla, sus causas y su localización. La Fig. 1 muestra el esquema conceptual del enfoque propuesto.

El enfoque se materializó como un asistente implementado dentro de la plataforma Eclipse, usando el motor de instrumentación de código fuente y los editores de Use Case Maps de FLABot [16]. De esta manera, el asistente permite ayudar en el proceso de localización de fallas en dos etapas denominadas *fase de configuración* y *fase de ejecución*.

En la *fase de configuración* se requiere especificar la arquitectura del sistema SOA a inspeccionar y relacionar esta especificación arquitectónica con su implementación orientada a objetos. Esta relación se denomina mapeo arquitectura-implementación, y es usual contar con ella en enfoques de desarrollos centrados en la arquitectura [16, 17]. La ventaja de esta configuración es que permite al enfoque recolectar información de ejecución del sistema que luego es utilizada para navegar la taxonomía.

Una vez configurado el asistente, la *fase de ejecución* tiene como objetivo asistir a los desarrolladores en la localización de fallas de la aplicación orientada a servicios. Entonces, el desarrollador ejecuta el escenario donde se manifiesta el error, por ejemplo mediante la aparición de una excepción. Durante esta ejecución, se instrumenta el código fuente y se registra en un archivo denominado *tracelog*. Además de la traza de ejecución, este archivo contiene la información sobre cada clase y su correspondencia con el escenario arquitectónico. Esto permite al asistente extraer la información de las excepciones que han ocurrido, es decir, los errores directamente observables de la aplicación.

Con la información del *tracelog*, el asistente navega la taxonomía de fallas en búsqueda del error observado. La información de la taxonomía permite ubicar cuál es la posible falla asociada a ese error, junto a información adicional sobre su frecuencia y posibles métodos para solucionarla. Además, el mapeo arquitectura-implementación permite localizar en el código fuente (si se tiene disponible) y en la especificación arquitectónica dónde se manifestó el error. En este punto, el asistente brinda esta información junto con la rama de la taxonomía que explica las causas de la falla al desarrollador.

De este modo, se espera que con esta información el desarrollador pueda circunscribir el análisis a las porciones del sistema realmente afectadas, excluyendo aquellas partes irrelevantes a la falla. En este sentido, es importante notar que el objetivo del enfoque no es localizar el punto exacto en el código donde se ubica la falla sino circunscribir la porción de código donde puede encontrarse.

A. Un ejemplo de localización de fallas

Con el objeto de clarificar el enfoque, se muestra un ejemplo basado en una aplicación prototipo relacionada con la gestión de jugadores de fútbol. Dicho sistema es orientado a servicios y posee una falla crítica que impide recuperar la información respectiva a un determinado jugador de fútbol. Se cuenta con el modelo arquitectónico realizado por el arquitecto del sistema, así como su correspondencia a la implementación. Básicamente, el modelo arquitectónico se describe en términos de escenarios en la notación de Use Case Maps [4]. De esta manera, este mapeo arquitectura-implementación completa la *fase de configuración*.

Por su parte, el desarrollador que tiene la tarea de localizar la falla y subsanarla, inicia la *fase de ejecución*. En esta fase se ejecutan los escenarios donde el desarrollador conoce que el error se manifiesta, en este caso, el escenario es *"Búsqueda de un jugador de fútbol"*. Luego de su ejecución, se obtiene la traza de ejecución mediante la instrumentación del código en tiempo de ejecución y se almacena en un *tracelog*. Esta traza es analizada por el algoritmo, que recolecta las excepciones que sucedieron, como por ej.: *"java.net.ConnectException"*. La excepción se identifica dentro de la taxonomía y ahí se obtiene una categoría e información acerca de la posible falla correspondiente al error manifestado.

La taxonomía de fallas SOA elaborada puede verse en la Fig. 2. Se construyó apuntando a tener en cuenta las fallas que pueden causar un gran número de errores observables en sistemas orientados a servicios, tanto fallas físicas, como de desarrollo e interacción entre servicios. En color rojo, se representan los errores o excepciones que corresponden a una determinada falla, mientras que las flechas indican relaciones causa-efecto [11]. De esta manera, se recorren las hojas de la taxonomía en búsqueda de la excepción, y de ahí hacia el tronco de la taxonomía para encontrar la información pertinente a la falla que ha ocurrido, junto con los efectos y posibles causas. Para el ejemplo, la Fig. 2 muestra en color rojo la rama de la taxonomía que incluye la excepción *"java.net.ConnectException"*.

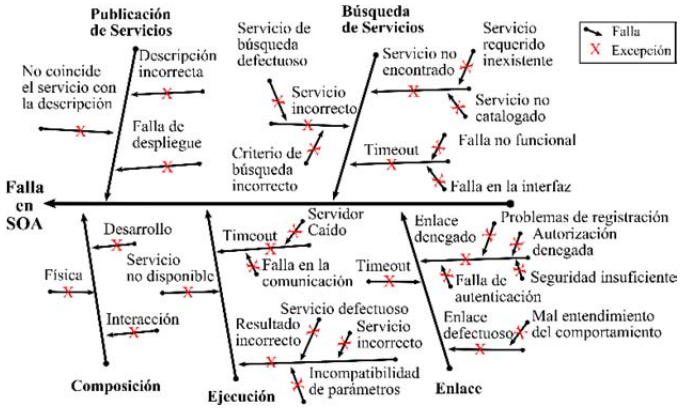

Figura 2. Diagrama causa-efecto de la taxonomía de fallas SOA.

Posteriormente, el algoritmo busca la responsabilidad del UCM especificado donde se manifestó el error. Cuando se localiza la responsabilidad, se buscan las clases relacionadas con ésta, así como los métodos involucrados en la ejecución. De esta manera, es posible informar al desarrollador de la porción de código donde se manifestó el error, así como el escenario arquitectónico posible al que se encuentra ligado e

información relacionada a la falla. Para este caso, el mensaje provisto será *"falla de búsqueda del servicio: conexión rechazada - asegúrese de que el servidor se encuentra activo"*. La Fig. 3 muestra una pantalla del asistente luego de localizar una falla SOA.

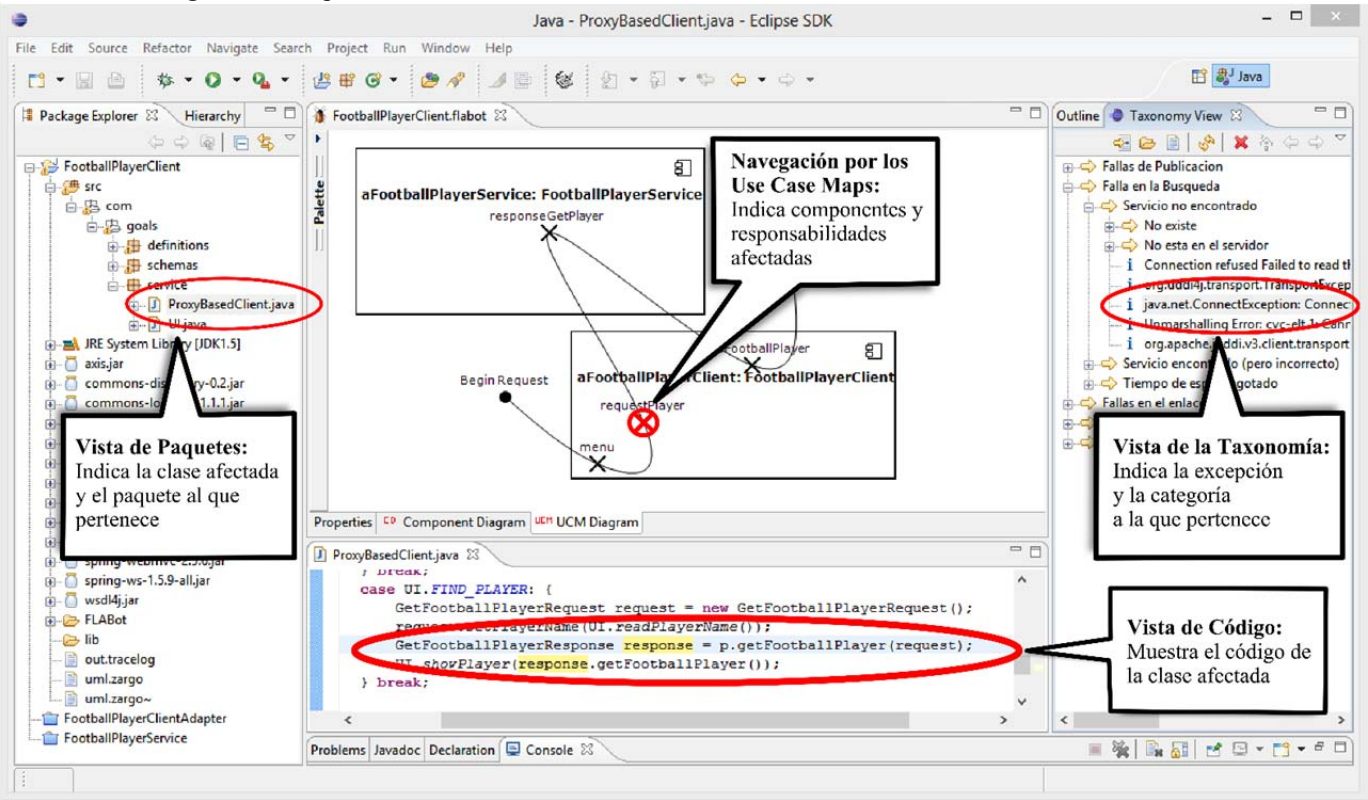

Figura 3. Pantalla del resultado del asistente luego de localizar la falla SOA.

Como resultado, la información provista mediante la taxonomía contribuye a que el desarrollador elimine una serie de opciones y reduzca el espacio de búsqueda necesario para localizar la falla. En el ejemplo, si el desarrollador no contara con dicha información, debería analizar todas las posibles causas por la cual ha sido rechazada la conexión con el servidor, (como por ejemplo podría ser que la conexión no funcione, haya problemas en los protocolos, el cliente se comporte erróneamente, etc.) e inspeccionar cada una de ellas en el código fuente, lo que sin duda implicaría mayor esfuerzo.

IV. Evaluación

El objetivo de la evaluación es analizar la efectividad en la localización de fallas de nuestro enfoque. Para esto, se contrastaron los resultados de llevar a cabo una localización de fallas tradicional con los obtenidos mediante el enfoque propuesto. La localización de fallas se llevó a cabo sobre una aplicación orientada a servicios, la cual ha sido utilizada previamente con fines experimentales [15]. Esta aplicación permite gestionar la información de jugadores de fútbol mediante consultas que son realizadas a través del acceso a servicios externos provistos por World Cup Football Pool. Sobre la aplicación se inyectaron cinco fallas de diferentes tipos, y se analizó el comportamiento de los desarrolladores en dos contextos: con la asistencia del enfoque y sin la asistencia.

Los usuarios que participaron de la evaluación fueron seleccionados de un grupo de estudiantes avanzados y graduados de la carrera de Ingeniería de Sistemas de la Facultad de Ciencias Exactas perteneciente a la UNICEN. Un total de 6 usuarios fueron asignados a dos grupos representando los usuarios no asistidos (grupo A) y los asistidos (grupo B). La Tabla I muestra las características de los usuarios, tales como la profesión, el porcentaje de avance en la carrera, los años de experiencia laboral, y la experiencia previa trabajando con SOA. Con el objetivo de que estas características no influyan significativamente en el proceso de localización de fallas, se buscó formar grupos homogéneos respecto a dichas características. Para esto, la asignación de participantes a grupos se realizó de forma emparejada, resultando en grupos cuyas características son similares. Los promedios y desvíos estándar de cada grupo para cada característica se pueden observar en la Tabla II. Adicionalmente, para mostrar que ambos grupos son similares respecto a sus características, se realizó una prueba *T-Student* para muestras independientes. Esta prueba busca contrastar si las medias de cada grupo son similares. Como se puede apreciar en la última fila de la Tabla II, los valores del estadístico *T* y del *p-valor* ratifican que la diferencia entre las medias de las características de ambos grupos no son significativamente diferentes (*p-valor* > *0.05*).

TABLA I
CARACTERÍSTICAS DE LOS DESARROLLADORES QUE PARTICIPARON EN EL EXPERIMENTO. LAS CARACTERÍSTICAS INDICADAS CON * UTILIZAN LA SIGUIENTE ESCALA: 1 = NUNCA O RARAMENTE; 2 = ALGUNAS OCASIONES; 3 = FRECUENTEMENTE

Usuario	Profesión	Avance en la carrera (%)	Experiencia laboral (años)	¿Ha trabajado con SOA? *	Grupo
#1	Desarrollador	100	6	3	A
#2	Analista Funcional	100	5	1	A
#3	Estudiante	90	1	1	A
#4	Analista Funcional	100	5	2	B
#5	Desarrollador	90	1	3	B
#6	Estudiante	80	0	1	B

TABLA II
PROMEDIOS, DESVÍOS ESTÁNDAR Y VALORES DE LA PRUEBA *T-STUDENT* PARA LAS CARACTERÍSTICAS DE LOS GRUPOS INVOLUCRADOS EN LA EVALUACIÓN

Grupo	Avance en la carrera (%)	Experiencia laboral (años)	¿Ha trabajado con SOA? *
A	96.67 ± 5.77	4 ± 2.65	1.67 ± 1.15
B	90 ± 10	2 ± 2.65	2 ± 1
T (p-valor)	2 (.183)	2 (.183)	.378 (.742)

Luego, a cada grupo se les asignaron 5 fallas. Al grupo A se le solicitó que localizara las fallas utilizando el ambiente de desarrollo Eclipse junto con un conjunto de documentos que describen el diseño de alto nivel del sistema, mientras que al grupo B se le solicitó además utilizar el asistente, teniendo también acceso a la documentación de diseño del sistema. Para monitorear el comportamiento de los participantes durante la localización de fallas se utilizó el *plug-in* Mylyn, ya que éste puede realizar el seguimiento de los métodos, clases, y atributos a medida que son accedidos por medio de Eclipse. Al finalizar el experimento, para cada grupo se registraron la cantidad de líneas de código (LOC) y el tiempo consumido. LOC mide la cantidad de líneas de código que fueron inspeccionadas, sin considerar blancos ni comentarios, mientras que el tiempo en minutos se utiliza como métrica de desempeño. En particular, la métrica LOC se obtiene mediante el *plug-in* Metrics evaluado sobre los métodos, clases y atributos detectados por Mylyn. Cabe destacar que las mediciones suponen que los artefactos requeridos en la *fase de configuración* han sido construidos previamente, ya que se supone el uso del enfoque en un marco de desarrollo dirigido por la arquitectura. Por lo tanto, la medición de tiempo no incluye el tiempo de elaboración de la especificación arquitectónica.

A. Resultados obtenidos

Los resultados obtenidos se muestran a continuación. Cada falla está representada como un eje en cada diagrama, sobre los cuales se marca el valor promedio de LOC y del tiempo necesario de los desarrolladores que fueron asistidos en la localización respecto a aquellos sin asistencia.

La Fig. 4 muestra la cantidad de LOC necesarias para localizar las fallas, pudiéndose observar que, en promedio, el asistente logra reducir el dominio de búsqueda sobre el cual se intenta localizar la falla, provocando consecuentemente una reducción del esfuerzo necesario. Por otro lado, la Fig. 5 muestra los tiempos promedios necesario para la localización de cada una de las fallas. Se puede observar que también los tiempos se ven reducidos, en promedio, en menos del 50% si el usuario se encuentra asistido por el enfoque.

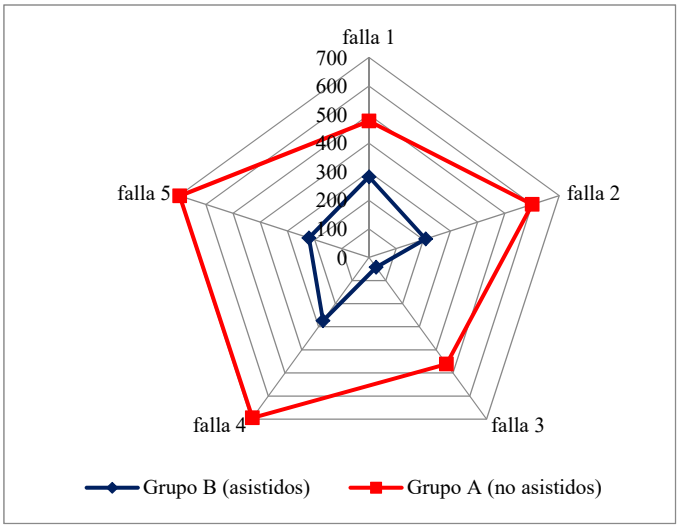

Figura 4. Líneas de código inspeccionadas.

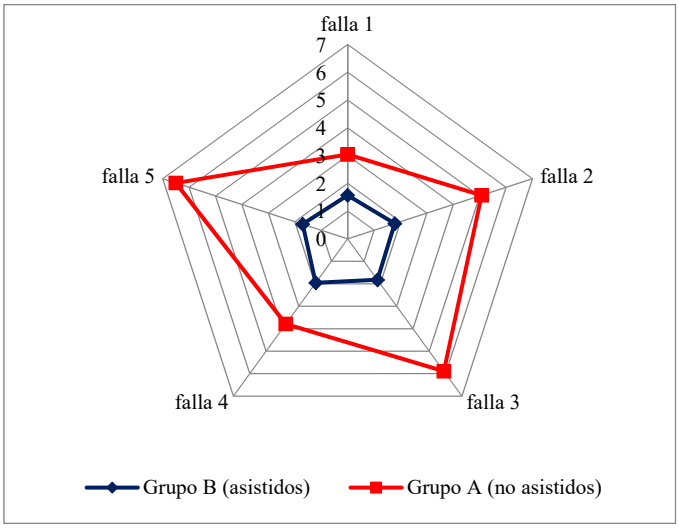

Figura 5. Tiempo de localización de fallas en minutos.

Por otro lado, se estudió la relación entre las características de los grupos y las métricas para cada una de las fallas (LOC y tiempo consumido). De esta manera, se intenta mostrar que dichas características no influyeron significativamente en el proceso de localización de fallas. Para esto, se calculó el coeficiente de correlación *(r)* entre las características y los valores de las métricas de cada falla. El cálculo se realizó usando el coeficiente de correlación de Pearson en la mayoría de los casos, con excepción de la característica *Profesión* donde se utilizó el coeficiente de correlación de Kendall por ser una variable de naturaleza categórica. Como se puede ver en la Tabla III, la mayoría de los resultados obtenidos muestra que no existe correlación entre las *(-.5 > r > .5)*, con

excepción de los 5 valores señalados con un asterisco. Estos valores muestran que existe una relación directa entre las métricas de la falla 1 y las características de *Avance en la carrera* y *Experiencia Laboral*. Esto puede deberse a que la falla 1: *"servidor no encontrado"* es una de las fallas más frecuentes en sistemas distribuidos y su localización resulta más sencilla a medida que los usuarios tienen mayor experiencia.

TABLA III
COEFICIENTES DE CORRELACIÓN ENTRE LAS CARACTERÍSTICAS DE LOS GRUPOS Y LAS FALLAS. LOS RESULTADOS DE LA FILA MARCADA CON EL SÍMBOLO § SON CALCULADOS EN BASE AL MÉTODO DE KENDALL MIENTRAS QUE EL RESTO EN BASE AL DE PEARSON

	Falla:	#1	#2	#3	#4	#5
LOC	Profesión §	.17	.17	-.08	-.08	-.08
	Avance en la carrera	**.54***	.49	.35	.15	.15
	Experiencia laboral	**.56***	**.52***	.30	.08	.08
	¿Ha trabajado en SOA?	-.09	.03	-.18	-.12	-.12
Tiempo	Profesión §	.30	-.15	-.30	0	.15
	Avance en la carrera	**.61***	.25	.40	.27	.35
	Experiencia laboral	**.64***	.21	.37	.33	.28
	¿Ha trabajado en SOA?	-.08	-.17	-.19	.01	-.10

B. Limitaciones

Es importante notar que este trabajo presenta ciertas limitaciones. En primer lugar, la principal limitación se encuentra relacionada con la generalización de los resultados. En este sentido, la muestra de usuarios y de fallas inyectadas en el sistema es de tamaño relativamente pequeño. Nuevos experimentos utilizando muestras de mayor tamaño podrían arrojar nuevos resultados reveladores.

Por otro lado, se encuentra la flexibilidad que posee la taxonomía y el esfuerzo que conlleva extender el enfoque. Si bien el enfoque propone una taxonomía de fallas SOA predefinida que resulta transparente al desarrollador, se ofrece la posibilidad de modificar dicha taxonomía agregando fallas propias del dominio de la aplicación. Proveer esta característica conlleva a un esfuerzo adicional que podría impactar en el desempeño de la asistencia. De la misma manera que el enfoque requiere de una taxonomía de fallas SOA, también necesita de los escenarios UCM para servir a la visualización de la falla de manera amigable al usuario. Pero la eficiencia y utilidad del enfoque varía en función del tiempo que implica construir estos escenarios. Aun así, cuanto mayor es el número de fallas localizadas, la reutilización de los escenarios y el tiempo de modelado provocan que el tiempo invertido en la instanciación del modelo SOA sea amortizado.

II. CONCLUSIÓN

Este trabajo presenta un enfoque de localización de fallas basado en una taxonomía específica para SOA. Dado que las aplicaciones orientadas a servicios poseen muchas diferencias respecto a los sistemas clásicos, resulta necesario organizar la información de las diferentes fallas que pueden ocurrir en esta taxonomía. Además, el enfoque permite unir la información de la taxonomía con la información sobre la ejecución del sistema y sus escenarios arquitectónicos correspondientes, permitiendo al desarrollador realizar un análisis en un nivel mayor de abstracción. El enfoque también facilita la comunicación de la falla y sus causas al desarrollador, permitiendo no sólo circunscribir el espacio de búsqueda en términos de líneas de código y tiempo requerido sino también asistiendo con información adicional para su solución.

Además, este trabajo plantea nuevas perspectivas de investigación en el área de localización de fallas. En este sentido, se proponen una serie de trabajos futuros que apuntan a resolver las limitaciones de este enfoque. En primer lugar, se propone realizar nuevos experimentos que involucren nuevas muestras, y de mayor tamaño, sobre el uso del enfoque con desarrolladores graduados provenientes de diferentes universidades. Esto contribuirá a la generalización de los resultados, y permitirá obtener nuevas conclusiones. En segundo lugar, el análisis y evaluación del enfoque con diferentes tipos de fallas también resulta atractivo. En particular, la aplicación de un enfoque probabilístico que complemente el algoritmo actual de localización de fallas podría impactar en la mejora de la asistencia generada.

La incorporación y evaluación de técnicas de Inteligencia Artificial como reemplazo del actual algoritmo de localización de fallas es otra perspectiva de investigación. De esta manera, los resultados actuales pueden ser tomados como línea base para la comparación con técnicas novedosas de Inteligencia Artificial aplicadas al mismo enfoque basado en taxonomía, e incluso servir para la comparación con otras herramientas o enfoques. Si bien hasta el momento no existen enfoques de localización de fallas especializados en aplicaciones orientadas a servicios, el enfoque propuesto puede sentar una línea base de comparación para futuros enfoques. Por lo tanto, entre los beneficios del enfoque propuesto podemos destacar que la utilización de una taxonomía como falla permite extender el enfoque no solo en cuanto al algoritmo utilizado sino en cuanto a la flexibilidad para incorporar nuevas fallas que no hubiesen sido contempladas en un inicio.

Finalmente, otra perspectiva futura se encuentra relacionada con explorar cómo afecta la escalabilidad y la característica distribuida de las aplicaciones orientadas a servicios en la localización de fallas. En este sentido, resulta conveniente analizar la información sobre las invocaciones de servicios, así como las topologías que surgen en el proceso de descubrimiento e invocación con el objetivo de mejorar la precisión de la localización. Otro factor interesante a analizar relacionado con la escalabilidad, es cómo varía el tiempo y las líneas de código inspeccionadas cuando se utiliza el enfoque en sistemas de alta, mediana y pequeña escala. De esta manera, podemos concluir que el presente enfoque ha contribuido a la localización de fallas en aplicaciones orientadas a servicios, además de servir como base de futuras investigaciones relevantes para el área.

REFERENCIAS

[1] ALGIRDAS AVIZIENIS, J.C. LAPRIE, AND BRIAN RANDELL. Dependability and its threats: a taxonomy. *Building the Information Society*, (July 1834), 2004.

[2] LUCIANO BARESI AND SAM GUINEA. An Introduction to Self-Healing Web Services. In *International Conference on Engineering of Complex Computer Systems*, 2005.

[3] STEFAN BRUNING, STEPHAN WEISSLEDER, MIROSLAW MALEK, STEFAN B.R., AND STEPHAN WEIß LEDER. A Fault Taxonomy for Service-Oriented Architecture. *Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, pages 367–368, 2007.

[4] R.J.A. BUHR. Use Case Maps as Architectural Entities for Complex Systems. *IEEE Transactions on Software Engineering*, 24:1131–1155, 1998.

[5] K. S. CHAN, JUDITH BISHOP, JOHAN STEYN, LUCIANO BARESI, AND SAM GUINEA. A Fault Taxonomy for Web Service Composition. In Elisabetta Nitto and Matei Ripeanu, editors, *Service-Oriented Computing - ICSOC 2007 Workshops*, pages 363–375. Springer-Verlag, Berlin, Heidelberg, 2009.

[6] USHA CHHILLAR AND SUCHETA BHASIN. Establishing Relationship between Complexity and Faults for Object-Oriented Software Systems. 8(5):437–442, 2011.

[7] HOLGER CLEVE AND ANDREAS ZELLER. Locating causes of program failures. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 342–351, New York, NY, USA, 2005. ACM.

[8] THOMAS ERL. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9] NEELAM GUPTA, HAIFENG HE, XIANGYU ZHANG, AND RAJIV GUPTA. Locating faulty code using failure-inducing chops. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 263–272, New York, NY, USA, 2005. ACM.

[10] ABDELWAHAB HAMOU-LHADJ. *Techniques to simplify the analysis of execution traces for program comprehension*. PhD thesis, Ottawa, Ont., Canada, 2006.

[11] K. ISHIKAWA. *Introduction to quality control*. Productivity Press, 1990.

[12] DAVID LO, HONG CHENG, JIAWEI HAN, SIAU-CHENG KHOO, AND CHENGNIAN SUN. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 557–566, New York, NY, USA, 2009. ACM.

[13] DAVID LO, SIAU-CHENG KHOO, AND CHAO LIU. Mining past-time temporal rules from execution traces. In *Proceedings of the 2008 International Workshop on Dynamic Analysis: held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008)*, WODA '08, pages 50–56, New York, NY, USA, 2008. ACM.

[14] LEONARDO MARIANI. A Fault Taxonomy for Component-Based Software. *Electronic Notes in Theoretical Computer Science*, 82(6):55–65, September 2003.

[15] GUILLERMO HORACIO RODRÍGUEZ, EZEQUIEL SCOTT, ÁLVARO SORIA, AND MARCELO CAMPO. Razonamiento basado en casos para la materialización de arquitecturas orientadas a servicios. In *XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Argentine Symposium on Artificial Intelligence (ASAI) (Buenos Aires, 2014)*, 2014.

[16] ÁLVARO SORIA, J. ANDRES DIAZ-PACE, AND MARCELO R. CAMPO. Tool Support for Fault Localization Using Architectural Models. In *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, pages 59–68, Washington, DC, USA, 2009. IEEE Computer Society.

[17] ÁLVARO SORIA, J. ANDRÉS DÍAZ-PACE, AND MARCELO R. CAMPO. Architecture-driven assistance for fault-localization tasks. *Expert Systems*, 32(1):1–22, 2013.

[18] W. ERIC WONG AND VIDROHA DEBROY. Software fault localization. *Encyclopedia of Software Engineering*, 1:1147–1156, 2010.

[19] GUOQUAN WU, JUN WEI, AND TAO HUANG. Towards self-healing web services composition. In *Proceedings of the First Asia-Pacific Symposium on Internetware*, Internetware '09, pages 15:1–15:5, New York, NY, USA, 2009. ACM.

[20] ANDREAS ZELLER AND RALF HILDEBRANDT. Simplifying and Isolating Failure-Inducing Input. *IEEE Trans. Softw. Eng.*, 28(2):183–200, 2002.

Ezequiel Scott es Ingeniero de Sistemas de la Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA-Argentina) desde 2012. Actualmente, se encuentra realizando el doctorado en Cs. de la Computación (UNCPBA) en el campo de Adaptación Personalizada de Mundos Virtuales para el entrenamiento en Scrum.

Álvaro Soria es investigador del Instituto Superior de Ingeniería de Software Tandil (ISISTAN-CONICET) desde 2001. Obtuvo el título de Ingeniero de Software (UNCPBA) en el año 2001 y el título de Doctor en Ciencias de la Computación (UNCPBA) en el año 2009. Sus principales áreas de investigación son Arquitecturas de Software, Diseño dirigido por la Calidad, Frameworks Orientados a Objetos y Localización de Fallas.

Marcelo Campo es profesor e investigador de la Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA-Argentina) desde 1988. Obtuvo el título de Ingeniero de Software (UNCPBA) en el año 1988 y el título de Doctor en Ciencias de la Computación (UFRGS-Brasil) en el año 1997. Sus principales áreas de investigación son Arquitecturas de Software, Agentes Inteligentes en Ingeniería de Software y Visualización de Software.