Recuperación de Trazas entre Documentos de Requerimientos y Arquitectura

Alejandro Rago*†[1], Claudia Marcos*‡[2], J. Andrés Diaz-Pace*†[3]
†*Facultad de Ciencias Exactas, UNICEN University,*
Paraje Arroyo Seco S/N, Tandil, Buenos Aires, Argentina
**Instituto Superior de Ingeniería de Software (ISISTAN), UNICEN University, CONICET*
Tandil, Argentina
‡*Instituto de Sistemas Tandil, UNICEN University, CIC*
Buenos Aires, Argentina
[1] [2] [3] {arago,cmarcos,adiaz}@exa.unicen.edu.ar

***Resumen*—Para satisfacer las necesidades de los stakeholders y adecuarse a las demandas del mercado, los desarrolladores de software deben tener en cuenta diversos atributos de calidad y asegurar su cumplimiento durante el desarrollo de un sistema. En este contexto, mantener relaciones de trazabilidad para verificar que los atributos de calidad asociados a ciertos requerimientos han sido tenidos en cuenta en el diseño arquitectónico (y viceversa) es fundamental. Desafortunadamente, establecer y mantener manualmente las trazas entre los artefactos de un sistema es una tarea compleja y tediosa. Algunos investigadores han desarrollado herramientas para identificar trazas de forma automática, pero las mismas no han sido aplicadas a documentos extensos como son los requerimientos y la arquitectura. En este trabajo se presenta una técnica para identificar trazas entre requerimientos y arquitectura basada en técnicas de procesamiento de lenguaje natural. La técnica filtra información relevante de la documentación para las trazas, haciendo hincapié en los atributos de calidad. Luego, se utiliza un algoritmo de Latent Semantic Analysis (LSA) para detectar trazas al nivel de oraciones. La técnica desarrollada fue evaluada en tres casos de estudio con resultados alentadores. Específicamente, se obtuvieron mejoras de desempeño entre 10 % y 40 % al recuperar las trazas.**

1. Introducción

La calidad de la documentación de requerimientos funcionales y decisiones arquitectónicas, realizadas durante las primeras fases del desarrollo de un sistema de software, son críticas para el éxito de un proyecto [1]. Por un lado, los documentos de requerimientos describen la funcionalidad y servicios que el sistema debe proveer de una forma clara y precisa [2]. Estos son elicitados mediante entrevistas y especificados en texto escrito en lenguaje natural, como por ejemplo con casos de uso, actuando como vehículo de comunicación entre los stakeholders a través de las etapas del ciclo de vida [3]. Por otro lado, la documentación arquitectónica describe la organización de los componentes de software, como así también las decisiones de diseño tomadas para satisfacer los atributos de calidad del sistema (e.g., performance o disponibilidad, entre otros) [1]. Los atributos de calidad condicionan los requerimientos funcionales y "moldean" la arquitectura de un sistema [4]. Existen situaciones en que losdesarrolladores toman malas decisiones por no contar con una acabada comprensión de los requerimientos y los atributos de calidad [5].

Una forma de verificar que la arquitectura y las decisiones de diseño tomadas satisfacen tanto los requerimientos funcionales como los atributos de calidad, es mantener una "buena" trazabilidad entre requerimientos, decisiones de diseño y arquitectura [6]. La trazabilidad se define como "la habilidad de describir y seguir la vida de un requerimiento tanto hacia adelante como hacia atrás (es decir, desde su origen, a través de sus especificaciones y desarrollo, su subsecuente despliegue y uso, y a través de períodos de refinamiento e iteraciones en cualquiera de estas fases)" [7]. Particularmente, la trazabilidad entre requerimientos y decisiones de diseño es importante porque permite verificar la cobertura entre los requerimientos con sus atributos de calidad asociados y el diseño arquitectónico. Cobertura se refiere a que los requerimientos estén contemplados por las decisiones de diseño tomadas, y a que las decisiones de diseño hayan sido motivadas por requerimientos existentes.

Sin embargo, determinar manualmente las trazas entre los diferentes artefactos de software suele ser una actividad costosa y tediosa para los analistas. Asimismo, dada la naturaleza evolutiva del software, es difícil mantener dichas trazas actualizadas debido a los cambios constantes realizados en el sistema. Por esta razón, varios investigadores han desarrollado técnicas que permiten descubrir trazas de manera semi-automática, utilizando algoritmos de recuperación de información y aprendizaje de máquina [8]. Dichas técnicas permiten vincular diferentes tipos de artefactos (clases y tests, o clases y componentes arquitectónicos) y operan a diferentes niveles de granularidad (paquete, clase, métodos, componentes, responsabilidades, entre otros). A pesar de la relevancia de la temática del descubrimiento

Este trabajo fue financiado parcialmente por ANPCyT (Argentina) mediante el Proyecto PICT 2015 No. 2565.

de trazas en la comunidad de Software, desafortunadamente muy pocos trabajos han explorado la recuperación de trazas entre documentación de requerimientos y arquitectura. Una posible causa está asociada al hecho que establecer estas relaciones de trazabilidad es una tarea compleja que requiere una interpretación y entendimiento de los documentos textuales [9]. Otra causa podría ser la dificultad de obtener buenos resultados porque no todos los requerimientos son relevantes para la arquitectura [10] y sólo una pequeña porción de la documentación de la arquitectura describe decisiones de diseño para satisfacer atributos de calidad [11].

En este contexto, se desarrolló una técnica para recuperar relaciones de trazabilidad entre especificaciones de casos de uso y documentación arquitectónica, la cual aprovecha peculiaridades del texto escrito en lenguaje natural para encontrar indicios que permiten inferir las trazas de forma automática. Nuestra hipótesis de trabajo se basa en dos observaciones. Primero, que gran parte de las trazas entre los documentos de requerimientos y arquitectura están vinculados semánticamente a través de atributos de calidad concretos. Segundo, que es posible limitar la búsqueda de trazas a ciertas partes de los documentos que describen explícitamente (o están indirectamente relacionadas con) los atributos de calidad de un sistema. Conceptualmente, nuestro enfoque está organizado en cuatro etapas bien definidas, a saber: análisis lingüístico, filtrado de requerimientos arquitecturales significativos, filtrado de decisiones de diseño y recuperación de trazas.

Para evaluar la técnica desarrollada, se llevaron a cabo experimentos con tres casos de estudio para determinar su desempeño al detectar decisiones de diseño y al buscar relaciones de trazabilidad entre los documentos. Como referencia, se analizaron los resultados logrados con los documentos filtrados en comparación con aquellos que utilizaron el contenido completo. Los experimentos arrojaron resultados prometedores, detectando la mayoría de las decisiones de diseño en los documentos y recuperando gran parte de las trazas sin cometer muchos errores. Asimismo, se apreció que el filtrado de los documentos permitió reducir significativamente la cantidad de falsos positivos.

El resto de este artículo está organizado de la siguiente forma. La Sección 2 introduce los conceptos principales de trazabilidad y explica su importancia para el desarrollo de sistemas. La Sección 3 enumera diversos trabajos abocados a la detección de trazas en artefactos de software. La Sección 4 explica la técnica de recuperación de trazas desarrollada, detallando cada una de las etapas de procesamiento de texto y búsqueda de relaciones de trazabilidad entre los documentos. La Sección 5 reporta los resultados de una evaluación experimental con casos de estudio. Finalmente, la Sección 6 presenta las conclusiones del trabajo, enumera limitaciones de la técnica y discute líneas de trabajo futuro.

2. Trazabilidad de Requerimientos

En el contexto de la Ingeniería de Software, la trazabilidad es la habilidad de relacionar dos artefactos de software y documentar el significado de dicha relación [6]. La trazabilidad se realiza comúnmente entre artefactos de distintas etapas de desarrollo, como por ejemplo: requerimientos y casos de test, requerimientos y diseño, diseño y código, entre otras combinaciones. La trazabilidad de requerimientos permite: (i) identificar la fuentes de un artefacto (quien lo creó y a partir de qué otros artefactos) y explorar su historia (quiénes participaron durante su desarrollo), (ii) realizar análisis de impacto mediante el seguimiento de trazas hacia los componentes afectados en respuesta a un cambio en particular en los requerimientos, o (iii) monitorear el progreso general de un proyecto, visualizando el número de requerimientos actualmente en análisis, en diseño o en implementación. Según la criticidad del sistema, mantener la trazabilidad entre los artefactos desarrollados puede no ser una opción sino una obligación impuesta por distintas entidades reguladoras [5]. Por ejemplo, los sistemas para el control de trafico aéreo o la administración de maquinaria médica están sujetos a inspecciones regulares para prevenir fallas. Desde la perspectiva de construcción de software, el análisis de trazabilidad permite verificar que el diseño arquitectónico del sistema implementa los requerimientos claves, y que los aspectos esenciales del diseño se hayan originado desde los requerimientos.

Por estas razones, la trazabilidad entre requerimientos y documentación de arquitectura es una de las más relevantes para el desarrollo de software. Por un lado, las trazas de requerimientos afectados por atributos de calidad hacia decisiones de diseño ayudan a identificar riesgos para implementar aspectos claves del sistema, estimar sus costos asociados y lograr un entendimiento más integral del esfuerzo requerido para satisfacer cada atributo de calidad [5]. Asimismo, estas trazas ayudan a seguir el progreso y grado de cumplimiento de diferentes atributos de calidad durante el desarrollo [1]. Las trazas entre requerimientos y decisiones de diseño también sirven para asegurar que todos los atributos de calidad han sido resueltos en la arquitectura y para identificar requerimientos que pueden causar conflictos durante el diseño, también denominados trade-offs [12]. Las trazas de decisiones de diseño hacia requerimientos afectados por atributos de calidad, por otro lado, permiten verificar y evaluar la completitud del diseño arquitectural y permiten entender el racional de cada elemento presente en el diseño [9]. Desafortunadamente, el costo necesario para crear y mantener relaciones de trazabilidad en un sistema de software en evolución puede ser alto [9]. Los analistas y arquitectos frecuentemente encuentran la trazabilidad como una actividad compleja, propensa a errores y que requiere mucho esfuerzo. Además, los beneficios de las relaciones de trazabilidad por lo general son desestimados debido a la falta de soporte de herramientas para identificar, mantener y utilizar las trazas [5].

3. Trabajos Relacionados

Durante los últimos años, diferentes autores han investigado técnicas para recuperar (o generar) trazas entre artefactos, como por ejemplo identificar semi-automáticamente

todas las clases implementadas relevantes a un requerimiento particular. También se ha estudiado la trazabilidad desde una perspectiva de conformidad, permitiendo identificar los requerimientos que cumplen con una norma específica de un ente regulador. Si bien se han desarrollado diferentes técnicas para automatizar la creación y el mantenimiento de trazas a diferentes niveles y entre distintos tipos de artefactos, pocos autores han tratado las trazas entre requerimientos afectados por atributos de calidad en las especificaciones y decisiones de diseño descriptas en documentos de arquitectura de software. Esto se debe a que este tipo de requerimientos son más difíciles de identificar que los requerimientos meramente funcionales, ya que afectan diferentes tipos de funcionalidades y tienen un impacto de amplio alcance en el sistema [3]. Asimismo, los documentos de arquitecturas están dirigidos a un gran número de stakeholders e incluyen diferentes secciones que no son relevantes para la búsqueda de trazas [11]. En esta línea de razonamiento, los trabajos relacionados se pueden organizar en dos grupos. El primer grupo está formado por herramientas cuyo objetivo es analizar documentos de requerimientos para detectar y clasificar requerimientos no funcionales, atributos de calidad o incluso requerimientos arquitecturales significativos. El segundo grupo abarca trabajos relacionados a la identificación semi-automática de relaciones de trazabilidad con diversos artefactos producidos durante el desarrollo, tales como el código fuente y los casos de prueba.

Con respecto al primer grupo, existen diferentes enfoques para buscar requerimientos arquitecturales en especificaciones de requerimientos mediante la utilización de algoritmos de Procesamiento de Lenguaje Natural (NLP) y Aprendizaje de Máquina (ML) [13]. Theme/Doc [14], por ejemplo, combina algunas técnicas tradicionales de NLP con información provista por los analistas, produciendo visualizaciones para facilitar el proceso de análisis de los requerimientos que impactan en más de un documento. La herramienta EAMiner [15] también permite a los analistas identificar potenciales requerimientos que impactan en la arquitectura de forma semi-automática. Para ello, trabaja en dos etapas, primero etiquetando cada palabra con su "parte del discurso" (POS) y luego utilizando una taxonomía para categorizar cada palabra de acuerdo a su significado semántico y su relación con aspectos del sistema preestablecidos (por ejemplo, seguridad, consistencia, performance, entre otros). La idea de utilizar una taxonomía para agrupar conceptos semánticamente similares podría llegar a aplicarse sobre los documentos de diseño y de esa forma agrupar patrones o tácticas arquitectónicas.

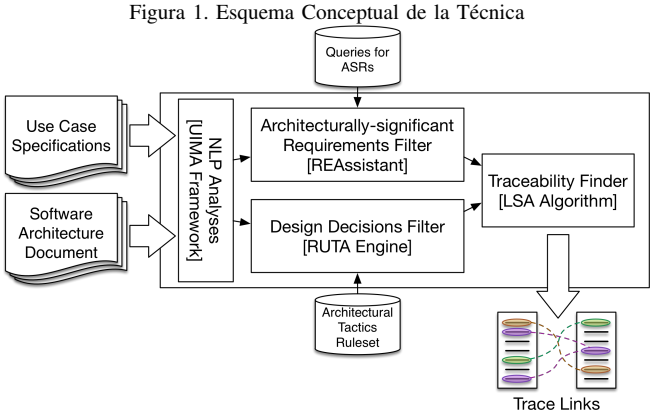
Otra herramienta interesante es *REAssistant* [16], la cual permite asistir a los analistas en la recuperación de requerimientos arquitecturales mediante un lenguaje y motor de consultas (es decir, de búsqueda) enriquecido con semántica. Dicho lenguaje permite caracterizar los diferentes requerimientos arquitecturales en término de constructos del lenguaje natural y conceptos específicos del dominio. Un aspecto importante de este lenguaje de consultas es que podría llegar a utilizarse para recuperar conceptos de diseño en documentos de arquitectura con pocas modificaciones. Otros trabajos han investigado el reconocimiento de atributos de calidad en documentos de requerimientos. Entre ellos, se destaca el uso de algoritmos de clasificación de texto para distinguir diferentes tipos de atributos de calidad [17], [18], así como también el uso de técnicas de clasificación semi-supervisadas para distinguir entre requerimientos funcionales y no funcionales de software [19], [20].

Con respecto al segundo grupo, Antoniol y otros sentaron la base de muchos trabajos relacionados que aplican técnicas clásicas de Information Retrieval (IR) para calcular la similitud entre dos artefactos y así identificar trazas [21]. Particularmente, su trabajo se enfocó en la identificación de trazas entre requerimientos y código fuente. Más recientemente, en cambio, se han utilizado técnicas de Vector Space Model (VSM) combinadas con métricas de distancia entre vectores (por ejemplo, coseno) y diccionarios de sinónimos para establecer las trazas. En [22], Hayes et al. introdujeron RETRO, una herramienta diseñada para manejar la trazabilidad de requerimientos y otros artefactos. Dicha herramienta implementa un conjunto de técnicas para la recuperación de trazas, a saber: TF-IDF, VSM, diccionarios de sinónimos, entre otras. De forma similar, la herramienta POIROT utiliza modelos jerárquicos para identificar relaciones de trazabilidad entre varios tipos de artefactos con los requerimientos en una interfaz Web [23]. Asimismo, la herramienta desarrollada por De Lucia et al. aprovecha las bondades de VSM combinadas con el feedback del usuario para mejorar la detección de trazas entre artefactos textuales que contengan un número acotado de términos [24]. Finalmente, debido a que las técnicas de IR y VSM producen demasiadas trazas incorrectas, también se han investigado algoritmos más sofisticados como Latent Semantic Analysis (LSA) que permiten analizar la semántica del texto. Marcus et al. desarrolló una herramienta que recupera vínculos desde el código fuente a los documentos de diseño utilizando técnicas de NLP en ambos documentos para identificar sustantivos y LSA para descubrir vínculos entre los documentos en base a dicha información [25]. De forma similar, Kuhn et al. exploró el uso de LSA para enriquecer la documentación de un sistema a partir del código fuente y así poder identificar agrupamientos de clases con sus relaciones [26].

4. Técnica para Buscar Trazas

Para abordar la problemática de trazabilidad entre artefactos textuales, se desarrolló una técnica que procesa documentos (casos de uso y documentos de arquitectura) con el fin de determinar información relevante para la trazabilidad y aplica una técnica automatizada para recuperar las trazas. En el caso de los documentos de requerimientos, se busca recuperar los requerimientos arquitecturales de forma tal de trabajar con aquellas porciones de las especificaciones que están relacionados con los atributos de calidad del sistema. En el caso de los documentos de arquitectura, se busca recuperar las decisiones de diseño tomadas para satisfacer uno o más atributos de calidad, tales como tácticas, patrones o estilos arquitectónicos y tecnologías. Para ambos documentos, se utilizaron diversas técnicas de procesamiento de

Figura 1. Esquema Conceptual de la Técnica



lenguaje natural (NLP) para analizar el texto y un lenguaje basado en reglas para recuperar la información deseada de los mismos. Luego, una vez filtrados los documentos, se utiliza un algoritmo para comparar las oraciones y descubrir las trazas entre los requerimientos arquitecturales y las decisiones de diseño. La comparación tiene como objetivo encontrar el «factor común» entre las partes de la traza: las responsabilidades funcionales del sistema. Es decir, todo requerimiento arquitectural tiene una parte funcional y toda táctica esta aplicada para satisfacer los atributos de calidad de una responsabilidad del sistema.

La Figura 1 muestra un diagrama conceptual de la técnica. Como entrada, la técnica recibe un conjunto de documentos de requerimientos, materializados en especificaciones de casos de uso, y un documento de arquitectura. Aunque éste último no requiere un formato predefinido, se prefirió utilizar documentos que sigan la organización de secciones definidas en [11]. Inicialmente, el componente NLP ANALYSES extrae el contenido de los documentos y genera un conjunto de meta-anotaciones lingüísticas, tales como los rangos de párrafos y oraciones, propiedades de las palabras (sustantivo, verbo, adjetivo, etc.), estructuras sintácticas y semánticas (dependencias, predicados, argumentos, etc.). Este componente fue implementado con el framework UIMA [27] para simplificar la creación de pipelines de análisis y el almacenamiento de la información.

El segundo componente, denominado ARCHITECTURALLY-SIGNIFICANT REQUIREMENTS FILTER, utiliza la herramienta *REAssistant* para filtrar aquellas porciones de los casos de uso que presentan requerimientos arquitecturales, es decir, que tienen un impacto en la arquitectura del sistema y por lo tanto están afectados por atributos de calidad. La herramienta aprovecha los análisis de NLP y realiza las búsquedas utilizando un conjunto de consultas pre-cargadas, tales como *Performance*, *Distribución*, *Correctitud*, *Persistencia*, entre otros. Como resultado, se genera un conjunto de oraciones extraídas de las especificaciones que representan a los requerimientos arquitecturales.

El tercer componente, denominado DESIGN DECISIONS FILTER, aprovecha el lenguaje de reglas provisto en RUTA [28] para identificar diferentes tipos de decisiones de diseño presentes en los documentos de arquitectura. Con este propósito, se estableció un conjunto de reglas organizadas por atributo de calidad para reconocer oraciones que describan distintos tipos de tácticas y estilos arquitectónicos, tales como detección de fallas (ping-echo, heart-beat), recuperación de fallas (replicación activa), administración de recursos (cache, colas de prioridad), interoperabilidad (fachadas e interfaces unificadas), entre otras. Las reglas codificadas con RUTA aprovechan la información del NLP para aumentar la capacidad de detección de la técnica. La salida de este componente es un conjunto de oraciones que representan a las decisiones de diseño.

Por último, el componente TRACEABILITY FINDER tiene la responsabilidad de encontrar relaciones de trazabilidad entre las oraciones filtradas de los documentos de requerimientos y arquitectura (detectadas con los dos componentes previos). La técnica desarrollada utiliza un algoritmo de Latent Semantic Analysis (LSA) [29] para analizar los términos utilizados por todas las oraciones y establecer grados de similitud entre las mismas. Si dos oraciones (provenientes de los requerimientos y la arquitectura, respectivamente) superan un umbral predeterminado de similitud, entonces son sugeridas como una traza candidata entre los documentos.

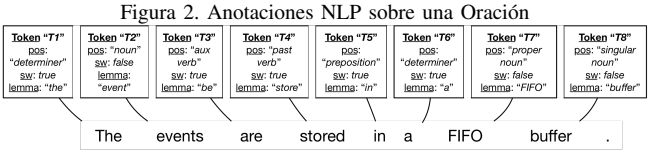
Las siguientes sub-secciones explican el funcionamiento de cada uno de estos componentes con mayor detalle.

4.1. Análisis de Lenguaje Natural

Este componente tiene la responsabilidad de llevar a cabo el análisis léxico y sintáctico de los documentos de requerimientos y arquitectura mediante la aplicación de algoritmos de NLP. El objetivo es descomponer el texto en constructos lingüísticos que faciliten la posterior detección de fragmentos de información relevantes para la recuperación de trazas. Para implementar este componente se utilizó el framework UIMA, una solución para el procesamiento de información no estructurada que provee interfaces bien definidas para adaptar los algoritmos de NLP y soporta tanto la importación de los documentos como la exportación de los resultados. En UIMA, la estructura de «anotadores» permite encapsular los diferentes algoritmos y encadenarlos en una misma secuencia de análisis compuesta, la cual puede ser adaptada según el tipo de documento que se procesa (por ejemplo, ciertos módulos para los documentos de requerimientos y otros para el documento de arquitectura). La salida de cada anotador es almacenada en el *Common Analysis Structure* (CAS), un repositorio que abstrae la meta-información de los documentos basada en la noción de anotaciones. Una anotación delimita una región específica del documento, marcando el principio y el final de un concepto de interés, así como también puede contener propiedades internas del concepto.

En nuestra técnica, se implementaron módulos que permiten importar los documentos de requerimientos y arquitectura a partir de archivos DOC, PDF o incluso algunos formatos específicos para casos de uso. También se adaptaron diversos algoritmos de NLP para generar anotaciones

Figura 2. Anotaciones NLP sobre una Oración



de las oraciones, palabras, partes del discurso, stop-words y lemas presentes en los documentos. Para identificar las oraciones y dividirlas en palabras, se utilizó el algoritmo *sentence splitter* y *tokenizer* provistos por OpenNLP[1]. El análisis de partes del discurso, también denominadas etiquetas POS, permite establecer el rol sintáctico de cada palabra en una oración. Esto significa saber si una palabra es un sustantivo, verbo, adjetivo, adverbio, proposición, entre otros roles. La integración de esta funcionalidad a UIMA fue efectuada con el algoritmo *postagger* implementado en Stanford CoreNLP[2]. En este caso, en vez de registrar las etiquetas como una nueva anotación, se decidió almacenar esta información como una propiedad de las palabras. Asimismo, se adaptaron algoritmos de *stopwords* y *lemmatizer* con el objetivo de marcar las palabras con poco valor discriminativo (pronombres, preposiciones, artículos, entre otras) y reducir las palabras a formas inflexionales (es decir, su raíz morfológica). Específicamente, se utilizaron diccionarios de stopwords públicos y la implementación de *lemmatizer* de la librería Mate-Tools[3]. La Figura 2 ilustra el resultado de los análisis de NLP realizados sobre una oración extraída de un documento de arquitectura. La oración es adornada con múltiples anotaciones *Token*, indicando el principio y final de cada palabra. Asimismo, cada una de las anotaciones se completan con sus respectivas partes del discurso, palabras irrelevantes y raíces morfológicas, representadas como propiedades de la anotación (pos, sw y lemma, respectivamente).

4.2. Filtrado de los Requerimientos Textuales

Este componente tiene como objetivo identificar aquellas partes de los requerimientos textuales que están relacionados con, o se ven afectados por, atributos de calidad del sistema. En vez de implementar dicha funcionalidad, se decidió integrar una herramienta existente denominada *REAssistant*[4], la cual permite reconocer requerimientos arquitecturales a partir del análisis semi-automático de los escenarios de casos de uso. La herramienta intenta comprender el significado de las interacciones descriptas en los escenarios para detectar funcionalidad que puede llegar a ser tenida en cuenta en el diseño arquitectónico, incluyendo aspectos tales como PERSISTENCIA, SEGURIDAD, PERFORMANCE, entre otros. Inicialmente, la herramienta identifica los predicados y argumentos de las oraciones en base a las anotaciones NLP generadas previamente. Luego, a partir de la descripción

1. Disponible en https://opennlp.apache.org
2. Disponible en http://stanfordnlp.github.io/CoreNLP
3. Disponible en http://code.google.com/p/mate-tools
4. Disponible en https://code.google.com/p/reassistant

Figura 3. Consulta para el Reconocer un Requerimiento Arquitectural

Architectural Requirement Queries for *Performance*

Direct Query #1

```
select S from [#Sentence#] as S, [#Token#] as T
where for T(lemma='response' or lemma='second' or
lemma='time' or lemma='delay' or lemma='throughput' or
lemma='latency' or lemma='deadline')
where T.begin >= S.begin where T.end <= S.end
```

Indirect Query #2

```
select S from [#Sentence#] as S, [#DomainAction#] as DA
where for DA(label='Calculation')
where DA.begin >= S.begin where DA.end <= S.end
```

Indirect Query #3

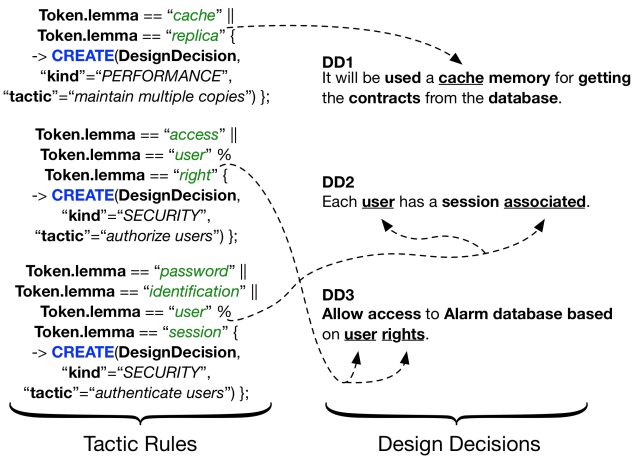
```
select S from [#Sentence#] as S,
[#DomainAction#] as DA, [#Token#] as T
where for DA(label='Process')
where for T(lemma='result' or lemma ='value')
where T.begin >= S.begin where T.end <= S.end
where DA.begin >= S.begin where DA.end <= S.end
```

de los predicados y argumentos, la herramienta categoriza las oraciones usando conceptos denominados *Acciones de Dominio* (*DA*), los cuales forman parte de una taxonomía de tipos de interacciones específicas de los casos de uso. La taxonomía esta compuesta por diversas *DAs*, tales como: «retrieval», «writing», «display», «entry», «selection», «verification», «calculation», entre otras. Este análisis adicional del contenido de los requerimientos textuales fue agregado al pipeline de NLP desarrollado en UIMA.

Una vez completado el procesamiento del texto, los analistas pueden realizar búsquedas de requerimientos arquitecturales utilizando un lenguaje de consultas símil a SQL. El rol del analista es codificar dichas consultas en base a anotaciones y sus propiedades, de forma tal de caracterizar requerimientos arquitecturales individuales. Las consultas pueden utilizar diferentes tipos de anotaciones, yendo desde constructos léxicos como los *Tokens* hasta conceptos semánticos como las *DAs*. La ventaja de utilizar las *DAs* en las consultas es que permiten encontrar requerimientos arquitecturales ocultos en las especificaciones de forma sencilla y precisa, especialmente en aquellas oraciones donde las referencias a los atributos de calidad son más sutiles y suelen ser mencionados de forma implícita.

La herramienta *REAssistant* cuenta con un conjunto de consultas pre-cargadas para identificar requerimientos arquitecturales particulares en las especificaciones de casos de uso. Sin embargo, estas consultas pueden ser personalizadas por los analistas para adaptarse a diferentes dominios de sistemas, tales como sistemas embebidos, críticos, o de negocios, entre otros. Las consultas describen cómo los requerimientos arquitecturales se relacionan semánticamente con las expresiones escritas en lenguaje natural. Existen dos tipos de consultas, a saber: (i) *consultas directas*, encargadas de detectar referencias explícitas a algún aspecto arquitectónico o de diseño presente en el requerimiento textual, y (ii) *consultas indirectas*, encargadas de detectar *DAs* que, debido a una interpretación semántica del requerimiento, pueden estar relacionadas con aspectos arquitectónicos del sistema. La Figura 3 presenta tres consultas provistas en

Figura 4. Reglas para Identificar Decisiones de Diseño



REAssistant para buscar un requerimiento arquitectural. La primera consulta intenta encontrar partes de los casos de uso vinculados con aspectos de PERFORMANCE a través del análisis de los lemas de los *Tokens*, tales como «response» y «second», entre otras. La segunda y tercera consulta tratan de revelar acciones potencialmente afectadas por restricciones de PERFORMANCE mediante la búsqueda de *DAs* de «calculation» y «process».

4.3. Filtrado del Documento de Arquitectura

El objetivo de este componente es recuperar aquellas oraciones del documento de arquitectura que describen decisiones tomadas por los arquitectos, particularmente si las mismas tienen un impacto sobre uno o más atributos de calidad. La búsqueda de decisiones de diseño se realiza a nivel de oración, ya que por lo general, el fragmento de texto donde se describe una decisión es pequeño y solo contiene algunas palabras clave específicas del dominio de arquitecturas de software. El resto del contenido del documento suele tratar sobre cómo esa decisión se materializó en el sistema. Como el documento de arquitectura está dirigido a diversos stakeholders y presenta descripciones técnicas de diversas vistas desarrolladas en consecuencia de las decisiones de diseño, se consideró que los párrafos son muy abarcativos para llevar a cabo este análisis.

Particularmente, se definió un conjunto de reglas que permiten identificar las decisiones de diseño de forma automática en base a terminología específica de arquitecturas de software. Las reglas están organizadas según la clasificación de atributos de calidad y tácticas arquitectónicas propuesta por Bass et al. [1]. Se estableció un conjunto de reglas para identificar decisiones de diseño relacionadas con atributos de calidad tales como: DISPONIBILIDAD, MODIFICABILIDAD, INTEROPERABILIDAD, TESTEABILIDAD, PERFORMANCE, USABILIDAD y SEGURIDAD. Para cada uno de estos grupos, se definieron reglas individuales para cada táctica arquitectural que permite satisfacerlos y las diferentes estrategias de implementación. A modo de ejemplo, en el caso DISPONIBILIDAD, se definieron reglas asociadas a las tácticas de *Detectar Fallas*, *Recuperarse de Fallos* y *Prevenir Fallos*. En el caso de *Detectar Fallas*, se codificaron dos reglas, una para buscar términos relacionados con la estrategia de «heartbeat» y otra para «ping-echo».

Desde un perspectiva de implementación, este componente de la técnica se materializó utilizando RUTA[5], una tecnología basada en UIMA que permite escribir reglas de búsqueda combinando un lenguaje de scripting muy potente con las anotaciones generadas por el análisis lingüístico. Este lenguaje permite extraer información a partir de expresiones y variables, importar y ejecutar componentes externos, como así también utilizar diccionarios de términos. Cada regla está compuesta por dos partes: un patrón de coincidencia y un conjunto de acciones. Los patrones de coincidencia expresan una serie de anotaciones a encontrar en el texto, mientras que las acciones asociadas permiten ejecutar operaciones tales como crear una nueva anotación. Algunas reglas son simples, en el sentido de que buscan términos claves a partir de las propiedades de las anotaciones *Token*, como son los lemas. Otras reglas, sin embargo, son más complejas y tienen en cuenta además del término su función sintáctica dentro de la oración mediante el análisis de la propiedad POS. Este tipo de reglas presenta ventajas con respecto a las técnicas tradicionales de IR basadas en keywords, ya que aprovechan información producida por el NLP.

La Figura 4 muestra tres reglas para detectar decisiones de diseño de PERFORMANCE y SEGURIDAD. En el caso de la primera regla, el objetivo es reconocer oraciones que contengan palabras cuyo lema sea «cache» o «replica». Si se encuentran dichas palabras en el documento de arquitectura, es muy probable que los arquitectos hayan utilizado una táctica de «mantener múltiples copias» para acelerar alguna funcionalidad del sistema. Entonces, cuando se encuentran estas palabras, se crea una nueva anotación denominada *DesignDecision* indicando la presencia de una potencial táctica de PERFORMANCE. En las otras reglas, en cambio, se utilizan operadores del lenguaje RUTA para encontrar fragmentos de texto que contengan dos palabras con lemas determinados en la misma oración. Esta característica posibilita diferenciar las tácticas de autenticar usuarios y autorizar usuarios, ambas relacionadas con SEGURIDAD. La táctica de autorizar usuarios, la cual se refiere al acceso controlado a la funcionalidad, se identifica cuando se encuentran las palabras «user» y «right» juntas en la misma oración. La táctica de autenticar usuarios, la cual se refiere a permitir el ingreso de un usuario al sistema, se identifica cuando se encuentran las palabras «user» y «session» juntas.

Para contar con un conjunto representativo de reglas sin sesgar la investigación, se consultó a un grupo externo de arquitectos y analistas funcionales, los cuales se encargaron de la codificación de las reglas en base a la taxonomía definida en [1]. Los participantes de este ejercicio recibieron una explicación acerca de las anotaciones disponibles y la sintaxis del lenguaje de reglas. Cada sujeto desarrolló

5. https://uima.apache.org/ruta.html

un conjunto inicial de reglas según su conocimiento de arquitecturas de software y experiencia en la industria. Luego, se realizó una reunión donde se validaron las reglas propuestas y se consolidaron en un único conjunto. En total, se escribieron 248 reglas para 7 tipos de atributos de calidad.

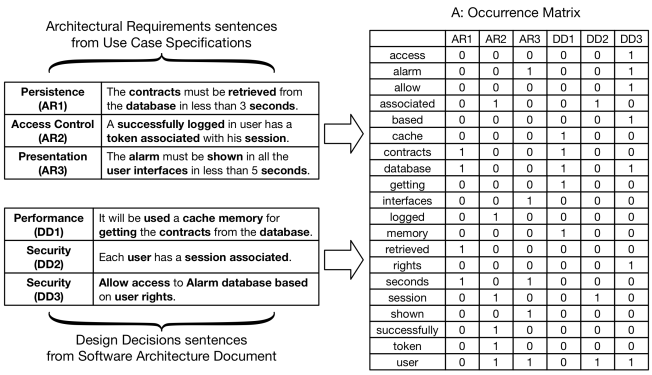
4.4. Búsqueda de Relaciones de Trazabilidad

Este componente tiene el objetivo de encontrar trazas entre las especificaciones de requerimientos y el documento de arquitectura. La técnica se enfoca en encontrar relaciones de trazabilidad a nivel de oración mediante la ayuda del filtrado de los requerimientos arquitecturales en los casos de uso y las decisiones de diseño en el documento de arquitectura. Para realizar dicha búsqueda, se utilizó un algoritmo denominado *Latent Semantic Analysis* (LSA) [29], el cual permite analizar la relación entre un conjunto de documentos y los términos que contienen mediante un conjunto de conceptos que los caracterizan. El algoritmo asume que las palabras que tienen significado similar tienden a aparecer en porciones de texto parecidas, y utilizando esta noción, simplifica el modelo de representación de los documentos para calcular la similitud entre los mismos. Nuestra hipótesis es que los documentos de arquitectura y de requerimientos tienen un vocabulario similar debido a que "hablan" sobre la misma funcionalidad dentro del dominio del sistema, por lo que debe haber palabras repetidas o con el mismo significado en ambos documentos.

El funcionamiento del algoritmo LSA depende de una matriz de ocurrencia «A» construida a partir de los documentos, la cual describe la frecuencia de términos por oración, y de la aplicación de una técnica matemática denominada descomposición de valores singulares (SVD), la cual permite reducir el número de filas (es decir, los términos) de la matriz de ocurrencia mientras se preserva la estructura de similitud entre las columnas (es decir, las oraciones). Las oraciones entonces se pueden comparar calculando la distancia entre aquellos dos vectores columna que los representen, ya que los valores en cada uno de ellos indican el grado de presencia de los conceptos. Aquellas oraciones que obtengan una distancia cercana a 1 son consideradas muy similares (ya que comparten muchos conceptos) y si la distancia es cercana a 0 significa que son disimilares (ya que no comparten conceptos).

Por ejemplo, considere las oraciones ilustradas en la Figura 5. En la parte superior izquierda, se puede apreciar un conjunto de tres oraciones provenientes de los casos de uso y que han sido marcadas como requerimientos arquitecturales de PERSISTENCIA, CONTROL DE ACCESO y PRESENTACIÓN. Análogamente, en la parte inferior izquierda se encuentran tres oraciones provenientes del documento de arquitectura que han sido marcadas como decisiones de diseño de PERFORMANCE y SEGURIDAD. Para completar la matriz A, solamente se consideran aquellos términos que no son stop-words y además se aplican tareas de posprocesamiento para transformar los términos a minúsculas y reducirlos a formas inflexionales mediante stemming.

Figura 5. Matriz de Ocurrencias



	AR1	AR2	AR3	DD1	DD2	DD3
access	0	0	0	0	0	1
alarm	0	0	1	0	0	1
allow	0	0	0	0	0	1
associated	0	1	0	0	1	0
based	0	0	0	0	0	1
cache	0	0	0	1	0	0
contracts	1	0	0	1	0	0
database	1	0	0	1	0	1
getting	0	0	0	1	0	0
interfaces	0	0	1	0	0	0
logged	0	1	0	0	0	0
memory	0	0	0	1	0	0
retrieved	1	0	0	0	0	0
rights	0	0	0	0	0	1
seconds	1	0	1	0	0	0
session	0	1	0	0	1	0
shown	0	0	1	0	0	0
successfully	0	1	0	0	0	0
token	0	1	0	0	0	0
user	0	1	1	0	1	1

Figura 6. Calculo de Similitud con la Matriz de Aproximación

V: Low-rank Approximation

AR1	AR2	AR3	DD1	DD2	DD3
0.257	0.49	0.447	0.168	0.49	0.472
-0.61	0.41	-0.12	-0.47	0.41	-0.24
-0.33	-0.29	0.624	-0.44	-0.29	0.362
0.454	0.044	0.477	-0.29	0.044	-0.69

Calculating Cosine Similarity

	DD1	DD2	DD3
AR1	0.794	0.092	0.339
AR2	0.004	0.993	0.107
AR3	0.06	0.231	0.339

Luego de construir dicha matriz, el algoritmo determina una aproximación de menor dimensión de la matriz A. Esta aproximación es interesante debido a que las matrices de ocurrencia suelen ser extensas y ralas, por lo que su utilización requiere de una importante capacidad de cómputo. Asimismo, las matrices de ocurrencia suelen ser ruidosas ya que contienen muchos términos que no suelen aportar información a la comparación de documentos (por ejemplo, términos que aparecen unas pocas veces de forma anecdótica). En algunos casos, la matriz de aproximación puede incluso llegar representar mejor cada una de las oraciones, ya que elimina las filas ruidosas de la matriz de ocurrencia. Otra ventaja de la aproximación es que debido a la reducción, algunas dimensiones son combinadas representando múltiples términos, los cuales por lo general tienen el mismo significado, permitiendo mitigar substancialmente los problemas de sinonimia y polisemia.

Desde una perspectiva algebraica, la descomposición en valores singulares trata de encontrar matrices que satisfagan la ecuación $A = U \cdot S \cdot V^t$. Por un lado, la matriz S contiene el vector de valores propios de A en la diagonal. Por otro lado, la matriz transpuesta V es la aproximación que se utiliza para comparar las oraciones. Dos aspectos a tener en cuenta al aplicar LSA es el número de dimensiones o "conceptos" que se desean utilizar a la hora de descomponer la matriz A y el umbral de similitud para comparar los documentos. La

parametrización de dimensiones «d» depende de la cantidad de documentos a analizar y el número de términos únicos que estos utilizan. Un valor «d» pequeño puede eliminar conceptos relevantes para la comparación, mientras que un valor grande puede llegar a tener los mismos problemas que la matriz de ocurrencia. Asimismo, otro parámetro a tener en cuenta es el umbral de similitud «u» utilizado para determinar si existe una traza entre dos oraciones. Como cada columna representa una oración, para realizar las comparaciones se debe determinar la métrica de distancia entre vectores más adecuada y su respectivo umbral de similitud entre par de vectores. Una alternativa popular en la literatura es la métrica del coseno [13], la cual calcula el coseno del ángulo entre dos vectores en espacios n-dimensionales.

A modo ilustrativo, considere las oraciones introducidas en el ejemplo anterior estableciendo un número de dimensiones $d = 4$ y el umbral $u = 0{,}75$. La Figura 6 presenta la matriz V resultante luego de realizar la descomposición de la matriz A con SVD. En la parte inferior se muestran los valores de similitud entre cada oración de requerimientos y arquitectura calculados con la métrica del coseno. Mediante el umbral, se puede determinar si existe una traza candidata entre los pares de oraciones. Luego del análisis, solo dos pares logran superar el umbral y ser sugeridas como trazas: (AR1, DD1) y (AR2, DD2). En la primer traza, a pesar de que «cache» fue la palabra utilizada para marcar la oración como una decisión de diseño, esta palabra no fue determinante para sugerir la traza. En cambio, las palabras decisivas para encontrar la traza fueron «contracts» y «database», las cuales están relacionadas con la funcionalidad del sistema y actuaron como nexo entre los requerimientos y la arquitectura. En la segunda traza, las palabras «user» y «session» fueron las que revelaron una decisión de diseño. Aunque las mismas fueron utilizadas en la comparación por LSA, la palabra «associated» también contribuyó a la detección de la traza.

5. Evaluación Experimental

Para determinar el desempeño de la técnica desarrollada, se llevaron a cabo tres casos de estudio con sistemas provenientes de diferentes dominios. El objetivo principal de la evaluación fue comparar las trazas generadas automáticamente con aquellas establecidas por expertos del área. Asimismo, un objetivo secundario de la evaluación fue cuantificar las ventajas de nuestra técnica en contraste con los enfoques de recuperación de trazas tradicionales. La hipótesis experimental es que la técnica es capaz de detectar una gran parte de las trazas entre documentos de requerimientos y el documento de arquitectura, cometiendo un número de errores manejable por los analistas. Para poner esta hipótesis a prueba, como así también guiar el análisis de resultados, se plantearon las siguientes preguntas de investigación:

1. ¿Pueden las reglas detectar la mayoría de las decisiones de diseño en el documento de arquitectura con una precisión alta?
2. ¿Puede la técnica de LSA detectar la mayoría de las trazas entre los documentos de requerimientos y arquitectura?
3. ¿Existen ventajas significativas al remover aquellas porciones de los documentos que no están afectadas o relacionadas con atributos de calidad?

Cuadro 1. PROPIEDADES DE LOS CASOS DE ESTUDIO

PROPIEDAD	CASO DE ESTUDIO		
	ADV. BUILDER	PET STORE	MSLITE
DOC. DE REQUERIMIENTOS			
Número de Páginas	5	8	38
Número de Casos de Uso	4	6	22
Tipos de Requerimientos Arquitecturales	8	8	8
Número de Oraciones Afectadas	64	63	99
DOC. DE ARQUITECTURA			
Número de Páginas	36	24	140
Tipos de Decisiones de Diseño	6	5	7
Número de Oraciones Afectadas	115	35	212
TRAZABILIDAD			
Número de Trazas	55	38	456

Los casos de estudio realizados en esta evaluación comprendieron el análisis manual y automático de la documentación de tres sistemas, a saber: ADVENTURE BUILDER, PET STORE y MSLITE. Los dos primeros sistemas fueron desarrollados por Sun Microsystems en el contexto del programa Java Blueprints [30]. Particularmente, ADVENTURE BUILDER y PET STORE son sistemas para administrar las operaciones de una agencia de viajes y manejar un sitio web de venta de mascotas, respectivamente. El tercer sistema, en cambio, fue desarrollado en conjunto por alumnos de la universidad de Carnegie Mellon e ingenieros de Siemens [31]. MSLITE es un sistema que monitorea y controla automáticamente varias funciones de un edificio, tales como: calefacción, ventilación, aire acondicionado, acceso y seguridad. Estos sistemas fueron seleccionados como casos de estudio por varias razones. En primer lugar, los mismos cuentan con una documentación muy completa disponible públicamente en la Web, incluyendo especificaciones de requerimientos, documento de arquitectura y código fuente. En segundo lugar, otros investigadores han utilizado estos casos de estudio para realizar evaluaciones empíricas [32], [33]. En tercer lugar, los documentos de requerimientos y el documento de arquitectura de cada uno de estos sistemas fueron inspeccionados previamente por expertos para verificar su aptitud para la búsqueda de relaciones de trazabilidad entre requerimientos y arquitectura. La Tabla 1 resume las propiedades los tres casos de estudio, detallando el tamaño y la complejidad de la documentación e informando el número oraciones afectadas por requerimientos arquitecturales y decisiones de diseño.

En este tipo de evaluaciones, es común que el desempeño de una nueva técnica sea contrastado con una solución «ideal», la cual establezca de antemano los resultados correctos. Por lo tanto, se requería contar con tres soluciones de referencia, una para cada caso de estudio. Las soluciones de referencia deben incluir tanto las decisiones de diseño presentes en el documento de arquitectura como las trazas

conformadas por pares de oraciones provenientes de los documentos de requerimientos y arquitectura. Considerando que la información original de los casos de estudio no es lo suficientemente detallada para derivar una solución de referencia, se les solicitó esta tarea a un grupo de analistas externos. Para facilitar su construcción y evitar sesgos, los analistas tuvieron acceso a documentación adicional del sistema y al código fuente de la aplicación para establecer las decisiones de diseño y las trazas con los requerimientos de cada sistema con mayor confianza.

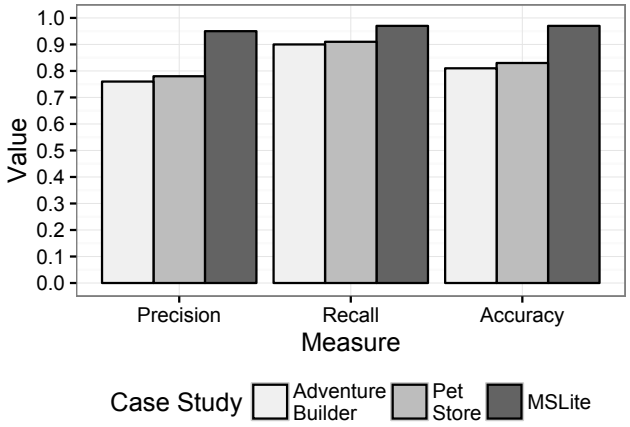
El desempeño de la técnica fue analizado mediante métricas cuantitativas provenientes del área de recuperación de información [34], las cuales son frecuentemente utilizadas para evaluar técnicas de clasificación de texto y recuperación de trazas [8], [18], [19], [24]. Básicamente, en base a valores de predicción binaria (tp: verdaderos positivos, tn: verdaderos negativos, fp: falsos positivos y fn: falsos negativos) se computan tres tipos de métricas: *precision*, *recall* y *accuracy*. La *precision* y el *recall* son dos indicadores interesantes para determinar la calidad de la salida de la técnica. En esta evaluación, *precision* mide cuántas de las decisiones de diseño o trazas detectadas son correctas, mientras que el *recall* mide cuántas decisiones de diseño o trazas fueron detectadas de la totalidad de decisiones o trazas existentes. Esto significa que obtener una buena *precision* implica que la técnica no comete errores, mientras que obtener un buen *recall* implica recuperar la mayoría de las decisiones de diseño o trazas. Otra métrica útil es *accuracy*, la cual no solamente mide la tasa de correctitud de las decisiones o trazas correctas, sino también de aquellas que son incorrectas y no debieran ser detectadas. Las fórmulas para las métricas son: $precision = \frac{tp}{tp+fp}$, $recall = \frac{tp}{tp+fn}$ y $accuracy = \frac{tp+tn}{tp+fp+tn+fn}$.

La evaluación fue dividida en dos partes. La primer parte tiene como propósito comparar las decisiones de diseño detectadas por la técnica en comparación con aquellas detectadas por analistas expertos. La segunda parte tiene como propósito comparar las trazas recuperadas por la técnica con aquellas definidas en las soluciones de referencia. Con respecto a la detección de requerimientos arquitecturales, una parte esencial de la recuperación de trazas, no se efectuó una evaluación experimental porque dicha herramienta ya ha sido analizada positivamente en otro trabajo [16].

5.1. Detección de Decisiones de Diseño

Para establecer el desempeño al filtrar el documento de arquitectura, se ejecutaron las reglas definidas con RUTA para cada atributo de calidad y táctica arquitectural sobre la documentación de los tres casos de estudio. Como las reglas fueron codificadas por expertos sin tener en cuenta un sistema o dominio particular, sino considerando conceptos generales de arquitecturas de software, la intuición era que los resultados iban a ser buenos pero no perfectos. Luego de establecer las oraciones afectadas, las mismas se compararon con la solución de referencia y se determinaron las métricas de *precision*, *recall* y *accuracy*.

Figura 7. Resultados de la Detección de Decisiones de Diseño

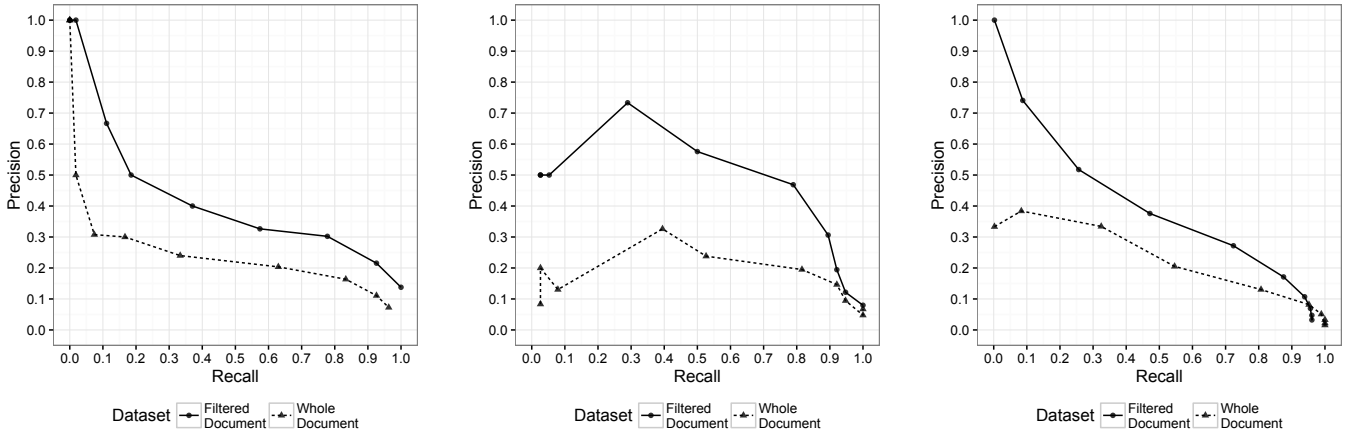


La Figura 7 muestra los resultados obtenidos en cada caso de estudio. Una observación inicial es que el *recall* obtenido por las reglas de decisiones de diseño esta por encima de 90 %, independientemente del caso de estudio. Este resultado es interesante, porque implica que las reglas pueden detectar la mayoría de las oraciones referidas a decisiones arquitectónicas. Con respecto a la *precision*, se obtuvieron resultados de ~78 % en Adventure Builder y Pet Store, indicando que el 20 % de las oraciones marcadas como decisiones eran incorrectas. Este comportamiento no fue observado en MSLite, donde las reglas fueron muy precisas y casi no se cometieron errores de detección (~95 %). La diferencia entre los casos de estudio se atribuye a la forma que están redactados los documentos de arquitectura, donde existen ciertas oraciones que utilizan palabras clave que son parte de las reglas pero no describen una decisión de diseño, sino que hacen referencia a oraciones anteriores. A pesar de estos problemas, los valores de *precision* obtenidos son aceptables para utilizarlos en la búsqueda de trazas, ya que se cree que los beneficios del filtrado van a compensar los errores cometidos en la detección de decisiones. El análisis de desempeño de las reglas respecto al *accuracy* produjo resultados similares a los observados con la *precision*. Se cree que en los dos primeros casos de estudio, la longitud de los documentos de arquitectura no es lo suficientemente importante para que los verdaderos negativos influyan en la métrica. En MSLite, sin embargo, el tamaño de la documentación sí tuvo un impacto positivo en el *accuracy*, donde los resultados obtenidos fueron superiores a aquellos de *precision* y *recall*.

5.2. Recuperación Automática de Trazas

La eficacia de una técnica automatizada de trazabilidad depende tanto de la cantidad de trazas correctas identificadas como también del porcentaje de trazas recuperadas respecto al total de existentes. A diferencia de una técnica de búsqueda tradicional, donde por lo general los resultados deben ser principalmente precisos, en el contexto de la trazabilidad

Figura 8. Resultados de la Recuperación de Trazas



a) Adventure Builder b) Pet Store c) MSLite

entre artefactos se prefiere que las técnicas obtengan un muy buen *recall* por sobre la *precision*. Esta afirmación se basa en el hecho de que a los analistas les resulta más fácil descartar trazas equivocadas que encontrar trazas faltantes [8], [22].

En la experimentación, se analizó el desempeño del algoritmo utilizando dos conjuntos de datos, a saber: la documentación completa de los casos de estudios y la documentación filtrada con las reglas. Asimismo, se exploraron diferentes combinaciones de parámetros para configurar el algoritmo de LSA. Con respecto al número de dimensiones al que se reduce la matriz de aproximación con SVD, en colecciones de documentos grandes con muchos términos se sugiere usar valores de $d \in [200; 500]$. Sin embargo, la cantidad de texto y documentos en el análisis de artefactos de software es mucho menor, y por lo tanto se suelen utilizar valores de $d \in [40; 100]$ para obtener buenos resultados [24]. De modo de abarcar diferentes escenarios, en la evaluación se probaron valores de $d = \{30; 45; 60; 75; 100\}$. Con respecto a la métrica de similitud y el umbral para identificar las trazas, se decidió utilizar la distancia del coseno entre vectores y valores de $u = \{0{,}45; 0{,}50; 0{,}55; 0{,}60; 0{,}65; 0{,}70; 0{,}75; 0{,}80; 0{,}85; 0{,}90\}$.

Luego de algunas ejecuciones preliminares del algoritmo de LSA, se pudo apreciar que a partir de un número de dimensiones $d = 60$ no se observaban mejoras en el desempeño independientemente del umbral u utilizado. Análogamente, cuando $d < 60$ el desempeño de la técnica se reducía progresivamente al disminuir el número de dimensiones. Por esta razón, solamente se analizaron los resultados de $d = 60$. La Figura 8 ilustra los valores de *precision* y *recall* producidos en cada sistema por las diferentes configuraciones. Las líneas que unen los puntos indican la variación de las métricas según el umbral de similitud utilizado. Los puntos circulares y las líneas llenas representan los resultados logrados utilizando la documentación filtrada, mientras que los puntos triangulares y las líneas punteadas representan los resultados logrados utilizando la totalidad del contenido textual de la documentación. A grandes rasgos, se observaron mejoras substanciales en los tres casos de estudio al filtrar la documentación mediante oraciones provenientes de requerimientos arquitecturales y decisiones de diseño.

En Adventure Builder, los resultados obtenidos mostraron mejoras de entre 15 % y 25 % en *recall* y *precision*. La curva de la documentación filtrada se comportó de manera similar a la documentación completa (quitando las mejoras), dando a entender un claro trade-off entre las dos métricas. Para valores de $u = 0{,}60$ se obtuvieron resultados de trazabilidad muy prometedores, alcanzando valores de *recall* y *precision* de ~80 % y ~30 %, respectivamente. Este desempeño está en la misma línea que las técnicas automatizadas de trazabilidad entre otros artefactos de software, tales como requerimientos y código o incluso requerimientos y tests.

En el caso de estudio Pet Store, sin embargo, las curvas mostraron ciertas irregularidades que requieren un análisis más detallado. Los resultados obtenidos con valores de $u > 0{,}85$ fueron llamativos, porque en vez de comenzar en el extremo superior izquierdo del gráfico (máxima *precision* y mínimo *recall*), se ubicaron en el centro izquierdo y el extremo inferior izquierdo para la documentación filtrada y completa, respectivamente. Esta anomalía se debió a que la técnica detectó algunas trazas incorrectas incluso con los umbrales más restrictivos. Como con estos valores de u el número de trazas recuperadas es muy pequeño, los valores de precisión fluctúan entre 0 % y 100 % con facilidad. Igualmente, las mejoras observadas al comparar los documentos filtrados con los documentos completos fueron importantes. En términos de *precision*, se observaron mejoras más notorias que en el caso de estudio previo, rondando en una mejora de entre 20 % y 40 %. En cuanto al *recall*, también se lograron mejoras más significativas que en Adventure Builder (entre 10 % y 30 %), pero inferiores a las obtenidas en *precision*. Se cree que esta diferencia entre casos de estudio se debe al menor número de trazas presentes en Pet Store (38 vs. 55). Al establecer una configuración

óptima, se observó un desempeño ejemplar de la técnica con $u = 0{,}65$, arrojando un *recall* y *precision* de ~90 y ~30 %, respectivamente. Una observación de ambos casos de estudio es los bajos valores de u necesarios para encontrar la mayoría de las trazas. Una posible explicación para este comportamiento es que el número total de oraciones en sistemas pequeños no es lo suficiente extensa para derivar una matriz de aproximación representativa.

En MSLite, el caso de estudio de mayor tamaño con 456 trazas, la técnica tuvo un desempeño similar al obtenido en Adventure Builder. En particular, se observaron mejoras tanto en *precision* como *recall* de entre 10 % y 15 %, ligeramente inferiores a las logradas en el primer caso de estudio. Los mejoras fueron consistentes a través de los diferentes valores de u, arrojando los mejores resultados cuando $u = 0{,}75$. En esta configuración, la cual utiliza un valor del umbral recomendado según la literatura, el *recall* y *precision* conseguidos fueron de ~90 % y ~18 %, respectivamente. En definitiva, la técnica propuesta pudo mejorar significativamente el desempeño al recuperar trazas en los tres casos de estudio mediante el filtrado previo de los documentos. Asimismo, la utilización de los documentos completos introduce un nivel de ruido considerable en la recuperación de trazas, convirtiendo las soluciones generadas inviables desde la perspectiva de un analista que tiene que utilizar y posiblemente validar esos resultados.

6. Conclusión

En este artículo se presentó una técnica para identificar relaciones de trazabilidad al nivel de oraciones entre documentación de requerimientos y el documento de arquitectura. La técnica hace uso de diferentes algoritmos de procesamiento de lenguaje natural para reconocer información léxica, sintáctica y semántica del texto. Luego, en base a dicho conocimiento, la técnica analiza los documentos para reconocer aquellas porciones de texto relevantes a las trazas. Por un lado, se utiliza la herramienta *REAssistant* para filtrar oraciones en las especificaciones de casos de uso que representen requerimientos arquitecturales, es decir, que hagan referencia o estén afectados por atributos de calidad. Por otro lado, se utilizó un lenguaje de reglas denominado RUTA que permite reconocer decisiones de diseño que permiten satisfacer diferentes atributos de calidad, tales como tácticas y estilos arquitectónicos, tecnologías de terceros, patrones de diseño, entre otras. Las reglas fueron definidas por expertos en base a la taxonomía de Bass et al. [1], codificando términos claves que revelan la presencia de las decisiones de diseño. Por último, la técnica hace uso de un algoritmo de LSA para transformar las oraciones a un espacio vectorial aproximado, el cual permite establecer trazas mediante métricas de distancia entre vectores.

La técnica desarrollada fue evaluada con tres casos de estudio de diferentes dominios. Los experimentos fueron divididos en dos partes, a saber: el desempeño de las reglas para decisiones de diseño y la calidad de las trazas recuperadas. Con respecto al desempeño de las reglas, se pudo observar que la gran mayoría de las decisiones de diseño fueron detectadas, aunque cometiendo algunas equivocaciones en el proceso. Específicamente, los resultados arrojaron valores de *recall* superiores al 90 %, mientras que se obtuvieron valores de *precision* del 78 %. Al analizar las trazas entre requerimientos y arquitectura, se evaluó el desempeño utilizando tanto los documentos filtrados como los documentos completos. Asimismo, se realizó un exploración de los diferentes parámetros del algoritmo de LSA, como son el número de dimensiones y el umbral de similitud. Como resultado, la técnica obtuvo una buena *precisión* para valores altos de *recall*. Específicamente, utilizando valores de umbral entre $0{,}65$ y $0{,}70$ y aplicando el filtrado de los documentos, se pudo mejorar la detección de trazas con respecto a los documentos completos, yendo desde un 10 % hasta un 40 % (dependiendo del caso de estudio).

Durante la experimentación también se encontraron ciertas limitaciones de la técnica. Muchas trazas fueron detectadas incorrectamente debido a errores acarreados por las reglas de decisiones de diseño o incluso el procesamiento de lenguaje natural. Otro problema detectado fue que el algoritmo de LSA era sensible al valor de umbral usado para calcular la similitud entre oraciones, variando según el caso de estudio. Como trabajo futuro, se espera poder evaluar la técnica con sistemas de mayor magnitud y complejidad. Asimismo, se planea refinar y ajustar las reglas para aumentar su capacidad de detección, posiblemente incorporando nuevos tipos de atributos de calidad y tácticas arquitectónicas. Por último, se pretende evaluar otros algoritmos para determinar las trazas (e.g, transformaciones a VSM y feedback de relevancia) y explorar otras métricas de similitud (e.g., Jaccard, Dice, Jenson-Shannon, etc.).

En resumen, se cree que la aplicación de técnicas avanzadas de procesamiento de lenguaje natural en el área de Ingeniería de Software pueden mejorar significativamente el análisis de la documentación textual, posibilitando la generación precisa de trazas para verificar la satisfacción de los atributos de calidad a lo largo del desarrollo.

Reconocimientos

Se agradece a German Attanasio y Rodrigo Gonzalez por haber implementado el prototipo de la técnica como parte de su tesina de grado. También se da las gracias a los revisores anónimos por sus sugerencias para mejorar el artículo.

Referencias

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed., ser. SEI Series in Software Engineering. Addison-Wesley Professional, October 2012.

[2] K. Wiegers and J. Beatty, *Software Requirements*, 3rd ed., ser. Developer Best Practices. Microsoft Press, 2013.

[3] A. Moreira, R. Chitchyan, J. Araujo, and A. Rashid, Eds., *Aspect-Oriented Requirements Engineering*. Springer Berlin Heidelberg, 2013, vol. XIX.

[4] L. Chung and J. do Prado Leite, "On non-functional requirements in software engineering," *Conceptual modeling: Foundations and applications*, pp. 363–379, 2009.

[5] J. Cleland-Huang, O. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: Trends and future directions," in *Future of Software Engineering (FOSE'14), held at the 36th International Conference on Software Engineering (ICSE'14)*. New York, NY, USA: ACM, 2014, pp. 55–69.

[6] J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., *Software and Systems Traceability*. Springer London, 2012.

[7] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *1st IEEE International Conference on Requirements Engineering (RE'94)*, April 1994, pp. 94–101.

[8] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: A systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, vol. 22, pp. 1–49, 2012.

[9] M. Mirakhorli and J. Cleland-Huang, "Tracing non-functional requirements," in *Software and Systems Traceability*. Springer London, 2012, pp. 299–320.

[10] L. Chen, M. Ali Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE Software*, vol. 30, no. 2, pp. 38–45, 2013.

[11] F. Bachmann, L. Bass, P. Clements, D. Garlan, J. Ivers, M. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2nd ed. Addison-Wesley Professional, 2010.

[12] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, ser. SEI Series in Software Engineering. Addison-Wesley, 2002.

[13] A. Clark, C. Fox, and S. Lappin, *The Handbook of Computational Linguistics and Natural Language Processing*, 1st ed. Wiley-Blackwell, 2012.

[14] E. Baniassad, P. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, "Discovering early aspects," *IEEE Software*, vol. 23, no. 1, pp. 61–70, 2006.

[15] A. Sampaio, A. Rashid, R. Chitchyan, and P. Rayson, "EA-Miner: towards automation in aspect-oriented requirements engineering," *Transactions on Aspect-Oriented Software Development III*, pp. 4–39, 2007.

[16] A. Rago, C. Marcos, and J. Diaz-Pace, "Assisting requirements analysts to find latent concerns with REAssistant," *Automated Software Engineering*, vol. 23, no. 2, pp. 219–252, 2016.

[17] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010.

[18] A. Rago, C. Marcos, and J. Diaz-Pace, "Uncovering quality-attribute concerns in use case specifications via early aspect mining," *Requirements Engineering*, vol. 18, no. 1, pp. 67–84, 2013.

[19] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements Engineering*, vol. 12, no. 2, pp. 103–120, May 2007.

[20] A. Casamayor, D. Godoy, and M. Campo, "Functional grouping of natural language requirements for assistance in architectural software design," *Knowledge-Based Systems*, vol. 30, pp. 78–86, 2012.

[21] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, Oct 2002.

[22] J. Hayes, A. Dekhtyar, and S. Sundaram, "Advancing candidate link generation for requirements tracing: the study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, Jan 2006.

[23] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou, "Poirot: A distributed tool supporting enterprise-wide automated traceability," in *14th IEEE International Requirements Engineering Conference (RE'06)*, 2006, pp. 363–364.

[24] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Enhancing an artefact management system with traceability recovery features," in *20th IEEE International Conference on Software Maintenance (ICSM'04)*, Sept 2004, pp. 306–315.

[25] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *25th International Conference on Software Engineering (ICSE'03)*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 125–135.

[26] A. Kuhn, S. Ducasse, and T. Girba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230 – 243, 2007, 12th Working Conference on Reverse Engineering.

[27] D. Ferrucci and A. Lally, "Uima: an architectural approach to unstructured information processing in the corporate research environment," *Natural Language Engineering*, vol. 10, no. 3-4, pp. 327–348, 2004.

[28] P. Kluegl, M. Toepfer, P.-D. Beck, G. Fette, and F. Puppe, "UIMA ruta: Rapid development of rule-based information extraction applications," *Natural Language Engineering*, pp. 1–40, 7 2015.

[29] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[30] Oracle Technology Network, "Java blueprints," http://www.oracle.com/technetwork/java/blueprints-141945.html, 2016.

[31] S. Gersmann, "Development of strategies for global software development," PhD. Thesis, Technische Universitat Munchen, 2005.

[32] N. Mullick, M. Bass, Z. Houda, P. Paulish, and M. Cataldo, "Siemens global studio project: experiences adopting an integrated gsd infrastructure," in *IEEE International Conference on Global Software Engineering (ICGSE'06)*. Florianopolis, Brazil: IEEE, 2006, pp. 203–212.

[33] N. Brown, R. Nord, I. Ozkaya, and M. Pais, "Analysis and management of architectural dependencies in iterative release planning," in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA'11)*. Boulder, Colorado, USA: IEEE, 2011, pp. 103–112.

[34] S. Ferilli and S. Ferilli, "Natural language processing," in *Automatic Digital Document Processing and Management*, ser. Advances in Pattern Recognition. Springer London, 2011, pp. 199–222.