**Identificación y priorización de Quality Attributes en requerimientos**

Tesis entregada para el grado de

**Ingeniero de Sistemas**

en la Facultad de Ciencias Exactas



Por

**Francisco Bertoni**

**Sebastián Villanueva**

Bajo la supervisión de la

Dra. Claudia Marcos

Dr. Andrés Díaz Pace

**Universidad Nacional del Centro**

**de la Provincia de Buenos Aires**

Tandil, Argentina

Agosto 2010

# INDICE

Contenido

[INDICE 2](#_Toc269239357)

[Capítulo I - Introducción 3](#_Toc269239358)

[Capítulo II - Dominio 4](#_Toc269239359)

[II.1 Arquitectura de Software 4](#_Toc269239360)

[II.2 Atributos de calidad 5](#_Toc269239361)

[II.2.1 Modificabilidad 6](#_Toc269239362)

[II.2.2 Disponibilidad 6](#_Toc269239363)

[II.2.3 Performance 6](#_Toc269239364)

[II.3 Escenarios de calidad 6](#_Toc269239365)

[II.4 Aspectos tempranos 7](#_Toc269239366)

[II.5 Especificación de requerimientos 8](#_Toc269239367)

[II.5.1 Casos de uso 8](#_Toc269239368)

[Capítulo III Trabajos Relacionados 11](#_Toc269239369)

[Capítulo IV Ontología 12](#_Toc269239370)

[IV.1 Componentes 12](#_Toc269239371)

[IV.2 Lenguajes de implementación de ontologías: OWL 13](#_Toc269239372)

[IV.3 Ontología para atributos de calidad 13](#_Toc269239373)

[Capítulo V Identificación de atributos de calidad 16](#_Toc269239374)

[V.1 Desarrollo de la técnica de identificación de atributos de calidad propuesta 16](#_Toc269239375)

[V.2 Generación de Tokens 16](#_Toc269239376)

[V.2.1 Procesamiento de la Entrada 17](#_Toc269239377)

[V.2.2 Filtrado de Tokens 18](#_Toc269239378)

[V.3 Análisis de los tokens 22](#_Toc269239379)

[5.3.1 Calcular Quality Attribute Theme 22](#_Toc269239380)

[Bibliografía (21) 36](#_Toc269239381)

# Capítulo I - Introducción

*Acá iría el capítulo 1*

# Capítulo II - Dominio

En este capítulo se presentará el dominio junto con los conceptos que se utilizarán a lo largo del presente trabajo. De esta forma, se introducirán los conceptos de “Arquitectura de software”, “Atributo de calidad”, “Escenario” y “Early Aspect” (Aspecto Temprano). Además, se introducirán algunas técnicas de especificaciones de requerimientos, centrándose en el concepto de “Caso de Usos” y su relación con los aspectos tempranos y atributos de calidad.

## II.1 Arquitectura de Software

En la actualidad no hay ninguna definición de Arquitectura de Software que esté unánimemente respaldada por la totalidad de la comunidad. El número de definiciones circulantes, que aumenta con el tiempo, alcanza un orden de tres dígitos, amenazando llegar a cuatro. De hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas, como la colección que se encuentra en el SEI (Software Engeenerning Institute), a la que cada quien puede agregar la suya (1). En general, las definiciones entremezclan el trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño (su lugar en el ciclo de vida), la configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción y la caracterización de la disciplina que se ocupa de uno de esos dos asuntos, o de ambos. (2).

Una definición reconocida es la de Clements (3):

*La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema.*

Sin embargo se debe mencionar que, algunos años después, el mismo Clements, junto con Bass, Kazman y Northrop, definen a la arquitectura de software de la siguiente forma (4):

*La arquitectura de software de un programa o sistema computarizado es la estructura o estructuras del sistema, que involucra elementos de software, las propiedades externas visibles de esos elementos.*

De esta manera, a despecho de la abundancia de definiciones del campo de la Arquitectura de Software, existe en general acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, estructura consistente en elementos, junto con sus propiedades y relaciones entre ellos. Estas cuestiones estructurales se vinculan con el diseño, pues la Arquitectura de Software es, después de todo, una forma de diseño de software que se manifiesta tempranamente en el proceso de creación de un sistema; pero este diseño ocurre a un nivel más abstracto que el de los algoritmos y las estructuras de datos. En el que muchos consideran un ensayo seminal de la disciplina, Mary Shaw y David Garlan (5) sugieren que dichas cuestiones estructurales incluyen organización a grandes rasgos y estructura global de control; protocolos para la comunicación, la sincronización y el acceso a datos; la asignación de funcionalidad a elementos del diseño; la distribución física; la composición de los elementos de diseño; escalabilidad y rendimiento; y selección entre alternativas de diseño.

## II.2 Atributos de calidad

La calidad de software se define como el grado en el cual éste posee una combinación deseada de atributos, tales como: Performance, Disponibilidad, Portabilidad, Seguridad, Usabilidad, etc. (6). Estos atributos son requerimientos adicionales del sistema (7) que hacen referencia a características o restricciones que éste debe satisfacer, y complementan los requerimientos funcionales del mismo. Estas características o atributos se conocen con el nombre de “atributos de calidad” (8).

En términos generales, Bass y otros autores (4) establece una clasificación de los atributos de calidad en dos categorías:

* Observables en tiempo de ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución (disponibilidad, performance, seguridad, etc.)
* No observables en tiempo de ejecución: aquellos atributos que se establecen durante el desarrollo del sistema (modificabilidad, reusabilidad, portabilidad, escalabilidad, etc.).

La “funcionalidad” es la habilidad de un sistema para realizar el trabajo. Los atributos de calidad de un sistema son independientes de la funcionalidad del mismo (4). Si esto no fuera así, una funcionalidad seleccionada dictaría los niveles de seguridad, modificabilidad, disponibilidad, etc. de un sistema. Esto no significa que cualquier nivel de cualquier atributo de calidad puede ser alcanzado para cualquier funcionalidad. Manipular gráficos complejos o realizar complejos cálculos matemáticos, por ejemplo, ciertamente tienen un impacto negativo sobre la performance. Sin embargo, lo que sí es posible es, bajo determinada funcionalidad, tomar ciertas decisiones que determinen un nivel relativo de calidad.

A pesar de estar muy relacionada con los atributos de calidad, en general es en la funcionalidad en donde se pone el foco de atención durante las distintas etapas de desarrollo, dejando a estos últimos, con mucha suerte, en un segundo plano. De esta manera la mayoría de los enfoques lidian con los atributos de calidad separadamente de la funcionalidad, dejando su integración para las etapas finales del ciclo de desarrollo (8). Sin embargo, los atributos de calidad deben ser considerados en todas las etapas del proceso. Ningún atributo de calidad es enteramente dependiente de la etapa de diseño o de la implementación, o de cualquier otra (4).

Particularmente la arquitectura es fundamental para la realización de varios atributos de calidad de un sistema. Es la primera etapa del ciclo de desarrollo en donde las estructuras deben ser diseñadas satisfaciendo a los atributos requeridos, y en donde estos pueden ser incluso testeados. Es por esta razón que el tener un conocimiento exacto de los atributos de calidad de un sistema en las etapas tempranas de desarrollo es fundamental para lograr un sistema exitoso.

Los sistemas no son generalmente rediseñados porque sean funcionalmente deficientes - de hecho los modulo reemplazados son, en general, funcionalmente idénticos - sino porque son difíciles de mantener, difíciles de escalar, demasiado lentos o se han visto comprometidos por hackers.

En este trabajo se denominará como un QAR (Quality Attribute Requirement) a un requerimiento de un atributo de calidad. En la bibliografía los QARs también son conocidos como “Non functional requirements”, o NFRs. Sin embargo se adopta como convención la primera denominación.

Nota para Claudia: ¿Estaría bien hacer una breve reseña de cada atributo de calidad o de algunos?

### II.2.1 Modificabilidad

### II.2.2 Disponibilidad

### II.2.3 Performance

## II.3 Escenarios de calidad

Los escenarios de calidad son una forma de representar, de una manera formal, los QARs de un sistema.

Un escenario consiste de seis partes (4):

1. Fuente de estímulo (Concrete Source). Alguna entidad (un humano, un sistema de computación u otro actor) que genera un estímulo.
2. Estímulo (Concrete Stimulus). Una condición que necesita ser considerada cuando arriba al sistema.
3. Ambiente (Concrete Enviroment). El estímulo ocurre bajo ciertas condiciones. El sistema puede estar sobrecargado o puede estar ejecutándose normalmente cuando el estímulo ocurre, o alguna otra condición puede ser verdadera.
4. Artefacto (Concrete Artifact). Algún artefacto es estimulado. Este puede ser el sistema completo o alguna parte de él.
5. Respuesta (Concrete Response). La respuesta es una actividad llevada a cabo luego del arribo del estímulo.
6. Medida de respuesta (Concrete Response Measure). Cuando la respuesta ocurre, ésta debería ser medida de alguna manera así el requerimiento puede ser testeado.

Se distinguen como escenarios generales de calidad (escenarios generales) a aquellos que son independientes del sistema y pueden, potencialmente, pertenecer a cualquier sistema; de aquellos denominados escenarios concretos de calidad (escenarios concretos), que son específicos de un sistema en particular que están bajo consideración. La caracterización de atributos de calidad puede ser hecha como una colección de escenarios generales, sin embargo, para traducir la caracterización de los atributos a requerimientos para un sistema específico, los escenarios generales deben transformarse en específicos.

Una colección de escenarios concretos puede ser usada como los QARs de un sistema. Cada escenario es suficientemente concreto para ser significativo para un arquitecto, y los detalles de la respuesta son suficientemente significativos para soportar tests convalidando que los requerimientos hayan sido cumplimentados.

En general, las definiciones provistas por un atributo de calidad no son operacionales. No tiene significado decir que un sistema será, por ejemplo, modificable, ya que todos los sistemas son modificables respecto a un conjunto de cambios y no modificable con respecto a otro conjunto. Los escenarios brindan una solución a esta cuestión al hacer posible su testeo a un nivel arquitectural.

También son de mucha utilidad a la hora de relacionar un aspecto del sistema a un atributo de calidad específico. Muchas veces es ambiguo si un aspecto pertenece a un atributo o a otro. (Una falla del sistema, por ejemplo, ¿pertenece a seguridad, disponibilidad o usabilidad?). Los escenarios son una solución directa a esta cuestión, relacionando una característica con atributo de calidad específico.

## II.4 Aspectos tempranos

Se define un “concern” como “cualquier asunto de interés en un sistema de software” (9). De esta manera el desarrollo de software tiene que tratar con un gran número de concerns. Algunos de estos están relacionados al producto (el software) que debe ser creado, como la funcionalidad y la performance. Otros concerns están relacionados con el proceso de desarrollo en sí mismo, como los tiempos y costos del desarrollo.

Mientras que (por medio de técnicas convencionales de modularización u orientación a objetos) un tipo de concern puede ser fácilmente encapsulado dentro de artefactos como módulos, clases y operaciones a nivel de diseño o implementación, esto mismo no es posible para otro tipo de concern. Estos cortan transversalmente (crosscut) el diseño o implementación de unos cuantos o incluso muchos artefactos y por lo tanto son llamados crosscutting concerns (9).

De esta manera, se define a un “aspecto temprano” como un concern que atraviesa el diseño de un sistema (10), y se manifiesta generalmente en las especificaciones de requerimientos u otros documentos preliminares producidos en el análisis de requerimientos. Muchos aspectos tempranos se corresponden con QARs de alto nivel como seguridad, performance, portabilidad y usabilidad (11). En consecuencia, el descubrimiento de aspectos tempranos puede proporcionar pistas para identificar QARs.

## II.5 Especificación de requerimientos

Tanto los requerimientos no funcionales como los requerimientos de atributos de calidad son capturados en un documento denominado “Especificación de Requerimientos de Software” (Software Requirements Specification), o SRS.

De esta manera el SRS es una completa descripción del comportamiento de un sistema a ser desarrollado. Principalmente está conformado por un conjunto de casos de uso, que especifican la funcionalidad del sistema y como este responde a distintos estímulos externos. Idealmente el SRS también incluye un una especificación de los requerimientos de atributos de calidad, como así también un prototipo del sistema a ser desarrollado. Sin embargo, esto último rara vez se cumple.

En un estudio realizado (11) en base a 15 especificaciones requerimientos se encontró una importante escasez de menciones a QARs. Esto pude indicar que los desarrolladores fallan a la hora de analizar la importancia de los QARs, o que falsamente asumen que éstos son ya entendidos y aceptados por todos los stakeholders.

En general los documentos son organizados en cuanto a la funcionalidad, poniendo muy poco, y casi nulo, énfasis en los QARs. De esta manera éstos quedan atravesados o esparcidos en varios documentos. Como ya se mencionó antes, esto conlleva a que los atributos de calidad no sean tenidos en cuenta en las primeras etapas del desarrollo, originando problemas potenciales sin detectar, diseños arquitectónicos que no cumplen con los requerimientos solicitados y productos de software que no satisfacen las necesidades del cliente.

Más aún la identificación de los QARs es una tarea poco sencilla y muy complicada, que puede consumir una gran cantidad de tiempo y esfuerzo. En general ésta es realizada “ad-hoc” entre varios documentos que describen al sistema. Como los QARs pueden estar esparcidos entre muchos de estos documentos, se corre el peligro de fácilmente malinterpretarlos o, directamente, ignorarlos por completo.

### II.5.1 Casos de uso

En ingeniería del software, un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final. En ocasiones, se utiliza a usuarios sin experiencia junto a los analistas para el desarrollo de casos de uso.

Los casos de uso se convirtieron en una de las prácticas más comunes para la captura de requisitos funcionales, especialmente con el desarrollo del paradigma de la programación orientada a objetos. También, como se mencionó anteriormente, representan la espina dorsal de los SRS y es en donde generalmente se pone el mayor énfasis, dejando en un segundo plano la especificación de los requerimientos de los atributos de calidad.

#### II.5.1.1 Especificación de un requerimiento mediante casos de uso (o Aplicación de Casos de Uso)

Los casos de uso evitan típicamente la jerga técnica, prefiriendo la lengua del usuario final o del experto del campo del saber al que se va a aplicar. Los casos del uso son a menudo elaborados en colaboración por los analistas de requerimientos y los clientes. Cada caso de uso se centra en describir cómo alcanzar una única meta o tarea de negocio.

Desde una perspectiva tradicional de la ingeniería de software, un caso de uso describe una característica del sistema. Para la mayoría de proyectos de software, esto significa que quizás a veces es necesario especificar diez o centenares de casos de uso para definir completamente el nuevo sistema. El grado de la formalidad de un proyecto particular del software y de la etapa del proyecto influenciará el nivel del detalle requerido en cada caso de uso. Los casos de uso pretenden ser herramientas simples para describir el comportamiento del software o de los sistemas. Un caso del uso contiene una descripción textual de todas las maneras que los actores previstos podrían trabajar con el software o el sistema. Los casos de uso no describen ninguna funcionalidad interna (oculta al exterior) del sistema, ni explican cómo se implementará. Simplemente muestran los pasos que el actor sigue para realizar una tarea.

Un caso de uso debe:

* tener un nivel apropiado del detalle;
* describir una tarea del negocio que sirva a una meta de negocio;
* ser bastante sencillo como que un desarrollador lo elabore en un único lanzamiento.

#### II.5.1.2 Ventajas

• La técnica de caso de uso tiene éxito en sistemas interactivos, ya que expresa la intención que tiene el actor (su usuario) al hacer uso del sistema.

• Como técnica de extracción de requerimiento permite que el analista se centre en las necesidades del usuario, qué espera éste lograr al utilizar el sistema, evitando que la gente especializada en informática dirija la funcionalidad del nuevo sistema basándose solamente en criterios tecnológicos.

• A su vez, durante la extracción (elicitation en inglés), el analista se concentra en las tareas centrales del usuario describiendo por lo tanto los casos de uso que mayor valor aportan al negocio. Esto facilita luego la priorización del requerimiento.

#### II.5.1.3 Templates de Casos de Uso

No existe un modelo estándar para documentar los casos uso. En la actualidad compiten una serie de esquemas existentes, y se anima a los individuos a utilizar el template o plantilla de casos de uso que mejor se adapte a las necesidades de su proyecto. De esta manera, la estandarización dentro de cada proyecto tiene mayor importancia que los detalles de una plantilla determinada. Existe, sin embargo, un acuerdo considerable entre la comunidad acerca de las principales secciones de los casos de uso, obteniendo cierta similaridad entre la mayoría de las plantillas.

Nota para Claudia: Aca iria la explicación de que vamos a utilizar el template del Rational (que esta en el Anexo I). ¿Iria una explicacion breve de cada sección?. Otro problema es que no encuentro una referencia “oficial”, de IBM o de quien sea, de que ese template es efectivamente el del “Rational Unified Process”

# Capítulo III Trabajos Relacionados

*Acá iría el capítulo 3*

# Capítulo IV Ontología

La literatura de Inteligencia Artificial contiene varias definiciones de ontologías, muchas veces contradictorias entre ellas. En este trabajo se acuerda en definir a una ontología de la siguiente manera:

*“Una ontología es un modelo de datos que describe conceptos (también llamados clases) en un dominio del discurso, propiedades de cada concepto que describen las diversas características y atributos del concepto, y restricciones sobre esas propiedades.”* (12)*.*

Un dominio es un área de temática específica o de conocimiento, tal como medicina, fabricación de herramientas, bienes inmuebles, reparación automovilística, gestión financiera, etc. (13). Las ontologías incluyen definiciones de conceptos básicos del dominio, y las relaciones entre ellos. También codifican el conocimiento de un dominio específico y el conocimiento que extiende de los dominios.

Un dominio específico es la parte del mundo que se quiere modelar. Representa el significado aplicado a los términos usados en la construcción de la ontología. Una ontología es la descripción de los conceptos que forman parte del dominio según un punto de vista. Un sistema sólo conoce lo que puede representar en algún lenguaje, por lo tanto, todo lo que no se exprese en la ontología no será conocido para el sistema que use la ontología.

Un beneficio principal cuando una ontología es creada es: “Compartir el entendimiento común de la estructura de información entre personas y sistemas de software” (14) . Las ontologías se usan para favorecer la comunicación y la comprensión común de la información entre personas, organizaciones y aplicaciones, permitiendo que entre ellas estén de acuerdo en los términos que usan cuando se comunican.

Los beneficios de utilizar ontologías se pueden resumir de la siguiente manera:

* Compartir un entendimiento común de la estructura de la información entre personas o agentes de software.
* Permitir el re uso de conocimiento de un dominio.
* Hacer explícitas presunciones de un dominio.
* Separar el conocimiento del dominio del conocimiento operacional.
* Analizar el conocimiento del dominio.

## IV.1 Componentes

Las ontologías cuentan con los siguientes componentes que sirven para representar el conocimiento de algún dominio (14). Los principales son:

1. Conceptos o Clases Generalmente el foco de cualquier ontología, las clases son las ideas básicas que se intentan formalizar. Por ejemplo, en una ontología de vinos, la clase “Vino” representa todos los vinos. Una clase también puede tener sub-clases que representan conceptos que son mas específicos que las superclases. Por ejemplo, subclases de la clase “Vino” podrían ser “Tinto”, “Blanco”, “Rosado”, etc.
2. Instancias Las instancias son representaciones de objetos determinados de un concepto. Siguiendo el ejemplo anterior, vinos específicos son instancias de la clase “Vino”. Una botella de New Age que se vende en el supermercado es una instancia de la clase “Vino”.
3. Propiedades o Slots Los slots describen propiedades de clases o instancias. En el ejemplo anterior “New Age” es la marca del vino. Por lo tanto, esa instancia posee un slot(propiedad), que poseen todas las instancias de la clase “Vino”, que podría denominarse “marca”, y cuyo valor es la cadena de caracteres “New Age”. El valor de una propiedad puede ser una cadena de caracteres, un número, otra instancia, etc. En el caso de que fuera otra instancia se forma lo que se denomina una relación.

## IV.2 Lenguajes de implementación de ontologías: OWL

OWL (Ontology Web Language) (15) es un lenguaje web para ontologías compatible con la World Wide Web. Este es utilizado cuando la información contenida en los documentos necesitan ser procesados por las aplicaciones, a diferencia de situaciones en que el contenido sólo requiere ser presetnado a los seres humanos. Puede ser utilizado para representar explícitamente el significado de los términos en los vocabularios y las relaciones entre esos términos (justamente una ontología). OWL tiene más facilidades para expresar el sentido y la semántica que otros lenguajes como XML [REF], RDF[REF] y RDF-S[REF], y por lo tanto OWL va más allá de estos lenguajes en su capacidad para representar a la máquina de contenido interpretable en la Web. Este lenguaje es una revisión del lenguaje de ontologías web DAML+OIL (16) que incorpora lecciones aprendidas desde el diseño y aplicaciones de DAML+OIL. La especificación de este lenguaje se encuentra dentro de las recomendaciones de la W3C. OWL se diferencia de otros lenguajes porque OWL es un lenguaje de ontologías Web. Lenguajes anteriores han sido utilizados para desarrollar herramientas y ontologías destinadas a comunidades específicas (especialmente para ciencias y aplicaciones específicas de comercio electrónico). Además no fueron definidos para ser compatibles con la arquitectura de la World Wide Web en general, y la Web Semántica en particular.

## IV.3 Ontología para atributos de calidad

En la figura [fig:Ontologia-de-escenarios] se muestra un diagrama con la ontología definida, que servirá de soporte en la técnica propuesta en capítulos siguientes. Su construcción se basa principalmente en conceptos, definiciones, y relaciones extraídas del libro “Software Architecture in Practice” (4).

El concepto de escenario de calidad fue presentado por Bass, Clements y Kazman, en el libro “Software Architecture in Practice”. Textualmente, la definición es:

“A quality attribute scenario is a quality-attribute-specific requirement.” (4)

Traduciendo, un escenario de calidad es un requerimiento específico de un atributo de calidad. De esta manera, los escenarios son utilizados para caracterizar atributos de calidad de una forma específica.

Es por ello que en la ontología se encuentran los conceptos QualityAttributeScenario y QualityAttribute relacionados por la propiedad isSpecificOf.

Un escenario de calidad está formado por seis partes:

1. Fuente de estímulo (Concrete Source). Alguna entidad (un humano, un sistema de computación u otro actor) que genera una estímulo.
2. Estímulo (Concrete Stimulus). Una condición que necesita ser considerada cuando arriba al sistema.
3. Ambiente (Concrete Enviroment). El estímulo ocurre bajo ciertas condiciones. El sistema puede estar sobrecargado o puede estar ejecutándose normalmente cuando el estímulo ocurre, o alguna otra condición puede ser verdadera.
4. Artefacto (Concrete Artifact). Algún artefacto es estimulado. Este puede ser el sistema completo o alguna parte de él.
5. Respuesta (Concrete Response). La respuesta es una actividad llevada a cabo luego del arribo del estímulo.
6. Medida de respuesta (Concrete Response Measure). Cuando la respuesta ocurre, ésta debería ser medida de alguna manera así el requerimiento puede ser testeado.

Se distinguen como escenarios generales de calidad a aquellos que son independientes del sistema y pueden, potencialmente, pertenecer a cualquier sistema, de aquellos denominados escenarios concretos de calidad, que son específicos de un sistema en particular que esta bajo consideración. Es por esto último que aparece el concepto abstracto Quality-Attribute Scenario, mientras que los conceptos Concrete Quality-Attribute Scenario y General Quality-Attribute Scenario derivan del mismo. A su vez la relación isDerivedFrom constata el hecho de que un escenario concreto deriva de uno general, haciendo las partes del últimos especificas de un sistema bajo consideración.

Una colección de escenarios concretos puede ser usada como los requerimientos de calidad para un sistema. Cada escenario es suficientemente concreto para ser significativo para un arquitecto, y los detalles de la respuesta son suficientemente significativos para soportar tests convalidando que los requerimientos hayan sido cumplimentados.

# Capítulo V Identificación de atributos de calidad

En este capítulo se describirá en detalle el enfoque elegido para la técnica de identificación de atributos de calidad en especificaciones de requerimientos. Esta técnica brindará una asistencia extra al analista, para identificar los atributos de calidad del sistema en etapas tempranas del desarrollo y evitar los problemas suscitados anteriormente.

## V.1 Desarrollo de la técnica de identificación de atributos de calidad propuesta

En la figura [fig:Tareas\_de\_la\_tecnica\_Propuesta] se pueden observar las tareas realizadas en la solución propuesta, junto con los artefactos de entrada y de salida.

En este trabajo se ha definido un proceso por medio del cual es posible identificar el grado de relación que tiene un conjunto de early aspects (aspectos tempranos) y casos de uso con los atributos de calidad del sistema. Este proceso utiliza como entrada una lista de casos de uso y early aspects, y agrega a cada conjunto <early aspect, casos de uso relacionados> información del atributo de calidad dominante para el conjunto en cuestión.

Básicamente, el proceso está constituido por dos etapas. La primera etapa, denominada generación de tokens, realiza un análisis sobre los elementos de entrada (casos de uso y early aspects) para representar la información en un formato interno y, mediante un filtrado, establecer qué información es relevante para la etapa de análisis.

En la segunda etapa, análisis de tokens, se calcula el porcentaje de pertenencia de cada elemento de entrada con los diferentes atributos de calidad del sistema. Posteriormente, esa información es combinada para obtener el atributo de calidad dominante para el conjunto de entrada.

Para soportar dicho proceso se ha implementado una herramienta como una extensión - plugin - del entorno de desarrollo integrado (IDE)Eclipse[REF]. La misma, permite al desarrollador seleccionar como entrada los early aspects que desea analizar, junto a los casos de uso que los relacionan. A partir de esta información, el desarrollador podrá contar con la información acerca de los atributos de calidad identificados, además de tener la posibilidad de visualizar la trazabilidad del proceso completo.

## V.2 Generación de Tokens

Este bloque de tareas tiene como finalidad llevar a cabo un análisis léxico y sintáctico sobre los textos definidos como entrada para la técnica propuesta. Esta entrada es la información generada por la herramienta Aspect Extractor Tool (17), luego del procesamiento con la técnica de “Análisis Semántico” (18). La salida producida por esta herramienta es la lista de early aspects detectados, los casos de uso analizados y las relaciones entre ellos. La especificación textual de los casos de uso respeta el estándar establecido por Rational (19). De igual modo, la especificación de los early aspects sigue el formato definido en Aspect Extract Tool, el cuál es semejante al de los casos de uso. A partir de esta información se confeccionan las listas de tokens, las cuales serán la entrada de la próxima etapa.



Para este análisis, se requiere la realización de dos actividades principales. En primer lugar, un procesamiento de los datos de entrada para representarlos en un formato común. Mientras que en segundo lugar se requiere un filtrado sobre los mismos para identificar qué información es relevante para la próxima etapa. En estas actividades, se logrará determinar un subconjunto del conjunto de palabras que forman la especificación de los casos de uso y la definición de los early aspects, las cuales serán relevantes para la identificación del atributo de calidad involucrado.

### V.2.1 Procesamiento de la Entrada

El objetivo de esta tarea es procesar la información de entrada para establecer la información en una representación interna uniforme. Este procesamiento es realizado a través de la separación de la información de las entidades de entrada (especificaciones de casos de uso y early aspects) en tokens. Se define un token como una unidad básica de texto que puede ser enriquecida con diferentes pares <atributos,valor>, como por ejemplo: <peso:1>, <ocurrencias:4>, etc.. Estos tokens representan al conjunto de palabras de entrada.



El resultado de este procesamiento es una lista de tokens para las palabras obtenidas de los casos de uso y otra lista de tokens para las palabras del aspecto temprano en cuestión. Al realizar la división de la entrada en tokens, se agregan los siguientes atributos para cada uno de ellos:

* Id: identificador único del token.
* Tipo: este atributo representa el tipo de documento del cual se extrajo el token (en este punto será Aspecto Temprano o Caso de Uso).
* Sección: se registra la sección a la cual pertenece el token. Para el caso de los casos de uso, será alguna de las secciones establecidas por el template de Rational (nombre, descripción, flujo básico, actor, etc.) mientras que para el aspecto temprano será alguna de las partes que conforman un aspecto temprano según lo establecido en la herramienta Aspect Extractor Tool (nombre o par <verbo, objeto directo>).

Por ejemplo, considérese la fracción de la especificación de un caso de uso mostrada en la tabla [tab:Fracción-de-caso]. Tomando esto como entrada, el resultado de aplicar el procesamiento descripto anteriormente daría como resultado la lista tokens mostrada en la tabla [tab: Tokens -generados].

### V.2.2 Filtrado de Tokens

Como se mencionó en la sección anterior, el filtrado de los tokens es otra de las tareas que se realiza sobre la información de entrada. En este caso, se ejecutan una serie de filtros sobre los tokens de las listas de casos de uso y de early aspects.



Se considera que un filtro es una unidad de procesamiento que realiza una modificación (enriquece, refina o transforma) sobre los datos de entrada y los copia a la salida para que otro filtro trabaje sobre los mismos datos. Mediante los filtros se pueden realizar transformaciones independientes sobre el flujo de datos.

El patrón de arquitectura Pipes & Filters (20) provee una estructura para procesar flujos de datos. Cada paso de procesamiento se encapsula en un filtro y es independiente del resto. Los datos se transmiten usando los pipes entre filtros adyacentes y mediante la combinación de éstos últimos se pueden conseguir diferentes salidas.

Las ventajas de utilizar este concepto son:

* flexibilidad: permite futuros cambios intercambiando o utilizando filtros nuevos;
* reusabilidad: reutilización de los filtros en diferentes secuencias de procesamiento;
* bajo acoplamiento: los filtros no comparten estados internos y son independientes;
* perfomance: procesamiento paralelo.

En este caso, la entrada para los filtros es una lista de tokens, por lo que el filtro ejecutará acciones sobre los tokens de la lista (modificar la palabra, agregar atributos), para luego devolver la lista de tokens modificada.

Para esta actividad, se han definido una serie de filtros que realizan transformaciones sobre los tokens de las listas. Los mismos fueron definidos de forma tal que puedan ser reutilizados y combinados para darle un formato común a cualquier otro texto que represente información relevante, como por ejemplo, las instancias de la ontología definida.

#### V.2.2.1 Filtro Lower Case

Este filtro es el encargado de pasar todos los caracteres de las palabras de los tokens a minúscula.

Continuando con los tokens generados en la tabla [tab: Tokens -generados], sólo se modifican los caracteres en mayúscula de las palabras de los tokens. En la tabla [tab:Ejemplo-FiltroLower] se observa la lista de tokens luego de la transformación aplicada por este filtro.

#### V.2.2.2 Filtro Stop Words

Se eliminan las denominadas stop-words, que son palabras que, desde el punto de vista no lingüístico, no contienen información relevante. Algunas de las stop words más comunes son los artículos, las preposiciones, etc. (por ejemplo: a, de, del, el, en, ella, los, se).

Por lo tanto, este filtro elimina de la lista de tokens aquellos en los cuales su palabra aparezca en la lista de stop-words dada. El analista/desarrollador tiene la posibilidad de incluir en esta lista las palabras, que para el dominio en el cuál este analizando, no representen información valiosa para el análisis, es decir palabras no relevantes o que no aportan información valiosa al análisis. De igual modo, se puede modificar la lista quitando aquellas palabras que en el domino resulten relevantes para el análisis y el desarrollador desee que sean procesadas.

La tabla [tab:Ejemplo-StopWords] muestra el estado de los tokens de la tabla [tab:Ejemplo-FiltroLower] luego de haberse aplicado este filtro.

#### V.2.2.3 Filtro Stemming

Diferentes palabras pueden tener el mismo significado desde el punto de vista semántico, pero no tener exactamente la misma secuencia de caracteres. Un ejemplo de esto pueden ser las palabras aprender, aprenden y aprendió, que, desde el enfoque en este trabajo, tienen el mismo significado. Por ello se decidió utilizar la técnica stemming [REF], que es el proceso de transformar una palabra en su raíz (stem). Para el ejemplo anterior, todas esas palabras estarían identificadas por su raíz, es decir, “aprend”. Para realizar el stemming se utiliza uno de los algoritmos más populares y conocidos, el algoritmo de Porter [REF].

Por lo tanto, a través de este filtro, se modifican los tokens de la lista llevando la palabra de cada token a su raíz.

Continuando con el ejemplo observado en la tabla [tab:Ejemplo-StopWords], se modificarión las palabras de los tokens, llevando cada una de ellas a su raíz. En la tabla [tab:Ejemplo - Stemming] se muestra el estado de los tokens luego de que se aplique el filtro en cuestión.

#### V.2.2.4 Filtro Ocurrencias

Este filtro elimina tokens duplicados en la lista de entrada y enriquece cada token con el número de ocurrencias. Se agrega el siguiente atributo al token:

* Ocurrencias: número de ocurrencias del token en la lista. Para los tokens obtenidos de los casos de uso se considera un token duplicado en caso de que el token sea la misma palabra y además aparezca en la misma sección. Esta particularidad se debe a que luego los tokens son ponderados y debe respetarse la sección en la cual aparece a la hora de asignarle su peso. Por lo tanto, existirán tokens con la misma palabra pero diferente sección.

Igualmente, para continuar con el ejemplo, solo se tendrá en cuenta la palabra del token para considerarlo repetido. De esta forma, se observa en la tabla [tab:Ejemplo-1.6.-Filtro] el nuevo atributo que se agrega a cada token.

#### V.2.2.5 Filtro Pesos

Este filtro es aplicado para enriquecer los tokens de la lista de tokens de casos de uso, asignando un peso a cada token. Este peso depende de la sección donde aparece la palabra. En la técnica propuesta las palabras tienen distinta importancia según la sección del caso de uso en la que aparezcan. Por ejemplo, se asignará un peso alto a una palabra perteneciente al nombre o descripción del caso de uso, mientras que se asignará un peso más bajo a una palabra perteneciente a un flujo alternativo.

Hemos realizado un estudio riguroso de la importancia que tiene cada una de las componentes del template que conforman los casos de uso, y por defecto se asignan los pesos mostrados en la tabla [tab:Ponderación-de-las] de acuerdo a la importancia de los textos y palabras que conforman las diferentes secciones. Sin embargo, el desarrollador podría modificar este peso dependiendo de las características de la definición de sus casos de uso. La herramienta soporta esta parametrización a través de un archivo de configuración donde se establecen estos pesos.

Considerando los pesos por defecto mostrados en la tabla [tab:Ponderación-de-las], la lista de tokens final quedaría tal como se muestra en la tabla [tab:Ejemplo-1.7.-Filtro].

Para los early aspects no es necesario aplicar el filtro de peso debido a que se considera que todas las palabras dentro del aspecto temprano tienen la misma importancia, mientras que el caso de uso está dividido en secciones, y se considera que hay secciones que deben ser ponderadas sobre otras.

## V.3 Análisis de los tokens

En la etapa de “Análisis de tokens” se encuentran las tareas para identificar los atributos de calidad a partir de una lista de tokens extraídos de los casos de uso y de los early aspects. En este punto la entrada ha sido analizada y ha sido construida la lista de tokens con la palabra de cada token pasada a minúscula, llevada a su raíz, ponderizandola de acuerdo al peso y dejando constancia del número de ocurrencias. De esta manera, a esta etapa llegan dos listas de tokens: una correspondiente a los casos de uso y otra correspondiente a los early aspects.

El objetivo de esta etapa es identificar el Quality Attribute Theme (QAT) correspondiente a esas listas de tokens. Para ello, por cada palabra de cada una de la listas se identifica el grado de asociación de esa palabra con los distintos atributos de calidad. Es decir, identificar qué tan ligada se encuentra esa palabra con los diferentes atributos de calidad y en qué porcentaje o grado. Para ello se desarrolla, por cada token, una tarea que relaciona una palabra con un conjunto de pares <atributo de calidad, porcentaje>, indicando el grado o porcentaje de asociación de esa palabra con los distintos atributos de calidad. Esta tarea, que se denomina “Asociar palabra con atributo de calidad”, arroja los resultados con la ayuda de una ontología.

El mecanismo anterior se repite para cada token y los resultados individuales de cada uno se combinan en una ecuación definida, que tiene en cuenta el peso de cada uno de los tokens y el número de ocurrencias del mismo. Esta ecuación da como resultado un conjunto de atributos de calidad y un porcentaje asociado a cada uno de ellos.

Los cálculos anteriores se aplican a cada una de las listas, la de los casos de uso y la del aspecto temprano. Finalmente estos resultados parciales se combinan, obteniendo los resultados finales.

### 5.3.1 Calcular Quality Attribute Theme

Esta tarea es la que tiene como objetivo asignar un conjunto de atributos de calidad, con un porcentaje cada uno, a un conjunto de casos de uso relacionados con early aspects. Vale la pena recordar que a esta etapa arriban dos listas de tokens: una lista de tokens extraídos de los casos de uso y otra del aspecto temprano. Cada uno de estos tokens posee atributos como peso, ocurrencias, etc.

Como se puede apreciar en la figura [fig:Tareas\_de\_la\_tecnica\_Propuesta] esta tarea utiliza a la tarea “Asociar Palabra con Atributos de Calidad”. El funcionamiento de la misma se explica detalladamente en la subsección [sub:Asociar-palabra-con], pero por ahora basta con detallar que la misma relaciona una palabra con el porcentaje o grado de asociación de la misma con los distintos atributos de calidad. Esta asociación se realiza mediante un conjunto de pares <atributo de calidad, porcentaje>, indicando, para los distintos atributos de calidad, el porcentaje de asociación de esa palabra con el mismo.

Para un mejor entendimiento de la tarea “Calcular Quality Attribute Theme” a continuación se muestran tres versiones de la misma, en las cuales se va aumentando la complejidad hasta llegar a la adoptada en la técnica propuesta.

La primera es una versión simplificada, que supone como entrada una sola lista de tokens y en donde no se tienen en cuenta las ocurrencias y los pesos de los tokens en el cálculo de los atributos de calidad que se le relacionan. La segunda versión de la tarea también supone como entrada una sola lista de tokens, pero se tienen en cuenta el peso y el número de ocurrencias de cada token. En la última versión, que es la que adopta la técnica propuesta, se supone como entrada dos listas de tokens, teniendo en cuenta el peso y el numero de ocurrencias de cada token en el cálculo de los atributos de calidad que se le relacionan.

Finalmente, en la subsección [sub:Definción-matemática-formal] enunciamos una serie de ecuaciones matemáticas que definen, formal y completamente, el funcionamiento de esta tarea.

#### Una sola lista de tokens, sin cálculo de ocurrencias ni pesos.

A continuación se muestra un fragmento de pseudocódigo, con una versión simplificada de la tarea, aplicada solo a una lista de tokens y en donde no se tiene en cuenta el peso y el número de ocurrencias de cada token.

1: Mapa<QA,porcentaje> porcentaje\_total\_map = crear

mapa con todos los atributos de calidad e igualar

porcentajes a 0;

2: cantidad\_de\_palabras = 0;

3: por c/ token de la lista de tokens

4: inicio

5: palabra = token.palabra

6: Mapa<QA,porcentaje> porcentaje\_palabra\_map =

asociar palabra con atributos de calidad(palabra);

7: porcentaje\_total = porcentaje\_total

+ porcentaje\_palabra ;

8: cantidad\_de\_palabras = cantidad\_de\_palabras + 1;

9: final

10: porcentaje\_total = porcentaje\_total/cantidad\_de\_palabras

En la línea 1 se declara un conjunto o mapa de pares <Atributo de calidad, porcentaje>. A este mapa se le agregan como pares todos los atributos de calidad, con los porcentajes de cada uno inicializados en 0.

En la línea 2 se inicializa la variable cantidad\_de\_palabras en 0. Esta variable cuenta la cantidad de palabras analizadas de los tokens de la lista.

En la línea 3 se inicializa una iteración por cada uno de los tokens de una lista de tokens. Por cada uno de esos tokens se recupera el atributo palabra, como se muestra en la linea 5; y se la asocia a a un conjunto de pares <Atributo de calidad, porcentaje>, como se muestra en la linea 6. Como se mencionó anteriormente, esta asociación entre la palabra y un conjunto de atributos de calidad es realizada por la tarea “Asociar palabra con atributos de calidad”, en adelante denominada AP, que toma como entrada a una palabra. La misma es detallada más adelante, en al subsección [sub:Asociar-palabra-con].

Los porcentajes de cada palabra, para cada atributo de calidad, se suman y almacenan en el mapa porcentaje\_total. La cantidad de palabras analizadas se va almacenando en la variable cantidad\_de\_palabras.

Finalmente, en la línea 10 se divide cada porcentaje de cada atributo de calidad del mapa porcentaje\_total por la cantidad de palabras analizadas.

Ejemplo:

Supóngase que se tiene sólo los atributos de calidad de Modificability, Performance y Aviability. La linea 1 inicializa un mapa con estos tres atributos en los pares, cada uno con valores en 0. En la línea 2 se inicializa el contador de palabras en 0. De esta manera, después de las inicializaciones, las variables quedan en el siguiente estado:

porcentaje\_total\_map = (<Modificability,0>; <Aviability,0>; <Performance,0>).

cantidad\_de\_palabras = 0;

Supóngase que la lista de tokens esta conformada por solo dos tokens. El primer token contiene la palabra “fast”. Supóngase que la tarea AP, relaciona, para esa palabra, el siguiente mapa

AP(“fast”) = (<Modificability,0.25>; <Aviability,0.0>; <Performance,0.75>)

Este resultado significa que la tarea asigna a la palabra “fast” una relación de 0.25 o 25% con el atributo de calidad “Modificability”, un 0% con “Aviability” y un 75% con el atributo “Performance”.

Como se dijo anteriormente los resultados parciales de cada palabra se van acumulando. Es por ello que este mapa se suma al mapa porcentaje\_total\_map durante la primera iteración. A su vez, el número de palabras aumenta en 1. De esta manera, el estado de las variables porcentaje\_total\_map y cantidad\_de\_palabras, después de la primera iteración, es el siguiente:

porcentaje\_total\_map = (<Modificability,0.25>; <Aviability,0>; <Performance,0.75>).

cantidad\_de\_palabras = 1;

Supóngase que la segunda palabra es “seconds”. Durante la segunda iteración, la tarea AP la relaciona con el siguiente mapa:

AP(“seconds”) = (<Modificability,0.25>;<Aviability,0.50>;<Performance,0.25>)

Este mapa se suma al de porcentaje\_total\_map. De esta manera, el estado de las variables porcentaje\_total\_map y cantidad\_de\_palabras después de la segunda iteración es el siguiente:

porcentaje\_total\_map = (<Modificability,0.50>; <Aviability,0.50>; <Performance,1.00>)

cantidad\_de\_palabras = 2;

Ya no hay más tokens en la lista, por lo que después de la segunda iteración se sale del bucle. La línea 10 del fragmento de pseudocódigo indica que esta suma de resultados parciales almacenados en porcentaje\_total\_map es dividida por la cantidad de palabras. De esta manera, luego de ejecutarse la línea 10, el contenido de porcentaje\_total\_map es la siguiente

porcentaje\_total\_map = (<Modificability,0.25>; <Aviability,0.25>; <Performance,0.50>)

La interpretación de este resultado es que la lista de tokens se relaciona en un 25,0% con Modificability, un 25% con Aviability y un 50,0 % con Performance.

#### Una sola lista de tokens, con cálculo de peso y ocurrencias

En el ejemplo anterior no se tuvo en cuenta el peso y el número de ocurrencias de los tokens. Éstas, junto con la palabra, son propiedades que los tokens poseen y que son tenidas en cuenta en la técnica propuesta en este trabajo. El siguiente fragmente de pseudocódigo es una modificación del anterior, para así tener en cuenta estas propiedades.

1: Mapa<QA,porcentaje> porcentaje\_total\_map = crear mapa con todos los atributos de calidad e igualar porcentajes a 0;

2: cantidad\_de\_palabras = 0;

3: por c/ token de la lista de tokens

4: inicio

5: palabra = token.palabra

6: Mapa<QA,porcentaje> porcentaje\_palabra\_map = asociar palabra con atributos de calidad(palabra);

7: porcentaje\_palabra = porcentaje\_palabra x token.ocurrencias x token.peso;

8: porcentaje\_total = porcentaje\_total + token.ocurrencias x token.peso;

9: cantidad\_de\_palabras = cantidad\_de\_palabras + 1;

10: final

11: porcentaje\_total = porcentaje\_total/cantidad\_de\_palabras

En el fragmento de pseudocódigo anterior se ha agregado una línea, la 7, y además se ha modificado la línea 8. Cada porcentaje de cada atributo de calidad del mapa al que se le asocia una palabra se multiplica por el número de ocurrencias de esa palabra y el peso de la misma.

Se supone que el número de ocurrencias es mayor o igual a uno (si no fuera así el token no aparecería en la lista). El peso también se supone mayor a cero, aunque un peso igual a cero no afectaría al algoritmo, ya que en este caso se interpretaría como que el token se encontraba en una sección que no debe ser tenida en cuenta para el análisis.

Ya que cada valor del mapa es multiplicados n veces, (siendo n el producto entre el peso y el número de ocurrencias), la cantidad de palabras no puede aumentar en uno por cada iteración, sino que aumenta n unidades. Es por esta razón que se ha modificado la línea 8.

A continuación se verá el mismo ejemplo del párrafo anterior, salvo que en este caso el token que tiene la palabra “fast” tiene un peso de 1, y un número de ocurrencias también de 1. Sin embargo, el token que posee la palabra “seconds” tendrá un valor de 2 para el número de ocurrencias y un valor de 3 para el peso.

Ejemplo:

El primer token contiene la palabra “fast”, por lo que el estado de las variables luego de la primera iteración es el siguiente:

porcentaje\_total\_map = (<Modificability,0.25>;<Aviability,0.0>;<Performance,0.75>).

cantidad\_de\_palabras = 1;

En este ejemplo, el estado de las variables, luego de la primera iteración, es exactamente el mismo del ejemplo anterior. En efecto, al tener el token un valor igual a 1 en el peso, cada valor del mapa de atributos de calidad es multiplicado por 1. Además la cantidad de palabras aumenta en el producto entre el peso y el número de ocurrencias, el cuál también es 1.

Distinto es el caso del segundo token, que posee la palabra “seconds”, con un peso de 3 y un número de ocurrencias igual a 2. En este caso el mapa asociado para la palabra posee un valor de 0.25 para Modificability, 0.50 para Aviability y 0.25 para Perfomance. Estos valores deben ser multiplicados por n=6 (ya que este es el valor del producto entre el peso y el numero de ocurrencias). Además la cantidad de palabras también aumenta 6 unidades durante la segunda iteración. De esta manera, el estado de las variables luego de la segunda iteración es el siguiente:

porcentaje\_total\_map = (<Modificability,1.75>;<Aviability,3.0>;<Performance,2.25>).

cantidad\_de\_palabras = 7;

Luego de esta iteración no hay más tokens, por los que los valores del mapa porcentaje\_total\_map deben ser divididos por cantidad\_de\_palabras, dando como resultado el siguiente mapa:

porcentaje\_total\_map = (<Modificability,0.25>;<Aviability,0.43>;<Performance,0.32>)

La interpretación de este resultado es que la lista de tokens se relaciona en un 25% con modificability, un 43% con aviability y un 32 % con Performance. Comparándolo con el resultado del ejemplo anterior, se puede notar que el resultado para “Performance” ha disminuido, mientras que el de “Aviability” ha aumentado hasta incluso a superar el a “Performance”.

La lista de tokens es la misma, sin embargo el peso y el numero de ocurrencias del segundo token, que en su conjunto son seis veces mayores que el primero, hacen inclinar la balanza hacia Aviability, que es el atributo de calidad con el que más se relaciona la palabra “seconds”.

#### Dos listas de tokens, con cálculo de peso y ocurrencias

En los ejemplos anteriores se detalla la forma en que una lista de tokens es analizada y relacionada con un conjunto de atributos de calidad, cada uno con un porcentaje de asociación. En este análisis interviene la palabra, el peso y el número de ocurrencias de cada token en la lista.

Sin embargo, como ya se ha mencionado, en la tarea analizada en esta sección intervienen dos listas distintas: una con tokens extraídos de los casos de uso y otra de tokens extraídos de los aspectos tempranos que relacionan a esos casos de uso. De esta manera, la tarea propuesta en este trabajo combina los resultados parciales de ambas listas. Es decir, lo detallado en el párrafo anterior se aplica para la lista de tokens extraídos de los casos de uso y luego se aplica a la lista de tokens extraídos de los aspectos tempranos. Luego, para combinar ambas listas se elige un factor k. Este factor es un número real entre 0 y 1 e indica la importancia que se le debe dar a cada una de las listas. La herramienta desarrollada en este trabajo permite que el desarrollador sea capaz de ingresar este valor, según sus preferencias.

Cada valor de cada atributo de calidad del mapa de la lista de tokens del caso de uso se multiplica por k, mientras que cada valor de la lista del mapa de tokens de los aspectos tempranos se multiplica por 1-k. Luego, para cada atributo de calidad de cada mapa, se suman los valores

Supóngase, por ejemplo, que el mapa A=(<Modificability,0.25>; <Aviability,0.25>; <Performance,0.50>) es el relacionado con la lista de los casos de uso y el mapa B=(<Modificability,0.10>; <Aviability,0.30>; <Performance,0.60>) es el relacionado con los aspectos tempranos. Eligiendo un valor de k=0.4 se tendría que multiplicar cada valor de A por 0.4 y cada valor de B por 1 - 0.4 = 0.6, para luego sumar, para cada atributo de calidad, los valores de ambas listas. Estos daría como resultado un mapa C=(<Modificability,0.16>; <Aviability,0.28>; <Performance,0.56>).

Nótese que eligiendo un k=1 el mapa resultantes es el mismo que el mapa que se relaciona con los casos de uso (en el caso anterior el mapa resultante, C, seria igual al A). Sin embargo, con un k=0, el mapa resultante es el mismo que el mapa identificado a partir de la lista de tokens de los aspectos tempranos. Un valor de k=0,5 implica que cada mapa aporta la misma proporción al mapa resultante.

Para la técnica propuesta en este trabajo se determinó que un valor de k igual a 0,5 es el más adecuado para el análisis, ya que balancea de forma equitativa la información extraída a partir de los casos de uso y de los aspectos tempranos.

#### 5.3.1.1 Definición matemática formal

En esta subsección se pasa a describir el funcionamiento de la tarea “Calcular Quality Attribute Theme” de una manera formal y detallada, utilizando herramientas y cálculos matemáticos. Para ello primero se introduce el concepto de “Mapa”, definiendo operaciones sobre el mismo. Este concepto es introducido por nosotros, para el desarrollo de la técnica propuesta. Luego se detallan las ecuaciones que conforman el funcionamiento de esta tarea.

De esta manera, definimos un mapa como una conjunto de tuplas <K,V> , donde a K se lo denomina “clave” y a V se lo denomina “valor”. Un mapa no puede tener dos tuplas con claves idénticas. Es decir, dado un mapa M no existe ningún par de tuplas <K1, V> , <K2,V2> tal que K1=K2.

En los mapas definidos en este trabajo se utiliza como “claves” los nombres de los atributos de calidad y como “valores” números reales entre 0 y 1.

#### Suma de dos mapas

Sean M1 y M2 dos mapas con igual número de tuplas. Además toda clave K que pertenece a alguna tupla de M1, también pertenece a alguna tupla de M2. Sea además cualquier valor de V perteneciente a M1 y M2 un número real. Definimos la suma M1+M2, como un mapa M3 en donde cada tupla que éste posee es el resultado de la suma de los valores de las tuplas de igual clave de M1 y M2. Por ejemplo, si se tiene el mapa M\_{1}=(<K\_{1},a>;<K\_{2},b>) y el mapa M\_{2}=(<K\_{1},c>;<K\_{2},d>) se define M\_{1}+M\_{2}=(<K\_{1},a+c>;<K\_{2},b+d>) siendo K1 y K2 claves; a, b, c y d números reales

#### División de un mapa por un numero real

Sea M un mapa compuesto por un conjunto de tuplas <K,V> donde K es la clave y V son números reales. Sea x un número real distinto de cero. La división de M por x da como resultado el mismo mapa, con todos los valores V divididos por ese número. Por ejemplo, si M=(<K\_{1},a>;<K\_{2},b>), entonces M/x=(<K\_{1},a/x>;<K\_{2},b/x>), siendo a,b y x numero reales; M un mapa; K1,K2 claves del mapa.

#### Multiplicación de un mapa por un número real

Sea M un mapa compuesto por un conjunto de tuplas <K,V> donde K es la clave y V son números reales. Sea x un número real. La multiplicación de M por x da como resultado el mismo mapa, con todos los valores V multiplicados por ese número. Por ejemplo, si M=(<K\_{1},a>;<K\_{2},b>), entonces M\times x=(<K\_{1},a\times x>;<K\_{2},b\times x>), siendo a,b y x números reales; M un mapa; K1,K2 claves del mapa.

Finalmente en esta subsección se detalla el funcionamiento de la tarea “Calcular Quality Attribute Theme” de una forma matemática formal. Al comienzo de la sección se mostró el funcionamiento de la tarea en un fragmento de pseudocódigo. En esta subsección se realiza lo mismo, pero mostrando el funcionamiento de la misma de una forma más completa, mediante ecuaciones matemáticas.

Como se ha dicho anteriormente, la tarea “Calcular Quality Attibute Theme” tiene como entrada dos listas de tokens extraídos de los casos de uso y de los early aspects. Así, se denomina como UCL[] a un arreglo compuesto por tokens extraídos de los casos de uso y a EAL[] como otro arreglo de tokens extraídos de los early aspects.

Aunque se ha mencionado anteriormente vale la pena recordar que un token x tiene varias propiedades, y en las siguientes ecuaciones se denomina como x.palabra al conjunto de caracteres del token que representan la palabra, x.ocurrencias a la propiedad que indica el numero de ocurrencias de ese token y x.peso al valor o peso que se le asigna a ese token.

AP son las siglas que denominan a la tarea “Asociar Palabra”. Esta tarea relaciona una palabra con un mapa de atributos de calidad y porcentajes, como se mostró en la figura [fig:Tareas\_de\_la\_tecnica\_Propuesta], siendo detallada en la subsección[sub:Asociar-palabra-con].

Entonces, sea i un número entero, UCL[i] se refiere al token en la posición i del arreglo UCL. Similar es el caso del arreglo EAL. De esta manera se definen las siguientes ecuaciones.

Sea totalUCM un mapa que se define como:

totalUCM=\frac{\overset{n}{\underset{i=1}{\sum}}AP(UCL[i].palabra)\times UCL[i].ocurrencias\times UCL[i].peso}{\overset{n}{\underset{i=1}{\sum}}UCL[i].ocurrencias\times UCL[i].peso}

Y sea totalEAM un mapa que se define como:

totalEAM=\frac{\overset{n}{\underset{i=1}{\sum}}AP(EAL[i].palabra)\times EAL[i].ocurrencias}{\underset{i=1}{\overset{n}{\sum}}EAL[i].ocurrencias}

Entonces se define el mapa totalMap como:

totalMap=k\times totalUCM+(1-k)\times totalEAM

en donde k es un numero real entre 0 y 1. La constante k indica la importancia correspondiente a la lista de tokens extraídos de los casos de uso y a la lista de tokens extraídos del aspecto temprano. Es decir, a la hora de calcular el totalMap, k indica que porcentaje sera extraído del totalUCM y qué porcentaje del totalEAM. Nótese que si k=1, entonces totalMap es igual a totalUCM; si k=0 entonces totalMap es igual a totalEAM; si k=0.5 totalMap es la mitad de totalUCM más la mitad de totalEAM.

#### 5.3.1.2 Asociar palabra con atributos de calidad

En esta subsección finalmente se detalla el funcionamiento de la tarea “Asociar palabra con atributos de calidad” (AP).

Esta tarea relaciona una secuencia de caracteres que representa una palabra con un conjunto de atributos de calidad, asociados cada uno con un porcentaje. Este porcentaje indica el grado de asociación o pertenencia que tiene ese palabra con cada atributo de calidad. Para realizar este cálculo, la tarea consulta las instancias de la ontología, formada por los conceptos y relaciones explicadas en la sección[sec:Ontología-para-atributos]. Se supone que las instancias de la ontología han sido creadas por un experto.

Básicamente esta tarea consta de dos partes. La primera es averiguar a que parte del escenario corresponde la palabra. Es decir, descubrir si corresponde a una “Fuente de estímulo”, a un “Estímulo” o a alguna otra. Una vez reconocida la instancia a la que se corresponde la palabra, se averigua cuántos y cuáles escenarios se relacionan con esa instancia. A partir de esto último, y de calcular los porcentajes en que ese grupo de escenarios se relacionan con los atributos de calidad, se obtiene un mapa de atributos de calidad y porcentajes.

Relación entre una palabra y una parte del escenario

En esta parte se intenta identificar si la palabra se corresponde con alguna instancia de los conceptos de ConcreteSource (Fuente), ConcreteStimulus (Estímulo), ConcreteEnviroment (Ambiente), ConcreteArtifact (Artefacto), ConcreteResponse (Respuesta) y ConcreteResponseMeasure (Medida de Respuesta). Básicamente, al arribar la palabra X se intenta responder a la pregunta, ¿ es X una “Fuente”, un “Estímulo”, un “Ambiente”, un “Artefacto”, una “Respuesta” o una “Medida de Respuesta”?

Esta tarea podría parecer trivial, pero no lo es tanto. El caso ideal es que exista una sola instancia, de alguna de esas seis, que se corresponda con la palabra. La figura [fig:Instancia-única] ilustra este caso.



En la figura [fig:Instancia-única] se muestra en un recuadro rojo la instancia, mientras que en un recuadro negro se representa la clase. A su vez la instancia posee una propiedad (de tipo “cadena de caracteres”) que se denomina “concreteScenarioPartDescription” y cuyo valor, en este caso, es “user”. Uno o mas escenario se pueden relacionar con esta instancia mediante la propiedad “concreteScenariohasSource”.

Se pretende identificar la palabra “user”. En este caso hay una sola instancia en la ontología, cuyo identificdor único de instancia es “Instancia\_1”, que es de tipo ConcreteSource, que mediante la propiedad “concreteScenarioPartDescription” posee el valor “user”, por lo que se retorna esta instancia.

Distinto es el caso en el que la misma palabra pueda representar, simultáneamente, distintas partes de un escenario. La figura [fig:Distintos-escenarios] ilustra este otro caso.



En este caso la palabra “system” podría ser un ConcreteSource o un ConcreteArtifact. Lo que lleva a la cuestión de decidir entre ambas opciones. Para ello se contabiliza con cuántos escenarios se relacionan cada una de las instancias. Será elegida como significado semántico de la palabra la instancia que se relacione con un mayor numero de escenarios. En este ultimo caso si la “instancia\_1” se relaciona con 3 escenarios y la “instancia\_2” con 6, se retorna la “instancia\_2”, tomándose como que la palabra “system” es un ConcreteArtifact.

En este punto cabe aclarar que no existen dos instancias que se relacionen con la misma palabra (mediante la propiedad “concreteScenraioPartDescription”) y que sea del mismo tipo (ConcreteScource, ConcreteArtifact, etc.) Es decir, por ejemplo, no puede haber dos instancias en las que ambas sean estímulos y además se relacionen con la palabra “developer”. Si esto ocurriera, se pasan todas las relaciones de una instancia a la otra, y se elimina la primera. Este es un proceso que se realiza al crear la ontología. Es por ello que en la descripción anterior, con devolver la instancia ya se asegura que es única para ese parte de escenario.

Obtención de porcentajes de atributos de calidad para una palabra

Una vez que se ha identificado la instancia de alguna parte de un escenario que se corresponde con una palabra, se dispone a obtener los porcentajes de los distintos atributos de calidad que se relacionan con esa palabra. Los atributos de calidad, juntos con los porcentajes asociados, se agrupan en un mapa como el descripto en la subsección [sub:Definción-matemática-formal].

Para encontrar estos porcentajes, primero se recuperan todos los escenarios que se relacionan con la instancia encontrada en el paso anterior. Cada uno de estos escenarios se relaciona con un atributo de calidad mediante la propiedad “isSpecifOf”. De esta manera, para cada atributo de calidad, se devuelven los porcentajes en que estos escenarios se relacionan con cada uno de ellos.

Por ejemplo, supóngase que como entrada se obtuvo el término "latency". Después del paso anterior, se encuentra que esa palabra se corresponde con la “Instancia\_48” que es de tipo ResponseMeasure. A su vez 40 escenarios se relacionan con la “Instancia\_48” mediante la propiedad “scenarioHasResponseMeasure”. De esos 40 escenarios, 30 de ellos se relacionan con el atributo de calidad "Performance", 8 con "Aviability" y 2 con "Modificability". De esta manera podemos concluir que ese concepto, “latency”, se relaciona en un 75% de certeza con "Performance", 20% con "Aviability" y 5% con "Modificability". En este caso la tarea relacionaría a la palabra “latency” con <Performance 0.75, Aviability 0.2, Modificability 0.05>.

ANEXO I

ANEXO II

Aca va el anexo 2

# Bibliografía (21)

1. [En línea] http://www.sei.cmu.edu/architecture/definitions.html.

2. **Reynoso, Carlos.** Introducción a la Arquitectura de Software - Version 1.0. [Universidad de Buenos Aires]. Marzo de 2004.

3. **Clements, Paul.** Survey of Architecture Description Languages. [Proceedings of the International Workshop on Software Specification and Design]. 1996.

4. **Bass, L, Clements, P y Kazman, R.** *Software Architecture in practice.* s.l. : Addison-Wesley, 1998.

5. **Garlan, David y Shaw, Mary.** An introduction to software architecture. *CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21.* 1994.

6. IEEE Standard 1061-1992. Standard for a Software Quality Metrics Meth. 1992.

7. **Kazman, R., Clements, P. y Klein, M.** Evaluating Software Architectures. Methods and case studies. s.l. : Addison Wesley, 2001.

8. **Barbacci, Mario; Klein, Mark; Longstaff, Thomas A.; Weinstock, Charles B.** Quality Attributes, Technical Report CMU/SEI-95-TR-021 ESC-TR-95-021. Diciembre de 1995.

9. *Identifying Crosscutting Concerns in Requirements Specifications,.* **Rosenhainer, L.** Vancouver : s.n., 2004. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design , held in Function with OOPSLA 2004 Conference Design, held in conjunction with OOPSLA 2004 Conferenc.

10. *Discovering early aspects.* **Baniassan, E., y otros.** EEE Software 23(1), págs. 61-70.

11. **Cleland-Huang, Jane, y otros.** Automated classiﬁcation of non-functional requirements. Londres, UK : Springer-Verlag, Abril de 2007. Vol. 12, 2, págs. 103-120.

12. *Ontology Development 101: A Guide to Creating Your First Ontology Escenario.*

13. *OWL Web Ontology Language - Use Cases and Requirements (W3C Recommendation- http://www.w3.org/TR/webont-req/).* 2004.

14. **Gruber, T.R.** *A translation approach to portable ontology specifications.* Junio, 1993. págs. 199-220. Vol. 5.

15. *OWL Web Ontology Language Overview.* W3C Recommendation. 10 de Febrero de 2004.

16. **Connolly, Dan, y otros.** *DAM+OIL, Reference Description.* W3C. 2001.

17. *Identificacion Temprana de Aspectos.* **Haak, B., y otros.** 2005, Revista SCC, (Workshop in SE), Vol. 6.

18. *Early Aspect Identification from Use Cases using NLP and WSD Techniques.* **Rago, y otros.** 2009. EA '09 Proceedings of the 15th workshop on Early aspects.

19. **Rational Software Corporation.** *UML Semantic Version.* Septiembre 1997.

20. **Buschmann, F., y otros.** *Pattern-Oriented Software Architecture: A System Of Patterns.* [ed.] England: John Wiley & Sons Ltd. West Sussex.

21. *A requirements model for quality attributes.* **Brito, Isabel, Moreira, Ana y Araujo, Joao.** 2002. Aspect-Oriented Requirements Engineering and Architecture Design Conference.

22. *IEEE recommended practice for software requirements specifications (IEEE Std 830-1998).* **The Institute of Electrical and Electronics Engineers.** New York : s.n., 20 de Octubre de 1998. ISBN 0-7381-0332-2.