

Recuperación de Trazabilidad entre Documentos de Diseño y Requerimientos mediante Técnicas semi-automáticas

Attanasio Ruiz German, Gonzalez Rodrigo Damian

{germanattanasio, rdgonzalez85}@gmail.com

Índice general

1. Introducción	7
1.1. Trazabilidad entre requerimientos y arquitectura	8
1.2. Dificultades a la hora de recuperar la trazabilidad	9
1.3. TRAS: Un enfoque automatizado para recuperar relaciones de trazabilidad	10
1.4. Organización del Trabajo	11
2. Marco teórico	13
2.1. Documentos de requerimientos	13
2.1.1. Casos de uso	14
2.1.2. <i>Concerns</i> y <i>crosscutting concerns</i>	15
2.2. Documentos de arquitectura	16
2.2.1. Atributos de calidad	16
2.2.2. Tácticas	18
2.2.3. Patrones de arquitectura	18
2.3. Decisiones de diseño	19
2.4. Trazabilidad	19
2.4.1. Trazabilidad entre documentos de requerimientos y arquitectura	20
2.4.2. Métodos automáticos para la recuperación de trazas	21
2.5. Resumen	22
3. Trabajos relacionados	24
3.1. Recuperación de información de artefactos de software	24
3.1.1. Procesamiento de documentos de requerimientos no estructurados	25
3.1.2. Identificación de <i>crosscutting concerns</i> en documentos de requerimientos .	26
3.1.3. Análisis comparativo de enfoques para identificar <i>crosscutting concerns</i> .	29
3.2. Recuperación de trazabilidad entre artefactos de software	31
3.2.1. Herramientas para recuperar trazas entre artefactos de software	32
3.2.2. Criterios de comparación para evaluar herramientas de recuperación de trazas	39
3.2.3. Comparación de herramientas analizadas	39
3.3. Resumen	41
4. TRAS: Una herramienta para la identificación de trazas entre documentos de requerimientos y de arquitectura	43
4.1. Esquema de la TRAS	43
4.2. Arquitectura de TRAS	47
4.2.1. Detección de <i>crosscutting concerns</i>	47
4.2.2. Detección de decisiones de diseño	48
4.2.3. Búsqueda de trazabilidad	53
4.3. Resumen	56

5. Evaluación	58
5.1. Casos de estudio	59
5.2. Análisis de los casos de estudio	59
5.3. Métricas	60
5.4. Experimentos	66
5.4.1. Evaluación de detección de decisiones de diseño	66
5.4.2. Evaluación de búsqueda de trazabilidad	67
5.5. Discusión de los resultados	72
5.6. Validez de los resultados	73
5.7. Resumen	73
6. Conclusiones	75
6.1. Ventajas y desventajas	76
6.2. Trabajos futuros	77
A. Reglas para detectar Decisiones de Diseño	80
A.1. Availability	80
A.2. Security	80
A.3. Modificability	81
A.4. Performance	82
A.5. Interoperability	82
A.6. Testability	82
A.7. Usability	83

Índice de figuras

1.1. Ejemplo de trazas en los diferentes artefactos de software	8
1.2. Enfoque Propuesto	10
2.1. Ejemplo de caso de uso	15
2.2. Vista de componentes y conectores	16
2.3. Ejemplo de vista de deployment	17
2.4. Listado de patrones arquitectónicos	19
2.5. Estructura de una traza	20
2.6. Representación vectorial de documentos en VSM	22
3.1. Visualizaciones de Temas en Theme/Doc	28
3.2. Interfaz gráfica de Poirot	34
3.3. Ejemplo de la representación jerárquica de artefactos y trazas por Poirot	35
3.4. Interfaz gráfica de RETRO	36
3.5. Arquitectura de RETRO	37
3.6. Pasos necesarios para recuperar la trazabilidad en ADAMS	38
3.7. Análisis de trazas sugeridas en ADAMS	38
3.8. Arquitectura de ADAMS	39
4.1. Arquitectura de TRAS	44
4.2. Lista de trazas entre <i>crosscutting concerns</i> y decisiones de diseño	45
4.3. Lista de decisiones de diseño producida por la herramienta	45
4.4. Lista de <i>crosscutting concerns</i> sin decisiones de diseño asociadas	46
4.5. Gráfico de distribución de decisiones de diseño	46
4.6. Distribución de decisiones de diseño dado un grupo de <i>crosscutting concerns</i>	47
4.7. Componentes encargados de la detección de decisiones de diseño	48
4.8. Ejemplo de las anotaciones producidas por los módulos de NLP	49
4.9. Taxonomía utilizada para la clasificación de decisiones de diseño	51
4.10. Estructura de una regla de Ruta	52
4.11. Regla encargada de detectar la táctica de disponibilidad <i>ping/echo</i>	52
4.12. Ejemplo de cómo las reglas recuperan decisiones de diseño	53
4.13. Conjuntos de oraciones extraídas de dos documentos	54
4.14. Descomposición en valores singulares	54
4.15. Representación gráfica de las matrices de SVD	55
5.1. MSLite. Distribución de las decisiones de diseño.	61
5.2. Pet Store. Distribución de las decisiones de diseño.	61
5.3. Adventure Builder. Distribución de las decisiones de diseño.	61
5.4. MSLite. Distribución de los <i>crosscutting concerns</i>	62
5.5. Pet Store. Distribución de los <i>crosscutting concerns</i>	62
5.6. Adventure Builder. Distribución de los <i>crosscutting concerns</i>	62
5.7. Notación formal de los <i>crosscutting concerns</i>	63

5.8.	Notación formal de las decisiones de diseño.	63
5.9.	Notación formal de las trazas.	64
5.10.	Detección de decisiones de diseño en <i>MSLite</i>	68
5.11.	Detección de decisiones de diseño en <i>Pet Store</i>	68
5.12.	Detección de decisiones de diseño en <i>Adventure Builder</i>	68
5.13.	Promedio de métricas para la detección de decisiones de diseño.	69
5.14.	<i>Precision</i> sobre <i>recall</i> - <i>MSLite</i>	70
5.15.	<i>Precision</i> sobre <i>recall</i> - <i>Pet Store</i>	70
5.16.	<i>Precision</i> sobre <i>recall</i> - <i>Adventure Builder</i>	71

Índice de cuadros

3.1.	Enfoques utilizados sobre artefactos de software	27
3.2.	Comparación de las herramientas de recuperación de <i>crosscutting concerns</i> descritas	30
3.3.	Resumen de trabajos para recuperar trazas entre artefactos de software	33
3.4.	Comparación herramientas de recuperación de trazabilidad	40
4.1.	Lista de etiquetas utilizadas por el anotador POS Tagger	50
4.2.	Distribución de palabras en los documentos	54
4.3.	Trazas recuperadas luego de aplicar LSA	56
5.1.	Casos de estudio.	59
5.2.	Detección de decisiones de diseño.	67
5.3.	Variación de valores para distintos thresholds - MSLite.	70
5.4.	Variación de valores para distintos thresholds - Pet Store.	71
5.5.	Variación de valores para distintos thresholds - Adventure Builder.	71
5.6.	REI	72

Nomenclatura

CAS	Common Analysis Structure
CCC	Crosscutting Concerns
CPL	Common Representation Language
DDD	Decisiones de diseño
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IR	Information Retrieval
LSA	Latent Semantic Analysis
ML	Machine Learning
NLP	Natural Language Processing
RCP	Rich Client Plugin
RE	Requirements Engineering
SAD	Software Architecture Document
SE	Software Engineering
SoC	Separation of Concerns
SVD	Singular Value Decomposition
TRAS	Traceability Assistant
UIMA	Unstructured Information Management Architecture
VSM	Vector Space Model

Introducción

La trazabilidad de software es la medida en la que se puede establecer una relación entre dos o más artefactos junto con la habilidad de poder examinar esta relación [?]. La capacidad para establecer la trazabilidad, por tanto, depende de la creación de relaciones entre la información contenida en diferentes tipos de artefactos [?, ?]. Estos artefactos se denominan artefactos de traza y pueden participar como la fuente o el destino en una relación de trazabilidad [?]. Por ejemplo, un artefacto puede ser un requerimiento, una clase UML o una clase Java.

Las trazas vinculan un artefacto fuente con un artefacto destino y tienen una naturaleza bidireccional [?]. Esto significa que si existe una traza desde A hacia B, entonces también existe la traza desde B hacia A. Existen tres maneras de clasificar las trazas, dependiendo del modo en que éstas son recuperadas. El primer tipo de trazas se denominan manuales, y se refieren a las relaciones entre artefactos que fueron identificadas por los analistas de forma manual [?]. El segundo tipo son trazas semi-automatizadas, las cuales son establecidas por los analistas mediante la asistencia de técnicas y herramientas automatizadas [?]. Por último, el tercer tipo son las trazas automatizadas, las cuales son establecidas automáticamente sin la intervención de los analistas a través de técnicas, métodos y herramientas para este fin [?].

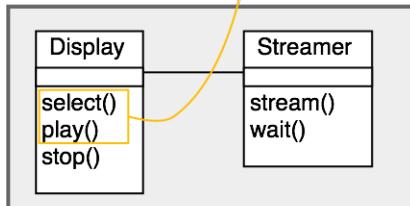
Los sistemas de software de hoy en día son complejos e incluyen una gran cantidad de artefactos complementarios producidos durante el ciclo de vida del producto. Además del código fuente, un sistema contiene artefactos tales como: documentos de arquitectura, documentos de requerimientos, casos de prueba, manuales de usuario, entre otros. Estos artefactos son creados y mantenidos a través de las fases de desarrollo por diferentes stakeholders del sistema, como por ejemplo analistas de requerimientos, arquitectos, desarrolladores, etc. Establecer y mantener la trazabilidad entre dichos artefactos de software facilita muchas de las tareas durante el desarrollo de software. Primero, permite identificar las fuentes de un artefacto (quién lo creó y a partir de qué otros artefactos) y explorar su historia (quiénes participaron durante su desarrollo). Segundo, facilita el análisis mediante el seguimiento de trazas hacia los componentes afectados en respuesta a un cambio en particular en los requerimientos. Tercero, simplifica el monitoreo general de un proyecto, visualizando el número de requerimientos actualmente en análisis, en diseño o en implementación.

En el contexto de la Ingeniería de Software(Software Engineer, SE), las relaciones de trazabilidad entre artefactos están estrechamente relacionadas con las etapas y actividades llevadas a cabo durante el desarrollo de un producto. Esto significa que las trazas suelen originarse en los requerimientos y terminan en el código fuente, pasando por los artefactos intermedios como diagramas de actividades y estados, componentes de diseño, casos de test, entre otros. Cada una de estas trazas contiene información relevante para la construcción del sistema. Por ejemplo, una traza entre requerimientos y componentes de diseño hace explícita la información de transición de un requerimiento (desde que el mismo fue definido) hasta que éste fue satisfecho por la arquitectura (mediante componentes y tácticas arquitectónicas), y viceversa. La Figura 1.1 muestra un ejemplo de varias trazas entre un requerimiento relacionado a reproducir una película, el código fuente que implementa y el diagrama de transición de estados donde se explica dicha

Requerimientos

Stop movie at request or at the end (no auto restart)
Start playing movie within 1 sec after selection

Código fuente



Diagramas de transición de estados

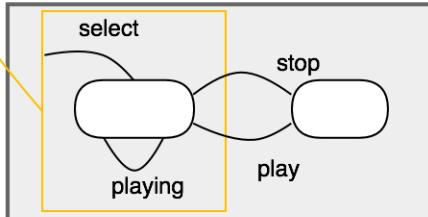


Figura 1.1: Ejemplo de trazas en los diferentes artefactos de software

funcionalidad.

1.1. Trazabilidad entre requerimientos y arquitectura

Muchas de las relaciones de trazabilidad más importantes se originan a partir de los requerimientos de software, uno de los primeros artefactos producidos al analizar un sistema y su funcionalidad. Los requerimientos representan las necesidades de los clientes, los servicios que los stakeholders (personas interesadas) desean que proporcione el sistema, y las restricciones sobre las que el software debe operar. Existen diferentes formas de organizar y documentar los requerimientos de un sistema. Una de las técnicas más utilizadas para documentar los requerimientos son los casos de uso y sus especificaciones textuales [?]. Los casos de uso contienen descripciones en lenguaje natural que ilustran las interacciones entre el sistema y los actores del mismo, divididas en escenarios “ideales” y escenarios “alternativos”.

A pesar de que los casos de uso tienen un carácter estrictamente funcional [?], estos suelen contener menciones o referencias a ciertas características no funcionales del sistema ocultas en las descripciones textuales de los requerimientos. Muchas de estas características pueden ser reveladas mediante el análisis de *crosscutting concerns* [?]. Un *crosscutting concern* (CCC) es un “asunto de interés en un sistema de software” que está descrito de forma parcial en múltiples documentos de requerimientos.

Algunas de las características no funcionales a las que los CCCs hacen referencia se conocen con el nombre de atributos de calidad (*Quality Attributes*, QA). Estos hablan de características específicas que debe tener el sistema. Por ejemplo: Modificabilidad, Performance o Disponibilidad. Existen diferentes formas de agrupar estas propiedades en categorías [?, ?]. Algunas propiedades pueden observarse durante el diseño del sistema como es el caso de la Modificabilidad y otras como Performance solo cuando el sistema está en ejecución.

Así como los documentos de requerimientos describen lo que el sistema tiene que hacer, el documento de la arquitectura (*Software Architecture Document*, SAD) describe la organización de componentes y sus relaciones entre sí para satisfacer la funcionalidad del sistema. Es decir, el SAD describe cómo el sistema va a cumplir con sus responsabilidades. El SAD es uno de los vehículos de comunicación entre los stakeholders [?] y habitualmente toma la forma de un documento o un conjunto de documentos textuales acompañados de diferentes ilustraciones gráficas, denominadas vistas arquitectónicas.

En el texto del SAD suele estar presente la descripción de las tácticas y estilos arquitectónicos utilizados, y en algunos casos la justificación de dichas elecciones. Las tácticas son organizaciones de componentes que permiten enfocarse en un único atributo de calidad y satisfacerlo adecuadamente. Existen diferentes tipos de tácticas y éstas pueden ser organizadas dependiendo del

atributo de calidad que buscan mejorar. Los patrones y estilos arquitectónicos permiten resolver problemas de diseño más complejos, abarcando un conjunto de atributos de calidad.

Uno de los aspectos principales del SAD es la documentación detallada de las decisiones de diseño (DDDs) tomadas por los arquitectos. Cada una de éstas puede verse como la implementación en la arquitectura de un conjunto de cambios [?]. Estos cambios son en algunos casos el resultado de elegir una táctica o estilo arquitectónico y permiten satisfacer tanto la funcionalidad como también los atributos de calidad asociados a dicha funcionalidad. Algunos ejemplos de decisiones de diseño son la elección de un protocolo de comunicación para aumentar la seguridad, la elección de un lenguaje de programación para mejorar la modificabilidad, o el uso de una arquitectura cliente-servidor.

Debido a que la funcionalidad documentada en las especificaciones de requerimientos suele tener menciones o referencias a determinados atributos de calidad y que estos pueden ser descubiertos mediante el análisis de crosscutting concerns, existe la posibilidad de poder encontrar trazas entre requerimientos y arquitectura. Es decir, las trazas entre documentos de requerimientos y el SAD emergen a través del análisis de atributos de calidad que satisfacen las decisiones de diseño y los atributos de calidad presentes en los crosscutting concerns. Por ejemplo, en los casos de uso se suelen observar oraciones tales como “el sistema genera el reporte de gastos en menos de 5 segundos”. En esta oración existe una clara referencia a un atributo de calidad de performance, el cual se encuentra descrito de forma parcial (en el contexto del resto del sistema) [?]. En el caso del SAD, se suelen observar oraciones como “El módulo de reportes utiliza una cache en memoria de hasta 2GB”. Ésta, es una decisión de diseño que busca satisfacer el atributo de calidad de performance mencionado en los requerimientos.

1.2. Dificultades a la hora de recuperar la trazabilidad

Si bien las etapas de captura de requerimientos y creación de la arquitectura están estrechamente relacionadas, las actividades realizadas en dichas etapas son llevadas a cabo por separado y con una interacción limitada [?]. En la industria, la ausencia de estos registros de trazabilidad entre artefactos se atribuye principalmente a que realizar los análisis pertinentes para obtenerlos es un proceso largo, costoso y propenso a errores. Adicionalmente, es necesario que la persona que realiza el análisis conozca detalladamente los requerimientos y la arquitectura del sistema.

Un desafío importante en la recuperación de trazas entre los artefactos de software es el hecho de que estos artefactos tienen diferentes objetivos y por lo tanto describen cosas distintas y en diferentes niveles de abstracción. En este contexto, es complicado establecer relaciones de trazabilidad sin conocer los puntos en común entre los artefactos. Hay que comprender los requerimientos para saber con qué parte de la arquitectura se corresponden. Por otro lado, hay que entender la arquitectura y las decisiones tomadas para determinar los requerimientos vinculados. El costo y esfuerzo necesario para crear y mantener relaciones de trazabilidad en un sistema de software en evolución puede ser muy alto [?]. Asimismo, 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 [?].

Existen diferentes técnicas para recuperar las trazas entre artefactos de software de forma automática, basadas en el análisis textual de los mismos. La suposición que hacen las técnicas es que si en el contenido textual de dos artefactos se mencionan conceptos similares, entonces los dos artefactos están conceptualmente relacionados y puede que exista una traza entre ellos. Los principales métodos utilizados para recuperar trazas son los modelos probabilísticos [?, ?, ?], los modelos de espacio vectorial (*Vector Space Model*, VSM) [?, ?, ?] y su extensión *Latent Semantic Analysis* (LSA) [?, ?, ?]. Este último método provee una mejora sustancial para documentos de gran tamaño con respecto al modelo de espacio vectorial estándar y los modelos probabilísticos [?, ?].

A pesar de los avances respecto a la búsqueda de trazas entre diversos artefactos de software, a nuestro saber y entender, los investigadores no han enfocado su esfuerzo en la recuperación de trazas entre documentos de requerimientos y el SAD. Adicionalmente, los enfoques existentes

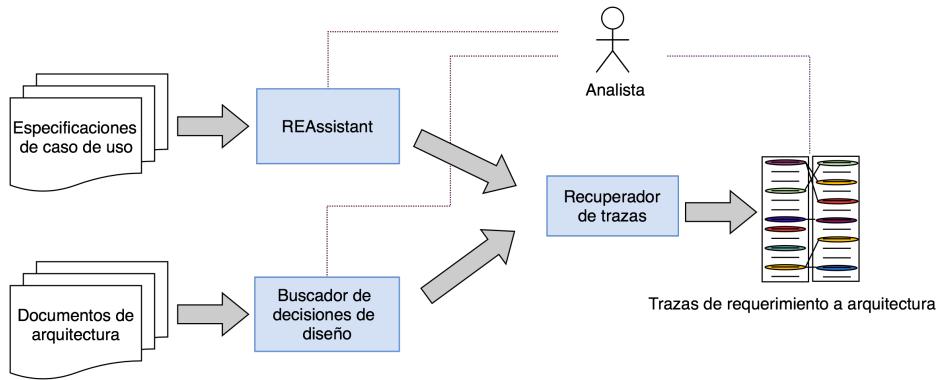


Figura 1.2: Enfoque Propuesto

y las técnicas utilizadas en estos, presentan ciertas limitaciones cuando son aplicados en documentos extensos y con mucho contenido textual. En el contexto del análisis de trazabilidad entre requerimientos y arquitectura, este tipo de limitaciones podría tener efectos negativos en los resultados de la recuperación, debido a que gran parte del texto presente en los documentos no está directamente relacionado con las trazas.

1.3. TRAS: Un enfoque automatizado para recuperar relaciones de trazabilidad

En este trabajo, se presenta un enfoque para asistir a los analistas en la recuperación de la trazabilidad entre documentos de requerimientos y el SAD. La hipótesis es que recuperar los vínculos entre los crosscutting concerns y las decisiones de diseño utilizando una técnica de recuperación de información como LSA, permitirá identificar la mayoría de las relaciones de trazabilidad presentes entre la documentación de requerimientos y arquitectura.

El enfoque está materializado en una herramienta de asistencia denominada TRAS (TRaceability ASSistant). La herramienta provee soporte para facilitar diversas tareas del analista. Primero, permite recuperar las decisiones de diseño del SAD utilizando análisis y reglas específicas del dominio. Segundo, permite recuperar las trazas entre decisiones de diseño y crosscutting concerns mediante LSA. Tercero, permite identificar aquellos crosscutting concerns que no tienen una decisión de diseño asociada y por lo tanto pueden no haber sido tenidos en cuenta en la arquitectura. Asimismo, la herramienta permite visualizar la distribución de decisiones de diseño de acuerdo al atributo de calidad que afectan y visualizar el impacto a alto nivel de cada grupo de crosscutting concerns sobre los atributos de calidad. Por último, la herramienta permite verificar que el diseño arquitectónico del sistema implementa los requerimientos claves, y que los aspectos esenciales del diseño tengan trazas hacia los requerimientos.

La herramienta desarrollada está organizada en tres componentes, a saber, la identificación de crosscutting concerns (REAssistant), la identificación de decisiones de diseño (Buscador de Decisiones de Diseño) y la identificación de relaciones de trazabilidad (Recuperador de Trazas). En la Figura 1.2 se ilustra un esquema de las entradas y salidas de la herramienta, sus componentes principales y la interacción del analista con ella. A continuación se describe en detalle cada uno de los componentes.

El componente REAssistant (REquirements Analysis Assistant) [?] (desarrollado en la UNICEN) procesa casos de uso textuales para descubrir *crosscutting concerns* candidatos que están ocultos en las especificaciones. Para ello, utiliza una combinación de técnicas de procesamiento de lenguaje natural (NLP) y estrategias semánticas de búsqueda. La herramienta recibe como entrada especificaciones de casos de uso y produce como salida un modelo que registra las oraciones afectadas por los *crosscutting concerns*.

El componente *Buscador de Decisiones de Diseño* tiene como objetivo identificar decisiones de diseño a partir del SAD. Para ello, utiliza diversos módulos de NLP que extraen información

léxica y sintáctica (palabras y partes del discurso) a partir del texto contenido en el SAD. Esta información es luego utilizada por un sistema basado en reglas para identificar oraciones que referencian (aunque sea parcialmente) decisiones de diseño que tienen impacto sobre uno o más atributos de calidad. Las reglas están agrupadas por tácticas y estas a su vez por atributos de calidad. Por ejemplo, para identificar decisiones de diseño asociadas al atributo disponibilidad, se utilizan reglas que buscan tácticas como *Ping/Echo*, *Exception* o *Heartbeat* entre otras. El desarrollo de este componente involucra diferentes tecnologías, tales como el framework de procesamiento de texto UIMA y el motor de reglas RUTA.

Por último, el componente Recuperador de Trazas infiere las relaciones trazabilidad a partir de los *crosscutting concerns* y las decisiones de diseño utilizando una técnica de recuperación de información denominada LSA (Latent Semantic Analysis). Esta técnica es una aproximación matemática para el descubrimiento de similitud entre documentos, fragmentos de documentos y las palabras dentro de los documentos [?].

Para determinar si existe una traza, LSA utiliza las oraciones afectadas por *crosscutting concerns* junto con las oraciones que contengan decisiones de diseño (recuperadas por REAssistant y el Buscador de Decisiones de Diseño) para generar un espacio vectorial. Este espacio representa cada par de crosscutting concern y decisión de diseño como vectores, donde la existencia de la traza está determinada por el coseno del ángulo comprendido entre estos. Un valor de coseno cercano a 1 indica que el contenido textual es similar, y por lo tanto, es una traza potencial. Por otro lado, un valor de coseno cercano a 0 indica que no tiene indicios para ser considerado una traza.

La herramienta y sus componentes fueron evaluados experimentalmente en dos partes utilizando 3 casos de estudio. Primero se evaluó el desempeño de la recuperación de decisiones de diseño, y luego se procedió a analizar las trazas identificadas por la herramienta. REAssistant no fue evaluada en esta tesis dado que ya ha sido evaluada independientemente en otros estudios empíricos [?, ?, ?]. En los experimentos llevados a cabo, se evaluaron los beneficios de generar las trazas utilizando solo las decisiones de diseño y los *crosscutting concerns* en vez de utilizar todo el contenido de los documentos. Asimismo, se analizaron métricas que permiten establecer el esfuerzo necesario para recuperar las trazas manualmente y compararlo con la herramienta. Con respecto al Buscador de Decisiones de Diseño, la herramienta mostró que es capaz de detectar la mayoría de las decisiones de diseño contenidas en un SAD. En el caso de la detección de trazas, el análisis reveló que el filtrado de los documentos mejora la precisión. Los resultados obtenidos fueron prometedores, observando un incremento notable en la calidad de las trazas recuperadas al filtrar los documentos como así también reduciendo significativamente el esfuerzo con respecto a un proceso enteramente manual.

Las principales contribuciones de este trabajo son:

- La combinación de técnicas de NLP y reglas específicas del dominio de arquitecturas de software para identificar decisiones de diseño en el SAD de forma automática.
- El desarrollo de una técnica para descubrir automáticamente las relaciones de trazabilidad entre documentos de requerimientos y el SAD mediante el análisis de *crosscutting concerns* y decisiones de diseño, respectivamente, para mejorar el desempeño general de la detección.
- La provisión de una herramienta que permite reducir significativamente de esfuerzo necesario para recuperar trazas con respecto a un análisis manual.

1.4. Organización del Trabajo

El resto de este trabajo está organizado de la siguiente manera. El Capítulo 2 explica el marco teórico del trabajo final, detallando conceptos que serán utilizados durante el desarrollo de los demás capítulos. Particularmente, se introducen conceptos del dominio de Ingeniería de Software, tales como: requerimientos de software, arquitectura de software, *crosscutting concerns*, decisiones de diseño, atributos de calidad, tácticas y estilos arquitectónicos. El capítulo también

introduce los conceptos principales de la trazabilidad de software, como así también presenta diversas estrategias para la recuperación semi-automática de relaciones de trazabilidad.

El Capítulo 3 discute trabajos relacionados con la tesis. Específicamente, los trabajos relacionados se organizan en dos líneas de investigación. Por un lado, se analizan trabajos que utilizan técnicas de recuperación de información para identificar conceptos de interés en documentación textual. Por otro lado, también se exploran trabajos enfocados en la recuperación de relaciones de trazabilidad entre distintos artefactos construidos durante el proceso de desarrollo.

El Capítulo 4 presenta el enfoque propuesto y la herramienta prototipo desarrollada, explicando detalladamente los componentes principales que permiten procesar los documentos de requerimientos y el SAD, identificar *crosscutting concerns* y decisiones de diseño y buscar las trazas entre estos.

En el Capítulo 5 se reportan los resultados de la experimentación con el prototipo en tres casos de estudio. El capítulo está dividido en dos secciones, referidas a la evaluación de la recuperación de decisiones de diseño y la evaluación de la recuperación de relaciones de trazabilidad entre documentos.

Finalmente, el Capítulo 6 presenta las conclusiones de esta tesis, discute las ventajas y limitaciones del enfoque. Asimismo, este capítulo enumera también algunas líneas de trabajo futuro. Adicionalmente, se incluye un apéndice que complementa el contenido de esta tesis. El Apéndice A enumera las reglas semánticas utilizadas para recuperar las decisiones de diseño presentes en el SAD.

Capítulo 2

Marco teórico

La especificación de requerimientos de un sistema es una actividad crítica en el desarrollo de software. El análisis de los mismos sirve fundamentalmente para comprender las necesidades de los stakeholders y, de esta manera, poder construir el sistema adecuado [?]. Los analistas tienen la responsabilidad de organizar y documentar dichos requerimientos de manera clara, lo que facilita su análisis, comunicación e implementación. Lalicitación de requerimientos incompleta o de mala calidad puede tener consecuencias negativas en las etapas posteriores del desarrollo [?].

Los documentos de requerimientos (en conjunto con otros artefactos) son utilizados en las etapas de análisis y diseño para construir la arquitectura de software del sistema. La arquitectura de software es el nexo entre los requerimientos y el resultado final del sistema. Los documentos de arquitectura son el vehículo principal para la comunicación entre los diferentes stakeholders [?] y presentan diferentes perspectivas y vistas del diseño de un sistema.

Si bien las etapas de captura de requerimientos y creación de la arquitectura están estrechamente relacionadas, las actividades asociadas con dichas etapas son llevadas a cabo por separado y con una interacción limitada [?]. En la práctica, el proceso de transición de requerimientos a arquitectura está (generalmente) basado en la experiencia, intuición, comunicación y conocimiento del dominio de arquitectos y diseñadores [?]. Evaluar la calidad de la transición es difícil debido a que la recuperación de la trazabilidad es compleja y muchas veces inexistente. Una traza entre requerimientos y componentes de diseño hace explícita la información de transición de un requerimiento (desde que el mismo fue definido) hasta que éste fue satisfecho por la arquitectura (mediante componentes y tácticas arquitectónicas), y viceversa.

Este capítulo está organizado en tres partes. La primer parte introduce conceptos de documentos de requerimientos tales como: casos de uso y *crosscutting concerns*. La segunda parte introduce conceptos de documentos de arquitectura como: atributos de calidad, decisiones de diseño o tácticas arquitectónicas, entre otros. La tercer y última parte describe conceptos de trazabilidad y técnicas utilizadas para recuperar la misma entre documentos de requerimientos y arquitectura.

2.1. Documentos de requerimientos

Los requerimientos definen el problema que tiene que ser resuelto: la finalidad del sistema y lo que se necesita para que cumpla su objetivo. Los requerimientos listan qué debe y no debe hacer el sistema, dejando para etapas posteriores cómo debería hacerse. Según la definición del Instituto de Ingeniería Electrónica y Eléctrica (IEEE), un requerimiento es (i) una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo, (ii) una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar o especificación, y (iii) una representación documentada de una condición o capacidad documentada como las descritas en (i) y (ii).

En el contexto de Ingeniería de Software (Software Engineering, SE), los requerimientos se definen como las necesidades de los clientes, los servicios que el usuario desea que proporcione

el sistema y las restricciones sobre las que el software debe operar. El resultado del proceso de requerimientos es la base para el diseño, la implementación y la evaluación del software. La manera más utilizada para expresar cada requerimiento es en texto plano escrito en lenguaje natural. Estudios recientes muestran que aproximadamente el 79 % de los requerimientos se encuentran especificados usando texto con poca estructura [?].

Los requerimientos pueden ser especificados a varios niveles de detalle. Un requerimiento complejo puede ser separado en requerimientos más simples que logran explicar dicho requerimiento en mayor detalle. También es posible escribir múltiples especificaciones de requerimientos a diferentes niveles.

Las actividades necesarias para definir, documentar y mantener requerimientos en forma adecuada se organizan en un proceso denominado Ingeniería de Requerimientos (Requirements Engineering, RE), el cual constituye el inicio de cualquier ciclo de vida del software. La RE desempeña un papel importante en la SE actual, ayudando a asegurar que la especificación del sistema cumpla con las necesidades de los stakeholders y, en última instancia, que se construirá el producto de software adecuado [?, ?].

Se denomina *documentos de requerimientos* al conjunto de documentos que contienen la descripción precisa de los requerimientos. El principal objetivo de estos documentos es captar información acerca de que debe y no debe hacer un sistema, en base a lo especificado por los stakeholders. Pueden contener gráficos y tablas además de texto y son usados como medio de comunicación entre los distintos stakeholders.

Existen diferentes formas de organizar y documentar los requerimientos [?], algunas de ellas dependen del modelo de desarrollo que se esté utilizando. En la sección 2.1.1 se describe una de las representaciones más utilizadas para documentar los requerimientos, los casos de uso [?]. Estos permiten organizar los requerimientos de un sistema de acuerdo a la perspectiva de sus usuarios finales y los servicios que debe proveer.

2.1.1. Casos de uso

Los casos de uso son una técnica para especificar el comportamiento de un sistema. Cada caso de uso describe una secuencia de interacciones entre un sistema y actores que usan alguno de sus servicios [?]. Los actores son personas u otros sistemas que interactúan con el sistema. Los casos de uso capturan quién (actor) hace qué cosa (interacción) con el sistema y con qué propósito (objetivo), sin enfocarse en las partes internas del sistema.

Un caso de uso es iniciado por un actor con un objetivo particular en mente, y se completa con éxito cuando se cumple ese objetivo. En él se describe la secuencia de interacciones entre actores y el sistema necesarias para prestar el servicio que cumpla el objetivo. También se describen las interacciones en situaciones de fallas y el comportamiento del sistema. El sistema es tratado como una "caja negra", se describen las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. Los casos de uso se deben describir en términos de negocio, cómo el actor interactúa con el sistema y lo que el sistema hace en respuesta.

La Figura 2.1 presenta un ejemplo de un caso de uso donde se pueden ver las distintas partes del mismo:

- Precondiciones: definen las condiciones necesarias para instanciar el caso de uso.
- Postcondiciones: reflejan el estado en el cual se encuentra el sistema luego de ejecutar el caso de uso.
- Flujo básico: describe los pasos de interacción entre los actores y el sistema donde no ocurren errores.
- Flujo alternativo: describe el comportamiento del sistema en situaciones de errores, o que no son consideradas comunes. Por ejemplo, una tarjeta de crédito vencida.

Caso de uso: Registro en el sistema
Descripción: Permite registrarse en el sistema como usuario.
Actores: Usuario.
Precondiciones: La página principal del sistema ya fue cargada.
Postcondiciones: Usuario registrado.
Flujo básico: 1. El usuario selecciona la opción Registrarse en el menú principal. 2. El sistema muestra la pantalla de ingreso de usuarios. 3. El usuario carga los datos y presiona el botón de registro. 4. El sistema almacena la información ingresada, creando un nuevo usuario. 5. El sistema muestra un mensaje informando que se procesó correctamente el registro.
Flujo alternativo: 3.1. El sistema muestra mensaje de error informando que la información ingresada está incompleta. 4.1. El sistema muestra mensaje de error informando que el usuario ingresado ya existe en el sistema.

Figura 2.1: Ejemplo de caso de uso

Existen dos formas de reutilizar el contenido de los casos de uso, estos son las relaciones de *include* y *extend*.

- La relación «*include*» se utiliza para extraer fragmentos de casos de uso que están duplicados en múltiples casos de uso. El caso de uso incluido no es un caso de uso en sí mismo, depende del caso de uso original, y el caso de uso original no estará completo sin el caso de uso incluido.
- La relación «*extend*» se utiliza para agregar comportamiento condicional a un caso de uso. A diferencia de la relación «*include*», donde el caso de uso base está incompleto, en este caso, se trata de un caso de uso totalmente completo y funcional. Normalmente se usa para agregar comportamiento opcional que puede no ser tan importante en el caso de uso original. Desde un punto de vista conceptual es similar a un flujo alternativo. La ventaja de la relación de «*include*» es que el flujo alternativo y los sub-flujos dependientes están contenidos en un caso de uso separado.

A pesar de que los casos de uso tienen un carácter estrictamente funcional [?], estos igualmente suelen tener menciones o referencias a ciertas características no funcionales del sistema, y estas características están frecuentemente relacionadas en mayor o menor medida con *cross-cutting concerns* [?].

2.1.2. *Concerns* y *crosscutting concerns*

Se define a un *concern* como "cualquier asunto de interés en un sistema de software" [?]. Por lo tanto, el desarrollo de software tiene que tratar con un gran número de *concerns*. Algunos están relacionados al producto (el software) que debe ser creado, como la funcionalidad y el rendimiento. Otros *concerns* están relacionados con el proceso de desarrollo en sí mismo, como los tiempos y costos del desarrollo.

Un sistema de software está generalmente sujeto a muchos *concerns*. Durante el modelamiento de dichos *concerns*, varias técnicas se pueden utilizar para representarlos, siguiendo un mecanismo de descomposición en particular, es decir, un criterio de separación de *concerns* (SoC). El resultado de tal mecanismo conducirá a una partición específica de *concerns*. Si el mecanismo de descomposición elegido permite descomponer los *concerns* en un pequeño número de dimensiones, entonces, algunos de los *concerns* no serán fáciles de aislar en un único modulo y eventualmente terminaran afectando otros modulos.

Un *concern* que corta a través de múltiples representaciones se llama un *crosscutting concern* (CCC) [?, ?]. Un *crosscutting concern* tiene dos características principales: (i) se encuentra

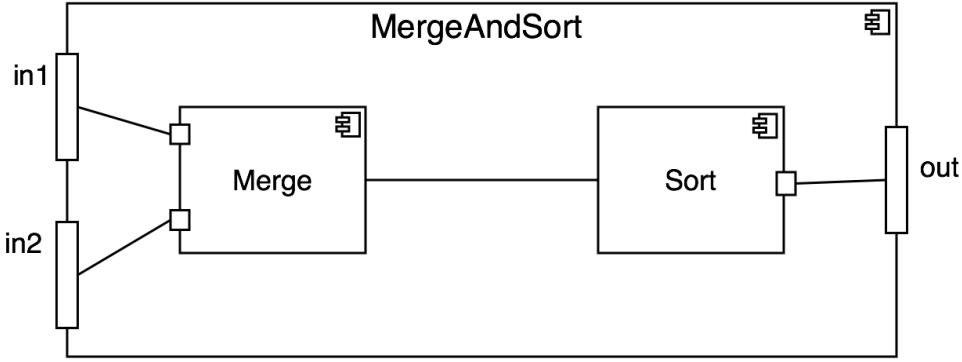


Figura 2.2: Vista de componentes y conectores

disperso entre varias representaciones, y (ii), más de un *concern* puede estar presente en una sola representación. La primer situación se llama *scattering*, mientras que la segunda se llama *tangling*. Ambas producen problemas en el desarrollo de software [?, ?]. Si un *crosscutting concern* debe cambiar, el cambio puede afectar a muchos elementos en la descomposición. Si la descomposición debe ser cambiada, numerosos *crosscutting concerns* pueden ser afectados. Generalmente los *crosscutting concerns* están relacionados con los atributos de calidad (descritos en la Sección 2.2.1) y como no adhieren a la forma de describir la funcionalidad, quedan dispersos en múltiples documentos [?].

2.2. Documentos de arquitectura

Para describir qué son los documentos de arquitectura primero hay que definir qué es una arquitectura de software. Esta puede definirse como: "la estructura o estructuras del sistema, la cual comprende los componentes del software, las propiedades externamente visibles de esos componentes, y las relaciones entre ellos" [?]. A la hora de comunicar una arquitectura entre los diferentes stakeholders, ésta debe ser documentada. Se denomina documento de la arquitectura (Software Architecture Document, SAD), al documento que busca describir la arquitectura de un sistema de software. Los documentos de arquitectura son el vehículo principal de comunicación entre los stakeholders [?].

Existen varias maneras conocidas para documentar una arquitectura, y la alternativa utilizada depende principalmente de los stakeholders y las personas involucradas en desarrollar la documentación. Habitualmente se suelen utilizar documentos de texto junto con diferentes notaciones gráficas, denominadas vistas [?, ?]. Estas son una representación de un conjunto coherente de elementos arquitectónicos a través de las cuales pueden especificarse los distintos aspectos técnicos y funcionales, así como también las decisiones involucradas durante el proceso de creación de la arquitectura [?].

Las Figuras 2.2 y 2.3 muestran un ejemplo de dos tipos de vistas que pueden encontrarse en los documentos de arquitectura. La primera es una vista de módulos que muestra los módulos del sistema junto con sus relaciones. Esta vista permite entre otras cosas ver como la funcionalidad fue distribuida y sus responsabilidades. La segunda figura muestra una diagrama de deployment donde se puede ver cómo los componentes de la arquitectura se organizan físicamente.

En los documentos de arquitectura, las vistas son acompañadas por una descripción textual donde se detallan, cada una de sus partes, como estas interactúan entre sí y sus responsabilidades. Esta no es la única información presente en los documentos de arquitectura. A continuación se describen algunos de los conceptos que forman parte de los documentos de arquitectura.

2.2.1. Atributos de calidad

Según Barbacci et al. [?], la calidad de software se define como el grado en el cual el software posee una combinación de atributos. Tales atributos no hacen referencia a la funcionalidad pro-

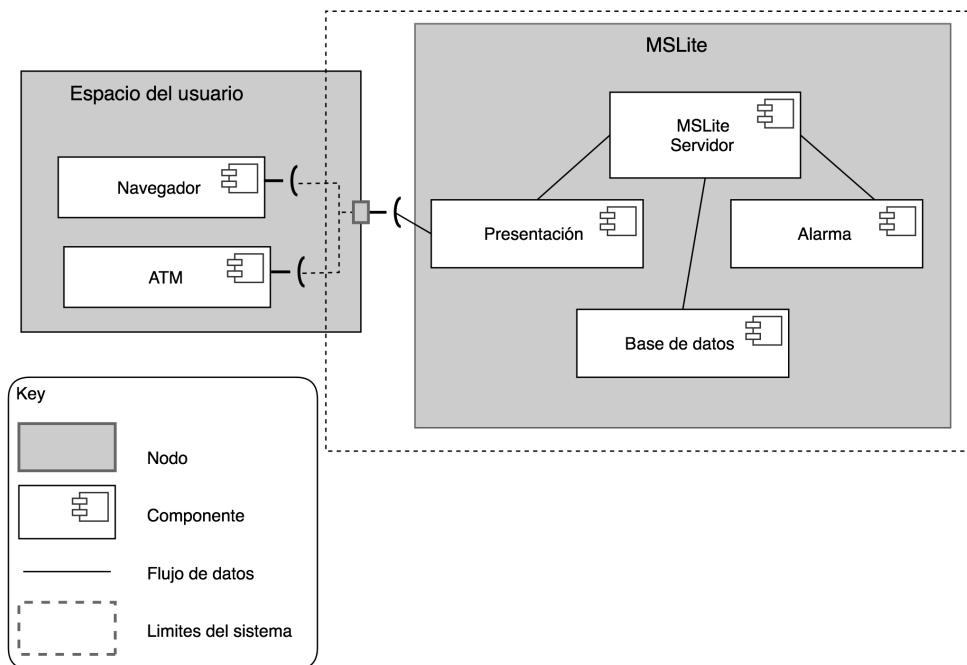


Figura 2.3: Ejemplo de vista de deployment

piamente dicha del sistema, sino que hacen referencia a características que éste debe satisfacer. Estas características se conocen con el nombre de atributos de calidad.

En el contexto del desarrollo de un sistema, los atributos de calidad no existen de forma aislada sino que están estrechamente relacionados con uno o más requerimientos funcionales los cuales deben satisfacer ciertas restricciones [?]. Un ejemplo de atributo de calidad es: "El sistema debe tener un tiempo de respuesta menor a 1 segundo". En este caso no se habla de nueva funcionalidad sino de cómo el sistema debe presentar cierta característica.

Los atributos de calidad pueden referirse al comportamiento de un sistema en tiempo de ejecución, al diseño arquitectónico del sistema, o la experiencia del usuario, entre otros. Algunos de estos atributos están relacionados al diseño general del sistema, mientras que otros son solo visibles durante el tiempo de ejecución o el tiempo de diseño. La medida en que la aplicación posee una combinación deseada de los atributos de calidad como usabilidad, performance, seguridad, etc. indica el nivel de éxito de la arquitectura y su calidad en general.

Debido a que el concepto de calidad de software es muy abstracto y subjetivo, se han definido diversos estándares y modelos de calidad que permiten cuantificar los atributos de calidad de un sistema. Uno de estos modelos de calidad es el descrito en [?], el cual permite analizar varios atributos de calidad dentro de seis categorías:

- **Modifiability (modificabilidad):** El costo de realizar un cambio, tomando en cuenta qué se va a cambiar y cuándo y quién va a realizar el cambio.
- **Availability (disponibilidad):** Tiene que ver con las fallas del sistema y las consecuencias asociadas a estas fallas.
- **Performance (desempeño):** Se refiere al tiempo de respuesta del sistema cuando ocurre un evento como interrupciones, mensajes, solicitudes del usuario, etc.
- **Usability (usabilidad):** Concierne a la facilidad de usar el sistema, por ejemplo, la facilidad de aprender las características, aprender a usar el sistema de una forma eficiente, minimizar el impacto de errores en su manejo, qué tan fácil se adapta el sistema a las necesidades del usuario y la satisfacción que tiene el usuario para con el sistema.
- **Security (seguridad):** Es la habilidad que tiene el sistema de rechazar usos no autorizados o ataques, sin dejar de proveer sus servicios a usuarios legítimos.

- Testability (facilidad de pruebas): Se refiere a la facilidad de encontrar las fallas del sistema a través de pruebas y de esta forma poder solucionarlas.

Es bastante común encontrar indicios acerca de atributos de calidad en documentos de requerimientos [?]. Estos se manifiestan en conjunto con los requerimientos funcionales. Por ejemplo, en los documentos de casos de uso se suelen observar oraciones tales como: "El sistema debe generar el reporte en menos de 5 segundos". En esta oración existe una clara referencia a un atributo de calidad de *performance*, el cual se encuentra descripto de forma parcial (en el contexto del resto del sistema). Esta situación es muy similar al descubrimiento de los *crosscutting concerns* en documentos de casos de uso.

Para satisfacer cualquiera de los requerimientos de los atributos de calidad, es necesario tener en cuenta el impacto potencial sobre otros requerimientos durante el diseño de la arquitectura [?]. Por ejemplo, si un arquitecto considera que para que el sistema opere más rápido se pueden eliminar ciertos componentes independientes, tiene que tener cuidado de que esta decisión no afecte negativamente la flexibilidad de cambios del sistema (es decir, que reduzca su modificabilidad). Por lo tanto, al realizar cambios en la arquitectura, se deben analizar las ventajas y desventajas entre los distintos atributos de calidad [?].

2.2.2. Tácticas

Las tácticas son un conjunto de decisiones sobre la arquitectura que buscan satisfacer un atributo de calidad [?]. Existen diferentes tipos de tácticas y estas pueden ser organizadas dependiendo del atributo de calidad que buscan mejorar. A continuación se listan algunas tácticas de diseño correspondiente a los atributos de calidad sugeridos en [?]:

- Tácticas de disponibilidad: Ping/echo, Latido (heartbeat), excepciones, voto, redundancia activa, entre otras.
- Tácticas de modificabilidad: Mantener coherencia semántica, anticipar cambios esperados, generalizar el módulo con el que se trabaja, encapsulamiento, polimorfismo, etc.
- Tácticas de desempeño: Aumento de eficiencia computacional, administrar cantidades de eventos, control de frecuencia de muestreo, limitar tiempos de ejecución, introducir concurrencia, etc.
- Tácticas de seguridad: Autentificación de usuarios, mantenimiento de confidencialidad de datos, mantenimiento de integridad, detección de intrusos, restauración de estado, etc.
- Tácticas de facilidad de pruebas: Separación de interfaz e implementación, interfaces de pruebas especializadas, monitores integrados, etc.
- Tácticas de usabilidad: Estas tácticas se dividen en las tácticas en tiempo de ejecución y tácticas en tiempo de diseño.

Las tácticas se utilizan para satisfacer algún atributo de calidad de cierta funcionalidad descrita en los documentos de requerimientos. Esta funcionalidad suele tener menciones o referencias a determinados atributos de calidad y estos suelen encontrarse agrupados en *crosscutting concerns*. Por lo tanto, las tácticas presentes en los documentos de arquitectura, están relacionadas con los *crosscutting concerns*.

2.2.3. Patrones de arquitectura

Se denomina patrón arquitectónico al conjunto de tácticas que suelen aplicarse en conjunto. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

Layered
Broker
Pipe and filter
Client server
Peer to peer
Server-oriented architecture
Publish-subscribe
Shared-data
Map-reduce
Multi-tier

Figura 2.4: Listado de patrones arquitectónicos

Aunque un patrón arquitectónico expresa una imagen de un sistema, no es una arquitectura como tal. Un patrón arquitectónico es más un concepto que captura elementos esenciales de una arquitectura de software. Muchas arquitecturas diferentes pueden implementar el mismo patrón y por lo tanto compartir las mismas características. Dado que el uso de patrones de arquitectura es muy común, existen varios catálogos de patrones de arquitectura. La Figura 2.4 enumera varios patrones arquitectónicos muy utilizados en la industria.

Uno de los aspectos más importantes de los patrones arquitectónicos es que permiten satisfacer diferentes atributos de calidad. Por ejemplo, algunos patrones representan soluciones a problemas de rendimiento y otros pueden ser utilizados con éxito en sistemas de alta disponibilidad. La elección de los patrones de arquitectura es comúnmente documentada y puede ser recuperada observando los documentos de arquitectura dado que es una de las primeras y mas importantes decisiones durante el diseño arquitectónico.

2.3. Decisiones de diseño

Una arquitectura de software puede ser vista como el resultado de un conjunto de decisiones de diseño (DDD) y los documentos que justifican esas decisiones [?]. Estas decisiones consisten de un conjunto de cambios tales como adiciones, sustracciones y modificaciones a la arquitectura de software [?]. Estos cambios son en algunos casos el resultado de elegir una táctica o estilo arquitectónico y permiten satisfacer tanto la funcionalidad como también los atributos de calidad asociados a dicha funcionalidad. Algunos ejemplos de decisiones de diseño son la elección de un protocolo de comunicación para aumentar la seguridad, la elección de un lenguaje de programación para mejorar la modificabilidad, o el uso de una arquitectura cliente-servidor.

2.4. Trazabilidad

La trazabilidad es la medida en la que se puede establecer una relación entre dos o más artefactos junto con la habilidad de poder examinar esta relación [?]. En este caso, los artefactos involucrados en la trazabilidad son artefactos del desarrollo de software como el código fuente, los documentos de requerimientos, arquitectura o los casos de prueba entre otros. A continuación se introducen conceptos básicos de trazabilidad, algunos de estos pueden además verse en la Figura 2.5:

- **Tipo de artefacto:** Una etiqueta que caracteriza a los artefactos de trazas que tienen una estructura o propósito similar. Por ejemplo, los requerimientos, el código fuente y casos de test pueden ser vistos como tipos de artefactos.
- **Artefacto fuente:** El artefacto del que se origina la traza.
- **Artefacto destino:** El artefacto en el destino de una traza.

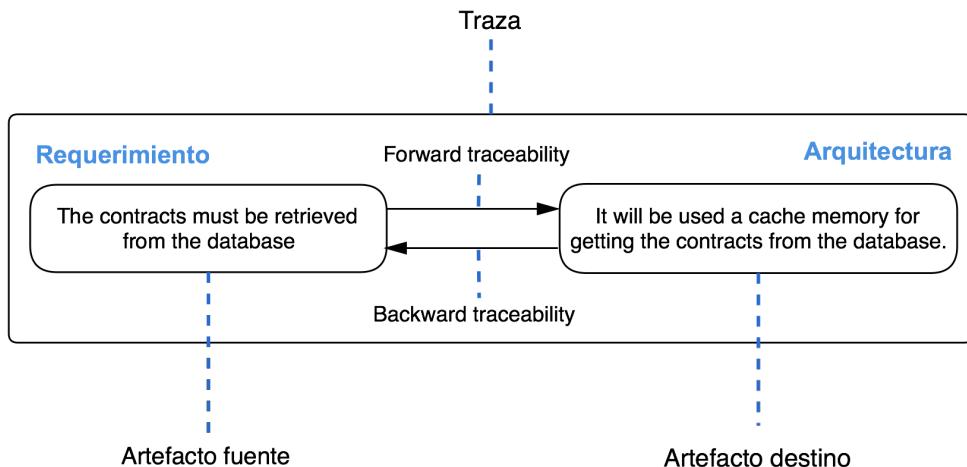


Figura 2.5: Estructura de una traza

- **Traza:** Una asociación específica entre un par de artefactos, un artefacto origen y un artefacto destino. Puede o no incluir información adicional como el tipo de enlace u otro atributo semántico.
- **Relación de trazabilidad:** Se llama relación de trazabilidad o traza al conjunto de vínculos creados entre dos tipos de artefactos. Por ejemplo, la relación de trazabilidad puede ser el conjunto de trazas entre requerimientos y casos de prueba. Esta es comúnmente documentada en una matriz de trazabilidad.
- **Matriz de trazabilidad:** Una matriz donde se documentan las trazas mostrando que pares de artefactos están asociados a través de trazas.

Existen tres maneras de clasificar las trazas entre artefactos de software y dependen del modo en que las trazas fueron recuperadas:

- **Trazas manuales:** La trazabilidad es establecida de forma manual por parte de un analista. En este caso, el analista puede usar una herramienta para almacenar las trazas, la matriz de trazabilidad o visualizar los artefactos, pero todo lo referido a recuperar las trazas es una tarea manual.
- **Trazas automatizadas:** Se establece la trazabilidad a través de técnicas, métodos y herramientas automatizadas. El analista solo está involucrado en la selección de los artefactos y su participación no es necesaria a la hora de recuperar las trazas.
- **Trazas semi-automáticas:** La trazabilidad se establece a través de una combinación de técnicas, métodos, herramientas automatizadas y actividades humanas. Por ejemplo, las técnicas automatizadas pueden sugerir trazas candidatas y luego es responsabilidad del analista el verificarlas.

Independientemente del modo en el que fueron recuperadas, las trazas siempre tienen un artefacto fuente y un artefacto destino y son bi-direccionales. Es decir que si existe una traza desde *A* hacia *B* entonces la traza de *B* a *A* también existe.

En el desarrollo de software, se considera *Forward traceability* cuando la traza recuperada sigue los pasos del desarrollo de software, es decir, desde requerimientos a través del diseño hasta el código fuente. Por otro lado si el orden es al revés, es decir, la traza comienza en el código fuente a través del diseño hasta los requerimientos, esta se denomina *Backward traceability*.

2.4.1. Trazabilidad entre documentos de requerimientos y arquitectura

Si bien las etapas de captura de requerimientos y creación de la arquitectura están estrechamente relacionadas, las actividades asociadas con dichas etapas son llevadas a cabo por separado

y con una interacción limitada [?]. En la práctica, el proceso de transición de requerimientos a arquitectura está generalmente basado en la experiencia, intuición, comunicación y conocimiento del dominio de los arquitectos del sistema [?].

Una traza entre requerimientos y componentes de diseño hace explícita la información de transición de un requerimiento (desde que el mismo fue definido) hasta que éste fue satisfecho por la arquitectura (mediante componentes y tácticas arquitectónicas), y viceversa [?]. Establecer la trazabilidad entre los requerimientos y los componentes arquitectónicos que los satisfacen, como así también mantener estas asociaciones actualizadas a medida que el sistema evoluciona es importante dado que: (i) permite verificar que el diseño arquitectónico del sistema implementa los requerimientos claves, y que los aspectos esenciales del diseño tengan trazas hacia los requerimientos, (ii) facilita las tareas de análisis ante cambios en los requerimientos, simplificando la incorporación de nueva funcionalidad y el estudio de su impacto en la arquitectura, y (iii) permite identificar decisiones de diseño tomadas por los arquitectos y que no están asociadas a requerimientos.

En la industria, la ausencia de estos registros de trazabilidad entre artefactos se atribuye principalmente a que realizar los análisis pertinentes para obtenerlos es un proceso largo, costoso y propenso a errores. Adicionalmente, es necesario que la persona que realiza el análisis tenga que conocer detalladamente los requerimientos y la arquitectura del sistema. Hay que comprender los requerimientos para saber con qué parte de la arquitectura se corresponden. Por otro lado, hay que entender la arquitectura y las decisiones tomadas para determinar los requerimientos vinculados. Existen técnicas que buscan recuperar las trazas de forma automática utilizando la información de los artefactos de software. En las siguientes secciones se describen 3 métodos para recuperar trazas automáticamente.

2.4.2. Métodos automáticos para la recuperación de trazas

Dada la complejidad de la tarea de recuperación de trazabilidad entre documentos de requerimientos y de arquitectura, existe una amplia variedad de métodos de recuperación de información que buscan ayudar al analista a recuperar trazas entre dichos documentos. A continuación se presentan los 3 métodos automáticos más utilizados para la obtención de trazas denominados: Modelos probabilísticos, modelos de espacio vectorial (*Vector Space Model*, VSM) y LSA (*Latent Semantic Analysis*). Estos métodos definen como documento a la unidad mínima de información (oración o un párrafo) que se utiliza como artefacto origen o destino en una traza.

Modelos probabilísticos

Este método se basa en modelos probabilísticos donde la similitud entre documentos está dada por el valor de probabilidad luego de aplicar la Fórmula 2.1 entre los documentos. En la fórmula, D_i el artefacto de destino y S_i es el artefacto de origen. $Pr(D_i \vee S_i)$ se computa utilizando un modelo de bayes [?].

$$(D_i, S_i) = Pr(D_i \vee S_i) \quad (2.1)$$

Vector space model

El modelo de espacio vectorial es un método matemático utilizado para la construcción de representaciones vectoriales a partir de documentos. Este método representa una colección de documentos en una matriz de término-por-documento cuyo elemento E_{ij} indica la asociación entre el término i y documento j.

A la hora de recuperar trazas de artefactos de software utilizando VSM, los documentos en VSM suelen ser las oraciones que se encuentran en los artefactos y los términos de VSM suelen ser las palabras dentro de las oraciones. La Figura 2.6 muestra un ejemplo de cómo suelen representarse los artefactos de software cuando se utiliza VSM. El ejemplo muestra documentos de 3 términos. Al usar esta técnica, cada documento es presentado como un vector donde cada término recibe un valor que mide la asociación del término para con el documento.

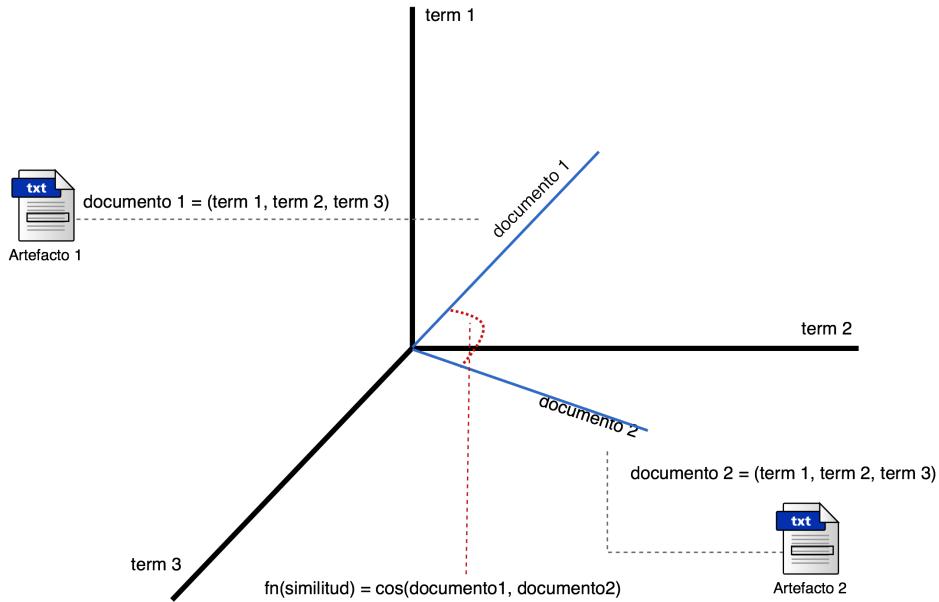


Figura 2.6: Representación vectorial de documentos en VSM

La similitud entre los documentos se suele medir por el coseno entre los vectores correspondientes, lo que aumenta a medida que el número de términos iguales en ambos documentos aumenta. En general, dos documentos se consideran similares si sus vectores correspondientes en el punto de espacio VSM apuntan en la misma dirección.

Latent semantic analysis

Latent Semantic Analysis [?] es una extensión de VSM desarrollada con el fin de superar el problema de las relaciones entre términos. LSA no requiere que dos documentos que son semánticamente similares tengan los mismos términos, como es el caso de VSM [?]. Esto significa que al utilizar LSA sobre artefactos de software, es posible recuperar una traza a pesar de que éstos no comparten muchos términos.

LSA busca recuperar y representar matemáticamente "estructuras latentes" a partir de los documentos. Para descubrir estas estructuras, el método utiliza una técnica estadística denominada Singular Value Decomposition (descomposición en valores singulares, SVD) [?]. Esta técnica así como VSM permite representar los documentos en una matriz cuyo tamaño se puede controlar mediante un parámetro. Esto es importante a la hora de trabajar con documentos de gran tamaño donde VSM generaría una matriz muy grande. La matriz generada utilizando SVD representa matemáticamente las estructuras contenidas en los documentos por lo tanto no es necesario que los términos en los documentos coincidan para que exista una relación entre ambos. Así como con VSM, la similitud de los documentos está dada por el coseno entre los vectores que representan dichos documentos.

2.5. Resumen

La definición de los requerimientos de un sistema es una actividad crítica en el desarrollo de software. El análisis de los mismos sirve fundamentalmente para comprender las necesidades de los stakeholders y, de esta manera, poder construir un sistema adecuado [?]. Los analistas tienen la responsabilidad de organizar y documentar dichos requerimientos de manera clara. Lalicitación de requerimientos incompleta o de mala calidad puede tener consecuencias negativas en las etapas posteriores [?] del desarrollo.

El objetivo principal de los documentos de requerimientos, es captar información acerca de que debe y no debe hacer un sistema, en base a lo especificado por los stakeholders. Estos

documentos en conjunto con otros artefactos son utilizados en las etapas de análisis y diseño para construir la arquitectura de software del sistema. La arquitectura de software es el nexo entre los requerimientos y el resultado final del sistema. Se denomina documento de la arquitectura (*Software Architecture Document*, SAD), al documento que busca describir los distintos aspectos técnicos y funcionales, así como también las decisiones involucradas durante el desarrollo la arquitectura de un sistema de software. Los documentos de arquitectura son el vehículo principal de comunicación entre los stakeholders [?].

Si bien las etapas de captura de requerimientos y creación de la arquitectura están estrechamente relacionadas, las actividades asociadas con dichas etapas son llevadas a cabo por separado y con una interacción limitada [?]. En la práctica, el proceso de transición de requerimientos a arquitectura está (generalmente) basado en la experiencia, intuición, comunicación y conocimiento del dominio de arquitectos y diseñadores [?]. Evaluar la calidad de la transición es difícil debido a que la recuperación de la trazabilidad es compleja y muchas veces inexistente. Una traza entre requerimientos y componentes de diseño hace explícita la información de transición de un requerimiento (desde que el mismo fue definido) hasta que éste fue satisfecho por la arquitectura (mediante componentes y tácticas arquitectónicas), y viceversa.

Establecer la trazabilidad entre los requerimientos y los componentes arquitectónicos que los satisfacen, es importante dado que: (i) ayuda a los stakeholders a ver el impacto de los requerimientos sobre la arquitectura, (ii) facilita las tareas de análisis ante cambios en los requerimientos y (iii) asegura que la documentación de requerimientos esté completa y que esta sea consistente con las decisiones tomadas en el diseño de la arquitectura.

En la industria, la ausencia de estos registros de trazabilidad entre artefactos se atribuye principalmente a que realizar los análisis pertinentes para obtenerlos es un proceso largo, costoso y propenso a errores. Adicionalmente, es necesario la persona que realiza el análisis tenga que conocer detalladamente los requerimientos y la arquitectura del sistema. Por este motivo, el uso de técnicas computacionales que simplifiquen este tipo de análisis puede llegar a ser muy valiosas para los analistas en términos de tiempo/esfuerzo. Gracias a los avances realizados en los últimos años en las áreas de lingüística computacional y recuperación de información es posible recuperar las trazas entre artefactos de software de manera automatizada. Específicamente, la aplicación de técnicas de IR, ML y NLP pueden facilitar la recuperación de trazas entre documentos de requerimientos y arquitectura.

Capítulo 3

Trabajos relacionados

En este capítulo se presentan algunos trabajos relevantes para esta tesis, discutiendo sus ventajas y desventajas. El análisis de los trabajos relacionados está dividido en dos grupos.

El primer grupo comprende trabajos que buscan recuperar información de artefactos de software (en particular documentos de requerimientos y de arquitectura) a través de la utilización de técnicas de procesamiento de lenguaje natural (*Natural Language Processing*, NLP) [?, ?, ?, ?] o aprendizaje de máquina (*Machine Learning*, ML) [?]. Dentro de estos trabajos, se distinguen aquellos que analizan documentos estructurados, tales como especificaciones de casos de uso o historias de usuario (*User Stories*) [?, ?], y también de aquellos enfocados en el análisis de documentos de arquitectura [?]. Además se describen y comparan tres herramientas de recuperación de información sobre documentos de requerimientos.

El segundo grupo comprende trabajos que buscan recuperar relaciones de trazabilidad entre distintos artefactos de software de forma semi-automática [?]. Estos trabajos utilizan modelos probabilísticos [?, ?, ?], de espacio vectorial (VSM) [?, ?] y *Latent Semantic Analysis* (LSA) [?, ?, ?, ?, ?]. Para cada una de estas técnicas, se describe una herramienta que la utiliza. Esto tiene como meta no solo comparar las técnicas, sino también la interacción con el analista, la flexibilidad y la velocidad de las mismas.

3.1. Recuperación de información de artefactos de software

La mayoría de los trabajos que recuperan información de documentos de requerimientos y de arquitectura, analizan el texto de los documentos porque se basan en la hipótesis de que la mayoría de la información que se requiere extraer es textual. En un estudio reciente, Mich et al. [?] estableció que casi el 79 % de los requerimientos son especificados en el lenguaje natural, con poca o ninguna estructura. Del mismo modo, los documentos de arquitectura son especificados en su mayoría en lenguaje natural [?].

Los documentos de requerimientos contienen información acerca del problema que tiene que ser resuelto: la finalidad del sistema y lo que se necesita para que cumpla su objetivo. Estos listan qué debe y no debe hacer el sistema, describen la interacción entre el sistema a desarrollar y los actores involucrados. Por otro lado, los documentos de arquitecturas o SAD (*Software Architecture Document*) contienen las descripciones de las decisiones arquitectónicas tomadas y las responsabilidades funcionales de los componentes de la arquitectura. Describen entre otras cosas como los requerimientos fueron implementados.

Para recuperar información de los documentos de requerimientos y arquitectura, las técnicas de recuperación de información (*Information Retrieval*, IR), NLP, ML e inteligencia artificial (*Artificial Intelligence*) han sido utilizados para automatizar la detección de patrones en el texto y extraer información útil para los analistas humanos. Los trabajos de Kof et al. [?] y Sawyer et al. [?] han estudiado la viabilidad de las técnicas de NLP para el análisis y mejora de las actividades de requerimientos. Kof et al. [?] examinaron la eficacia de varias técnicas de NLP, como por ejemplo, parsing profundo (*Deep Parsing*), resolución de anáforas (*Anaphora Resolution*).

tion) y agrupamientos por similitud (*Similarity Clustering*) [?], en el contexto de documentos de requerimientos. En Jansen et al. [?] se utilizan técnicas de NLP para recuperar decisiones de diseño a partir de documentos de arquitectura. Estos trabajos destacan los beneficios del uso de técnicas de NLP y IR para analizar documentos de requerimientos y de arquitectura.

En la siguiente sección se describen dos categorías de trabajos, aquellos enfoques que analizan requerimientos descritos en lenguaje natural y los que identifican *crosscutting concerns* en los documentos de requerimientos

3.1.1. Procesamiento de documentos de requerimientos no estructurados

A continuación se presentan una serie de trabajos que procesan el contenido textual y no estructurado de documentos de requerimientos. Algunos de estos trabajos presentan técnicas que pueden ser utilizadas para procesar el contenido textual de documentos de arquitectura sin grandes cambios.

La herramienta denominada CIRCE [?] procesa el texto de los requerimientos dados como entrada y produce un modelo intermedio. Después, el desarrollador es capaz de analizar diferentes vistas del modelo, tales como Entidad-Relación, flujo de datos, orientado a objetos, redundancia, entre otras. Otra herramienta interesante es COLOR-X [?], que también toma como entrada una especificación de requerimientos y genera un modelo intermedio. Este modelo utiliza una notación formal definida en la Representación del Lenguaje Común (Common representation language, CPL). Aunque COLOR-X aplica varias técnicas de NLP automatizadas, la generación del modelo intermedio no se produce automáticamente, y se necesita la intervención del analista a lo largo del proceso. Una vez obtenido el modelo intermedio, la herramienta genera dos salidas: un modelo de objetos estáticos (similar a un diagrama de clases UML) y un modelo basado en eventos (similar a una máquina de estados UML). En la misma línea, AbstFinder [?] es una herramienta que ayuda a encontrar abstracciones de texto en lenguaje natural para ser utilizadas en la obtención de requerimientos. Las abstracciones son conceptos relevantes que se pueden entender sin tener que comprenderlos en detalle. Esta herramienta utiliza técnicas de comparación de patrones para identificar las abstracciones. Sawyer et al. desarrolló una versión mejorada de AbstFinder, denominada Relevance-driven Abstraction Identification (RAI) [?]. En lugar de utilizar técnicas de búsqueda de patrones, RAI aplica una combinación de técnicas de NLP que pueden analizar tanto términos individuales como palabras compuestas, e identificar abstracciones candidatas a partir de ellas. Además, RAI incluye métodos estadísticos, tales como simple frequency y corpus-based frequency profiling, para clasificar las abstracciones detectadas. Logical Structure Extraction (LSE) [?] es un framework que permite describir las "estructuras y componentes lógicos" y proporciona mecanismos para extraerlos de documentos de texto enriquecidos. En este trabajo, una estructura lógica refiere a un artefacto de alto nivel, como los requerimientos funcionales o casos de uso, y un componente lógico se refiere a partes de nivel inferior de estructuras lógicas, tales como un actor o un flujo principal dentro de un caso de uso. LSE genera contenido estructurado de templates de requerimientos específicos (por ejemplo, documentos de casos de uso basados en plantillas de RequisitePro¹) accesibles para su posterior procesamiento automático. El framework permite entonces buscar a través de estructuras y componentes lógicos con la ayuda de un lenguaje de consulta y recuperar aquel texto relevante para las inspecciones de los documentos.

En el caso de los documentos de arquitectura, la herramienta ADDRA [?] fue diseñada para recuperar decisiones de diseño a partir de documentos de especificaciones de arquitecturas. La herramienta utiliza una plantilla basada en un modelo conceptual para describir las decisiones de diseño arquitectónicas. La hipótesis de ADDRA es que los cambios en la arquitectura se deben a decisiones de diseño arquitectónico. La herramienta utiliza el conocimiento tácito del arquitecto, el cual obtiene a partir de un cuestionario de preguntas, junto con documentos de arquitectura, para identificar los cambios y las decisiones de diseño relacionadas.

¹RequisitePro es una herramienta para el manejo de requerimientos que se distribuye como plugin en Rational Suite Analyst Studio

ADRA consta de cinco pasos, que se organizan en un proceso iterativo: Paso 1: definir y seleccionar los releases. Paso 2: diseño detallado. Paso 3: puntos de vista de la arquitectura de software. Paso 4: delta de la arquitectura. Paso 5: decisiones de diseño arquitectónicas. El paso 4 es uno de los más importantes y dentro del cual se definen los llamados delta de la arquitectura. Estos muestran los cambios en la arquitectura los cuales son el resultado de las decisiones de diseño que luego son recuperadas en el paso 5. Una de las desventajas de ADRA es que este paso es completamente manual y es responsabilidad del arquitecto identificar los cambios en la arquitectura.

La herramienta MIA permite identificar características definidas en los documentos de requerimientos que pueden más tarde ser consideradas puntos de variabilidad en una familia de productos de software [?]. Con este objetivo, MIA aplica una serie de módulos de análisis léxicos (por ejemplo, etiquetado POS, lematización, eliminación de duplicados y stopwords, entre otros). La extracción de características puede ayudar a aprovechar oportunidades de reuso que eventualmente reducen los costos de producción. Una ventaja de MIA es la aplicación de técnicas de NLP robustas, que permiten extraer requerimientos similares en familias de productos de software. Desafortunadamente, la identificación de estos requerimientos esta basado en análisis léxicos y diccionarios, lo cual puede llegar a traer problemas con el uso de sinónimos en las descripciones textuales [?].

La Tabla 3.1 presenta un resumen de los trabajos discutidos en esta sección. Estos trabajos fueron comparados de acuerdo a cinco características: objetivo, técnicas, características, ventajas y desventajas. El objetivo se refiere al propósito del enfoque, técnicas se refiere a los algoritmos subyacentes y las estrategias aplicadas en el enfoque, las características se refieren a las propiedades destacables de los enfoques, y las ventajas y desventajas enuncian los pro y contra de cada enfoque.

3.1.2. Identificación de *crosscutting concerns* en documentos de requerimientos

Se define un *concern* como "cualquier asunto de interés en un sistema de software". Cuando éste corta a través de múltiples representaciones se denomina un *crosscutting concern* (CCC) [?]. Recuperar los CCCs a partir de documentos de requerimientos así como las decisiones de diseño de los documentos de arquitectura es de gran utilidad dado que ambos están relacionados con los atributos de calidad del sistema [?]. En esta sección se describen tres herramientas que recuperan *crosscutting concerns* a partir de documentos de requerimientos. Estas herramientas fueron seleccionadas teniendo en cuenta las técnicas que utilizan y su posible adaptación a nuevos artefactos de software como son los documentos de arquitectura.

Theme/Doc

Baniassad y Clarke desarrollaron el enfoque Theme [?, ?], el cual expone las relaciones entre los comportamientos del sistema introduciendo la noción de temas (*themes*) tanto en los niveles de requerimientos como de diseño [?]. Para identificar los temas el enfoque recibe como entrada una lista de acciones y entidades clave, las cuales deben ser provistas por el analista. La idea principal del enfoque es que permite ir refinando las diferentes vistas de requerimientos provistas para que se pueda determinar qué temas son crosscutting y cuáles no, además de explicitar en qué lugares se manifiestan [?].

Un tema es un elemento de diseño: una colección de estructuras y comportamiento que representan una característica. Temas múltiples pueden ser combinados o integrados para formar un "todo funcional" de acuerdo a un modelo multidimensional, es decir, formar el sistema en sí. Existen dos tipos de temas: los temas base, los cuales pueden compartir algunas estructuras y comportamientos con otros temas base, y los temas crosscutting, los cuales tienen comportamientos que se solapan con la funcionalidad de los temas base.

Como se aprecia en la Figura 3.1, utilizando las acciones y entidades ingresadas a la herramienta puede generar diferentes vistas de los requerimientos de forma automática. Los analistas

Autores [Enfoque / Herramienta]	Objetivo	Técnicas	Características	Ventajas	Desventajas
Ambriola and Gervaci "CIRCE" [?]	Produce un modelo intermedio y deriva múltiples vistas de él	Canonization y Tokenization. Matching con reglas difusas	Procesamiento flexible mediante reglas textuales editables	Ajustable para diferentes dominios de software	Pipeline de procesamiento básico. Requiere un glosario de requerimientos etiquetados manualmente
Burg "COLOR-X" [?]	Genera un modelo en CPL y deriva diagramas de clases y transición de estados	Etiquetado POS. Análisis gramatical (verbo, objeto directo, etc.). Etiquetado semántico con WordNET	Ánalisis semántico y sintáctico de los requerimientos	Técnica específica, personalizable con conceptos específicos del dominio	La notación CPL es compleja. Requiere la intervención de los analistas
Goldin and Berry "AbstFinder" [?]	Infiere abstracciones (conceptos relevantes) de documentos de requerimientos	Representación del texto como una secuencia de bytes. Técnicas de comparación por patrones	Uno de los primeros en atacar este problema, aplicando desplazamientos circulares de palabras clave	Poca complejidad temporal y espacial	Falta de análisis sintáctico y semántico. Salida no necesariamente útil para la mejora de los artefactos de RE
Gacitúa, Sawyer and Gervaci "RAR" [?]	Mismo objetivo que "AbstFinder"	Etiquetado POS, lemmatización y eliminación de stopwords. Tratamiento de términos de múltiples palabras. Perfiles de frecuencia basados en Corpus	Análisis sintáctico, semántico y léxico para la detección de abstracciones. Ranking de importancia estadística de las abstracciones	Combinación avanzada de técnicas	Ánalisis semántico "ligeró" Salida no necesariamente útil para mejorar los artefactos de RE
Rauf et al. "LSE" [?]	Extracción de información relevante a partir de documentación de requerimientos enriquecida	Reconocimiento de patrones de regiones mediante plantillas. Soporte de consultas para recuperar regiones de texto	Paso previo para utilizar técnicas de procesamiento en artefactos textuales	Técnica ajustable para diversos formatos de documentos y plantillas (PDF, DOC, HTML, etc.)	Solo sirve para extraer información de documentos, no para analizarlos
Boutkova y otros "MIA" [?]	Identificación de características que pueden variar en el futuro en artefactos de requerimientos	Etiquetado de POS, stemming y lemmatización	Extracción semi-automática de características de software. Soporte para análisis de la variabilidad de las características	Robustez por la omisión de conocimiento gramatical e información pragmática	Se basa solo en análisis léxicos. Requiere un diccionario
Jansen et al. "ADDRA" [?]	Identificación de decisiones de diseño en documentos de arquitectura	Modelo conceptual para caracterizar y descubrir decisiones de diseño	Diferentes técnicas para conversión del conocimiento del arquitecto	Enfoque sistemático, uso explícito del conocimiento tácito. Recupera el tipo de arquitectura.	Depende del conocimiento del arquitecto. Su desempeño depende de cómo se configura la herramienta.

Cuadro 3.1: Enfoques utilizados sobre artefactos de software

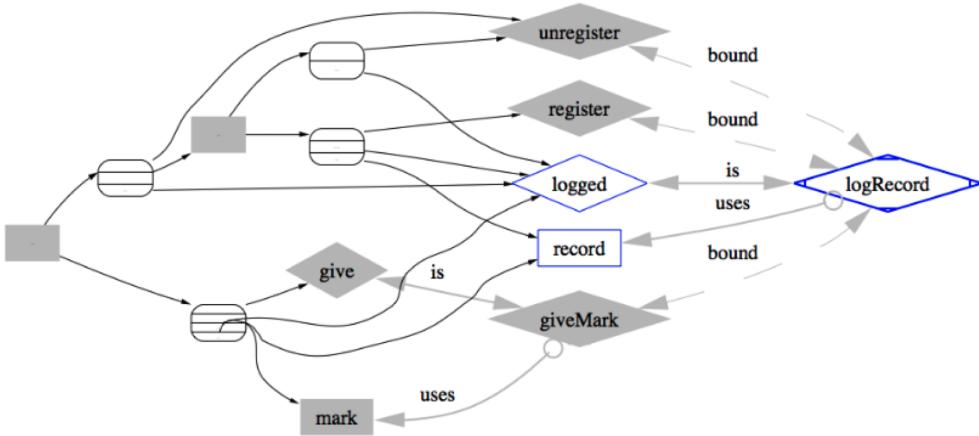


Figura 3.1: Visualizaciones de Temas en Theme/Doc

pueden navegar las relaciones entre los comportamientos y decidir cuáles de ellos son *cross-cutting* y cuáles no. Theme/Doc permite gestionar la detección de los *crosscutting concerns*, destacando aquellos *concerns* que cortan trasversalmente múltiples artefactos. Sin embargo, no queda en claro si una herramienta visual tal como Theme/Doc puede escalar a documentos de requerimientos de gran longitud.

Theme provee vistas que exponen aquellos comportamientos que están descritos en varios documentos de requerimientos. La *major action view* (vista de acción mayor) es utilizada para mostrar gráficamente la relación entre los requerimientos y las acciones. La vista de acción recortada es usada para representar temas crosscutting. Aparte de estas vistas, la vista de tema es utilizada para planificar el diseño y el modelamiento de los temas identificados. La vista de tema difiere de las vistas de acción en que no sólo muestra requerimientos y acciones, sino que también muestra elementos clave del sistema que deben ser considerados para cada tema de diseño.

EA-Miner

Uno de los enfoques más relevantes para encontrar *concerns* en documentos de requerimientos fue presentado por Sampaio et al. [?]. Este enfoque describe cómo diferentes fuentes de requerimientos no estructuradas (entrevistas, descripciones del sistema en lenguaje natural, etc.) pueden ser extraídas automáticamente para ayudar al ingeniero de requerimientos a rápidamente identificar y construir un modelo orientado a aspectos de los requerimientos del sistema.

La herramienta EA-Miner se basa en el enfoque anterior y permite a los analistas encontrar *crosscutting concerns* [?, ?, ?]. EA-Miner realiza dos actividades principales para llevar a cabo la detección: (i) etiquetado POS y (ii) análisis de semántica. La primer actividad asigna etiquetas de parte del discurso a cada palabra, como por ejemplo, sustantivo, verbo o adjetivo. La segunda actividad categoriza cada palabra de acuerdo a su significado. Esta actividad se enfoca sobre los verbos y cómo éstos están distribuidos en los distintos casos de uso. La herramienta reconoce tres tipos de *concern* "concerns base", *crosscutting concerns* no funcionales y *crosscutting concerns* funcionales.

Para utilizar EA-Miner el usuario debe proporcionar un documento de texto el cual la herramienta envía a WMATRIX. Luego de analizar el documento, la herramienta le presenta los puntos de vista candidatos, los cuales el usuario puede editar. A partir de estos puntos de vista, la herramienta recupera *crosscutting concerns*. El usuario puede elegir varias opciones para el filtrado de *crosscutting concerns*, como por ejemplo, excluir los verbos que no representen una acción como son los verbos auxiliares *is*, *are* y *be*.

REAssistant

REAssistant (REquirements Analysis Assistant) [?] es una herramienta que procesa casos de uso textuales para descubrir *crosscutting concerns* candidatos que están ocultos en estas especificaciones. Para ello, la herramienta hace uso de técnicas de NLP para procesar los requerimientos, y utiliza una combinación de estrategias semánticas de búsqueda. Particularmente, REAssistant implementa dos técnicas para la búsqueda de *concerns*: clustering semántico y consultas enriquecidas semánticamente. La primera técnica utiliza un algoritmo clustering para agrupar comportamientos semánticamente similares, los cuales luego son propuestos como *crosscutting concerns* candidatos. La segunda técnica implementa una motor de búsqueda que posibilita a los analistas identificar los *concerns* mediante reglas que aprovechan la información para encontrar referencias implícitas y explícitas de los *crosscutting concerns*.

Para utilizar REAssistant, el analista debe especificar los casos de uso utilizando la herramienta. Posteriormente, debe seleccionar una de las dos técnicas provistas para ejecutar la herramienta, consultas semánticamente enriquecidas y cluster semántico. Como salida, la herramienta genera diferentes vistas, con las cuales el analista puede validar los resultados obtenidos.

3.1.3. Análisis comparativo de enfoques para identificar *crosscutting concerns*

En esta sección se discutirán las propiedades de los enfoques para identificar *crosscutting concerns*. La comparación considera diferentes aspectos de los enfoques/herramientas. Inicialmente, se consideraron cinco aspectos, a saber:

- Dependencia de la estructura: Analiza sobre qué tipo de documentos es aplicable la herramienta. Cada herramienta puede llegar a tener limitaciones o restricciones sobre la estructura de los documentos que acepta como entrada. También puede ocurrir que la herramienta no tenga ninguna limitación y pueda trabajar sobre cualquier tipo de documento, independientemente de la estructura que éste posea.
- Nivel de automatización: Se refiere al grado de interacción que tiene el analista con la herramienta. Como generalmente las herramientas son del tipo semi-automatizadas, este criterio se enfoca principalmente en si es necesario tener un conocimiento previo de los documentos o del sistema en sí, si es necesario ingresar información adicional con respecto al sistema, la cantidad de etapas donde el desarrollador debe tomar decisiones, entre otras.
- Tipo de análisis: Clasifica el análisis del texto realizado según su nivel léxico, sintáctico, o semántico. El nivel léxico se logra cuando se analizan las palabras, el sintáctico cuando se analiza la estructura de las sentencias escritas en los documentos, y el semántico cuando se interpreta el significado de las palabras según el contexto donde se encuentran.
- Escalabilidad: Describe las capacidades que proveen las herramientas para poder aplicar los enfoques a problemas en sistemas de gran tamaño y de qué manera lo realizan. Además, debe tenerse en cuenta si la herramienta es capaz de analizar grandes cantidades de información en un tiempo aceptable, es decir, que no se produzcan demoras considerables al usar las herramientas.
- Efectividad: Describe la precisión en la recuperación de información de los artefactos de software.
- Integrabilidad: Se refiere a la capacidad de la herramienta analizada de acoplarse a diferentes metodologías y procesos de desarrollo de software.
- Velocidad: Se refiere a la medición del tiempo empleado para llevar a cabo el análisis del texto, para poder compararlas tanto con respecto al análisis manual como con respecto a las otras técnicas.

Criterio / Enfoque	Theme/Doc	EA-Miner	REAssistant
Dependencia de la estructura	Estructurados (textos amoldados a la herramienta)	Independiente de la estructura	Estructurados (documentos de caso de uso)
Nivel de automatización	Entrada de información adicional (se debe leer toda la documentación)	No necesita la entrada de información adicional. No depende del nivel de conocimiento del analista	No necesita la entrada de información adicional. No depende del nivel de conocimiento del analista
Tipo de análisis	Léxico	Léxico, sintáctico y semántico	Léxico, sintáctico y semántico
Escalabilidad	Semi-escalable (tiene problemas para manejar el tamaño de las vistas)	Escalable (soporta filtros para manejar grandes cantidades de información)	Escalable (se pueden añadir reglas fácilmente para detectar nuevos crosscutting concerns)
Efectividad	Algo efectivo para encontrar todos los concerns de un sistema. Problemas con palabras semánticamente similares	Muy efectivo. Discierne entre aspectos funcionales y no funcionales	Muy efectivo. Detecta los crosscutting concerns relacionados explícita e implícitamente con la funcionalidad
Integrabilidad	Se integra al diseño con UML.	Está pensado para ser integrable con cualquier metodología o proceso	Está pensado para ser integrable con cualquier metodología o proceso
Velocidad	No tan rápida	Rápida	Rápida

Cuadro 3.2: Comparación de las herramientas de recuperación de *crosscutting concerns* descritas

La Tabla 3.2 presenta un resumen del análisis comparativo. Con respecto al nivel de automatización, Theme/Doc depende de información adicional por parte del analista para su ejecución. Asimismo, esta herramienta requiere que el analista ingrese una lista de palabras a partir de las cuales se realiza la búsqueda de *crosscutting concerns*. Mientras que, EAMiner y REAssistant no requieren información adicional para ser utilizadas, ya que aprovechan la información léxica y semántica a la hora de recuperar *crosscutting concerns*.

Uno de las principales desventajas de EAMiner es el tiempo que puede tomarle al analista validar la lista de *crosscutting concerns* candidatos para llegar a tener aquellos que son relevantes. Además, el uso de filtros hace que algunos conceptos importantes puedan quedar fuera de la lista. En el caso de REAssistant, las técnicas de consultas permiten que la herramienta pueda ser utilizada sin necesidad de información adicional por parte del analista. Además, esta técnica podría utilizarse en documentos de arquitectura. Se podrían escribir consultas para identificar decisiones de diseño dado que la técnica es fácilmente adaptable, y permite utilizar información tanto léxica como semántica.

REAssistant y EA-Miner presentan mejores resultados en cuanto a nivel de automatización, tipo de análisis, escalabilidad, efectividad y velocidad. Con respecto a la efectividad, REAssistant tiene mejores resultados que EA-Miner [?]. Asimismo, REAssistant está diseñada de forma tal que la intervención del analista es menor que en EA-Miner.

Tanto REAssistant como EA-Miner recuperan *crosscutting concerns* a partir de documentos de requerimientos, esto es de gran utilidad dado que estos están relacionados con los atributos de calidad así como las decisiones de diseño presentes en los documentos de arquitectura. Por lo tanto, recuperar esta información es muy importante para recuperar la trazabilidad entre documentos de requerimientos y arquitectura.

3.2. Recuperación de trazabilidad entre artefactos de software

En la última década, varios autores han aplicado técnicas de IR [?, ?, ?] para el problema de la recuperación de las trazas entre artefactos software. En general, una herramienta de recuperación de trazabilidad basada en IR compara un conjunto de objetos origen contra otro conjunto de objetos destino y mide la similitud entre todas las posibles combinaciones. La hipótesis es que los artefactos que tienen una gran similitud léxica probablemente estén relacionados, por lo que existiría una potencial relación de trazabilidad (es decir, una traza). Por esta razón, este tipo de técnicas también utilizan algún método para acotar la lista de soluciones, presentando al analista sólo el subconjunto que cumple dichas restricciones (por ejemplo, un threshold en el nivel de similitud) [?, ?, ?, ?, ?].

Los primeros métodos utilizados para recuperar trazas entre artefactos de software fueron los modelos probabilísticos [?, ?, ?] y los modelos de espacio vectorial (*vector space model*, VSM) [?, ?, ?]. El método probabilístico calcula el valor de similitud como la probabilidad de que un artefacto se relacione con otro artefacto. En el contexto de recuperación de trazabilidad, los modelos probabilísticos han sido usados para recuperar la trazabilidad entre diferentes requerimientos y código fuente [?, ?, ?], y manuales de usuario y código fuente [?]. En estos trabajos se proponen además diferentes estrategias de mejora como clustering [?, ?], donde la hipótesis es que las trazas tienden a ocurrir en grupos, y hierarchical [?], donde se busca agrupar las trazas en una jerarquía de acuerdo al artefacto al que pertenecen.

El método VSM representa objetos (es decir, los documentos) como vectores a partir de sus términos, y calcula la similitud entre dos objetos como el coseno del ángulo entre los vectores correspondientes [?, ?]. El método VSM se ha utilizado [?, ?] para recuperar trazas entre requerimientos [?, ?, ?], entre requerimientos y código fuente [?, ?, ?, ?], entre las solicitudes de mantenimiento y documentos de software [?, ?], entre manuales de usuario y código fuente [?, ?, ?], entre los requerimientos e informes de defectos [?], y entre varios otros tipos de artefactos (por ejemplo, casos de uso, diagramas UML, casos de prueba [?, ?]). Una crítica común a VSM es que no tiene en cuenta las relaciones entre términos (por ejemplo, sinónimos y relaciones de hiperonimia entre palabras).

Con el objetivo de solucionar problemas de relaciones entre términos, se ha propuesto una extensión a VSM denominada *Latent Semantic Analysis* (LSA) [?]. Esta asume que hay una cierta "estructura latente" o subyacente en el uso de la palabra que está parcialmente oscurecida por la variabilidad en la selección de palabras. Utiliza una técnica estadística denominada descomposición en valores singulares (*Singular Value Decomposition*, SVD) [?] para estimar esta estructura latente. LSA se basa en el contenido semántico de los artefactos en lugar de su contenido léxico. Como consecuencia, una traza puede ser recuperada incluso si los artefactos involucrados no comparten muchos términos literales [?].

LSA ha sido utilizado para recuperar trazas entre requerimientos de alto y bajo nivel [?, ?], entre varios tipos de artefactos de software incluidos los casos de prueba, código fuente y documentos de arquitectura [?, ?, ?, ?, ?, ?]. Se han obtenido resultados prometedores al aplicar LSA para la recuperación de trazas. En Marcus y Maletic (2003) [?] se comparó la precisión de las trazas recuperadas con LSA contra las obtenidas con VSM y modelos probabilísticos. Las evaluaciones fueron llevadas a cabo en los casos de estudio presentados en Antonio et al. (2002) [?]. Como resultado de la experimentación, se determinó que tanto VSM como los modelos probabilísticos requieren de un análisis morfológico del texto para conseguir el mismo rendimiento que LSA [?].

La Tabla 3.3 resume los enfoques basados en IR que se utilizan para abordar el problema de recuperación de la trazabilidad. Para cada uno, se muestra el método IR utilizado, la estrategia para mejorar el método si es que se utiliza alguna y qué tipos de trazas fueron recuperadas. Algunos de estos enfoques fueron materializados en herramientas, tal es el caso de Poirot [?], ADAMS [?] y RETRO [?], las cuales se describen y comparan en la siguiente sección.

3.2.1. Herramientas para recuperar trazas entre artefactos de software

En esta sección se describen y comparan las herramientas Poirot, RETRO y ADAMS. Estas fueron seleccionadas debido a que cada herramienta utiliza una técnica diferente, modelos probabilísticos, VSM y LSA, respectivamente. Para cada herramienta se describe, cómo los analistas la utilizan, su arquitectura, ventajas y desventajas. A su vez, se describen los criterios de comparación definidos para evaluar las herramientas. Por último, se presentan los resultados de la comparación.

Poirot

Poirot es una herramienta web que permite recuperar la trazabilidad entre diferentes artefactos de software utilizando modelos probabilísticos. Permite recuperar trazas entre requerimientos, elementos de diseño, código fuente y otros artefactos de software los cuales pueden estar almacenados en herramientas de terceros como Rational DOORS², Rational Rose³ o repositorios de código fuente. Resultados experimentales sobre múltiples artefactos mostraron que Poirot es capaz de obtener una precisión de 15-30 % dado un *recall* de entre 90-95 % [?].

A la hora de utilizar la herramienta el analista necesita importar los artefactos de software como documentos XML. Estos pueden ser obtenidos a partir de herramientas de terceros como DOORS o Rose. Una vez importados los documentos el analista puede recuperar la trazabilidad entre dos artefactos previamente seleccionados o realizar consultas y recuperar documentos relevantes a esa consulta sobre alguno de estos.

La Figura 3.2 muestra la interfaz gráfica (GUI) de Poirot. En ésta, se puede ver como las trazas candidatas son presentadas al usuario, el nivel de confianza de cada traza y el usuario puede aceptar o no la misma. Poirot también proporciona una visualización de la dispersión y probabilidades de las trazas recuperadas la cual permite identificar fácilmente las trazas asociadas a cierto documento. Además, la herramienta permite ver aquellas trazas que considera que no están relacionadas, las cuales denomina "*unlikely links*". A la hora de interactuar con otros sistemas, Poirot permite exportar las trazas en formato XML.

²<http://www-03.ibm.com/software/products/es/ratidoor>

³<http://www-03.ibm.com/software/products/es/ratirosefami>

Autores [Enfoque / Herramienta]	Método utilizado	Estrategia de mejora	Tipo de trazas	Ventajas	Desventajas
Zou et al. (2007) [?]	Modelos probabilístico	Cobertura de término de consulta y frases	Trazas entre requerimientos y artefactos de diseño	Funciona bien sobre código fuente donde la mayoría de los términos se repiten. Construye el modelo probabilístico rápidamente	No funciona bien con sinónimos o mucha cantidad de términos diferentes
Zou and Cleland-Huang (2007) [?]	Modelos probabilístico	Busca agrupar las trazas candidatas en clusters para facilitar la validación del analista.	Trazas entre requerimientos y artefactos de diseño	Agrupa las trazas candidatas en clusters para mejorar la validación	La presencia de sinónimos genera problemas cuando se arman los clusters
Lin et al. (2006) "Poirot: Trace Maker" [?]	Modelos probabilístico	Modelos basados en jerarquía	Trazas entre diferentes tipos de requerimientos (alto y bajo nivel) y estos con el código fuente, manuales de usuario o casos de prueba	Mejora la precisión cuando los artefactos de origen o destino presentan una jerarquía	Los datos tienen que estar en un formato XML específico de la herramienta
De Lucia et al. (2006) [?]	VSM	Feedback del usuario	Trazas entre diferentes tipos de requerimientos (alto y bajo nivel) y entre otros tipos de artefactos como casos de uso, diagramas de interacción, código fuente y casos de prueba	Buenos resultados para documentos que no tienen una gran cantidad de términos. No tiene en cuenta las relaciones entre términos	La precisión está relacionada al número de términos distintos en los documentos
Hayes et al.(2006) "RETRO" [?]	VSM	Feedback del usuario, palabras clave y uso de sinónimos	Trazas entre diferentes tipos de requerimientos (alto y bajo nivel) y estos con el código fuente	Buenos resultados para documentos que no tienen una gran cantidad de términos. Soporta sinónimos gracias a un diccionario	Se deben especificar las palabras claves
Marcus and Maletic (2003) [?]	LSA	Ninguna	Trazas entre código fuente en C++ y manuales de usuario y entre código fuente Java y requerimientos funcionales	Buenos resultados para gran cantidad de documentos	Calcular el número de dimensiones
Lormans and van Deursen (2005) [?]	LSA	Nueva estrategia para la selección de enlaces de trazabilidad	Trazas entre requerimientos y artefactos de diseño y entre requerimientos y casos de prueba	La selección del método de filtrado es automática	Calcular el número de dimensiones
De Lucia et al.(2008) "ADAMS" [?]	LSA	Feedback del usuario	Trazas entre muchos tipos de artefactos como casos de uso, diagramas de interacción código fuente, casos de prueba, etc	Buenos resultados para gran cantidad de documentos. Muy buenos resultados sobre documentos de texto no estructurados	Calcular el número de dimensiones

Cuadro 3.3: Resumen de trabajos para recuperar trazas entre artefactos de software

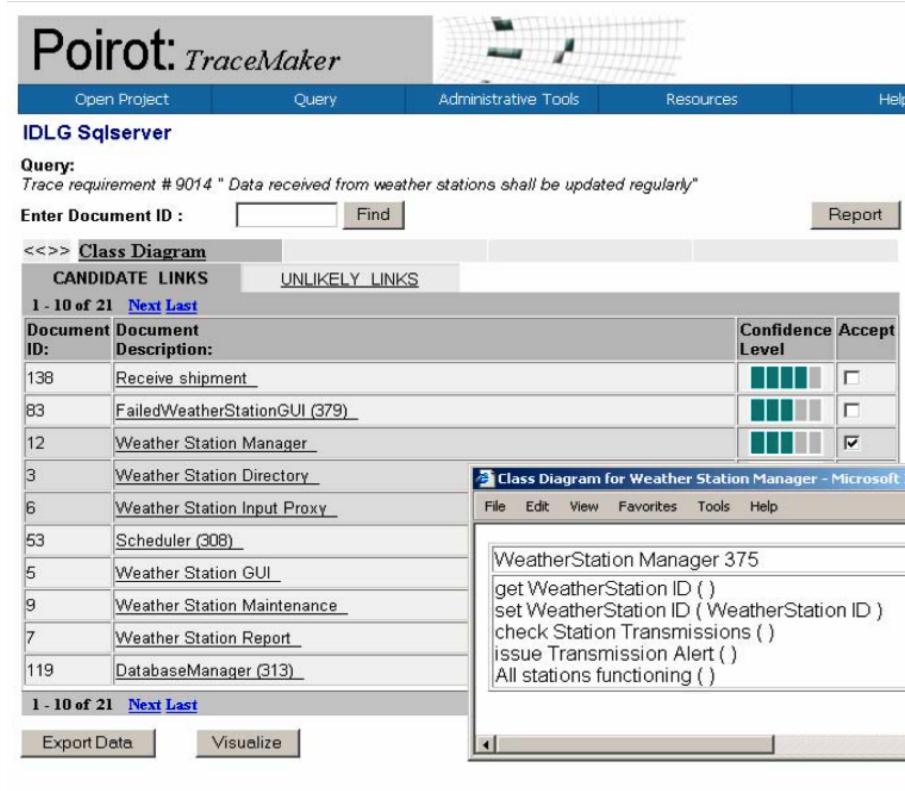


Figura 3.2: Interfaz gráfica de Poirot

Para recuperar trazas, Poirot decide si existe una traza entre dos documentos en base al valor de probabilidad $Pr(\frac{q}{d})$ siendo q el documento origen y d el documento destino. La probabilidad $Pr(\frac{q}{d})$ es calculada utilizando el contenido textual tanto de q como d . La Formula 3.1 muestra como se calcula dicha probabilidad. La función depende de la frecuencia de los términos t_i que ocurren tanto en q como en d [?].

$$Pr\left(\frac{q}{d}\right) = \left[\sum_{i=1}^k Pr\left(\frac{q}{t_i}\right) Pr(q, t_i) \right] / Pr(q) \quad (3.1)$$

Dado que el valor de probabilidad entre documentos depende principalmente del contenido textual, Poirot incluye una técnica de mejora basada en modelos de jerarquía. Ésta asume que los documentos dentro de los artefactos están organizados de forma jerárquica y por lo tanto las palabras utilizadas para nombrar y describir componentes de alto nivel capturan el contenido general de los componentes de bajo nivel. Por lo tanto, para calcular la probabilidad, Poirot tiene en cuenta la frecuencia de los términos que aparecen en los componentes que contiene a q y d [?]. La Figura 3.3 muestra un ejemplo de trazas entre requerimientos y código fuente. El hecho de que existan una traza T1 entre P1 y R3, aumenta la probabilidad de que T2 sea considerada como traza candidata.

RETRO

La herramienta RETRO (*REquirements TRacing On target*) [?, ?] fue diseñada para recuperar la trazabilidad entre diferentes tipos de documentos de requerimientos que contienen información textual sin estructura. RETRO es una herramienta independiente pero puede ser utilizada en conjunto con otras herramientas de administración de proyectos.

Inicialmente el analista debe cargar el contenido textual de los requerimientos en la herramienta. Luego, la herramienta puede ser utilizada para recuperar las trazas. La herramienta solo soporta documentos de texto sin formato, por lo tanto no distingue entre documentos estructurados y no estructurados. Asimismo, la herramienta tiene funcionalidad para exportar las trazas recuperadas en formato XML.

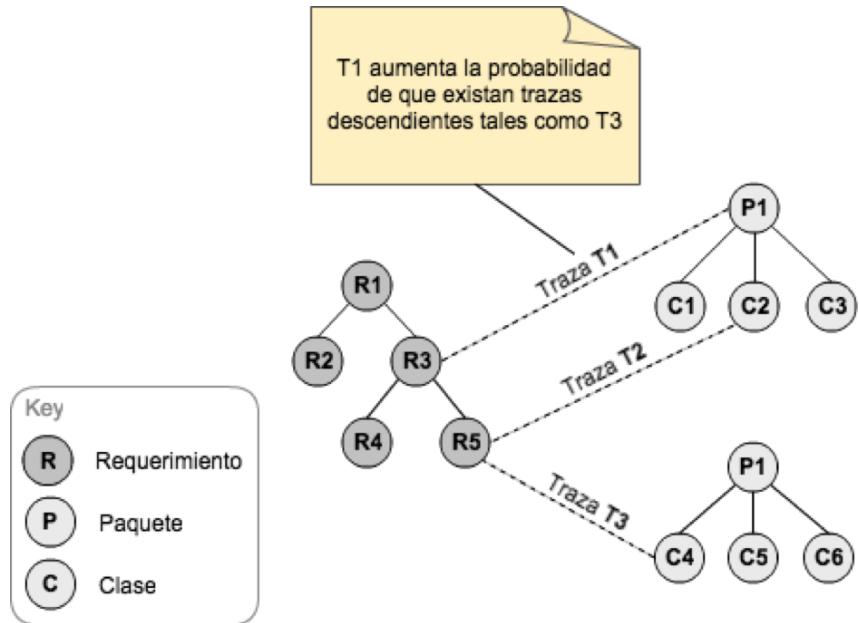


Figura 3.3: Ejemplo de la representación jerárquica de artefactos y trazas por Poirot

La Figura 3.4 muestra la interfaz gráfica (GUI) de RETRO. En la parte superior izquierda de la pantalla se encuentra la lista de requerimientos de alto nivel, mientras que a la derecha se encuentra la lista de bajo nivel. El analista puede seleccionar cualquiera de los requerimientos de alto nivel y la herramienta actualizará la lista de la derecha con aquellos requerimientos de bajo nivel asociados mediante relaciones de trazabilidad.

La Figura 3.5 muestra la arquitectura de RETRO. El componente IR Toolbox contiene los algoritmos de IR adaptados para la recuperación de trazas. Estos algoritmos se encargan de procesar los documentos de requerimientos provistos por los analistas. El componente Filter and analysis permite filtrar la lista de trazas candidatas en base a las preferencias del analista. En la parte inferior de la pantalla, el analista puede modificar los parámetros de filtrado. El analista puede elegir filtrar las trazas cuyo valor de similitud sea menor a un valor predeterminado, o obtener sólo un porcentaje del total de trazas candidatas, entre otros filtros. La Figura 3.4 muestra un valor de 0,07 como filtro y 4 trazas que superan ese valor de peso.

La herramienta usa VSM para recuperar la trazabilidad. Este método representa cada documento de una colección como un vector, y asigna a cada palabra (o término) un valor numérico denominado peso. Cada documento es procesado y se eliminan las palabras que no tienen importancia (stopwords) como "a", "and", "to" o "shall" entre otras. Luego se aplica un proceso de stemming a cada palabra con el fin de que las palabras "information", "informational" y "informative" sean tratadas de igual forma [?]. El conjunto de términos resultantes $D = \{k_1, k_2, k_3, \dots, k_n\}$ se forma con la unión de los términos hallados en todos los documentos. Cada documento d_i es luego representado como un vector $d_i = (w_{i1}, w_{i2}, w_{i3}, \dots, w_{ik})$ de términos/palabras claves. La Formula 3.2 describe como calcular la frecuencia de ocurrencia del término en la colección de documentos.

$$w_{ij} = tf_{ij} \cdot \log \left(\frac{n}{df_i} \right) \quad (3.2)$$

Con el objetivo de asignar un peso a cada término en D para luego construir los vectores, diferentes esquemas pueden ser usados. La herramienta utiliza *tf-ifd* (*Term Frequency–Inverse Document Frequency*). Este esquema se basa en la fórmula 3.2, donde w_{ij} denominado frecuencia del término (*term frequency*) k_j en el documento d_i , es la frecuencia de ocurrencia normalizada de k_j en d_i , mientras que $\log \left(\frac{n}{df_i} \right)$ es la denominada frecuencia d_j , inversa del documento (*inverse document frequency*) para el término k_j . Por lo tanto, el peso del término es proporcional a la frecuencia con la que aparece en el documento e inversamente proporcional a la frecuencia con

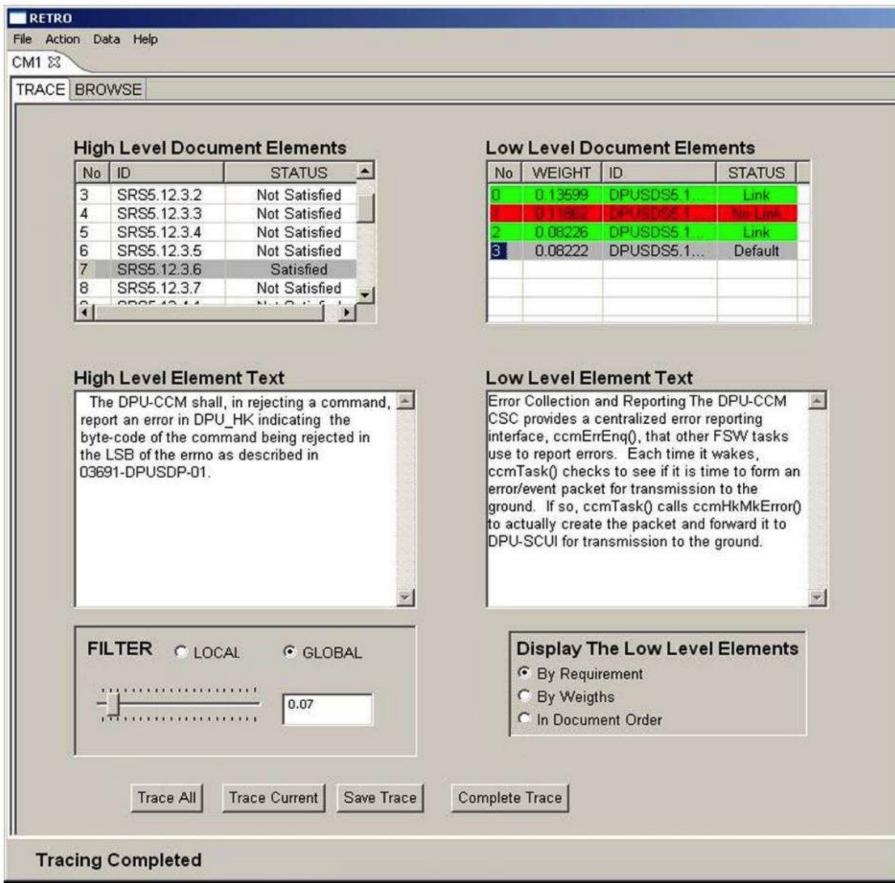


Figura 3.4: Interfaz gráfica de RETRO

la que aparece en el resto de los documentos.

La herramienta realiza los cálculos necesarios para manipular el texto de entrada y representar los documentos como vectores mediante el componente *Build* (ver Figura 3.5). Los vectores son almacenados en el componente *Representation*. En el momento de generar las trazas candidatas, la herramienta calcula el coseno del ángulo comprendido entre los vectores y en caso de pasar el filtro del componente *Filter and Analysis* es presentado al analista mediante el componente *GUI* (ver Figura 3.5).

ADAMS

ADvanced Artefact Management System (ADAMS) es una herramienta que utiliza LSA para recuperar relaciones de trazabilidad entre diferentes artefactos de software [?]. Dicha herramienta provee soporte para el manejo de versiones, posibilitando administrar la trazabilidad de forma activa, propagando notificaciones de cambios cuando nuevos artefactos son agregados, actualizados o modificados [?].

La herramienta ayuda al analista en la identificación de posibles problemas en la descripción textual de los artefactos trazados generando trazas especiales denominadas "trazas de advertencia". Estas son aquellas trazas recuperadas manualmente por analistas y que no fueron descubiertas por la herramienta. Las trazas de advertencia pueden deberse a problemas en la documentación donde los artefactos involucrados pueden no estar claramente descritos.

La Figura 3.6 muestra los pasos necesarios para utilizar la herramienta. En primer lugar, el analista selecciona los tipos de artefactos de software a partir de los cuales se quiere recuperar las trazas. Estos artefactos pueden ser casos de uso, código fuente o casos de prueba, entre otros. Dado que cada artefacto puede estar representado por más de un documento, el segundo paso permite seleccionar los documentos pertinentes para recuperar las trazas. Por último, el analista elige el método para filtrar las trazas candidatas. La Figura 3.7 ilustra cómo las trazas sugeridas

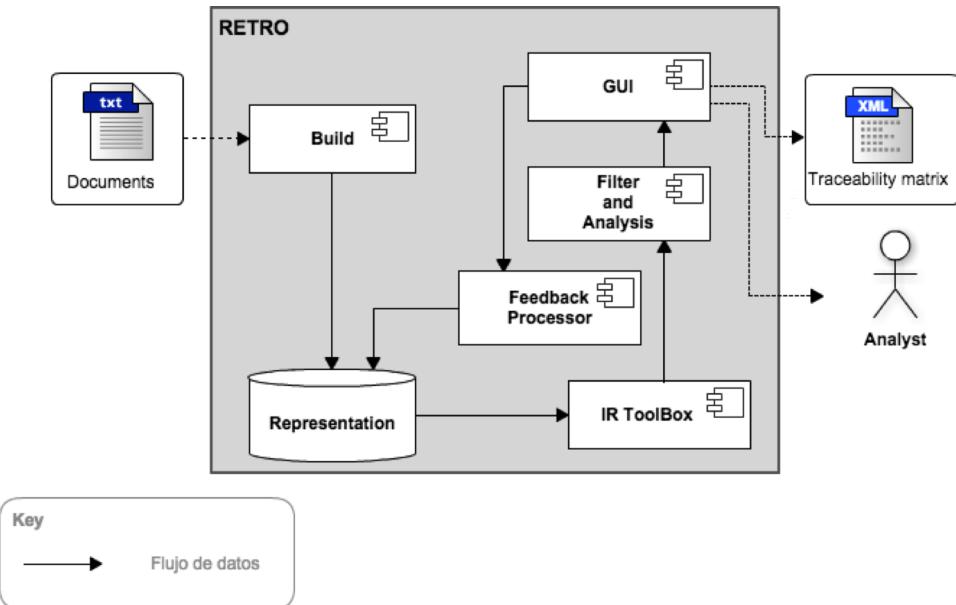


Figura 3.5: Arquitectura de RETRO

son presentadas al analista para que éste las verifique.

La herramienta le permite al analista filtrar las trazas candidatas de dos maneras distintas::

- Threshold: Este método es el estándar usado en la literatura [?, ?], consiste en definir un valor y seleccionar aquellas trazas cuya similitud sea mayor a este valor.
- Threshold variable: Este método consiste en una extensión del Threshold [?], permitiendo especificar un porcentaje predefinido (relativo al número total de trazas), en lugar de un número absoluto fijo. En este caso el número de trazas que superan el filtro depende del número de trazas candidatas. Por ejemplo, el 30 % de 100 trazas serían 30 trazas.

La Figura 3.8 ilustra la arquitectura de ADAMS. Dicho diseño está compuesto por diferentes componentes y fue construido para satisfacer funcionalidades invocadas tanto desde una aplicación web como de un plugin de Eclipse. Los componentes principales de la arquitectura son:

- Web y Eclipse: Componentes encargados de gestionar la interacción del analista con la herramienta, el componente Web define las interfaces para una aplicación web mientras que el componente Eclipse define las interfaces para un plugin de Eclipse.
- Indexer: Este componente es responsable de extraer y almacenar el texto de diferentes formatos de documentos como HTML, TXT o PDF, entre otros. Además, el componente normaliza el texto eliminando los caracteres especiales, símbolos de puntuación y removiendo las stop words.
- LSA: Este componente es responsable de ejecutar el algoritmo de LSA sobre los documentos y almacenar los resultados (parciales) en el repositorio Artifacts. Entre los resultados generados se encuentran: la matriz de SVD, la matriz de términos por documento y los vectores asociados a cada documento.
- Query: Este componente permite filtrar de diferentes formas las trazas candidatas y provee funcionalidad para buscar trazas mediante palabras clave. Las trazas recuperadas por el módulo de LSA son comparadas con las provistas por el analista y, de esta forma, es posible identificar aquellas relaciones de trazabilidad que no fueron detectadas por la herramienta como trazas de advertencia.

Traceability Link Recovery

Artefact Type Filtering

SOURCE	TARGET
Class Sequence Diagram Test Case Use Case	Class Sequence Diagram Test Case Use Case

Artefact Filter

Contains Contains

Traceability Link Recovery

Artefact Filtering

SOURCE	TARGET
20 - (20) Valida paziente 21 - (21) Visualizza cartella clinica 19 - (19) Accesso sezione paziente 6 - (06) Elimina anagrafica medico 23 - (23) Visualizza anagrafica 3 - (03) Elimina anagrafica laboratorio 24 - (24) Accesso Box Tower 25 - (25) Valida paziente - Box Tower 10 - (10) Elimina visita	131 - (132) GUILoginPaziente 118 - (118) GUICartellaClinica 120 - (120) GUILogin - Operatore 121 - (122) GUIAnagrafica 115 - (115) GUIPrincipale 133 - (134) GUIPrenotaVisita 129 - (130) GUITempiMedi 137 - (138) GUIEliminaPrenotazione 117 - (117) GUIPrenotazioni

Select all source Select all target

< Back Next >

(a) (b)

Traceability Link Recovery

Approach

Approach type	Settings
Threshold based <input type="button"/>	<input type="radio"/> Use the last threshold <input checked="" type="radio"/> Use the lowest threshold <input type="radio"/> Use the default threshold (95%)

< Back Next >

(c)

Figura 3.6: Pasos necesarios para recuperar la trazabilidad en ADAMS

Traceability Link Recovery

Proposed Links

(Threshold based approach)

Threshold: %

Start recovery wizard

Previous iteration statistics

Threshold:	90%	Traced links:	2
Suggested links:	2	False positives:	0
Session time:	00:00:08		

Start Time: 2008-06-23 (10:19:36) | End Time: —

15 items found, displaying all items.

ID	Source Artefact	ID	Target Artefact	Similarity	Trace link	False positive	Stereotype	Direction
6	(06) Elimina anagrafica medico	121	(122) GUIAnagrafica	89.68	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
23	(23) Visualizza anagrafica	150	(151) Persona	89.18	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
3	(03) Elimina anagrafica laboratorio	121	(122) GUIAnagrafica	87.3	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
24	(24) Accesso Box Tower	115	(115) GUIPrincipale	86.54	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
25	(25) Valida paziente - Box Tower	131	(132) GUILoginPaziente	85.52	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
10	(10) Elimina visita	133	(134) GUIPrenotaVisita	84.29	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
29	(29) Tempi medi di attesa	129	(130) GUITempiMedi	84.12	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
22	(22) Visualizza prenotazioni	137	(138) GUIEliminaPrenotazione	83.77	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
22	(22) Visualizza prenotazioni	117	(117) GUIPrenotazioni	83.48	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
14	(14) Prenota visita	133	(134) GUIPrenotaVisita	83.35	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
23	(23) Visualizza anagrafica	116	(116) GUIAnagraficaPaziente	83.31	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
5	(05) Modifica anagrafica medico	121	(122) GUIAnagrafica	82.55	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
2	(02) Modifica anagrafica laboratorio	121	(122) GUIAnagrafica	82.47	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
17	(17) Modifica anagrafica paziente	135	(136) GUIAnagraficaPaziente	81.81	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH
3	(03) Elimina anagrafica laboratorio	122	(123) GUIAnagraficalaboratorio	81.72	<input type="checkbox"/>	<input type="checkbox"/>	extends	<input type="checkbox"/> BOTH

Classify all as corrects links Classify all as false positives

Figura 3.7: Análisis de trazas sugeridas en ADAMS

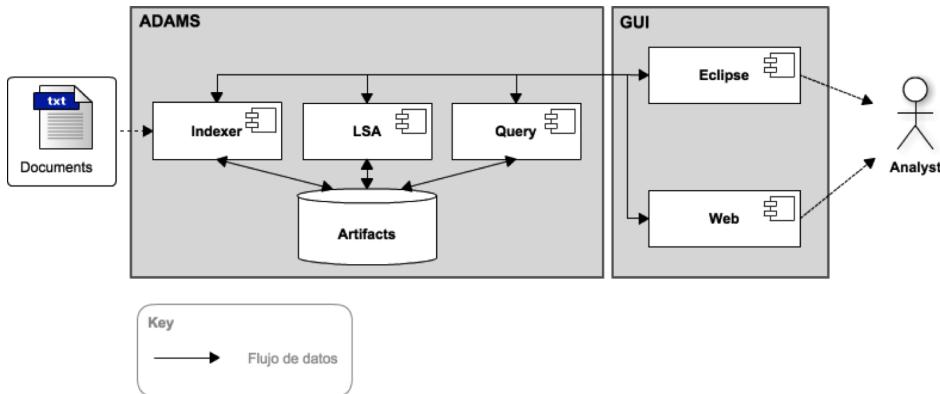


Figura 3.8: Arquitectura de ADAMS

3.2.2. Criterios de comparación para evaluar herramientas de recuperación de trazas

En esta sección se listan los criterios de comparación utilizados para evaluar las características de las herramientas de recuperación de trazabilidad. Los criterios de comparación definidos son:

- Nivel de automatización: Se refiere al grado de interacción que tiene el analista con la herramienta. Este criterio se enfoca principalmente en si es necesario tener un conocimiento previo de los documentos o del sistema en si, si es necesario ingresar información adicional con respecto al sistema, la cantidad de etapas donde el analista debe tomar decisiones, entre otras.
- Tipo de análisis: Describe el método de IR utilizado para recuperar la trazabilidad, y las técnicas de mejora en caso de existir.
- Escalabilidad: Describe las capacidades que proveen las herramientas, para poder aplicar los enfoques a problemas en sistemas de gran tamaño, de qué manera lo realizan. Además, debe tenerse en cuenta si la herramienta es capaz de analizar grandes cantidades de información en un tiempo aceptable, es decir, que no se produzcan demoras considerables al usar las herramientas.
- Efectividad: Mide la capacidad de lograr una buena precisión, para un determinado nivel de *recall*, a la hora de recuperar la trazabilidad de los artefactos de software.
- Velocidad: Se refiere a la medición del tiempo empleado para llevar a cabo la recuperación de trazabilidad con respecto a las otras técnicas.

3.2.3. Comparación de herramientas analizadas

En esta sección se analizan las tres herramientas utilizando los criterios de comparación descritos anteriormente. La Tabla 3.4 presenta un resumen de la comparación. En el caso de RETRO y Poirot, las ventajas residen en la velocidad y el nivel de automatización, mientras que ADAMS tiene una mejor efectividad debido a la técnica de SVD. Sin embargo, la técnica SVD tiene un costo computacional mayor, y por lo tanto, la herramienta ADAMS es más lenta que las otras dos. Tanto ADAMS como Poirot y RETRO reducen significativamente el tiempo necesario para recuperar trazas [?, ?, ?]. Poirot además muestra aquellas trazas que considera true negative las cuales denomina "unlikely link", esto permite mejorar el tiempo necesario para validar las trazas candidatas [?].

RETRO está basado en la técnica de VSM y por lo tanto tiene las ventajas y desventajas de la misma. Si bien esta herramienta es más rápida que ADAMS (la cual está basada en LSA), su mecanismo de generación de relaciones de trazabilidad puede producir un mayor número de falsos positivos. Tanto RETRO como Poirot necesitan que los documentos comparten ciertos

Criterio / Enfoque	Poirot	RETRO	ADAMS
Nivel de automatización	No necesita la entrada de información adicional. No depende del nivel de conocimiento del analista	No necesita la entrada de información adicional. No depende del nivel de conocimiento del analista	El analista debe seleccionar el número de dimensiones para construir la matriz de SVD
Tipo de análisis	Modelos probabilísticos los cuales son mejorados mediante el uso de modelos jerárquicos	VSM junto con palabras claves y el uso de sinónimos para reducir el número de términos (tamaño de los vectores)	LSA, tiene en cuenta los sinónimos y las estructuras latentes en los documentos
Escalabilidad	Puede escalar sin problemas	Tiene dificultad con grandes documentos debido a que el número de términos es mayor	Puede escalar sin problemas
Efectividad	Poco efectivo. Tiene problemas cuando los documentos no presentan una estructura jerárquica que pueda ser fácilmente descubierta por la herramienta	Poco Efectivo. Recomendado para documentos donde el orden de los términos varía. Tiene problemas con documentos que tienen oraciones grandes o muchos sinónimos	Efectivo. No necesita hacer un análisis léxico sobre los documentos. Permite modificar el número de dimensiones de los vectores que representan los documentos. Requiere de una gran cantidad de documentos
Velocidad	Rápida	Rápida	No tan rápida debido a que construir la matriz de SVD es un proceso costoso

Cuadro 3.4: Comparación herramientas de recuperación de trazabilidad

términos iguales para identificar trazas. Es por eso que dependen de técnicas como Stemming la cual busca reducir las palabras a su raíz o lema para obtener buenos resultados. ADAMS no necesita modificar las palabras originales para identificar trazas.

Además, RETRO y Poirot tienen problemas cuando en los documentos se utilizan sinónimos, debido a que si bien significan lo mismo, los términos son distintos. Dado que ADAMS utiliza LSA, si la cantidad de documentos es grande, esta puede manejar sinónimos. Esto es una gran ventaja dado que permite analizar documentos que puedan haber sido escritos por distintas personas, las cuales pueden utilizar un vocabulario distinto para referirse a lo mismo.

El tamaño de los documentos de entrada también afecta la eficacia de las herramientas. ADAMS funciona mejor con documentos de gran tamaño dado que LSA puede recuperar más estructuras semánticas. Por otro lado RETRO pierde eficacia a medida que el tamaño de los documentos crece, ya que el número de palabras aumenta junto con el número de documentos. Por lo tanto, ADAMS es recomendable para documentos de gran tamaño como son los documentos de arquitectura.

Para documentos de gran tamaño, el número de trazas candidatas generadas por las tres herramientas puede ser muy grande. Por lo tanto, más allá de que las herramientas reducen el trabajo necesario para recuperar la trazabilidad, el analista aún tiene que evaluar un gran número de trazas candidatas. Domges et al. observó que un número excesivo de trazas candidatas puede resultar complejo de manejar para el analista [?].

3.3. Resumen

En este capítulo se presentó un estudio de los enfoques más relevantes para el análisis de documentación de requerimientos y arquitecturas de software y la recuperación de trazabilidad entre los mismos. Se describieron los diferentes enfoques y herramientas existentes, destacando sus beneficios y problemática.

En primer lugar se describieron enfoques que buscan recuperar información de artefactos de software. Estos enfoques utilizan en su mayoría el análisis léxico y en algunos casos semánticos para recuperar información de artefactos tales como documentos de requerimientos y de arquitectura de software.

Dentro de los enfoques que trabajan sobre documentos de requerimiento, se destaca REAssistant [?], una herramienta que recupera *crosscutting concerns* a partir de especificaciones de casos de uso. Esta herramienta fue comparada con otras que realizan tareas similares y mostró resultados alentadores [?]. Además, REAssistant utiliza una técnica denominada consultas enriquecidas semánticamente, la cual se basa en reglas que utilizan información semántica y que podrían ser fácilmente aplicadas sobre documentos de arquitectura, con el fin de recuperar decisiones de diseño.

Luego, se analizaron diferentes enfoques para recuperar la trazabilidad entre artefactos de software. Se reconocieron tres tipos de enfoques para llevar a cabo esta tarea: métodos probabilísticos, métodos del espacio vectorial y LSA. Estas técnicas buscan recuperar trazas entre distintos artefactos de software tales como código fuente y manuales de usuario o código fuente y casos de test, entre otros.

Recuperar las trazas entre los requerimientos y los componentes arquitectónicos que los satisfacen, como así también mantener estas asociaciones actualizadas a medida que el sistema evoluciona, es importante dado que: (i) ayuda a los stakeholders a ver el impacto de los requerimientos sobre la arquitectura, (ii) facilita las tareas de análisis ante cambios en los requerimientos, simplificando la incorporación de nueva funcionalidad y el estudio de su impacto en la arquitectura, (iii) asegura que la arquitectura contemple todos los requerimientos del sistema, y (iv) asegura que la documentación de requerimientos esté completa.

Existen un gran número de herramientas y enfoques para encontrar trazas entre diversos artefactos, pero ninguno se enfocó en recuperar trazas entre documentos de requerimientos y arquitectura. Además, los enfoques analizados trabajan sobre todo el contenido de los documentos. Esto implica que el contenido no relevante genera ruido, el cual es tenido en cuenta e

impacta en las técnicas de recuperación de trazabilidad.

Dado que los *crosscutting concerns* presentes en los documentos de requerimientos están relacionados con los atributos de calidad, y las decisiones de diseño están relacionadas con los atributos de calidad. Se cree que la identificación de *crosscutting concerns* y decisiones de diseño junto con el uso de una de las técnicas de recuperación de trazabilidad permitirá recuperar las trazas entre los documentos de requerimientos y arquitectura de manera eficiente.

Capítulo 4

TRAS: Una herramienta para la identificación de trazas entre documentos de requerimientos y de arquitectura

La trazabilidad entre documentos de requerimientos y documentos de arquitectura es de gran importancia para el análisis de un sistema [?, ?, ?]. Documentar dicha trazabilidad permite entre otras cosas:

- Entender cómo los requerimientos dieron origen a la arquitectura, qué decisiones de diseño pueden haber sido afectadas por ciertos requerimientos y cuáles no.
- Descubrir si los requerimientos fueron tenidos en cuenta a la hora de diseñar la arquitectura del sistema.
- Prever el impacto de futuros cambios en los requerimientos sobre la arquitectura diseñada.

En este capítulo se presenta una herramienta que procesa los documentos de requerimientos y de arquitectura con el fin de utilizar solamente la información relevante a la hora de recuperar la trazabilidad entre estos. En el caso de los documentos de requerimientos, se busca recuperar los *crosscutting concerns* de forma tal de trabajar con aquellas porciones de las especificaciones que impactan en diversas partes de la arquitectura del sistema. En el caso de los documentos de arquitectura, se busca recuperar las decisiones de diseño, en particular, aquellas que afectan a uno o más atributos de calidad. Una vez analizados ambos documentos, se utiliza una técnica de recuperación de información para descubrir las trazas ocultas entre los *crosscutting concerns* y las decisiones de diseño. Los aspectos clave de esta propuesta son:

- Utilización de varias técnicas de procesamiento de lenguaje natural (NLP) para el análisis de texto.
- Utilización de un lenguaje basado en reglas para recuperar información de los documentos de arquitectura
- Uso de *Latent Semantic Analysis* (LSA) [?] como técnica de recuperación de información, para determinar la trazabilidad entre los documentos
- Separación del enfoque propuesto en tres partes independientes entre sí y extensibles

4.1. Esquema de la TRAS

El enfoque se materializa mediante una herramienta denominada TRAS (*TRaceability ASSistant*). La herramienta asiste a analistas y arquitectos en la identificación de *crosscutting concerns*

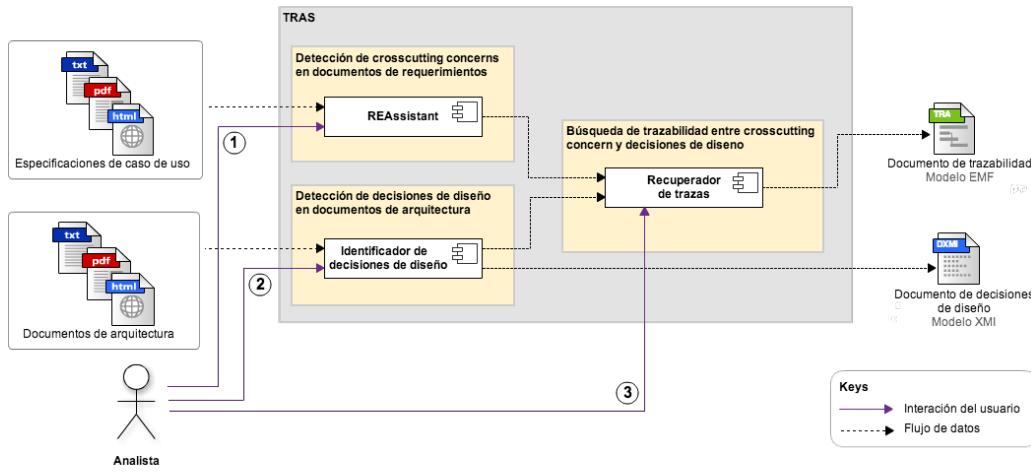


Figura 4.1: Arquitectura de TRAS

en documentos de requerimientos y decisiones de diseño en documentos de arquitectura, tanto así como la recuperación y análisis de la trazabilidad entre documentos de requerimientos y documentos de arquitectura.

TRAS permite, entre otras funcionalidades:

- Recuperar y posteriormente editar a partir de los documentos de arquitectura, las decisiones de diseño utilizando diferentes análisis lingüísticos.
- Recuperar y posteriormente editar trazas entre decisiones de diseño y *crosscutting concerns*.
- Identificar aquellos *crosscutting concerns* que no tienen una decisión de diseño asociada y por lo tanto pueden no haberse tenido en cuenta en la arquitectura.
- Visualizar la distribución de decisiones de diseño de acuerdo al atributo de calidad que afectan.
- Visualizar el impacto a alto nivel de cada grupo de *crosscutting concern* sobre los atributos de calidad.

La Figura 4.1 muestra un diagrama de alto nivel de TRAS, los documentos de requerimientos y arquitectura que reciben como entrada cada componente y el documento de trazabilidad que se genera como resultado. Además, se muestran tres componentes:

- REAssistant: Una herramienta desarrollada en la UNICEN¹ [?, ?] que permite recuperar los *crosscutting concerns* de documentos de requerimientos.
- Buscador de decisiones de diseño: componente encargado de recuperar y clasificar decisiones de diseño a partir de documentos de arquitectura.
- Recuperador de trazas: Componente encargado de recuperar la trazabilidad entre los *crosscutting concerns* y las decisiones de diseño.

Los documentos de arquitectura que recibe la herramienta no necesariamente siguen una estructura definida, pueden ser templates de SAD o documentos con notas tomadas por los arquitectos del sistema. Por otro lado, los documentos de requerimientos de los cuales se extraen los *crosscutting concerns* son especificaciones de casos de uso.

¹Universidad Nacional del Centro. www.unicen.edu.ar

Traceability Editor

Requirements Concerns	
Type	Label
Latency	The contracts must be retrieved from the database in less than 3 seconds.
User Access Rights	A successfully logged in user has a token associated with his session.
Latency	The alarm must be shown in all the user interfaces in less than 5 seconds.
Persistence	All hierarchies are user specific and are persisted in the database.
Error Management	System notifies the user of command failure and reason of failure.

Remove

Architectural design decisions	
Type	Label
Security	Each user has a session associated
Security	The user session will be available for 30 minutes

Links **Graph** **New Link** **Source**

Figura 4.2: Lista de trazas entre *crosscutting concerns* y decisiones de diseño

DXMI Editor

Design decisions	
Type	Label
Performance	It will be used a cache memory for getting the contracts from the database.
Security	Each user has a session associated.
Security	Allow access to Alarm database based on user rights.
Performance	Stores events in a First In First Out (FIFO) queue.
Modifiability	The Event Publisher(s) will notify their subscribers concurrently.

Source **List** **Graph**

Figura 4.3: Lista de decisiones de diseño producida por la herramienta

A la hora de recuperar las trazas, el usuario (analista o arquitecto del sistema) debe realizar 3 pasos, cada uno asociado a cada componente descrito anteriormente. En primer lugar, el analista utiliza la herramienta REAssistant para recuperar los *crosscutting concerns* de documentos de requerimientos. Esta herramienta soporta descripciones de casos de uso como documentos de entrada. Los *crosscutting concerns* son almacenados para luego ser utilizados en una etapa posterior.

En segundo lugar, el analista provee a TRAS con documentos de arquitectura que contienen decisiones de diseño. TRAS ejecuta una serie de análisis de NLP sobre estos documentos creando anotaciones que representan información sintáctica y semántica del texto. Estas anotaciones son utilizadas en conjunto con reglas semánticas para recuperar las decisiones de diseño. Una vez recuperadas, las decisiones son presentadas al analista para que éste pueda eliminar las incorrectas o agregar aquellas faltantes. Las oraciones que contienen decisiones son almacenadas para ser utilizadas luego en la búsqueda de trazabilidad. La Figura 4.3 muestra cómo TRAS presenta las decisiones de diseño al analista.

Por último, el analista ingresa los *crosscutting concerns* recuperados previamente por la herramienta REAssistant y el resultado de recuperar las decisiones de diseño y utiliza TRAS para recuperar la trazabilidad entre estos. TRAS utiliza una técnica de recuperación de información denominada LSA [?]. Esta técnica permite analizar las relaciones entre los términos utilizados por los *crosscutting concerns* y las decisiones de diseño. Una vez ejecutada la técnica, el analista visualiza las trazas candidatas, las cuales puede eliminar si no son relevantes o son incorrectas, como así también puede agregar aquellas que no hayan sido recuperadas.

TRAS presenta dos vistas que pueden ser utilizadas para analizar la trazabilidad. Por un lado, se puede utilizar una vista de alto nivel, como se muestra en la Figura 4.6, donde se puede

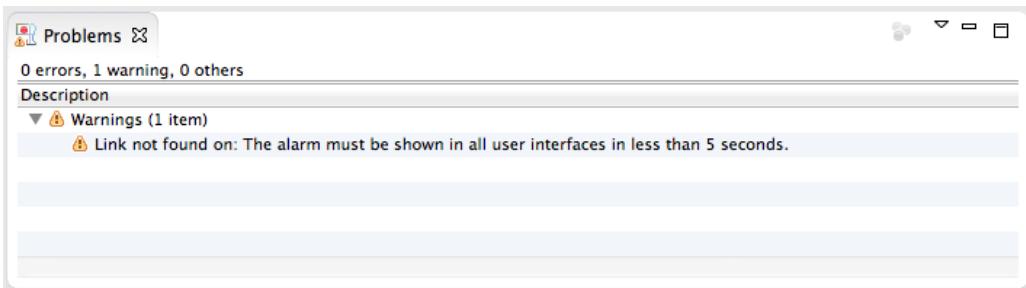


Figura 4.4: Lista de *crosscutting concerns* sin decisiones de diseño asociadas

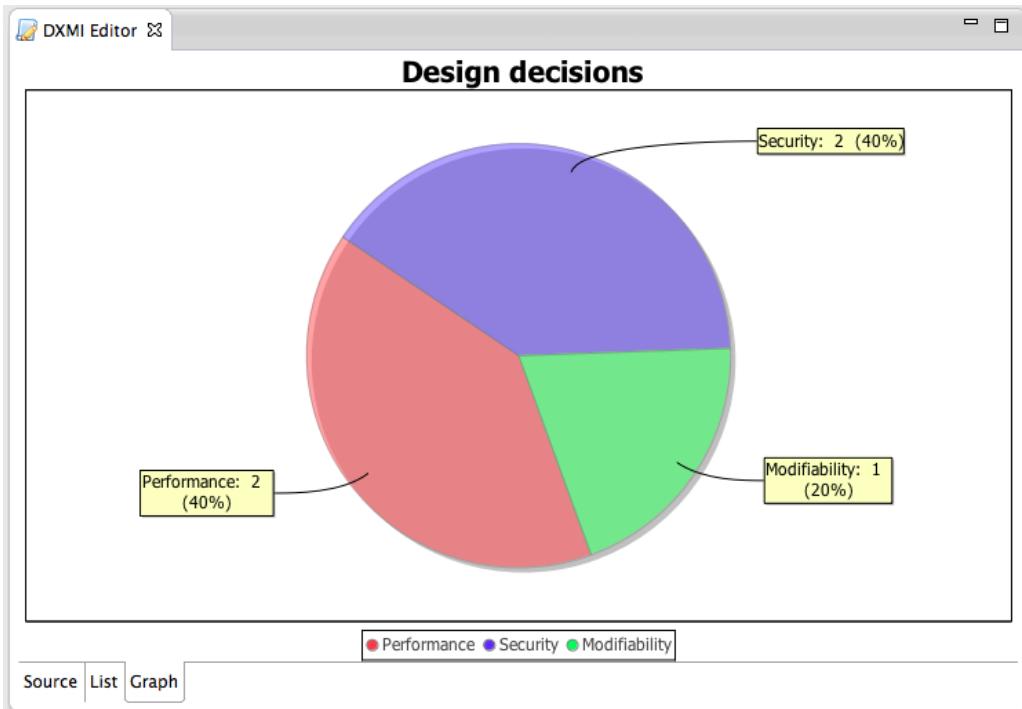


Figura 4.5: Gráfico de distribución de decisiones de diseño

observar la distribución de decisiones de diseño asociadas a un grupo de determinado de *crosscutting concerns* denominado *persistence*. En esta vista se agrupan los *crosscutting concerns* en base al grupo definido por REAssistant y las decisiones de diseño de acuerdo al atributo de calidad que afectan. El analista puede seleccionar un *crosscutting concern* como *persistence* y ver los atributos de calidad que fueron afectados. En la Figura 4.2 se observa que los *crosscutting concerns* de *user access rights* afectaron mayormente al atributo de calidad de *security*.

El analista puede tener un mayor detalle de las trazas utilizando la vista detallada de trazabilidad, como se puede ver en la Figura 4.2, donde se observan las trazas entre un *crosscutting concern* y dos decisiones de diseño asociadas a la seguridad del sistema. Esta vista muestra la lista de *crosscutting concerns* y la de las decisiones de diseño al mismo tiempo, permitiendo al analista seleccionar un *crosscutting concern* y listar las decisiones de diseño asociadas al mismo. En la Figura 4.3 se muestra en detalle las decisiones de diseño asociadas al *crosscutting concern*: "A successfully logged in user has a token associated with his session". Por último, la herramienta también identifica aquellos *crosscutting concerns* para los cuales no fue posible detectar decisiones de diseño asociadas ver Figura 4.4.

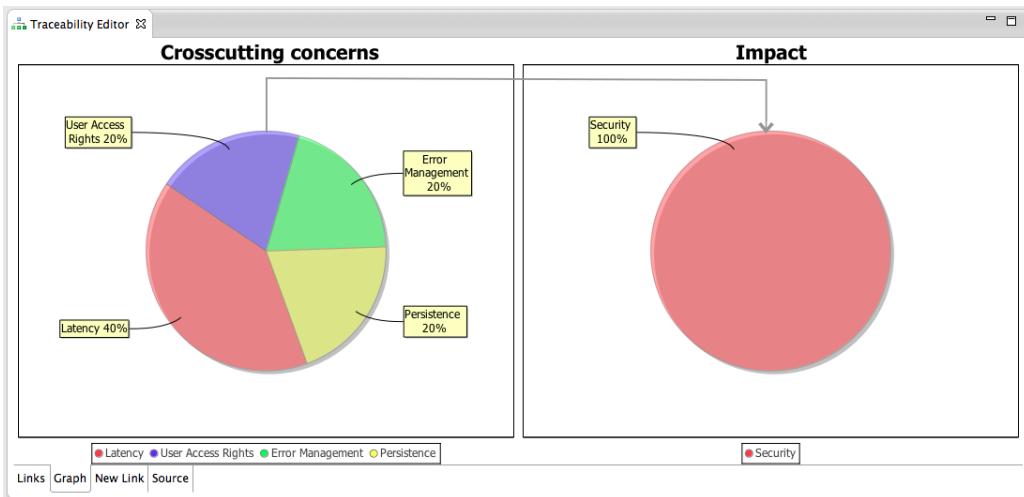


Figura 4.6: Distribución de decisiones de diseño dado un grupo de *crosscutting concerns*

4.2. Arquitectura de TRAS

La arquitectura de TRAS comprende diferentes componentes, y su diseño involucra la integración de dos tecnologías, principalmente: UIMA² (*Unstructured Information Management*) y Eclipse RCP³ (*Rich Client Plugin*). El framework UIMA permite el procesamiento de información no estructurada y proporciona interfaces bien definidas, soporte para la importación y exportación de documentos en diferentes formatos, y una arquitectura que permite encapsular fácilmente algoritmos existentes. Eclipse RCP es un framework de desarrollo que sirve para extender el entorno de desarrollo de Eclipse con nuevas aplicaciones o funcionalidad.

La herramienta consta principalmente de tres partes diferentes:

- Detección de *crosscutting concerns* en documentos de requerimientos.
- Detección de decisiones de diseño en documentos de arquitectura.
- Búsqueda de trazabilidad entre *crosscutting concerns* y decisiones de diseño.

Esta separación permite que cada parte pueda ser fácilmente extensible. Cada una de estas partes define una entrada, un objetivo y una salida, las cuales son descritas a continuación.

4.2.1. Detección de *crosscutting concerns*

Esta sección describe el proceso de recuperar los *crosscutting concerns* de los documentos de requerimientos. Para ello, se decidió utilizar la herramienta existente llamada REAssistant [?]. Esta herramienta detecta de manera automática *crosscutting concerns* a partir de descripciones de casos de uso.

Para llevar a cabo la detección de *crosscutting concerns*, REAssistant recibe como entrada las especificaciones de casos de uso y ofrece dos técnicas distintas para descubrir *crosscutting concerns* de una manera semi-automática. La primera técnica, *Clasificación Semántica*, aplica algoritmos de clasificación para agrupar comportamientos similares que están esparcidos en múltiples documentos. La segunda técnica, *Reglas Semánticamente Enriquecidas*, provee un motor de reglas para detectar *crosscutting concerns* usando conocimiento semántico.

El resultado obtenido luego de ejecutar las técnicas de descubrimiento de *crosscutting concerns* mencionadas anteriormente, le permite a TRAS filtrar aquella información que no es relevante a la hora de buscar la trazabilidad del sistema en los documentos de requerimientos.

²UIMA: <http://uima.apache.org>

³Eclipse RCP: <http://www.eclipse.org/home/categories/rcp.php>

TRAS define como información relevante a aquellos *concerns* que tienen efectos de amplio alcance sobre otros componentes como son los *crosscutting concerns*. Estos *concerns* tienen, por lo general, una o más decisiones de diseño asociada en los documentos de arquitectura.

4.2.2. Detección de decisiones de diseño

Esta sección describe el proceso que se lleva a cabo para detectar decisiones de diseño en documentos de arquitectura. Dicho proceso fue dividido en dos etapas (ver Figura 4.7):

- Analizador de documentos de arquitectura: recupera información léxica y semántica de los documentos de arquitectura.
- Identificador de decisiones de diseño: utiliza la información de la etapa anterior y un sistema de reglas para recuperar y clasificar las decisiones de diseño.

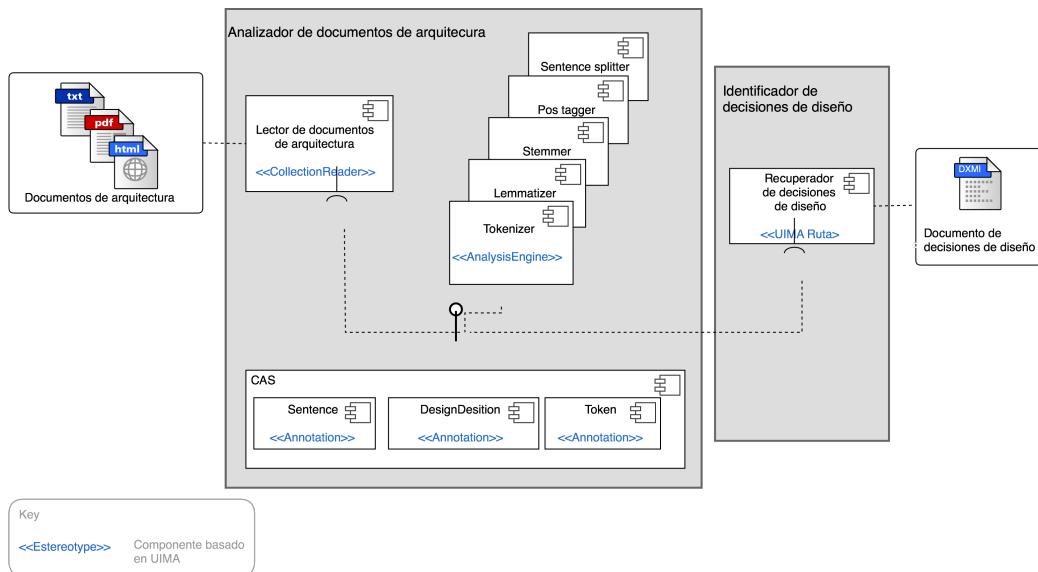


Figura 4.7: Componentes encargados de la detección de decisiones de diseño

Analizador de documentos de arquitectura

Esta etapa contiene los componentes que se encargan de llevar a cabo el análisis léxico y semántico de los documentos de arquitectura. Este análisis se realiza aplicando técnicas de NLP y busca recuperar la información que se encuentra en los documentos de arquitectura.

Los documentos de arquitectura son procesados mediante una implementación de la interfaz de *CollectionReader* de UIMA. Este componente tiene la finalidad de extraer el texto del cual se van a detectar las decisiones de diseño. En la implementación actual, los documentos procesados comprenden texto sin formato (por ejemplo, archivos TXT) pero es posible implementar otros *CollectionReader* y así extraer texto de otros tipos de documentos como PDF, DOC, Wikis, etc. La salida de este componente es almacenada en una estructura de datos provista por UIMA denominada *Common-Analysis-Structure* (CAS).

La Figura 4.7 muestra los componentes involucrados en el *Analizador de documentos de arquitectura* y cómo éstos utilizan el CAS como interfaz común para recuperar y almacenar información. Esta información tiene una estructura definida que se denomina "anotación" e identifica una región específica del texto y permite asociarle propiedades.

A continuación se describen los módulos de NLP utilizados en la herramienta. Estos módulos tienen como objetivo generar las anotaciones correspondientes para identificar palabras, oraciones y almacenar resultados de análisis más complejos como la etiqueta gramatical o el lema de cada palabra:

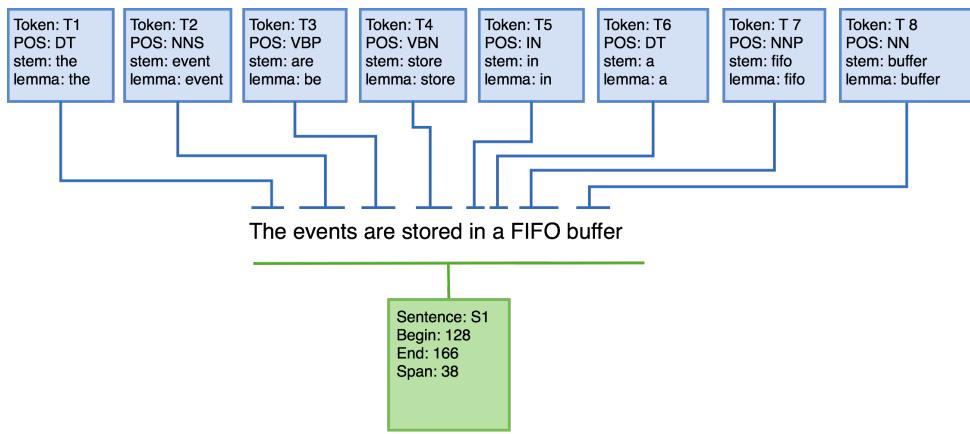


Figura 4.8: Ejemplo de las anotaciones producidas por los módulos de NLP

- *Tokenizer*: a partir de una cadena de texto separa palabras y crea anotaciones denominadas *Token* (ver Figura 4.8). Esta actividad es usualmente difícil debido a los apóstrofes, signos de puntuación y contracciones del lenguaje inglés.
- *Sentence Splitter*: dado un conjunto de *Tokens*, se encarga de separarlos en oraciones. Para hacer esto, los signos de puntuación son considerados potenciales marcadores de fin de oración y cada vez que se encuentra uno de estos caracteres, se evalúan una serie de condiciones para determinar si es un marcador de fin de oración o no. Esto es necesario dado que las oraciones son utilizadas como la mínima unidad de información a la hora de buscar decisiones de diseño.
- *POS Tagger*: Esta tarea tiene como finalidad realizar la asignación de etiquetas de parte del discurso a cada palabra recibida de la tarea anterior. Esto es necesario para determinar si una palabra es un verbo, sustantivo, artículo, etc. En TRAS, las etiquetas se almacenan como propiedades *POS* de las anotaciones *Token* y se utilizan como información para saber el uso de la palabra en la oración. La Tabla 4.1 muestra todas las etiquetas que una palabra puede tener y que son detectadas por el etiquetador gramatical de TRAS. Cada palabra puede tener una sola etiqueta asociada. La Figura 4.8 muestra que la palabra *stored* con la anotación *Token:T4* tiene la etiqueta "VBN" es decir, que esta siendo utilizada como verbo en pasado participio en la oración "**The events are stored** in a FIFO buffer".
- *Stemmer* y *Lemmatizer*: por razones gramaticales, los documentos de texto pueden tener una palabra en diferentes formas, como *organize*, *organizes*, *organizing*. El *Stemmer* busca identificar la raíz de las palabras mientras que el *Lemmatizer* busca eliminar las formas de inflexión y las formas de derivación de las palabras. El objetivo de estas tareas es asignarle una única representación a las palabras que escritas diferentes tengan un significado similar. Para ejemplificar esta diferencia entre el *lemma* y el *stem*, se puede observar en la Figura 4.8 que el *Token:T3* contiene la palabra *are*, cuyo lemma es *be* mientras que su stem es *are*.

La Figura 4.8 muestra un ejemplo de anotaciones como *Token* y *Sentence* asociadas a una palabra y una oración respectivamente, además se muestran las propiedades de cada anotación. Muchas de las implementaciones de las tareas de NLP descritas anteriormente fueron adaptadas a partir de componentes ya disponibles, entre ellas, se destacan las provistas por el grupo de Stanford NLP⁴: *Tokenizer*, *Sentence Splitter*, *POS Tagger*, *Lemmatizer*. Adicionalmente la implementación del *Stemmer* provista por SnowBall⁵.

⁴<http://nlp.stanford.edu/software/corenlp.shtml>

⁵<http://snowball.tartarus.org/>

Tag	Descripción	Tag	Descripción
CC	Coordinating conjunction	RP	Particle
CD	Cardinal number	SYM	Symbol
DT	Determiner	TO	<i>to</i>
EX	Existential <i>there</i>	UH	Interjection
FW	Foreign word	VB	Verb, base form
IN	Preposition	VBD	Verb, past tense
JJ	Adjective	VBG	Verb, gerund or present participle
JJR	Adjective, comparative	VBN	Verb, past participle
JJS	Adjective, superlative	VBP	Verb, non-3rd person singular present
LS	List item marker	VBZ	Verb, 3rd person singular present
MD	Modal	WDT	Wh-determiner
NN	Noun, singular or mass	WP	Wh-pronoun
NNS	Noun, plural	WP\$	Possessive wh-pronoun
NNP	Proper noun, singular	WRB	Wh-adverb
NNPS	Proper noun, plural	-LRB-	Left parenthesis
PDT	Predeterminer	-RRB-	Right parenthesis
POS	Possessive ending	#	Pound sign
PRP	Personal pronoun	\$	dollar sign
PRP\$	Possessive pronoun	;	Colon, Semi-colon
RB	Adverb	,	Comma
RBR	Adverb, comparative		
RBS	Adverb, superlative		

Cuadro 4.1: Lista de etiquetas utilizadas por el anotador POS Tagger

Identificador de decisiones de diseño

Esta sección describe el componente encargado de recuperar decisiones de diseño a partir de documentos de arquitectura. Este módulo emplea un conjunto de reglas y las anotaciones generadas por el *Analizador de documentos de arquitectura*.

Una decisión de diseño es el resultado de un proceso de decisión que toma lugar cuando se está desarrollando la arquitectura de un sistema [?]. Dentro del conjunto de decisiones de diseño presentes en los documentos de arquitectura. Este enfoque se especializa en aquellas decisiones que tienen impacto sobre uno o más atributos de calidad. A este tipo de decisiones se las denominan tácticas arquitecturales [?].

Dado que las tácticas/decisiones de diseño están asociadas a los atributos de calidad que afectan, TRAS utiliza una taxonomía que busca agrupar las decisiones de diseño que detecta teniendo en cuenta el atributo de calidad asociado. La Figura 4.9 muestra la taxonomía (adaptación de Bass et al.[?]) utilizada por TRAS para clasificar las decisiones de diseño. Por ejemplo, para el atributo de calidad de *Availability*, se buscan identificar las tácticas asociadas de *detect faults*, *recover from faults* y *prevent faults*. La taxonomía completa junto con una descripción de la misma está descrita en el Capítulo 2.

La búsqueda de decisiones de diseño se realiza a nivel de oración. Se considera que un párrafo es demasiado abarcativo, en especial en documentos no estructurados donde hay una gran presencia de contenido que es irrelevante. Utilizar oraciones permite tener una mayor granularidad a la hora de detectar decisiones de diseño. Por otro lado, utilizar oraciones introduce problemas cuando las decisiones están descritas en mas de una oración. En este caso pueden no ser detectadas o puede que solo se detecte una parte de las mismas.

Este componente se implementó mediante la utilización de UIMA RUTA⁶. Esta tecnología utiliza un conjunto de reglas predefinidas que mediante anotaciones presentes en los documentos,

⁶<https://uima.apache.org/ruta.html>

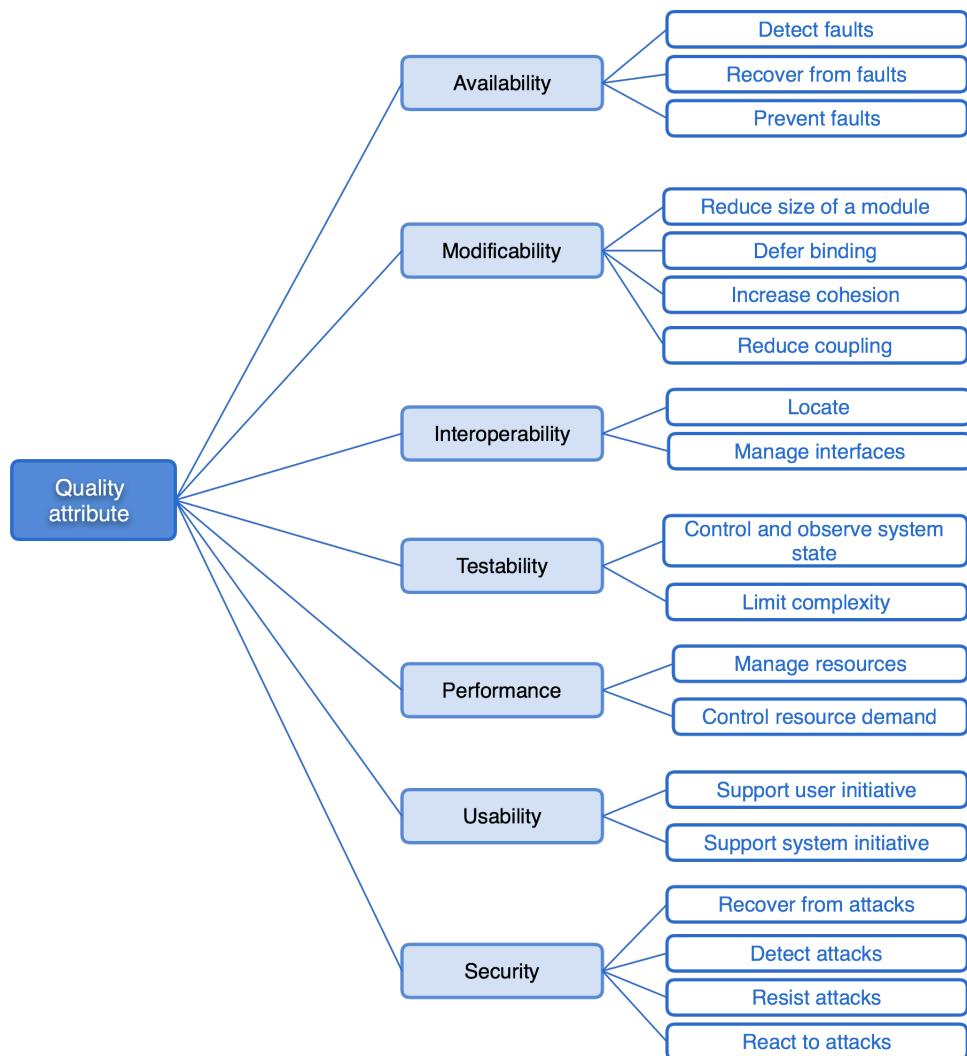


Figura 4.9: Taxonomía utilizada para la clasificación de decisiones de diseño

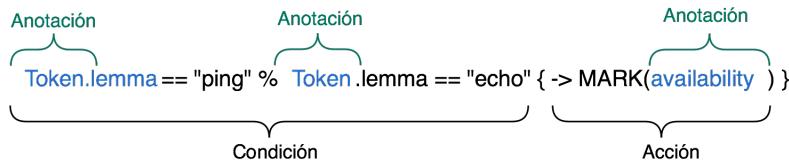


Figura 4.10: Estructura de una regla de Ruta

```

Token.lemma == "ping" % Token.lemma == "echo" {->MARK(availability)};
Sentence[CONTAINS(availability) ->
        CREATE(DesignDecision, "kind" = "Availability", "type" = "tactic")];
  
```

Figura 4.11: Regla encargada de detectar la táctica de disponibilidad *ping/echo*

permiten identificar las decisiones de diseño siguiendo la taxonomía antes descrita. La herramienta define al menos una regla por cada táctica presente en la taxonomía. La lista completa de reglas y sus definiciones se encuentran listadas en el Apéndice A.

Se eligió UIMA Ruta debido a que provee un lenguaje de reglas sobre las anotaciones potente y flexible. La principal ventaja es ser capaz de extraer información a partir de:

- Expresiones y variables
- Importación y ejecución de los componentes externos
- Diccionarios

UIMA Ruta es un lenguaje de reglas imperativas con elementos de scripting. Una regla (ver Figura 4.10) define un patrón. Si el patrón es coincidente con una porción del texto (es decir, sus anotaciones), las acciones asociadas a la condición son ejecutadas (por ejemplo, crear una nueva anotación).

TRAS utiliza diferentes tipos de reglas para recuperar decisiones de diseño. Algunas de estas reglas son sencillas, buscando identificar una palabra en particular mediante anotaciones como *Token*. Otras reglas son más complejas y tienen en cuenta además de la palabra, su función dentro de la oración, si es un verbo o si es un sustantivo. Esta es una ventaja respecto a otras técnicas de recuperación de información (*keywords*) dado que las reglas utilizan la información producida por los anotadores de NLP como el *Stemmer* o el *POS Tagger*.

En la Figura 4.11 se muestra una regla encargada de detectar la táctica de detección de fallas mediante el uso de la estrategia *ping/echo* y una segunda regla encargada de crear una anotación *DesignDecision*. Oraciones que tengan las palabras con lema "ping" y "echo" serán anotadas como tácticas de disponibilidad.

La Figura 4.12 muestra un ejemplo de cómo las reglas detectan decisiones de diseño. En este caso dos de esas reglas son tácticas de seguridad, mientras que la restante es una táctica de *performance*. Esta última detecta oraciones que contengan la palabra "cache", que es una técnica que se utiliza en la administración de recursos para mejorar la *performance*. El conjunto completo de reglas utilizado por la herramienta es listado en el Apéndice A.

Las reglas fueron inicialmente desarrolladas por analistas externos quienes expresaron su experiencia y conocimiento de arquitecturas de software. Cada una de estas reglas busca detectar una táctica asociada a un atributo de calidad. Algunas tácticas pueden tener más de una regla asociada, debido a que existen más de una alternativa para implementar dicha táctica. Por ejemplo, para el atributo de calidad de disponibilidad, existen varias estrategias para detectar fallas tales como *ping/echo*, *heartbeat* o *voting*. Dichas reglas son extensibles y se pueden agregar nuevas reglas y editar las existentes para soportar la identificación de decisiones de diseño específicas de un dominio en particular.

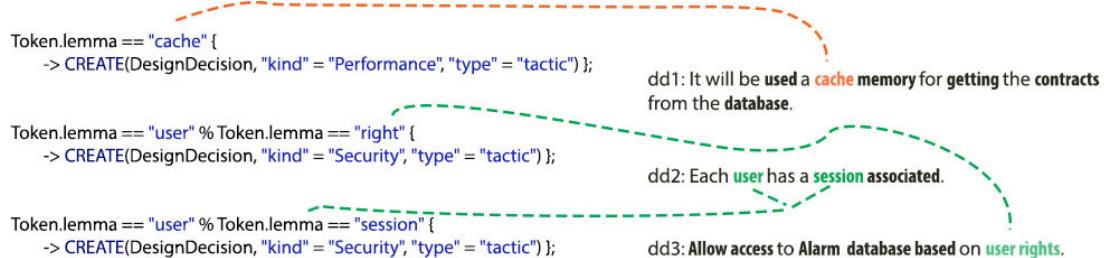


Figura 4.12: Ejemplo de cómo las reglas recuperan decisiones de diseño

4.2.3. Búsqueda de trazabilidad

Esta sección describe el proceso de recuperar trazas entre la lista de *crosscutting concerns* y la lista de decisiones de diseño (conjuntos de oraciones que pueden pertenecer a *crosscutting concerns* o decisiones de diseño en los documentos de requerimientos o arquitectura, respectivamente) mediante una técnica de recuperación de información. La primera lista es provista por la herramienta REAssistant en la etapa de *Detección de crosscutting concerns*. Mientras que la segunda lista es el resultado de la etapa de *Detección de decisiones de diseño*.

A la hora de realizar la recuperación de trazas, este módulo utiliza una técnica de recuperación de información llamada *LSA (Latent Semantic Analysis)* [?, ?]. Esta técnica es una aproximación matemática para el descubrimiento de similitud entre documentos, fragmentos de documentos y las palabras dentro de los documentos. LSA asume que las palabras que están semánticamente relacionadas aparecerán en piezas similares de texto [?].

Dentro de las técnicas evaluadas y teniendo en cuenta este problema en particular, se decidió utilizar LSA debido a que:

- Provee una mejora sustancial con respecto al modelo de espacio vectorial estándar [?, ?, ?].
- Existen estudios como [?, ?] que utilizan LSA para recuperar la trazabilidad entre el código fuente y la documentación donde se obtuvieron resultados alentadores.
- Los documentos de arquitectura y de requerimientos tienen un vocabulario similar debido a que ”hablan” sobre el mismo dominio, de forma tal que hay palabras repetidas en ambos documentos.
- Debido a que los términos en los documentos no son independientes, LSA mejora el manejo de palabras polisémicas, ya que tiene en cuenta que la mayoría de los términos en los documentos no son independientes entre si. De esta manera, la técnica soporta el análisis de palabras que tienen más de un significado y su entendimiento depende fuertemente del contexto en la oración [?].

Para demostrar cómo esta técnica se adapta al problema a resolver, se explicará su funcionamiento mediante un ejemplo. La Figura 4.13 muestra un conjunto de *crosscutting concerns* cc_1, cc_2, cc_3 y un conjunto de decisiones de diseño dd_1, dd_2, dd_3 que van a ser analizados mediante la técnica de LSA. La idea es recuperar las trazas entre ambos conjuntos. A lo largo del ejemplo cada cc_i o dd_j va a ser denominado documento con el fin de mantener una uniformidad dado que ambos conjuntos son tratados de igual manera.

Antes de utilizar la técnica, los documentos son pre-procesados con el fin de reducir el ruido producido por símbolos de puntuación, palabras mal escritas, palabras escritas en mayúscula o minúscula, etc. En este caso las oraciones se convierten a minúscula, se eliminan los signos de puntuación, los símbolos y las *stopwords*. Estas últimas son palabras que tienen frecuencias de aparición muy altas como los artículos o preposiciones y que en este caso no aportan información relevante [?]. Con las palabras resultantes de ambos conjuntos, se construye una matriz de palabras por documento, las filas representan palabras únicas y las columnas representan los documentos (ver Tabla 4.2).

cc1: The **contracts** must be **retrieved** from the **database** in less than 3 **seconds**.
cc2: A **successfully logged in user** has a **token** associated with his **session**.
cc3: The **alarm** must be **shown** in all the **user interfaces** in less than 5 **seconds**.

dd1: It will be **used** a **CACHE** **memory** for **getting** the **contracts** from the **database**.
dd2: Each **user** has a **session** **associated**.
dd3: **Allow access to Alarm database based on user rights.**

Figura 4.13: Conjuntos de oraciones extraídas de dos documentos

$$A = USV^t$$

Figura 4.14: Descomposición en valores singulares

	cc1	cc2	cc3	dd1	dd2	dd3
access						1
alarm			1			1
allow						1
associated		1			1	
based						1
cache				1		
contracts	1			1		
database	1			1		1
getting				1		
interfaces			1			
logged		1				
memory				1		
retrieved	1					
rights						1
seconds	1		1			
session		1			1	
shown			1			
successfully		1				
token		1				
user		1	1		1	1

Cuadro 4.2: Distribución de palabras en los documentos

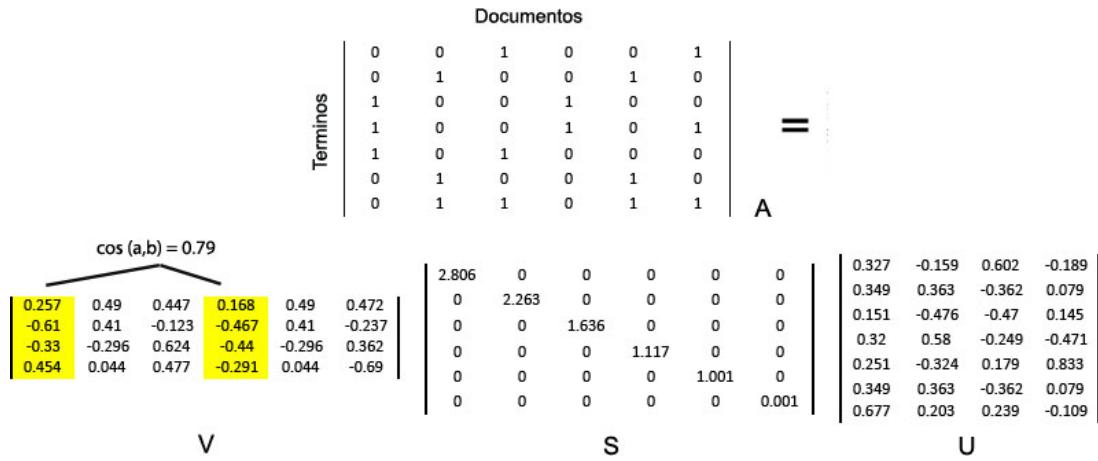


Figura 4.15: Representación gráfica de las matrices de SVD

Una vez obtenida la matriz, se dejan afuera las palabras que aparecen solamente en un documento dado que éstas no son tenidas en cuenta por LSA, en la Tabla 4.2 se muestran resaltadas las palabras que aparecen en más de un documento. A la matriz resultante, denominada matriz A , se le aplica una técnica matemática (ver Figura 4.14) denominada descomposición en valores singulares (*Singular Value Decomposition*, SVD). Esta técnica se utiliza para encontrar relaciones entre las palabras que aparecen en un documento, e intenta reconstruir la matriz A con la mínima cantidad de información, tratando de evitar el ruido producido por términos que no tienen relación entre sí. En el ejemplo las palabras *contract-database* ó *user-associated* tienen una estructura definida que se repite en dos documentos respectivamente.

Uno de los principales elementos a tener en cuenta a la hora de aplicar SVD es el número de dimensiones o "conceptos" que se desean utilizar a la hora de descomponer la matriz A . El número de dimensiones depende del número de documentos a analizar y el número de palabras únicas que estos utilizan [?, ?]. Un número pequeño de dimensiones puede dejar patrones importantes afuera de la comparación, mientras que un número grande puede introducir ruido causado por palabras aleatorias a la comparación entre documentos. Siguiendo con el ejemplo y dado 4 como número de dimensiones, se aplica la fórmula de la Figura 4.14 sobre la matriz A . Las diferentes matrices resultantes de esta operación se muestran en la Figura 4.15.

La matriz U , es una representación uniforme de los documentos [?] y puede usarse para compararlos. Para ello se toman dos columnas, se arman dos vectores $v_1 = [0,257, -0,61, -0,33, 0,454]$ y $v_2 = [0,168, -0,467, -0,44, -0,291]$ y se calcula el coseno entre estos. Los valores cercanos a 1 representan documentos muy similares, mientras que valores cercanos a 0 representan documentos muy disímiles.

En este caso, cada *crosscutting concern* cc_i (cc_1, cc_2, cc_3) es comparado con cada una de las decisiones de diseño dd_j (dd_1, dd_2, dd_3). Para decidir si existe una traza, se calcula el coseno entre los vectores de cada uno de los documentos (oraciones de las listas de *crosscutting concerns* y decisiones de diseño), y si este valor supera una cota definida (0,75 para el ejemplo), los documentos están lo suficientemente relacionados como para establecer una traza entre ambos.

La Tabla 4.3 muestra la tabla con los valores de coseno entre todos los pares cc_i y dd_j luego de aplicar LSA. Las únicas trazas que superan la cota definida (fijada en 0.75) son:

- cc_1 : "The **contracts** must be retrieved from the **database** in less than 3 seconds"
- dd_1 : "It will be used a cache memory for getting the **contracts** from the **database**"
- cc_2 : "A successfully logged in user has a token **associated** with his **session**"
- dd_2 : "Each user has a **session associated**"

Las palabras resaltadas son las que aparecen en más de un documento, y que LSA utilizó para detectar la trazabilidad entre los mismos. Para la primer traza $cc_1 - dd_1$, *cache* fue la palabra utilizada para marcar la oración como una decisión de diseño, sin embargo *contracts* y

Crosscutting concern (cc)	Decisión de diseño (dd)	Peso
cc1	dd1	0.794
cc1	dd2	0.092
cc1	dd3	0.339
cc2	dd1	0.004
cc2	dd2	0.993
cc2	dd3	0.107
cc3	dd1	0.06
cc3	dd2	0.231
cc3	dd3	0.339

Cuadro 4.3: Trazas recuperadas luego de aplicar LSA

database fueron las palabras que dieron lugar a la identificación de la traza. Esas palabras están relacionadas a partir de la funcionalidad del sistema. En el caso la traza $cc_2 - dd_2$, las palabras *user* y *session* fueron las que se utilizaron para marcar la oración como una decisión de diseño, y las mismas fueron también utilizadas en la comparación por LSA. Un análisis especial merece la traza $cc_3 - dd_3$ la cual recibió un valor de 0,399. En este caso las palabras *user* y *alarm* están presentes en ambas oraciones, pero el algoritmo no encontró similitud dado que palabras como *second* y *database* también estaban presentes. Para un ejemplo pequeño como el planteado, esas palabras pueden matemáticamente, cambiar la dirección de los vectores asociados a las oraciones, resultando en un valor de coseno menor a la cota definida.

Del mismo modo que con el ejemplo propuesto TRAS utiliza LSA para, a partir de un número de dimensiones y una cota definida por el analista, generar la matriz U y realizar la comparación entre la lista de *crosscutting concerns* y la lista de decisiones de diseño. El resultado de esta operación es presentado al analista el cual puede validar las trazas recuperadas.

4.3. Resumen

En este capítulo, se presentó una herramienta que asiste al analista en el análisis y recuperación de trazabilidad entre documentos de requerimientos (escritos en forma de casos de uso) y documentos de arquitectura. La herramienta busca trazas solamente en lugares relevantes, como lo son los *crosscutting concerns* (en los documentos de requerimientos) y las decisiones de diseño (en los documentos de arquitectura), para así acotar el espacio de búsqueda. La herramienta divide el problema de recuperar la trazabilidad en tres etapas, cada una de ellas abocada a un problema en particular e independientes entre sí. Esta división permite que los componentes puedan mejorarse o cambiarse fácilmente.

La primera etapa tiene como objetivo recuperar *crosscutting concerns* a partir de documentos de requerimientos, identificando sólo la información relevante en la búsqueda de trazabilidad. Esta etapa es provista por una herramienta desarrollada en la UNICEN denominada REAssistant.

La segunda etapa está dividida en dos partes. La primer parte se encarga de procesar documentos de arquitectura mediante técnicas de NLP con el fin de recuperar información léxica y semántica. La segunda parte utiliza esta información junto con un conjunto de reglas para detectar decisiones de diseño. Estas decisiones son clasificadas de acuerdo al atributo de calidad que afectan siguiendo la taxonomía descrita en el Capítulo 2.

El sistema de reglas utilizado se basa en UIMA Ruta, un lenguaje de reglas imperativas con elementos de scripting. Cada regla define un patrón, y si el patrón coincide con una porción de texto dentro de una oración, ésta es anotada como una decisión de diseño. Además, la decisión de diseño es clasificada de acuerdo al tipo de regla que la descubrió.

La búsqueda de decisiones de diseño se realiza a nivel de oración. Se considera que un párrafo es demasiado abarcativo, en especial en documentos no estructurados donde hay una gran presencia de contenido que es irrelevante. Por otro lado, utilizar oraciones introduce proble-

mas cuando las decisiones están descritas en mas de una oración. En este caso pueden no ser detectadas o puede solo se detecte una parte de las mismas.

Una vez recuperadas, las decisiones son presentadas al usuario mediante diferentes vistas. La primer vista muestra la distribución de decisiones de diseño y los atributos de calidad en un gráfico de torta. La segunda vista lista las decisiones recuperadas. En este caso, el analista puede editar los resultados obtenidos ya sea eliminando o añadiendo nuevas oraciones que contengan decisiones de diseño.

La tercer etapa recibe como entrada los *crosscutting concerns* y las decisiones de diseño detectadas previamente y aplica una técnica de recuperación de información denominada LSA para recuperar relaciones de trazabilidad. Este algoritmo busca extraer el contenido conceptual de un documento, estableciendo asociaciones entre aquellos términos que ocurren en contextos similares. Ésto permite representar *crosscutting concerns* o decisiones de diseño como vectores, los cuales luego son comparados midiendo el coseno del ángulo que forman ambos vectores.

Dado que la herramienta busca recuperar la trazabilidad a partir de *crosscutting concerns* y decisiones de diseño, información presente en los documentos que no es relevante durante la recuperación de estas no es tenida en cuenta. Esto mejora la precisión permitiéndole al algoritmo de LSA enfocarse en aquellas oraciones que tienen relevancia a la hora de recuperar la trazabilidad.

Las trazas recuperadas son presentadas al analista mediante dos vistas. Una de las vistas presenta las trazas a alto nivel, enfocándose en la distribución de *crosscutting concerns* y decisiones de diseño. Del mismo modo, el analista puede acceder a una segunda vista más detallada donde se listan todos los *crosscutting concerns* y decisiones de diseño encontradas. Al final, el analista tiene la posibilidad de validarlas, agregando o eliminando aquellas trazas que considere necesario.

Capítulo 5

Evaluación

En este capítulo se reportan los resultados obtenidos luego de evaluar la herramienta propuesta con 3 casos de estudio. El objetivo de la evaluación es comparar los resultados obtenidos mediante el uso de la herramienta con las soluciones ideales provistas por analistas humanos. Los resultados de la evaluación serán analizados y se describirán las observaciones encontradas de la herramienta propuesta.

La hipótesis experimental es que la herramienta propuesta es capaz de detectar la mayoría de las trazas entre documentos de requerimientos y documentos de arquitectura, cometiendo pocos errores en el proceso. Para probar la hipótesis, como así también guiar los experimentos y el análisis de resultados, se plantearon las siguientes preguntas de investigación:

1. ¿Puede la herramienta detectar las decisiones de diseño a partir de los documentos de arquitectura correctamente?
2. ¿Puede la herramienta detectar la mayoría de las trazas entre los documentos de requerimientos y de arquitectura?
3. ¿Es beneficioso utilizar las decisiones de diseño y no todo el documento de arquitectura para detectar las trazas?

Los casos de estudio utilizados en la evaluación provienen de diferentes dominios, los cuales abarcan el desarrollo de un sistema de monitoreo centralizado, un sitio web de venta de mascotas y una agencia de viajes. Se eligieron estos casos de estudio por varias razones. En primer lugar, estos sistemas cuentan con un amplio conjunto de artefactos de software disponible públicamente [?, ?, ?], incluyendo especificaciones de requerimientos, documentos de arquitectura y código fuente. En segundo lugar, otros investigadores han utilizado estos casos de estudio en evaluaciones similares [?, ?]. En tercer lugar, los documentos de requerimientos y los documentos de arquitectura fueron revisados por expertos los cuales aceptaron su uso como documentos de evaluación.

La evaluación fue dividida en dos partes. La primer parte tiene como objetivo evaluar las decisiones de diseño descubiertas por la herramienta con aquellas detectadas por analistas humanos. Del mismo modo, la segunda parte tiene como objetivo comparar las trazas recuperadas por la herramienta con las detectadas por analistas humanos. A la hora de recuperar las trazas, se utilizó REAssistant para obtener los *crosscutting concerns* a partir de los documentos de requerimientos. Esta herramienta ya fue evaluada en [?] y por lo tanto no va a ser evaluada en este trabajo. Dado que la herramienta depende de REAssistant para recuperar la trazabilidad, además de *crosscutting concerns* recuperados por REAssistant, se van a utilizar soluciones ideales para cada caso de estudio. El objetivo es evaluar el desempeño de TRAS independientemente del desempeño de REAssistant.

Caso de estudio	MSLite	Pet Store	Adventure Builder
Tamaño de los documentos de casos de uso	38	8	5
Número de casos de uso	22	6	4
Tipos de crosscutting concerns	8	8	8
Número de ocurrencias de crosscutting concerns	99	63	64
Tamaño de los documentos de arquitectura	140 páginas	24 páginas	36 páginas
Tipos de decisiones de diseño	7	5	6
Número de ocurrencias de decisiones de diseño	212	35	115
Trazas	456	38	55

Cuadro 5.1: Casos de estudio.

5.1. Casos de estudio

Los tres casos de estudio utilizados en la evaluación son: *Management System Lightweight (MSLite)*, *Pet Store* y *Adventure Builder*. Estos casos de estudio son descritos brevemente a continuación.

- Caso de estudio #1: *MSLite*. Este es un sistema desarrollado en conjunto por alumnos de la universidad de Carnegie Mellon y Siemens [?]. *MSLite* es un sistema que monitorea y controla automáticamente varias funciones de un edificio, tales como: calefacción, ventilación, aire acondicionado, acceso y seguridad. Los documentos de requerimientos constan de 22 casos de uso (alrededor de 38 páginas textuales) y los documentos de arquitectura tienen alrededor de 140 páginas textuales.
- Caso de estudio #2: *Pet Store*. Desarrollado por Sun Microsystems¹ en el contexto del programa Java BluePrints [?]. *Pet Store* es un sistema que permite la venta de todo tipo de mascota a través de internet. Los usuarios pueden elegir mascotas de un catálogo, realizar una compra y controlar el estado de la misma. El sistema consta de 6 casos de uso descritos en alrededor de 8 páginas textuales mientras que los documentos de arquitectura tienen alrededor de 24 páginas textuales.
- Caso de estudio #3: *Adventure Builder*. Desarrollado por Sun Microsystems en el contexto del programa Java BluePrints [?]. *Adventure Builder* es una compañía ficticia que vende paquetes de aventuras para vacacionistas a través de internet. Estos paquetes se generan a partir de vuelos, hoteles y actividades que el usuario selecciona. El sistema interactúa con diferentes agentes externos tales como bancos, hoteles y compañías aéreas. Los documentos de requerimientos constan de 4 casos de uso descritos en alrededor de 5 páginas textuales, mientras que los documentos de arquitectura tienen alrededor de 36 páginas textuales.

5.2. Análisis de los casos de estudio

Dado que los resultados de la herramienta deben ser contrastados con una solución ideal, se necesitan tres soluciones ideales, una para cada caso de estudio. Estas deben incluir las decisiones de diseño, los *crosscutting concerns* y las trazas entre *crosscutting concerns* y decisiones de diseño que se cree que son correctas. Considerando que la información recuperada de los casos de estudio no es lo suficientemente detallada para derivar una solución ideal, hemos pedido a analistas externos que generen la solución ideal a partir de cada caso de estudio.

La Tabla 5.1 muestra entre otras cosas, la cantidad de ocurrencias de decisiones de diseño, *crosscutting concerns* y trazas obtenidas para los 3 casos de estudio:

- *MSLite*: Al poseer una documentación de arquitectura y requerimientos extensa, presenta una mayor cantidad de oraciones acerca de decisiones de diseño y *crosscutting concerns*. El mismo tiene 212 oraciones relacionadas con decisiones de diseño, agrupadas en tácticas, que se asocian a 7 atributos de calidad diferentes. Además, tiene 99 oraciones afectadas por

crosscutting concerns, agrupadas en 8 tipos de *concerns*. Las relaciones entre las oraciones de *crosscutting concerns* y decisiones de diseño resultan en 456 trazas.

- *Pet Store*: Al poseer una documentación de arquitectura y requerimientos más breve, presenta la menor cantidad de oraciones acerca de decisiones de diseño y *crosscutting concerns* de los 3 casos de estudio. Tiene 35 oraciones relacionadas con decisiones de diseño asociadas a 5 atributos de calidad diferentes. Además, tiene 63 oraciones relacionadas con *crosscutting concerns* agrupados en 8 tipos. Las relaciones entre las oraciones de *crosscutting concerns* y decisiones de diseño resultan en 38 trazas.
- *Adventure Builder*: Posee 115 oraciones relacionadas con decisiones de diseño asociadas a 6 atributos de calidad diferentes. Tiene 64 oraciones relacionadas con *crosscutting concerns* agrupados en 8 tipos. Las relaciones entre las oraciones de *crosscutting concerns* y decisiones de diseño resultan en 55 trazas.

La Figura 5.1 muestra la distribución de decisiones de diseño agrupadas en atributos de calidad para *MSLite*. En la imagen, se puede apreciar que los atributos de calidad de *security* (41%) y *modifiability* (34%) cubren el 75 % del total. Esto se debe a que durante el diseño de *MSLite* se prestó mayor atención a la seguridad del sistema. En la documentación se hace mucho énfasis en los permisos que tienen los usuarios que utilizan el sistema para modificar las alarmas y que debe hacer el sistema ante esos cambios.

Las Figuras 5.2 y 5.3 muestran la distribución de decisiones de diseño agrupadas en atributos de calidad para *Pet Store* y *Adventure Builder*. Se observa que las decisiones de diseño se distribuyen de forma similar, con más del 70 % de decisiones relacionadas con *modifiability* e *interoperability* (71 % y 75 %, respectivamente) y en tercer lugar *availability* (20 % y 13 %, respectivamente). Esto se debe a que ambos sistemas son bastante similares desde el punto de vista de su solución arquitectónica. Ambas son aplicaciones web de comercio electrónico. Tanto *Pet Store* como *Adventure Builder*, al ser sitios de ventas que interactúan con otros sistemas como bancos, es normal que posean decisiones de diseño orientadas a la modificabilidad e interoperabilidad del sistema.

En la Figura 5.4 se puede ver la distribución de *crosscutting concerns* dentro de *MSLite*. Se puede apreciar que tanto *Graphical User Interface* (GUI) como *Error Management* contienen más del 50 % de las oraciones afectadas por *crosscutting concerns*. Por último, las Figuras 5.5 y 5.6 muestran la distribución de *crosscutting concerns* para *Pet Store* y *Adventure Builder* respectivamente, las cuales muestran que GUI y *Persistence* presentan los mayores valores para ambos casos con 54 % y 46 % respectivamente.

5.3. Métricas

Para evaluar el desempeño de la herramienta y poder compararla con las soluciones ideales, se tuvieron en cuenta varias métricas. Estas provienen del área de recuperación de información, y son comúnmente utilizadas para evaluar técnicas de descubrimiento de decisiones de diseño y *crosscutting concerns* [?, ?, ?], así como también técnicas para evaluar la recuperación de trazas [?, ?]. Además, se buscó comparar el beneficio de utilizar una herramienta para recuperar la trazabilidad y hacer el mismo trabajo manualmente.

Para describir las métricas utilizadas en la evaluación de detección de decisiones de diseño y recuperación de trazabilidad, es necesario definir los siguientes términos:

Se considera a los casos de uso como un conjunto de documentos $D = d_1, d_2, \dots, d_n$, donde $n = |D|$ es el total de casos de uso para un sistema particular. Cada documento d_i está compuesto por oraciones de la forma: $d_i = s_{i1}, s_{i2}, \dots, s_{im}$, donde $m_i = |d_i|$ es el número total de oraciones dentro de un documento. Sea $C = c_1, c_2, \dots, c_p$ el conjunto "correcto" de oraciones afectadas por *crosscutting concerns* de un sistema. Un *concern* particular $c_j \subset C$ para $1 \leq j \leq p$ está compuesto de un número de ocurrencias $L_j = l_{j1}, l_{j2}, \dots, l_{jq}$ que denota la presencia de un *concern* en una oración de D. La Figura 5.7 ilustra el razonamiento introducido anteriormente.

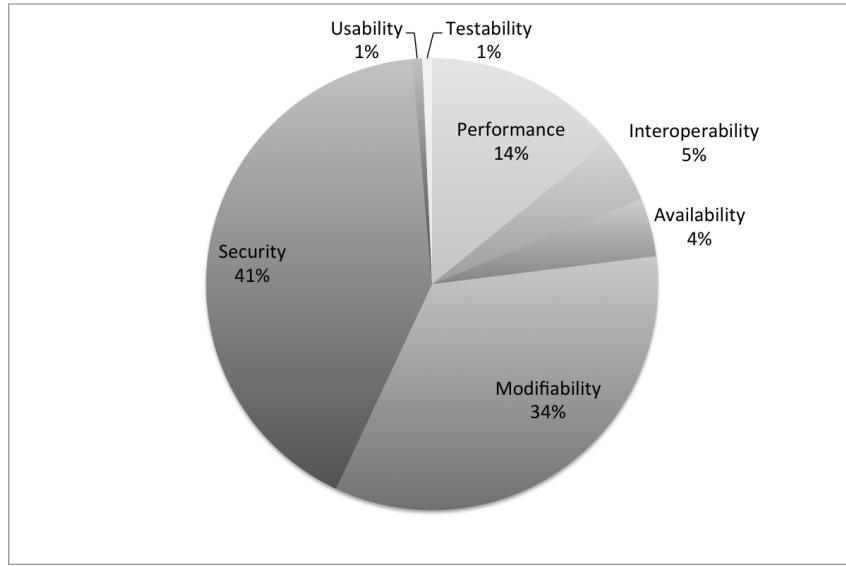


Figura 5.1: MSLite. Distribución de las decisiones de diseño.

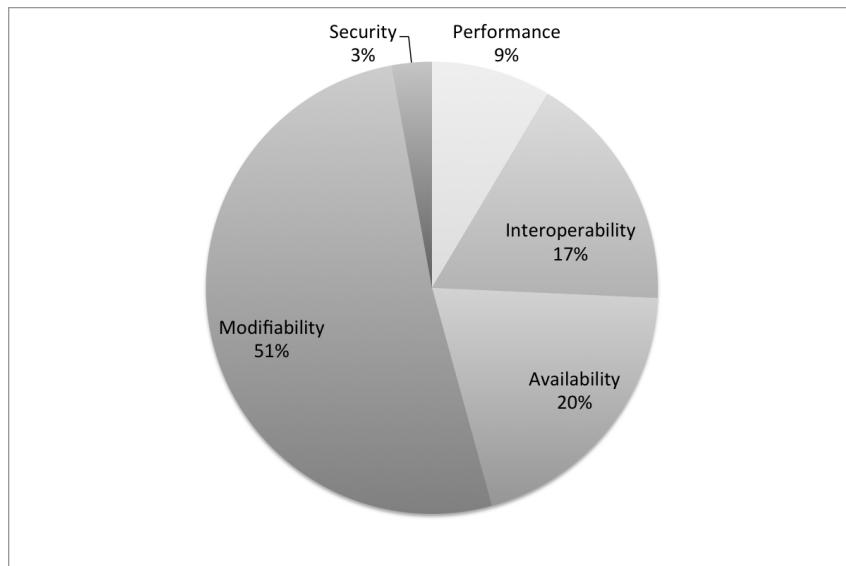


Figura 5.2: Pet Store. Distribución de las decisiones de diseño.

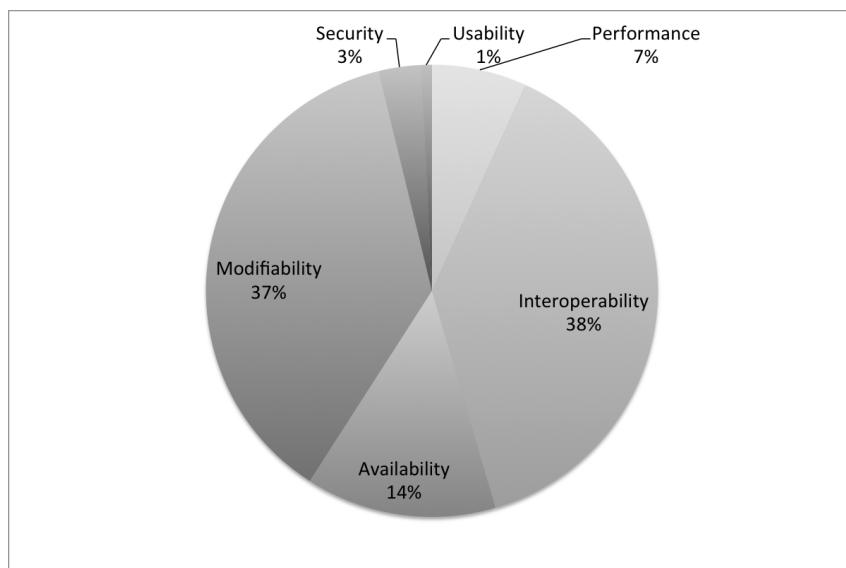


Figura 5.3: Adventure Builder. Distribución de las decisiones de diseño.

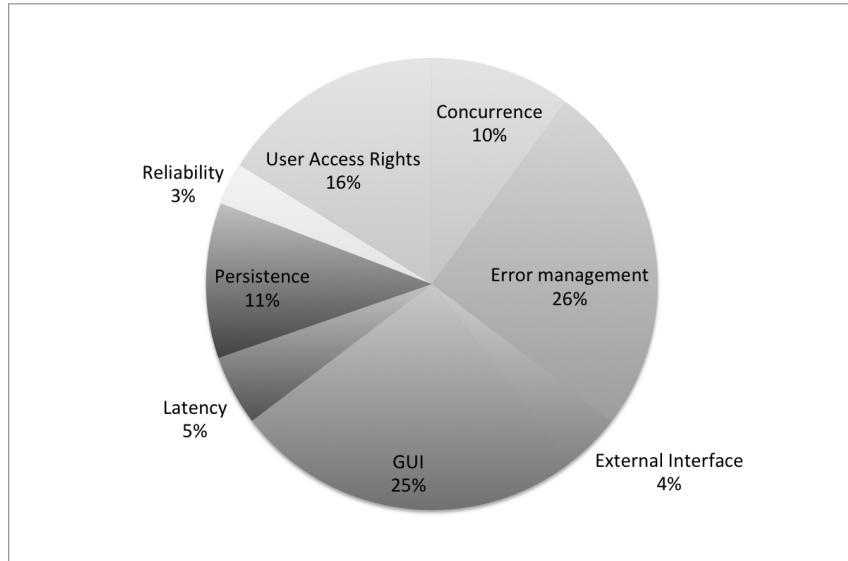


Figura 5.4: MSLite. Distribución de los *crosscutting concerns*.

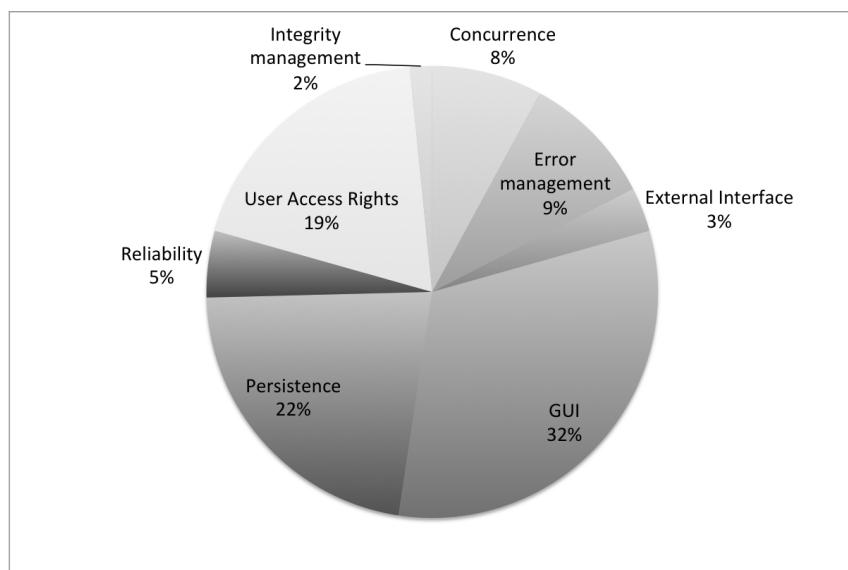


Figura 5.5: Pet Store. Distribución de los *crosscutting concerns*.

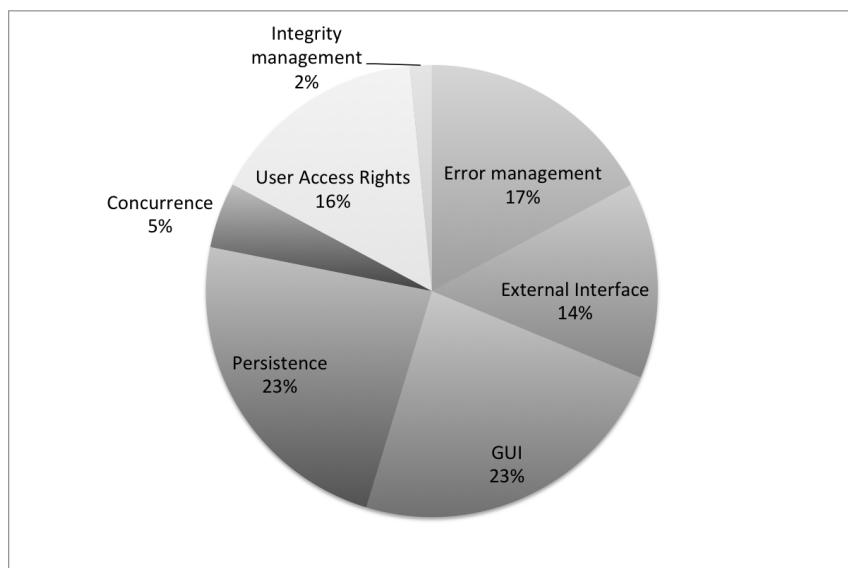


Figura 5.6: Adventure Builder. Distribución de los *crosscutting concerns*.

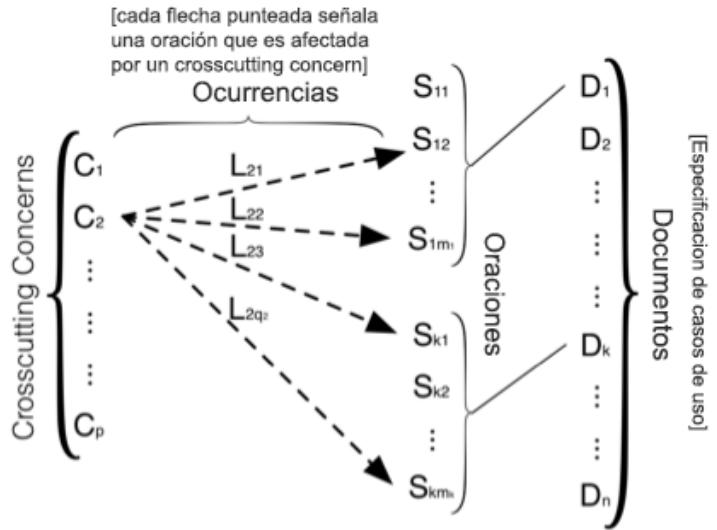


Figura 5.7: Notación formal de los *crosscutting concerns*.

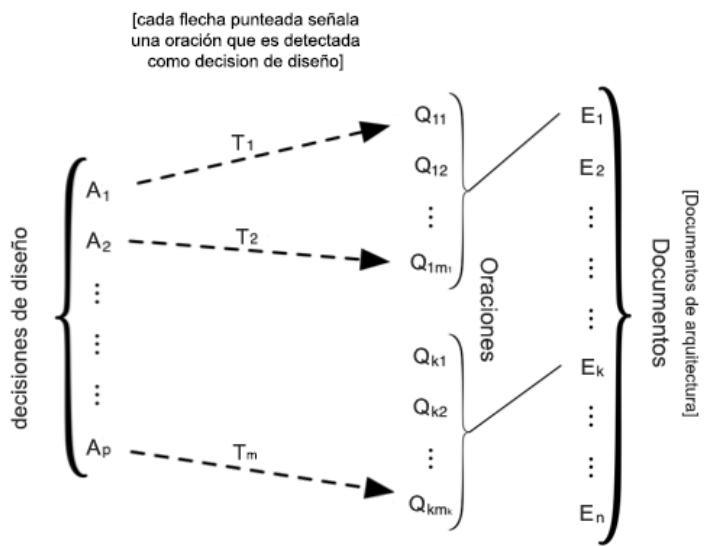


Figura 5.8: Notación formal de las decisiones de diseño.

Del mismo modo, se considera a los documentos de arquitectura como un conjunto de documentos $E = E_1, E_2, \dots, E_n$ donde $B = |E|$ es el total de documentos de arquitectura para un sistema particular. Cada documento E_i esta compuesto por oraciones de la forma $E_i = q_{i1}, q_{i2}, \dots, q_{im}$, donde $m_i = |e_i|$ es el número total de oraciones dentro de un documento. Sea $A = a_1, a_2, \dots, a_f$ el conjunto "correcto" de oraciones que contienen referencias a una decisiones de diseño. Una decisión de diseño particular $a_j \subset A$ para $1 \leq j \leq f$ está representada como una oración de E . De esta forma, $A \subset E$. La Figura 5.8 ilustra el razonamiento gráficamente para los documentos de arquitectura.

Por último, dados los pares $l_i - t_j$, se denomina traza z_k al par $l_i - t_j$ donde $1 \leq i \leq n$ y $1 \leq j \leq u$, siendo n y u el numero de ocurrencias de *crosscutting concerns* y decisiones de diseño respectivamente. Se denomina $Z = |z_k|$ al número total de trazas del sistema, este razonamiento se muestra en la Figura 5.9.

Con esta formalización, dada una herramienta que genera un conjunto potencial de decisiones de diseño $T' \subset T$ y trazas $Z' \subset Z$, se puede analizar qué decisiones están incluidas o no en T y qué trazas z están incluidas o no en Z . Estos resultados se expresan a menudo en términos de operadores de clasificación binaria, que cuentan el número de resultados de predicción: true/false

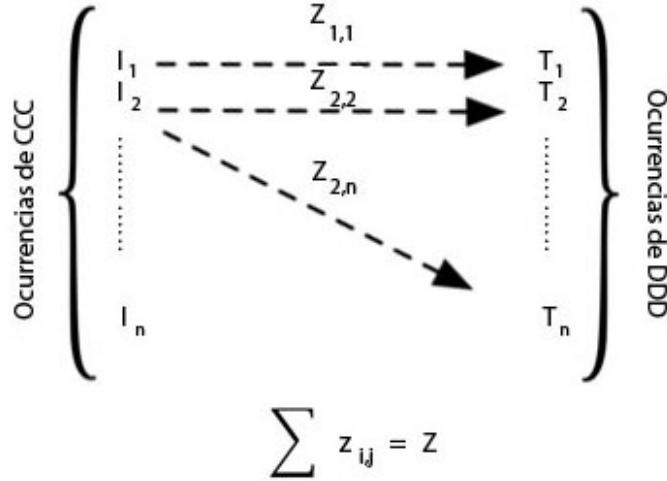


Figura 5.9: Notación formal de las trazas.

positives/negatives.

- *true positive*: una decisión de diseño (o una traza) es considerada como un true positive si además de ser detectada por la herramienta, es parte de la solución ideal.
- *true negative*: es una decisión de diseño (o una traza) que no es parte de la solución ideal, y la herramienta tampoco detecta.
- *false negative*: es una decisión de diseño (o una traza) que es parte de la solución ideal, pero no fue detectada por la herramienta.
- *false positive*: es una decisión de diseño (o una traza) que fue detectada por la herramienta pero que no es parte de la solución ideal.

Las Funciones 5.1, 5.2, 5.3 y 5.4 muestran los cálculos de precisión general para un conjunto particular de decisiones de diseño. Del mismo modo la funciones 5.5, 5.6, 5.7 y 5.8 muestran los cálculos para un conjunto de trazas.

$$tp_j = |T_j \cap T_j| = \text{sum}(\{q : q \in T_j \wedge q \in T_j\}) \quad (5.1)$$

$$tn_j = |\overline{T}_j \cap \overline{T}_j| = \text{sum}(\{q : q \notin T_j \wedge q \notin T_j\}) \quad (5.2)$$

$$fn_j = |\overline{T}_j \cap T_j| = \text{sum}(\{q : q \notin T_j \wedge q \in T_j\}) \quad (5.3)$$

$$fp_j = |T_j \cap \overline{T}_j| = \text{sum}(\{q : q \in T_j \wedge q \notin T_j\}) \quad (5.4)$$

$$tp_j = |Z_j \cap Z_j| = \text{sum}(\{w : w \in Z_j \wedge z \in Z_j\}) \quad (5.5)$$

$$tn_j = |\overline{Z}_j \cap \overline{Z}_j| = \text{sum}(\{w : w \notin Z_j \wedge z \notin Z_j\}) \quad (5.6)$$

$$fn_j = |\overline{Z}_j \cap Z_j| = \text{sum}(\{w : w \notin Z_j \wedge z \in Z_j\}) \quad (5.7)$$

$$fp_j = |Z_j \cap \overline{Z}_j| = \text{sum}(\{w : w \in Z_j \wedge z \notin Z_j\}) \quad (5.8)$$

El cálculo de los valores globales de predicción se derivan directamente de las fórmulas anteriores:

$$TP = \sum_{\forall j} tp_j \quad (5.9)$$

$$TN = \sum_{\forall j} tn_j \quad (5.10)$$

$$FN = \sum_{\forall j} fn_j \quad (5.11)$$

$$FP = \sum_{\forall j} fp_j \quad (5.12)$$

Basados en los valores de predicción generales (TP, TN, FN y FP), las métricas de interés para la evaluación son simples de calcular. Dos métricas importantes son la *precision* y el *recall* [?]. En este enfoque, *precision* mide cuántas de las decisiones de diseño o trazas detectadas por la herramienta (o un analista) son correctas, mientras que el *recall* mide, del total de decisiones de diseño o trazas existentes, cuantas fueron detectadas por la herramienta [?]. Esto significa que alcanzar una buena *precision* implica que la herramienta (o un analista) no comete errores, mientras que al obtener un buen *recall* se recuperan la mayoría de las decisiones de diseño o trazas. Las funciones 5.13 y 5.14 muestran los cálculos para obtener los valores de *precision* y *recall*, respectivamente.

$$precision = \frac{TP}{TP + TF} \quad (5.13)$$

$$recall = \frac{TP}{TP + FN} \quad (5.14)$$

Otra métrica interesante para evaluar el desempeño de la herramienta es la *F-Measure* (f_b) [?]. f_b constituye la media armónica entre el *recall* y la *precision* (ver función 5.15). Una característica de f_b es que puede ser utilizada para favorecer una métrica o la otra a través del parámetro b . La media f_1 ($b=1$) es comúnmente usada en la comunidad [?, ?], y considera el *recall* y *precision* por igual. Para una herramienta semi-automática, lo que se busca es obtener un *recall* alto y una *precision* aceptable. Es más fácil para el analista descartar sugerencias incorrectas que encontrar las faltantes [?]. Por lo tanto, se va a utilizar la medida f_2 ($b=2$), la cual favorece el *recall* por sobre *precision*.

$$\begin{aligned} f_b &= \frac{1 + b^2}{\frac{b^2}{recall} + \frac{1}{precision}} = (1 + b^2) \cdot \frac{recall \cdot precision}{b^2 \cdot precision + recall} \\ f_b &= \frac{1 + b^2}{\frac{b^2}{recall} + \frac{1}{precision}} = (1 + b^2) \cdot \frac{recall \cdot precision}{b^2 \cdot precision + recall} \\ f_2 &= \frac{5}{\frac{4}{recall} + \frac{1}{precision}} = 5 \cdot \frac{recall \cdot precision}{4 \cdot precision + recall} \end{aligned} \quad (5.15)$$

De la misma manera que las métricas de *precision* y *recall* establecen que tan bien la herramienta (o los analistas) detectaron las decisiones de diseño o relaciones de trazabilidad, existe otra métrica denominada *accuracy* [?]. La *accuracy* mide la proximidad entre los resultados obtenidos y el valor real. La fórmula 5.16 muestra la fórmula para calcular *accuracy*, la cual hace uso de los cuatro operadores de predicción (TP, TN, FP y FN).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.16)$$

Por último, para obtener una indicador del beneficio al usar la herramienta propuesta por sobre un análisis manual se utiliza una medida denominada *Recovery Effort Index* (REI, ver fórmula 5.17) [?]. REI se define como la relación entre el número de trazas recuperadas, sobre el total de trazas posibles y puede ser utilizada para estimar el porcentaje de esfuerzo requerido para analizar manualmente los resultados obtenidos con la herramienta [?].

$$REI_i = \frac{|trazas\ recuperadas|}{|trazas\ posibles|} \quad (5.17)$$

Las métricas explicadas con anterioridad permiten evaluar la calidad de la herramienta desde distintos ángulos y dan indicadores para realizar observaciones sobre los experimentos descritos a continuación.

5.4. Experimentos

Los experimentos están organizados de acuerdo a las preguntas de investigación en dos partes. En primer lugar se ejecutó la herramienta para detectar las decisiones de diseño sobre los tres casos de estudio. Estos resultados fueron comparados con las soluciones ideales. Se utilizaron las métricas de *recall*, *precision*, *f₂* y *accuracy*. Para satisfacer la pregunta de investigación N°1, la herramienta debe obtener un *recall* lo más cercano posible a 100 %, ya que un bajo *recall* implicaría que decisiones de diseño que puedan tener trazas asociadas no son detectadas. En esta parte de la evaluación se realizaron 3 experimentos en total, uno por cada caso de estudio.

La segunda parte de los experimentos se focaliza en la recuperación de trazas y las preguntas de investigación N°2 y N°3. Dicha recuperación se realiza a partir de los *crosscutting concerns* detectados por REAssistant y las decisiones de diseño que surgen de los experimentos de la primera parte. A su vez, para evaluar la etapa de forma individual, se utilizaron 3 fuentes diferentes de decisiones de diseño. Las provistas por analistas, las provistas luego de ejecutar la herramienta y una tercera fuente donde se toma todo el documento de arquitectura como un conjunto de decisiones de diseño. Este último experimento busca responder la pregunta de investigación N°3, en la cual se considera que es más efectivo identificar trazas sobre la porción relevante del documento que sobre todo el documento. En total se realizaron 3 experimentos por cada caso de estudio dando un total de 9 experimentos para esta etapa.

Para todos los experimentos de búsqueda de trazabilidad se utilizan los *crosscutting concerns* de las soluciones ideales. Se busca evitar el posible acarreo de errores provenientes de REAssistant. Además, se utilizaron las métricas de *precision*, *recall* y *REI*, esta última de vital importancia dado que se busca poder medir el beneficio de la herramienta con respecto al análisis manual. Se comparó la relación entre *precision* y *recall* a partir de variaciones en la configuración de la herramienta. Específicamente, y con el propósito de establecer como los parámetros del algoritmo afectan los resultados de la herramienta, se variaron los valores del threshold y el número de dimensiones al recuperar las trazas.

5.4.1. Evaluación de detección de decisiones de diseño

A la hora de evaluar las decisiones de diseño, la herramienta utiliza una técnica basada en reglas. Estas utilizan la información recuperada luego de utilizar anotadores del framework de UIMA. La idea es permitir a los analistas definir sus propias reglas, que utilicen propiedades lingüísticas y semánticas para detectar decisiones de diseño [?]. Para evaluar el desempeño de la detección, la herramienta utiliza reglas definidas por expertos. Éstas fueron desarrolladas sin tener en cuenta el dominio de los casos de estudio utilizados en los experimentos.

Parámetros y configuración de la herramienta

Para realizar los experimentos propuestos, es necesario configurar los anotadores de UIMA. En este caso, los anotadores son implementaciones de terceros que realizan tareas de procesa-

	Precision	Recall	Accuracy	F_2
MSLite	0.95	0.97	0.97	0.97
Pet Store	0.78	0.91	0.83	0.87
Adventure Builder	0.76	0.90	0.81	0.87

Cuadro 5.2: Detección de decisiones de diseño.

miento de lenguaje natural. Los anotadores utilizados son parte de Stanford CoreNLP e incluyen los siguientes módulos: *Tokenizer*, *Sentence Splitter*, *POS Tagger*, *Stemmer* y *Lemmatizer*.

El desarrollo de un conjunto de reglas fue de vital importancia para esta etapa. Para producir este tipo de reglas sin sesgar la investigación, se consultó a un grupo externo de arquitectos y analistas funcionales de alto nivel. Se les pidió que desarrollen un conjunto inicial de reglas para lo cual trabajaron de manera individual utilizando información que explicaba cuales eran las anotaciones que la herramienta genera y como era la sintaxis para crear reglas. Cada analista generó reglas basadas en su propia experiencia con respecto a decisiones de diseño presentes en los documentos de arquitectura. Por último se realizó una reunión donde todos los analistas validaron las reglas propuestas y las consolidaron en un único conjunto. Se cree que este conjunto inicial es lo suficientemente general para ser aplicado a diferentes dominios.

El conjunto de reglas que se utiliza actualmente fue agrupado de acuerdo a la táctica y atributo de calidad que busca identificar. La herramienta tiene entre otras, reglas para tácticas como Detección de fallas, Recuperación de fallas o Detección de ataques asociadas a la disponibilidad del sistema. En total hay 248 reglas que detectan 7 atributos de calidad (ver Apéndice A).

Análisis de los resultados

Las Figuras 5.10, 5.11 y 5.12 muestran los resultados de *precision*, *recall*, f_2 y *accuracy* obtenidos luego de ejecutar la detección de decisiones de diseño sobre los casos de estudio *MSLite*, *Pet Store* y *Adventure Builder*, respectivamente. Se observa que en todos los casos el *recall* está por encima del 90 %, mientras que la *precision* varía entre el peor caso con un 76 % para *Adventure Builder*, un 78 % para *Pet Store* y un 95 % para *MSLite*. Estos valores permiten validar la pregunta de investigación N°1, ya que la herramienta puede ser utilizada para recuperar las decisiones de diseño a partir de los documentos de arquitectura.

La Tabla 5.2 muestra los valores de *accuracy* y *F-Measure* obtenidos. En el caso de *accuracy* el valor es en todos los casos mayor al 81 %. Esto se debe principalmente a que hay muchas oraciones que son detectadas como true negatives, es decir, información dentro de los documentos de arquitectura que no es parte de las decisiones de diseño y por lo tanto es descartada para la búsqueda de trazas. Para documentos de gran tamaño como *MSLite*, el *accuracy* es mayor, 97 %.

En el caso de F-Measure, los valores son en todos los casos superiores al 87 %, esto se debe a que se tiene un *recall* por sobre el 90 % y la métrica fue definida para favorecer el *recall* por sobre la *precision*.

5.4.2. Evaluación de búsqueda de trazabilidad

Para ser eficaz, una herramienta de recuperación de trazas debe recuperar la mayor cantidad de trazas correctas como sea posible. A diferencia de una herramienta de búsqueda tradicional, en la que la *precision* es más importante que el *recall*, a la hora de buscar trazas las herramientas deben recuperar un alto porcentaje de trazas si desean ser útiles [?].

Parámetros y configuración de la herramienta

Dado que esta etapa está basada en el algoritmo de LSA, su evaluación tiene en cuenta diferentes configuraciones con las que este algoritmo puede ejecutarse para recuperar trazas. El uso efectivo de LSA es un proceso que requiere un ajuste sofisticado. Son muchos los factores que

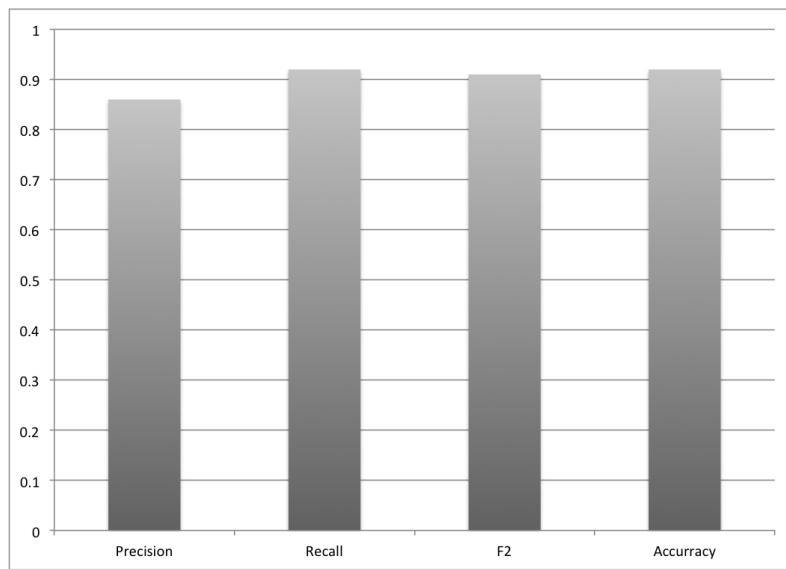


Figura 5.10: Detección de decisiones de diseño en *MSLite*.

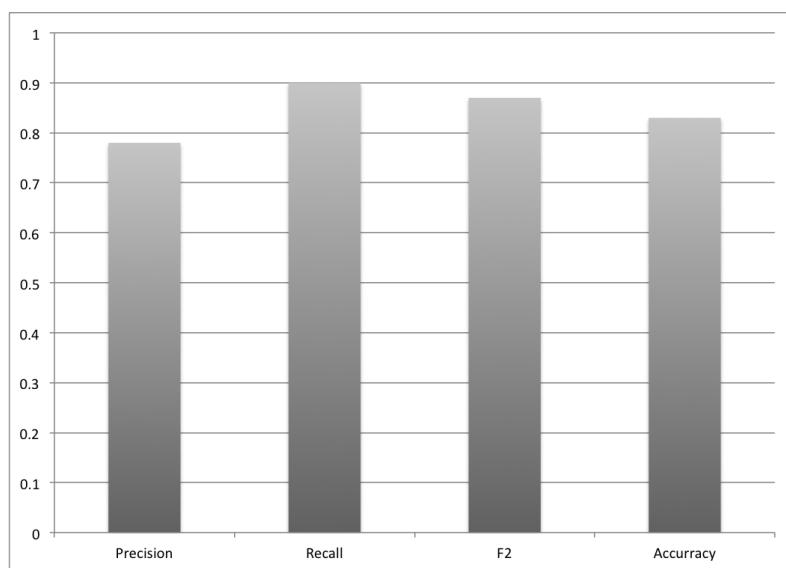


Figura 5.11: Detección de decisiones de diseño en *Pet Store*.

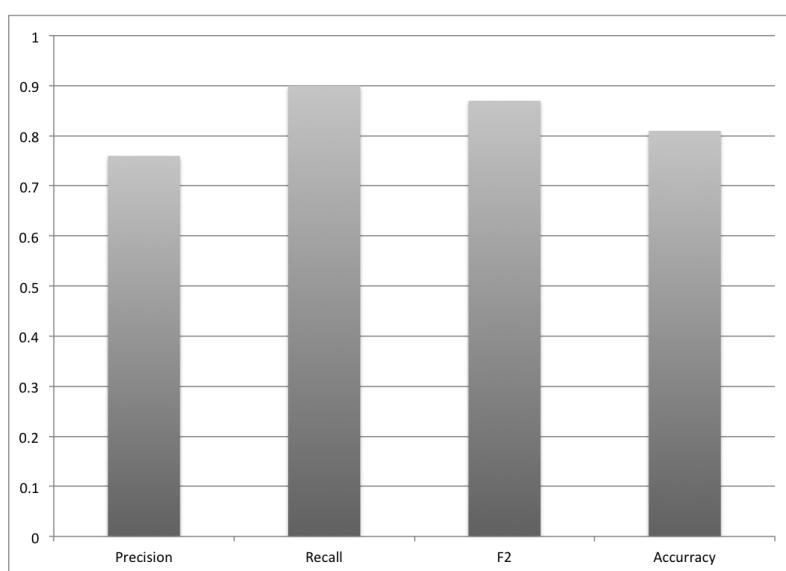


Figura 5.12: Detección de decisiones de diseño en *Adventure Builder*.

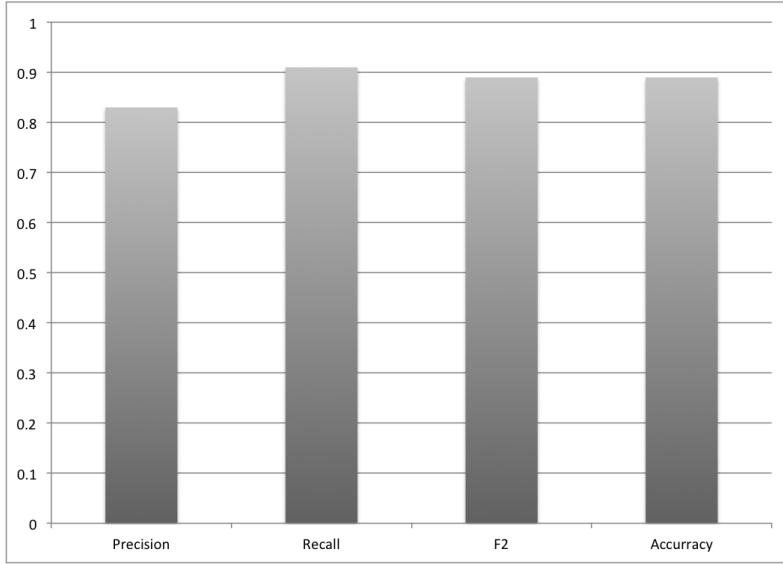


Figura 5.13: Promedio de métricas para la detección de decisiones de diseño.

pueden influenciar el rendimiento de esta técnica [?, ?]. Estos factores son el pre-procesamiento de texto utilizado para generar la matriz dimensional (quitar las stopwords, los caracteres especiales, separar las palabras en CamelCase), la elección del número de dimensiones para la matriz y la elección del valor de threshold a partir del cual se descartan las posibles trazas [?, ?].

LSA utiliza un modelo matemático de aproximación para recuperar trazas y, cuando se está trabajando con lenguaje natural que contiene miles de documentos y millones de términos, la mayoría de los autores sugieren usar una aproximación de entre 200 y 500 dimensiones. En el análisis de software, la cantidad de documentos es mucho menor, por lo tanto se deben utilizar dimensiones de entre 40 y 100 para obtener resultados aceptables [?, ?, ?].

Luego de algunos experimentos con diferentes valores, se decidió utilizar 60 como el número de dimensiones para el análisis de trazas, y por lo tanto, los experimentos desarrollados en este capítulo utilizan ese valor. De todas formas, la herramienta soporta dimensiones y thresholds personalizables. Los analistas pueden cambiar estos valores en base a los dominios sobre los cuales desean utilizar la herramienta.

Análisis de los resultados

Esta sección describe los resultados de los experimentos descritos en la sección 5.3. Las figuras 5.14, 5.15 y 5.16 muestran la variación de los valores de *precision* dado diferentes valores de *recall* para *MSLite*, *Pet Store* y *Adventure Builder*, respectivamente. Para obtener los diferentes valores de *recall* y *precision*, se utilizó 60 como número de dimensiones y se varió el valor de *threshold*. Cada Figura posee 3 líneas punteadas verticales, denotando el valor ideal de *recall*.

Los resultados muestran que para los 3 casos de estudio, utilizar las decisiones de diseño y no todo el contenido de los documentos de arquitectura mejora sustancialmente la precisión en la recuperación de trazas. Esto permite responder la pregunta de investigación N°3, ya que es beneficioso utilizar las decisiones de diseño y no todo el documento de arquitectura para detectar las trazas.

En el caso de *Pet Store*, la Figura 5.15 muestra valores más altos de precisión para distintos niveles de *recall* con respecto a los otros casos de estudio. Esto se debe a que el número de trazas a recuperar no difiere mucho del número de posibles trazas. Como en los otros casos de estudio la diferencia es mayor, la *precision* es más baja..

Las Tablas 5.3, 5.4 y 5.5 muestran los valores de *precision* y *recall* obtenidos para distintos valores de *threshold*, todos ellos entre 0.5 y 1.0. Las filas resaltadas indican el valor óptimo de *threshold*, con el cual se obtuvieron los mejores resultados. En el caso de *Pet Store* y *Adventure Builder*, dado que el número de decisiones de diseño y *crosscutting concerns* no es grande

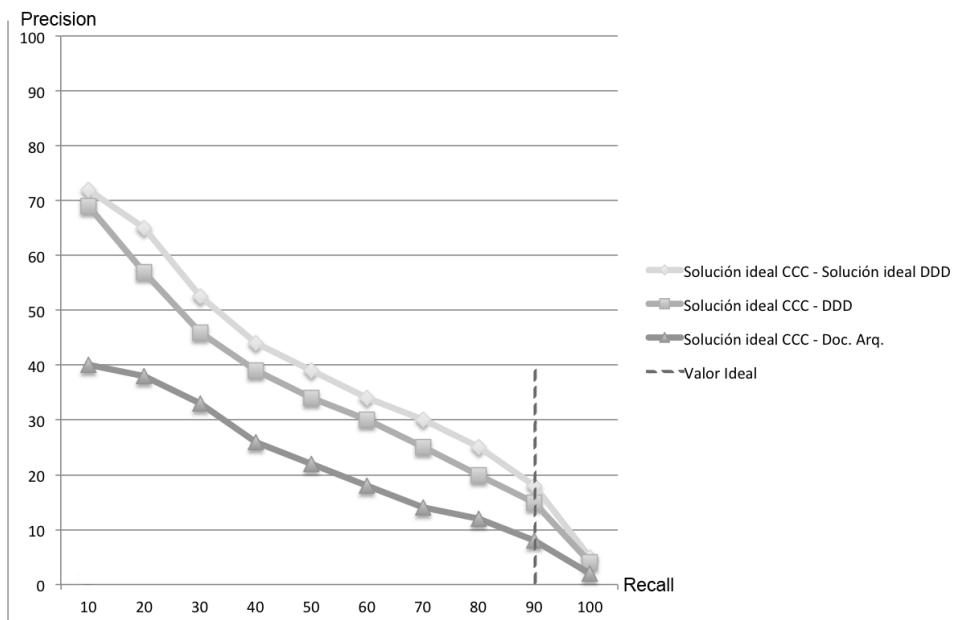


Figura 5.14: *Precision* sobre *recall* - MSLite.

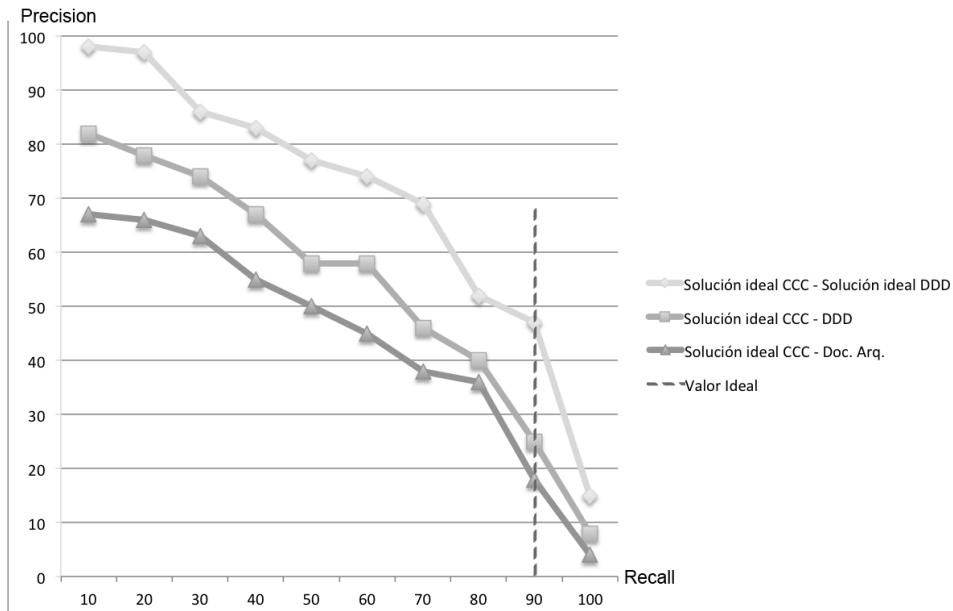


Figura 5.15: *Precision* sobre *recall* - Pet Store.

Threshold	Trazas recuperadas		Precisión (%)	Recall (%)
	Correctas (true positives)	Incorrectas (false positives)		
0.95	3	0	100.00 %	0.66 %
0.9	37	14	72.55 %	8.11 %
0.85	121	100	54.75 %	26.54 %
0.8	218	309	41.37 %	47.81 %
0.75	334	792	29.66 %	73.25 %
0.7	413	1685	19.69 %	90.57 %
0.65	444	3120	12.46 %	97.37 %
0.6	452	5090	8.16 %	99.12 %
0.55	456	7663	5.62 %	100.00 %
0.5	456	11073	3.96 %	100.00 %

Cuadro 5.3: Variación de valores para distintos thresholds - MSLite.

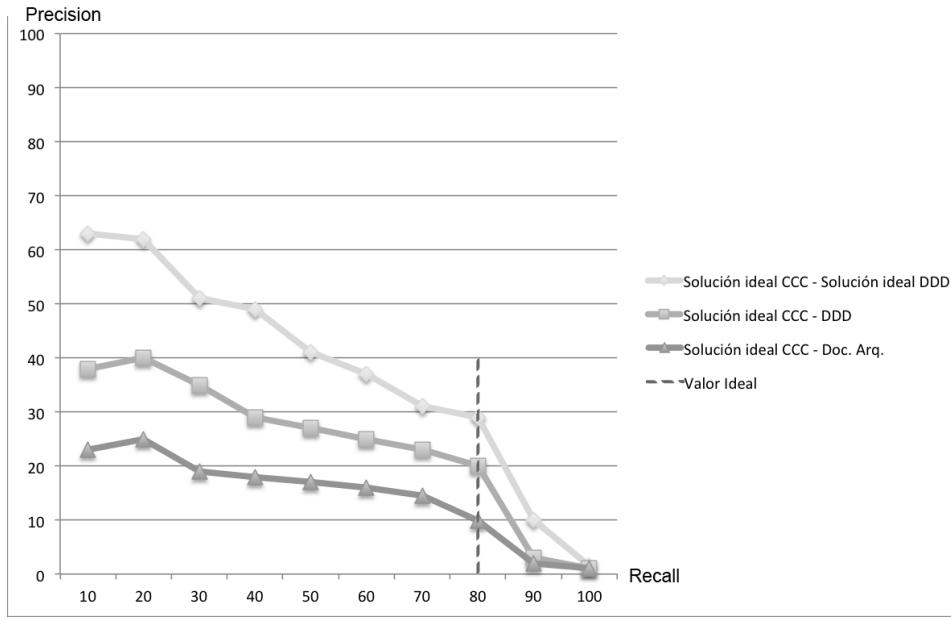


Figura 5.16: *Precision* sobre *recall* - Adventure Builder.

Threshold	Trazas recuperadas		Precisión (%)	Recall (%)
	Correctas (true positives)	Incorrectas (<i>false positives</i>)		
0.95	1	0	100.00 %	2.63 %
0.9	1	0	100.00 %	2.63 %
0.85	1	0	100.00 %	2.63 %
0.8	11	3	78.57 %	28.95 %
0.75	21	5	80.77 %	55.26 %
0.7	28	18	60.87 %	73.68 %
0.65	35	33	51.47 %	92.11 %
0.6	37	81	31.36 %	97.37 %
0.55	37	155	19.27 %	97.37 %
0.5	38	272	12.26 %	100.00 %

Cuadro 5.4: Variación de valores para distintos thresholds - Pet Store.

Threshold	Trazas recuperadas		Precisión (%)	Recall (%)
	Correctas (true positives)	Incorrectas (<i>false positives</i>)		
0.95	0	0	0.00 %	0.00 %
0.9	0	05.11	0.00 %	0.00 %
0.85	1	0	100.00 %	1.82 %
0.8	2	2	50.00 %	3.64 %
0.75	5	3	62.50 %	9.09 %
0.7	14	10	58.33 %	25.45 %
0.65	27	26	50.94 %	49.09 %
0.6	38	50	43.18 %	69.09 %
0.55	51	103	33.12 %	92.73 %
0.5	54	200	21.26 %	98.18 %

Cuadro 5.5: Variación de valores para distintos thresholds - Adventure Builder.

	Trazas		REI
	Recuperadas	Posibles	
MSLite	456	84148	0.99
Pet Store	38	10609	1.00
Adventure Builder	55	12740	1.00

Cuadro 5.6: REI

(aproximadamente 500), el algoritmo de LSA obtiene los mejores resultados cuando se utiliza un threshold de 0.65 y 0.55 (valores bajos de threshold). Esto se debe a que el algoritmo no tiene la información suficiente para generar las estructuras que permitan obtener trazas. En estos casos, es necesario reducir el threshold para poder recuperar más trazas aun a costa de que las trazas recuperadas no sean en su mayoría correctas. En el caso de *MSLite* el threshold utilizado es mayor a 0.7, el cual es un valor estándar según en la literatura [?, ?].

La Tabla 5.6 muestra los valores de REI para los tres casos de estudio, obteniendo para todos los casos de estudio valores mayores al 93 %. Estos resultados implican que el uso de la herramienta reduce en promedio un 93 % el esfuerzo necesario para recuperar las trazas en comparación con un análisis completamente manual. Se observa un valor mayor para *Pet Store* y *Adventure Builder* con respecto a *MSLite*. Esto ocurre debido a que el numero de falsos positivos en *MSLite* para un *recall* 100 % es mayor en promedio a los otros casos de estudio. Los valores de REI obtenidos permiten además responder la pregunta de investigación N°2, donde la herramienta puede detectar la mayoría de las trazas entre los documentos de requerimientos y de arquitectura.

5.5. Discusión de los resultados

De los experimentos anteriores se pueden hacer ciertas observaciones. En el caso de la etapa de recuperación de decisiones de diseño, las Figuras 5.10, 5.11 y 5.12 muestran que para todos los casos de estudio el *recall* estuvo por sobre el 90 % y la *precision* sobre el 78 %. Solo en el caso de *Adventure Builder* la *precision* y el *recall* se vieron afectados, aunque se cree que esto es debido a la pobre calidad con la que fueron escritos los documentos de arquitectura. Los valores obtenidos avalan que el método utilizado permite recuperar una gran cantidad de las decisiones de diseño.

La segunda etapa fue evaluada midiendo la *precision* sobre distintos valores de *recall* dadas tres variaciones del origen de las decisiones de diseño:

1. Resultantes de ejecutar la herramienta (reglas).
2. Provistas por la solución ideal (según los expertos del área).
3. Tomando todo el documento de arquitectura como fuente de decisiones de diseño.

Las Figuras 5.14, 5.15 y 5.16 muestran los resultados para cada caso de estudio. Para todos los casos de estudio, recuperar y utilizar solamente las decisiones de diseño mejoró la precisión en un 20 % a la hora de recuperar las trazas. En especial, si las decisiones de diseño recuperadas por la herramienta son validadas por un analista antes de ser utilizadas en la recuperación de trazas.

Se cree que la mejora en la *precision* se debe a que cuando se utilizan decisiones de diseño, no se están generando trazas donde la similitud es netamente sintáctica como es el caso de oraciones dentro del documento de arquitectura que contienen palabras similares a las utilizadas en los *crosscutting concerns* pero que no están relacionados con QAs del sistema..

La Tabla 5.6 muestra los valores de REI para los casos de estudio, donde en todos los casos el valor es mayor al 99 %, es decir, que el uso de la herramienta (aún cuando el analista debe validar las trazas recuperadas) mejora en al menos un 99 % con respecto a realizar el análisis manual.

Tomando como ejemplo *MSLite*, donde el espacio de solución comprende ≈ 84.000 posibles trazas, el analista sólo tendría que analizar 5.600, un número mucho menor que resultaría en un ahorro considerable de tiempo.

Por último, los experimentos permitieron validar las tres preguntas de investigación y probar la hipótesis planteada. La herramienta propuesta es capaz de detectar la mayoría de las trazas entre documentos de requerimientos y documentos de arquitectura, cometiendo pocos errores en el proceso.

5.6. Validez de los resultados

En esta sección se describen los problemas que pueden haber afectado a los resultados de la experimentación. Estos son importantes para el estudio ya que el objetivo de la evaluación es demostrar que a pesar de las limitaciones, la herramienta de recuperación de trazabilidad todavía representa un apoyo útil durante la identificación trazas. A continuación se describen algunos de los inconvenientes observados durante el desarrollo de los experimentos:

- La determinación de las decisiones de diseño para un sistema es una tarea subjetiva. Por lo tanto, el conjunto de decisiones de diseño puede variar de acuerdo al conocimiento y experiencia de la gente involucrada en la actividad. Aunque las soluciones ideales para los casos de estudio fueron desarrolladas por gente externa a esta evaluación, pueden haberse visto condicionadas por la experiencia y conocimiento de los analistas que las definieron.
- La distribución de decisiones de diseño en los documentos reflejan las aptitudes de los arquitectos que desarrollaron el documento y es natural que describan con más detalle cuestiones de su interés (por ejemplo, un arquitecto experto en seguridad va a describir más cantidad de decisiones de diseño de este tipo). Lo que conlleva a que no todas las decisiones tengan la misma distribución y presencia en los documentos.
- Determinar un valor óptimo de threshold y cantidad de dimensiones para LSA con la cual obtener los mejores resultados no es una tarea sencilla, y por lo general estos valores dependen del sistema sobre el cual se utilice la herramienta.
- Las reglas utilizadas para detectar las decisiones de diseño pueden haberse visto condicionadas por la experiencia y conocimiento de los analistas que las definieron.
- Aunque las reglas para la detección de decisiones de diseño han sido definidas de manera tal que pueden ser reutilizadas en otros dominios, puede haber casos en los cuales dichas reglas necesiten ser adaptadas. Si esto ocurriese, nuevas reglas para detectar otro tipo de decisiones de diseño pueden ser fácilmente agregadas.
- El resultado producido por la herramienta está afectado por la calidad de los documentos de entrada (especificaciones de casos de uso y documentos de arquitectura). Si dichos documentos están escritos de una manera pobre y tienen poco detalle, la herramienta probablemente no tenga un buen desempeño a la hora de identificar decisiones de diseño o recuperar trazas.
- Debido al número pequeño de proyectos sobre los cuales se evaluó la herramienta, es probable que la evaluación en otros sistemas produzca diferentes resultados.

5.7. Resumen

En este capítulo, se reportaron los resultados de la evaluación de la herramienta desarrollada en tres casos de estudio. Los casos de estudio cubren diversos dominios de software. Se utilizaron métricas del área de Recuperación de Información (IR) para evaluar la *performance* de las técnicas de detección de decisiones de diseño y recuperación de relaciones de trazabilidad (*precision*, *recall*, *f-measure*, *accuracy* y *REI*).

Para poder llevar a cabo las comparaciones, se solicitó a expertos en el área que inspeccionen manualmente los casos de estudio para obtener soluciones de referencia con la cual evaluar la herramienta propuesta. El objetivo de la evaluación consistió en determinar el desempeño de la técnica de detección de decisiones de diseño utilizando la solución ideal provista por los expertos en el área como parámetro de comparación y observar la calidad de las trazas obtenidas mediante el análisis de los casos de uso y las decisiones de diseño.

A la hora de recuperar las decisiones de diseño, los experimentos mostraron que el uso de la herramienta produce muy buenos resultados tanto en términos de *precision* como de *recall*. Esta última métrica es de gran importancia dado que lo que se busca es recuperar la mayor cantidad de decisiones de diseño, aun a costa de que existan falsos positivos. Tener un buen *recall* implica que el analista solo tiene que validar los verdaderos positivos recuperadas por la herramienta y no dedicar tiempo a buscar los falsos negativos.

Las reglas utilizadas arrojaron un $recall > 90\%$ y una $precision > 75\%$ a la hora de recuperar decisiones de diseño. Estos valores implican que las reglas fueron capaces de detectar en gran medida las decisiones de diseño. Para los casos que las reglas no funcionaron, notamos que las decisiones que no fueron detectadas estaban escritas en más de una oración, o con terminología que no estaba cubierta con las reglas utilizadas. Para estos casos, la herramienta permite modificar las reglas y adaptarlas a dominios específicos.

La recuperación de trazabilidad evidenció buenos resultados de *precision* para valores altos de *recall*. Uno de los problemas al utilizar LSA es la elección del valor de *threshold*, el cual depende del tamaño de los documentos de entrada. Se observó que para documentos de gran tamaño (*MSLite*), el uso de valores de 0.7 (valor por defecto) arrojaron los mejores resultados. Mientras que para documentos pequeños (*Adventure Builder*), los mejores resultados se obtuvieron con valores de 0.55. Esto se debe a que documentos pequeños tienen una menor cantidad de palabras y LSA no es capaz de detectar las estructuras que se forman a partir de las mismas.

El principal aporte está dado en la reducción significativa de esfuerzo necesario con respecto a un análisis manual. Para el caso de *MSLite*, solicitar los documentos y trabajar sobre los *crosscutting concerns* y las decisiones de diseño redujo significativamente el dominio de búsqueda de trazas, eliminando información que no era relevante. El uso de LSA permitió tener una *recall* en ~90% y una *precision* ~30%. Esto implica que en comparación a un análisis manual, la herramienta permite reducir el esfuerzo necesario para recuperar trazas en un 93% en promedio.

Conclusiones

Los sistemas de software de hoy en día son complejos e incluyen una gran cantidad de artefactos complementarios producidos durante el ciclo de vida del producto. Estos artefactos son creados y mantenidos por diferentes stakeholders, como por ejemplo analistas de requerimientos, arquitectos, desarrolladores, etc. Establecer y mantener conexiones explícitas entre artefactos de software como los documentos de requerimientos y arquitectura (SAD) es importante dado que permite, entre otras cosas, verificar que la arquitectura contempla todos los requerimientos del sistema, que la documentación de requerimientos esté completa y que esta sea consistente con las decisiones tomadas en el diseño de la arquitectura, como así también analizar el posible impacto en la arquitectura de cambios en los requerimientos.

Si bien las etapas de captura de requerimientos y creación de la arquitectura están estrechamente relacionadas, las actividades realizadas en dichas etapas son llevadas a cabo por separado y con una interacción limitada [?]. En la industria, la ausencia de estos registros de trazabilidad entre artefactos se atribuye principalmente a que realizar los análisis pertinentes para obtenerlos es un proceso largo, costoso y propenso a errores además, la persona que realiza el análisis tiene que conocer detalladamente los requerimientos y la arquitectura del sistema. A su vez, estos documentos pueden cambiar durante el ciclo de vida del sistema y estos cambios pueden alterar las trazas identificadas inicialmente. Asimismo, los beneficios de las relaciones de trazabilidad por lo general son desestimados debido a la falta de soporte de herramientas para recuperar, mantener y utilizar las trazas [?].

A pesar de los avances respecto a la búsqueda de trazas entre diversos artefactos de software, a nuestro saber y entender, los investigadores no han enfocado su esfuerzo en la recuperación de trazas entre documentos de requerimientos y de arquitectura. Adicionalmente, los enfoques existentes y las técnicas utilizadas en estos presentan ciertas limitaciones cuando son aplicados en documentos extensos y con mucho contenido textual. En el contexto del análisis de trazabilidad entre requerimientos y arquitectura, este tipo de limitaciones podría tener efectos negativos en los resultados de la recuperación, debido a que gran parte del texto presente en los documentos no está directamente relacionado con las trazas. Para asistir a los desarrolladores, como así también resolver los problemas encontrados en las técnicas existentes, se desarrolló un enfoque denominado TRAS que automatiza la identificación de relaciones de trazabilidad entre documentos de requerimientos y arquitectura, combinando técnicas de NLP, reglas específicas del dominio de arquitecturas y el algoritmo de LSA. El aspecto principal de TRAS es que no analiza todo el texto de los documentos para buscar las trazas, sino que reduce el espacio de búsqueda a aquellas porciones de texto que tienen más posibilidades de contener información relevante. En el caso de los documentos de requerimientos, se utiliza el concepto de crosscutting concerns para filtrar el contenido de los casos de uso a aquella funcionalidad que está relacionada con atributos de calidad. En el caso de los documentos de arquitectura, se filtran las descripciones de las decisiones de diseño tomadas, las cuales suelen tener un objetivo de calidad concreto en la materialización del producto.

El enfoque se materializó en una herramienta prototípico que opera en tres etapas. En primer

lugar, se utiliza una herramienta denominada REAssistant para procesar casos de uso textuales y descubrir crosscutting concerns candidatos que están ocultos en estas especificaciones. En segundo lugar, se utilizan diversos módulos de NLP que extraen información léxica y sintáctica (palabras y partes del discurso) a partir del texto contenido en los documentos de la arquitectura. Esta información es luego utilizada por un sistema basado en reglas para identificar oraciones que describan decisiones de diseño. Las reglas definidas están agrupadas por atributo de calidad, por ejemplo, para identificar decisiones de diseño asociadas a disponibilidad. En este caso, se utilizan consultas para detectar tácticas como Ping/echo, Exception o Heartbeat, entre otras.

Por último, los *crosscutting concerns* y las decisiones de diseño previamente descubiertos se utilizan como entrada de una técnica de recuperación de información denominada LSA (Latent Semantic Analysis). Esta técnica es una aproximación matemática para el descubrimiento de similitud entre documentos, fragmentos de documentos y las palabras dentro de los mismos [?]. Para determinar si existe una traza, LSA genera un espacio vectorial donde cada vector representa a un crosscutting concern o una decisión de diseño. La existencia de la traza entre el par dado por un crosscutting concerns y una decisión de diseño está determinada por el coseno del ángulo comprendido entre ambos.

La herramienta y sus componentes fueron evaluados experimentalmente en dos partes utilizando tres casos de estudio. Primero se evaluó el desempeño de la recuperación de decisiones de diseño y luego se procedió a analizar las trazas identificadas por la herramienta.

Con respecto al buscador de decisiones de diseño, la hipótesis experimental fue que mediante el uso de un sistema de reglas y técnicas semánticas de procesamiento de texto la herramienta es capaz de detectar gran parte de las decisiones de diseño en el SAD correctamente. La herramienta desarrollada obtuvo un recall del 90% y una precisión entre 75%-90% en los tres casos de estudio. Esto significa que la herramienta es capaz de detectar la mayoría de decisiones de diseño contenidas en un SAD cometiendo pocos errores en el proceso. Asimismo, otra observación interesante fue que el buscador se desempeñó mejor al analizar SADs de gran tamaño.

Con respecto a la detección de trazas, la hipótesis experimental fue que la herramienta es capaz de detectar la mayoría de las relaciones de trazabilidad entre los documentos al nivel de oraciones sin cometer muchos errores en el proceso. Se realizaron diversos experimentos, estudiando las variaciones de precisión y recall obtenidos por la herramienta bajo diferentes parametrizaciones del algoritmo de LSA. Los resultados obtenidos permitieron establecer que ciertas configuraciones de LSA fueron capaces de identificar la mayoría de las trazas entre los documentos. El análisis de variaciones de las métricas reveló que para valores de recall del 90%, la herramienta obtuvo valores de entre 20-50% de precisión. Por último, se realizó un análisis de esfuerzo para identificar las trazas utilizando la métrica REI. En este sentido, se pudo establecer que mediante el uso de la herramienta se reducen las tareas de los analistas en aproximadamente un 93%. Los resultados obtenidos fueron prometedores, observando un incremento notable en la calidad de las trazas recuperadas al filtrar los documentos como así también reduciendo significativamente el esfuerzo con respecto a un proceso enteramente manual.

6.1. Ventajas y desventajas

Luego de desarrollar la herramienta TRAS y evaluarla sobre los distintos casos de estudio, se identificaron diversas ventajas y desventajas de la técnica propuesta. A continuación, se listan las ventajas encontradas según diferentes criterios:

- Recall: Desarrollo de una herramienta que permite analizar la trazabilidad entre requerimientos y arquitectura al nivel de oración, detectando la mayoría de las trazas existentes.
- Detección de decisiones de diseño: La técnica propuesta para recuperar decisiones de diseño utilizando información léxica y sintáctica (palabras y partes del discurso) permite filtrar los documentos de arquitectura mediante reglas de tácticas y estilos arquitectónicos con muy buenos resultados.

- Automatización: Si bien el analista puede supervisar cada una de las tres etapas de la herramienta, esto no es necesario. La interacción con el analista es mínima, y el usuario puede ejecutar la herramienta una vez provistos los documentos sin necesidad de agregar o clasificar información durante el proceso. Este nivel de automatización y simpleza es deseable ya que facilita la tarea del analista, especialmente cuando se cuenta con un gran número de documentos.
- Flexibilidad: La técnica propuesta puede ser fácilmente adaptada para recuperar trazas a partir de otros documentos textuales, como código fuente, minutos, notas, entrevistas, etc. Es frecuente que, durante la elicitation de requerimientos o la creacion de la arquitectura, se produzcan una gran variedad de documentos complementarios en lenguaje natural que pueden ser importantes para el análisis de trazabilidad. Con solo algunas modificaciones, se podría utilizar esta técnica para recuperar trazas sobre otros documentos textuales.

A continuación se listan algunas de las desventajas encontradas según distintos criterios.

- Recuperación de decisiones de diseño mediante un sistema de reglas: La técnica propuesta se basa en identificar decisiones de diseño a partir de un sistema de reglas el cual debe ser definido a priori. Se necesitan ciertos conocimientos del dominio para generar un conjunto de reglas que permita obtener un desempeño aceptable de la herramienta.
- Limitaciones del lenguaje: Al procesar los documentos de requerimientos y arquitectura, estos pueden presentar errores ortográficos, gramaticales, o tener poca información relevante para los análisis. Esto repercute negativamente en las técnicas utilizadas para recuperar crosscutting concerns y decisiones de diseño, impidiendo en algunos casos identificar las oraciones afectadas, y por lo tanto, las trazas asociadas.
- Aprendizaje automático: La herramienta no prevé la interacción con un analista que pueda proveer cierta clase de feedback en base a los resultados obtenidos. Este tipo de información podría ser de ayuda para mejorar la recuperación de trazas.
- Gran número de falsos positivos: El SAD suele contener un gran número de decisiones de diseño, repercutiendo en el número de posibles trazas a analizar. Debido a que el análisis de trazabilidad se basa en LSA (y a su vez este se basa en el contenido textual), puede suceder que una traza válida sea descrita con diferentes palabras y la herramienta no la detecte. Del mismo modo puede ser que se utilicen las mismas palabras para referenciar dos artefactos distintos y estos sean presentados como una traza candidata por la herramienta.

6.2. Trabajos futuros

La técnica propuesta en este trabajo se basa en la combinación de técnicas de NLP, filtrado y LSA. Esta es una idea novedosa que no ha sido contemplada anteriormente en la literatura. Por esta razón, existen diferentes líneas de trabajo que se derivan de esta tesis y podrían ser profundizadas más adelante. Entre estas líneas de trabajo futuro, las más relevantes son:

Explorar nuevas líneas de investigación:

- Soportar otras fuentes de información en los documentos: La técnica propuesta podría ser extendida para el tratamiento de otros aspectos de los documentos, tanto de requerimientos como de arquitectura. Por ejemplo, se podría recuperar información a partir de gráficos o tablas presentes en los documentos. Existen varios trabajos interesantes en esta área [?, ?, ?]. Se cree que estas nuevas fuentes de información permitirán mejorar la recuperación de decisiones de diseño.
- Aprendizaje automático: sería de utilidad el desarrollo de alguna técnica de aprendizaje automatico (Machine Learning) para que la herramienta pueda aprender las reglas para recuperar decisiones de diseño automáticamente.

Mejoras en el enfoque presentado:

- Realizar más experimentos: Analizar más casos de estudio y llevar a cabo más experimentos. Contar con nuevos casos de estudio permitirá entender mejor el desempeño de la herramienta y el conjunto actual de reglas utilizadas para recuperar decisiones de diseño.
- Variar la técnica de trazabilidad: Actualmente la herramienta solo utiliza LSA para recuperar trazas. La herramienta podría adaptarse a nuevas técnicas como métodos probabilísticos o VSM y permitir al analista seleccionar que técnica utilizar.
- Realizar más comparaciones: Sería interesante evaluar el desempeño de los analistas cuando utilizan TRAS para recuperar la trazabilidad. Este tipo de estudio permitirá establecer si la salida de la herramienta es útil cuando se combina con el criterio de los analistas y si los analistas se comportan mejor que con otras herramientas de recuperación de trazabilidad.

Bibliografía

Apéndice A

Reglas para detectar Decisiones de Diseño

A.1. Availability

```

DECLARE availability;
WORDLIST availabilityKeywordList = 'tactics/availability/keywords.txt';
WORDLIST availabilityLemmaList = 'tactics/availability/lemma.txt';
WORDLIST availabilityStemList = 'tactics/availability/stem.txt';
Document{-> MARKFAST(availability, availabilityKeywordList,true)};
Token{INLIST(availabilityStemList, Token.stem) ->
    MARK(availability)};
Token{INLIST(availabilityLemmaList, Token.lemma) ->
    MARK(availability)};
Token.lemma == 'client' % Token.lemma == 'server' {->
    MARK(availability)};
Token{AND(FEATURE('lemma','voting'),FEATURE('pos','NN')) ->
    MARK(availability)};
Sentence{CONTAINS(availability) ->
    CREATE(DesignDecision, 'kind' = 'Availability',
        'typex' = 'availability') };
availability { -> UNMARK (availability)};

DECLARE faultDetection;
WORDLIST faultDetList = 'tactics/availability/faultDetection/keywords.txt';
Document{-> MARKFAST(faultDetection, faultDetList,true)};
Sentence{CONTAINS(faultDetection) ->
    CREATE(DesignDecision, 'kind' = 'Availability', 'typex' = 'fault detection') };
faultDetection { -> UNMARK (faultDetection)};

DECLARE faultPrevention;
WORDLIST faultPreventionList = 'tactics/availability/faultPrevention/keywords.txt';
Document{-> MARKFAST(faultPrevention, faultPreventionList,true)};
Sentence{CONTAINS(faultPrevention) ->
    CREATE(DesignDecision, 'kind' = 'Availability', 'typex' = 'fault prevention') };
faultPrevention { -> UNMARK (faultPrevention)};

DECLARE recoverFromFaults;
WORDLIST recoverFromFaultsList = 'tactics/availability/recoverFromFaults/keywords.txt';
Document{-> MARKFAST(recoverFromFaults, recoverFromFaultsList,true)};
Sentence{CONTAINS(recoverFromFaults) ->
    CREATE(DesignDecision, 'kind' = 'Availability', 'typex' = 'recover from faults') };
recoverFromFaults { -> UNMARK (recoverFromFaults)};

```

A.2. Security

```

DECLARE security;
WORDLIST securityKeywordList = 'tactics/security/keywords.txt';
WORDLIST securityLemmaList = 'tactics/security/lemma.txt';
WORDLIST securityStemList = 'tactics/security/stem.txt';
Document{-> MARKFAST(security, securityKeywordList,true)};
Token{INLIST(securityStemList, Token.stem) -> MARK(security)};
Token{INLIST(securityLemmaList, Token.lemma) -> MARK(security)};
Token.lemma == 'digital' % Token.lemma == 'certificate' {->MARK(security)};
Token.lemma == 'authenticate' % Token.lemma == 'user' {->MARK(security)};
Token.lemma == 'throw' % Token.lemma == 'exception' {->MARK(security)};
Sentence{CONTAINS(security) ->

```

```

CREATE(DesignDecision, 'kind' = 'Security', 'typex' = 'security') };
security { -> UNMARK (security)};

DECLARE detectAttacks;
WORDLIST detectAttacksList = 'tactics/security/detectAttacks/keywords.txt';
Document{-> MARKFAST(detectAttacks, detectAttacksList,true)};
Sentence{CONTAINS(detectAttacks) ->
CREATE(DesignDecision, 'kind' = 'Security', 'typex" = "detect attacks") };
detectAttacks { -> UNMARK (detectAttacks)};

DECLARE reactToAttacks;
WORDLIST reactToAttacksList = 'tactics/security/reactToAttacks/keywords.txt';
Document{-> MARKFAST(reactToAttacks, reactToAttacksList,true)};
Sentence{CONTAINS(reactToAttacks) ->
CREATE(DesignDecision, "kind" = "Security", "typex" = "react to attacks") };
reactToAttacks { -> UNMARK (reactToAttacks)};

DECLARE recoverFromAttacks;
WORDLIST recoverFromAttacksList = 'tactics/security/recoverFromAttacks/keywords.txt';
Document{-> MARKFAST(recoverFromAttacks, recoverFromAttacksList,true)};
Sentence{CONTAINS(recoverFromAttacks) ->
CREATE(DesignDecision, "kind" = "Security", "typex" = "recover from faults") };
recoverFromAttacks { -> UNMARK (recoverFromAttacks)};

DECLARE resistAttacks;
WORDLIST resistAttacksList = 'tactics/security/resistAttacks/keywords.txt';
Document{-> MARKFAST(resistAttacks, resistAttacksList,true)};
Sentence{CONTAINS(resistAttacks) ->
CREATE(DesignDecision, "kind" = "Security", "typex" = "resist attacks") };
resistAttacks { -> UNMARK (resistAttacks)};

```

A.3. Modifiability

```

DECLARE modifiability;
WORDLIST modifiabilityKeywordList = 'tactics/modifiability/keywords.txt';
WORDLIST modifiabilityLemmaList = 'tactics/modifiability/lemma.txt';
WORDLIST modifiabilityStemList = 'tactics/modifiability/stem.txt';
Document{-> MARKFAST(modifiability, modifiabilityKeywordList,true)};
Token{INLIST(modifiabilityStemList, Token.stem) -> MARK(modifiability)};
Token{INLIST(modifiabilityLemmaList, Token.lemma) -> MARK(modifiability)};
Token.lemma == "model" % Token.lemma == "view"
    % Token.lemma == "controller" {-> MARK(modifiability)};
Token.lemma == "client" % Token.lemma == "server" {->MARK(modifiability)};
Token.lemma == "publish" % Token.lemma == "subscribe" {->MARK(modifiability)};
Sentence{CONTAINS(modifiability) ->
CREATE(DesignDecision, "kind" = "Modifiability", "typex" = "modifiability") };
modifiability { -> UNMARK (modifiability)};

DECLARE deferBinding;
WORDLIST deferBindingList = 'tactics/modifiability/deferBinding/keywords.txt';
Document{-> MARKFAST(deferBinding, deferBindingList,true)};
Sentence{CONTAINS(deferBinding) ->
CREATE(DesignDecision, "kind" = "Modifiability", "typex" = "defer binding") };
deferBinding { -> UNMARK (deferBinding)};

DECLARE increaseCohesion;
WORDLIST increaseCohesionList = 'tactics/modifiability/increaseCohesion/keywords.txt';
Document{-> MARKFAST(increaseCohesion, increaseCohesionList,true)};
Sentence{CONTAINS(increaseCohesion) ->
CREATE(DesignDecision, "kind" = "Modifiability", "typex" = "increaseCohesion") };
increaseCohesion { -> UNMARK (increaseCohesion)};

DECLARE reduceCoupling;
WORDLIST reduceCouplingList = 'tactics/modifiability/reduceCoupling/keywords.txt';
Document{-> MARKFAST(reduceCoupling, reduceCouplingList,true)};
Sentence{CONTAINS(reduceCoupling) ->
CREATE(DesignDecision, "kind" = "Modifiability", "typex" = "reduce coupling") };
reduceCoupling { -> UNMARK (reduceCoupling)};

DECLARE reduceSizeOfAModule;
WORDLIST reduceSizeOfAModuleList = 'tactics/modifiability/reduceSizeOfAModule/keywords.txt';
Document{-> MARKFAST(reduceSizeOfAModule, reduceSizeOfAModuleList,true)};
Sentence{CONTAINS(reduceSizeOfAModule) ->
CREATE(DesignDecision, "kind" = "Modifiability", "typex" = "reduce size of a module") };
reduceSizeOfAModule { -> UNMARK (reduceSizeOfAModule)};

```

A.4. Performance

```
DECLARE performance;
WORDLIST performanceKeywordList = 'tactics/performance/keywords.txt';
WORDLIST performanceLemmaList = 'tactics/performance/lemma.txt';
WORDLIST performanceStemList = 'tactics/performance/stem.txt';
Document{-> MARKFAST(performance, performanceKeywordList,true)};
Token{INLIST(performanceStemList, Token.stem) -> MARK(performance)};
Token{INLIST(performanceLemmaList, Token.lemma) -> MARK(performance)};
Sentence{CONTAINS(performance) ->
    CREATE(DesignDecision, "kind" = "Performance", "typex" = "performance") };
performance { ->UNMARK(performance)};

DECLARE controlResDemand;
WORDLIST controlResDemandList = 'tactics/performance/controlResDemand/keywords.txt';
Document{-> MARKFAST(controlResDemand, controlResDemandList,true)};
Sentence{CONTAINS(controlResDemand) ->
    CREATE(DesignDecision, "kind" = "Performance", "typex" = "control resource demand") };
controlResDemand { -> UNMARK (controlResDemand)};

DECLARE manageResources;
WORDLIST manageResourcesList = 'tactics/performance/manageResources/keywords.txt';
Document{-> MARKFAST(manageResources, manageResourcesList,true)};
Sentence{CONTAINS(reduceSizeOfAModule) ->
    CREATE(DesignDecision, "kind" = "Performance", "typex" = "manage resources") };
manageResources { -> UNMARK (manageResources)};
```

A.5. Interoperability

```
DECLARE interoperability;
WORDLIST interoperabilityKeywordList = 'tactics/interoperability/keywords.txt';
WORDLIST interoperabilityLemmaList = 'tactics/interoperability/lemma.txt';
WORDLIST interoperabilityStemList = 'tactics/interoperability/stem.txt';
Document{-> MARKFAST(interoperability, interoperabilityKeywordList,true)};
Token{INLIST(interoperabilityStemList, Token.stem) -> MARK(interoperability)};
Token{INLIST(interoperabilityLemmaList, Token.lemma) -> MARK(interoperability)};
Sentence{CONTAINS(interoperability) ->
    CREATE(DesignDecision, "kind" = "Interoperability", "typex" = "interoperability") };
interoperability { ->UNMARK(interoperability)};

DECLARE locate;
WORDLIST locateList = 'tactics/interoperability/locate/keywords.txt';
Document{-> MARKFAST(locate, locateList,true)};
Sentence{CONTAINS(locate) ->
    CREATE(DesignDecision, "kind" = "Interoperability", "typex" = "locate") };
locate { -> UNMARK (locate)};

DECLARE mngInterfaces;
WORDLIST mngInterfacesList = 'tactics/interoperability/mngInterfaces/keywords.txt';
Document{-> MARKFAST(mngInterfaces, mngInterfacesList,true)};
Sentence{CONTAINS(mngInterfaces) ->
    CREATE(DesignDecision, "kind" = "Interoperability", "typex" = "manage interfaces") };
mngInterfaces { -> UNMARK (mngInterfaces)};
```

A.6. Testability

```
DECLARE testability;
WORDLIST testabilityKeywordList = 'tactics/testability/keywords.txt';
WORDLIST testabilityLemmaList = 'tactics/testability/lemma.txt';
WORDLIST testabilityStemList = 'tactics/testability/stem.txt';
Document{-> MARKFAST(testability, testabilityKeywordList,true)};
Token{INLIST(testabilityStemList, Token.stem) -> MARK(testability)};
Token{INLIST(testabilityLemmaList, Token.lemma) -> MARK(testability)};
Sentence{CONTAINS(testability) ->
    CREATE(DesignDecision, "kind" = "Testability", "typex" = "testability") };
testability { ->UNMARK(testability)};

DECLARE controlSystemState;
WORDLIST controlSystemStateList = 'tactics/testability/controlSystemState/keywords.txt';
Document{-> MARKFAST(controlSystemState, controlSystemStateList,true)};
Sentence{CONTAINS(controlSystemState) ->
    CREATE(DesignDecision, "kind" = "Testability",
           "typex" = "control and observe system state") };
```

```

controlSystemState { -> UNMARK (controlSystemState)};

DECLARE limitComplexity;
WORDLIST limitComplexityList = 'tactics/testability/limitComplexity/keywords.txt';
Document{-> MARKFAST(limitComplexity, limitComplexityList,true)};
Sentence{CONTAINS(limitComplexity) ->
    CREATE(DesignDecision, 'kind' = 'Testability', 'typex' = 'limit complexity') };
limitComplexity { -> UNMARK (limitComplexity)};

```

A.7. Usability

```

DECLARE usability;
WORDLIST usabilityKeywordList = 'tactics/usability/keywords.txt';
WORDLIST usabilityLemmaList = 'tactics/usability/lemma.txt';
WORDLIST usabilityStemList = 'tactics/usability/stem.txt';
Document{-> MARKFAST(usability, usabilityKeywordList,true)};
Token{INLIST(usabilityStemList, Token.stem) -> MARK(usability)};
Token{INLIST(usabilityLemmaList, Token.lemma) -> MARK(usability)};
Sentence{CONTAINS(usability) ->
    CREATE(DesignDecision, 'kind' = 'Usability', 'typex' = 'usability') };
usability { ->UNMARK(usability)};

DECLARE supSysIni;
WORDLIST supSysIniList = 'tactics/usability/supSysIni/keywords.txt';
Document{-> MARKFAST(supSysIni, supSysIniList,true)};
Sentence{CONTAINS(supSysIni) ->
    CREATE(DesignDecision, 'kind' = 'Usability',
          'typex' = 'support system initiative') };
supSysIni { -> UNMARK (supSysIni)};

DECLARE supUserIni;
WORDLIST supUserIniList = 'tactics/usability/supUserIni/keywords.txt';
Document{-> MARKFAST(supUserIni, supUserIniList,true)};
Sentence{CONTAINS(supUserIni) ->
    CREATE(DesignDecision, 'kind' = 'Usability', 'typex' = 'support user initiative') };
supUserIni { -> UNMARK (supUserIni)};

```