Extensión de Aspects Extractor con Ontología

Tesis de Grado de la Carrera de Ingeniería de Sistemas
Facultad de Ciencias Exactas de la UNCPBA

Por

Gladis N. Errecalde
Directora: Dra. Claudia Marcos

Septiembre, 2008

Índice de contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción

1.1 Motivación

El desarrollo de grandes sistemas de software presenta dificultades debido a su complejidad, se encuentran muchos problemas para los cuales las técnicas de programación procedural u orientadas a objetos no son suficientes [BM06]. A menudo, nos encontramos con situaciones en las que un requerimiento está diseminado sobre varios módulos del sistema. A este problema se lo conoce como código disperso, generalmente provoca dificultades de mantenimiento y desarrollo. Otro problema que puede aparecer, es que un mismo módulo implemente múltiples requerimientos del sistema de forma simultánea, es el problema del código enmarañado. El hecho es que hay ciertas decisiones de diseño que son difíciles de capturar, debido a que determinados problemas no se pueden encapsular claramente de igual forma que los que habitualmente se resuelven con funciones u objetos. A menudo esta situación desemboca en la construcción de los sistemas de software con poca calidad, cuya evolución y mantenimiento es muy costosa [NPME04].

Nuevas técnicas han surgido para disminuir los problemas de enmarañamiento y dispersión del código. El Desarrollo de Software Orientado a Aspectos (DSOA) intenta resolver estos problemas utilizando el principio de separación de concerns. La separación de concerns se refiere a la habilidad para identificar, encapsular y manipular sólo las partes del software que son relevantes a un concepto, meta o propósito en particular [DI76].

El DSOA proporciona mecanismos y abstracciones para la implementación de la funcionalidad secundaria de manera separada y aislada, permite encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. El DSOA ofrece un conjunto de mecanismos que permite gestionar los intereses que aparecen dispersos en el sistema, cruzando o cortando diferentes elementos del mismo. El DSOA identifica estos concerns expresándolos como aspectos y tratándolos de forma explícita [NLRA04].

La estructura de los sistemas software no es estática sino que cambia y evoluciona para adaptarse a nuevas circunstancias. Por ello, cada vez es más importante que los desarrollos sean fácilmente adaptables y que su evolución suponga un esfuerzo reducido. Para que la construcción de software obtenga beneficios en reutilización, modularidad y evolución, el principio de separación de concerns es fundamental. Las técnicas orientadas a aspectos demuestran ser una excelente solución para manejar la separación de esos tipos de intereses. El DSOA es una disciplina muy prometedora que constituye una alternativa válida para mejorar el proceso de desarrollo de software, en un intento de superar la creciente complejidad de los sistemas de software [BCC05].

El DSOA incentiva la utilización del concepto de aspecto en todas las fases del ciclo de vida del desarrollo de software [BCC05], si lo aspectos pueden ser detectados en etapas tempranas se podrán obtener beneficios tales como la reducción de costos de desarrollo, mantenimiento y evolución.

La ingeniería de requisitos es un campo muy activo dentro de la ingeniería del software, el resultado que se obtiene es clave para el resto de las fases en el desarrollo de software. [CLMC05]. Las prácticas de desarrollo de software actuales solo tienden a consideran los requisitos funcionales desde las primeras etapas del ciclo de vida de los productos de software, relegando los no funcionales a etapas tardías del desarrollo, causando enmarañamiento y dispersión de código y dificultando el mantenimiento. La identificación, comprensión y la importancia de la correcta y precisa especificación de los requisitos, se incrementa drásticamente con la magnitud y complejidad de los sistemas de software, particularmente si estos tienen que responder a intereses que van más allá del alcance de sus funcionalidades principales. Estos intereses deben ser identificados de manera temprana y deben ser

considerados oportunamente por los ingenieros de software, en el desarrollo de sus aplicaciones [CLMC05].

La ingeniería de requerimientos orientada a aspectos reduce el peligro de cambios no esperados en los productos de software a través de una identificación temprana de aspectos y resolución de conflictos [HDMP05]. O sea, si no se tratan los aspectos tempranamente, podrían surgir cambios en el sistema que se está desarrollando en etapas posteriores aumentando los costos de mantenimiento, desarrollo y evolución.

Existen varios enfoques en la ingeniería de requisitos orientada a aspectos que están enfocados en identificar y modelar aspectos durante las etapas tempranas de desarrollo de software [LHB06]. El tratamiento de los requisitos se soporta en un conjunto de enfoques de la Ingeniería de Requisitos Orientada por Aspectos (Aspect-Oriented Software Requirement - AORE). Las propuestas que existen tienen el propósito de fortalecer, en la fase inicial, el análisis de los concerns involucrados en un problema, así como el soporte a su evolución en la especificación del sistema, la solución de conflictos, y la toma de decisiones de diseño [LHB06]. Los enfoques orientados por aspectos asociados a la ingeniería de los requisitos proponen mecanismos que facilitan la identificación, separación y clasificación de intereses y la composición de intereses transversales [LHB06]. Diversas propuestas han surgido que proponen una identificación de aspectos candidatos en la etapa de captura y especificación de requisitos tales como AORE [TAA07], Theme/Doc [BA06], Exploración de Aspectos en Requerimientos [SLRR05], [RA03], Buscando Aspectos en la Especificación de los Requerimientos [RO04], Aspect Extractor [HD05], etc.

1.2 Problemática

Los aspectos constituyen una disciplina emergente y por lo tanto no esta exenta de cuestiones y problemas aún no resueltos completamente. El DSOA presenta algunos problemas entre los que principalmente se destacan:

- Falta de unificación de los conceptos que determinan los crosscutting concerns en las aplicaciones de software.
- Necesidad de un enfoque que agilice la identificación, especificación, integración y evaluación de los crosscutting concerns en etapas tempranas del desarrollo de software.

Los enfoques existentes para la identificación, especificación, integración y evaluación de los concerns no son completamente apropiados. Existen varias implementaciones de estos enfoques, pero ninguno satisface de forma completa las principales necesidades de los ingenieros de software. Los métodos actuales poseen deficiencias tales como la falta de precisión en los resultados obtenidos, necesidad de un conocimiento previo del proyecto por parte del ingeniero de software, necesidad de documentos con algún formato específico, es tedioso el trabajo con proyectos de grandes volúmenes de información.

Se puede concluir que la necesidad actual de los ingenieros de software en el área DSOA es la identificación rápida y precisa de los crosscutting concerns, ofreciendo términos unificados de éstos. De esta manera se logra una mejor calidad en la aplicación desarrollada, reduciendo los costos de desarrollo y mantenimiento, y evitando complicaciones que puedan ocurrir en el futuro.

1.3 Trabajo propuesto

En este trabajo se presenta una extensión a la herramienta Aspect Extractor Tool. Esta herramienta implementa el enfoque de identificación, especificación, integración y evaluación de aspectos, Aspect Extractor. La extensión a la herramienta presenta una forma distinta de identificar los aspectos candidatos obteniendo mejores resultados desde las fases tempranas del ciclo de vida de desarrollo de software.

La extensión presenta otra manera de realizar la sub-tarea Identificar crosscutting concern de la tarea Elección de aspectos candidatos. Para realizar esta sub-tarea se desarrolló una ontología. En la ontología se modelan los términos que representan a los aspectos en el área de conocimiento del DSOA. La ontología permitió estandarizar y unificar los conceptos que representan a aspectos, los conceptos que se modelaron son Acceso a Memoria, Actualización de Pantalla, Comunicación entre Procesos, Concurrencia, Debugging, Distribución de Recursos, Manejo de Errores, Monitoreo, Logging, Tracing, Profiling, Performance, Restricciones de Tiempo Real, Seguridad, Auditoria, Autenticación, Control de Acceso, Disponibilidad e Integrabilidad. La ontología permite realizar búsquedas sobre los datos que modela logrando obtener resultados más concretos. La ontología permite también realizar consultas sobre el conocimiento no explícito en el modelo original, lo que permitió derivar conocimiento tal como las situaciones de conflictos existentes entre los aspectos candidatos propuestos y cuales son las palabras de los casos de uso del proyecto que hicieron que el aspecto sea propuesto como aspectos candidato. Fue realizada una ontología en el idioma español y otra en el idioma inglés, de esta manera Aspect Extractor Tools puede continuar trabajando con proyectos en ambos idiomas.

Para obtener el listado de aspectos candidatos se recorre la ontología una vez por cada caso de uso del proyecto y una vez por los requerimientos especiales que el proyecto tenga. Al recorrer la ontología se van buscando coincidencias entre las palabras del caso de uso y las palabras que representan a los aspectos en la ontología. Si en las palabras del caso de uso se encuentra una palabra que sea igual a alguna de las palabras que representan el aspecto o a las palabras sinónimos de estas palabras, se considera que se ha encontrado una coincidencia. Se cuenta la cantidad de coincidencias existentes para un aspecto en el caso de uso y se la compara con la cantidad mínima de coincidencias que el aspecto requiere para ser propuesto como aspecto candidato. Un aspecto es considerado como aspecto candidato si tiene una cantidad de coincidencias mínima que se calcula para cada aspecto según la cantidad de palabras que este tenga.

Con la incorporación del algoritmo usando la ontología se logró obtener una lista de aspectos candidatos más apropiada para presentar al analista. Los resultados son precisos y dan la idea de la responsabilidad que representan. De esta manera de obtiene una mejor comunicación entre los stakeholders del sistema donde todos sabrán cuál es el significado de cada término del que se esta hablando, se facilita la tarea manual Elección de Aspectos Candidatos del enfoque Aspect Extractor y se reducen tiempos de desarrollo y mantenimiento.

1.4 Organización del trabajo

El trabajo que se presenta esta organizado de la siguiente manera:

En el capítulo 2 se presenta al Desarrollo de Software Orientado a Aspectos. Se hace una introducción a su origen y a los conceptos básicos que lo componen. Se presentan los aspectos en la ingeniería de requerimientos estudiando algunos enfoques para identificar y modelar aspectos en etapas tempranas del desarrollo de software y se los compara a través de diferentes criterios. Se identifican algunos problemas que se presentan actualmente en esta área.

En el capítulo 3 se describe una ontología. Se presentan los componentes, la clasificación y los diferentes tipos. Se describen los principales beneficios que se obtienen al utilizar una ontología. Se presenta una visión general de los lenguajes de implementación y las herramientas de creación y edición más usadas.

En el capítulo 4 se presenta la herramienta Aspect Extractor Tool. Esta herramienta automatiza el enfoque Aspect Extractor presentado en el capítulo 2. Se presenta la funcionalidad de la herramienta mediante distintos ejemplos. Se describen algunos problemas encontrados, y contrarrestando a estos problemas, se presenta una propuesta de incorporación de un algoritmo de identificación de aspectos candidatos mediante el uso de una ontología.

En el capítulo 5 se describe la ontología desarrollada que modela los conceptos que representan aspectos en dominio DSOA. Se presenta la herramienta Aspect Extractor Tool con la incorporación del mecanismo de identificación de aspectos candidatos con el uso de la ontología. Se describe el proceso de identificación y la inferencia de información.

En el capítulo 6 se describen distintos casos de estudio que muestran el funcionamiento y los resultados que se obtienen al usar la herramienta Aspect Extractor Tool con el algoritmo que usa la ontología.

En el capítulo 7 se comparan los resultados obtenidos al usar los dos algoritmos de identificación de aspectos aplicados a mismos casos de estudio.

Finalmente, en el capítulo 8 se presentan las conclusiones obtenidas del trabajo realizado.

Capítulo 2: Desarrollo de Software Orientado a Aspectos

La separación de concerns se refiere a la *"habilidad para identificar, encapsular y manipular sólo las partes del software que son relevantes a un concepto, meta o propósito en particular"* [DI76].

Si un sistema es desarrollado usando alguna de las técnicas de desarrollo existentes, se encontrarán muchas partes del sistema que tienen fragmentos de código que no están directamente relacionados con la funcionalidad que el sistema tiene que hacer. Esto es debido a que las técnicas de desarrollo de software existentes tienden a tratar los concerns usando metodologías de una sola dimensión. Esta dimensión es efectiva para la funcionalidad básica del sistema, pero no lo es para los requerimientos restantes, los cuales quedan mezclados a lo largo de la dimensión dominante, resultando en un mapeo deficiente de los requerimientos a sus implementaciones. Los dos síntomas más significativos de este problema son el código mezclado y el código diseminado [RA02].

- **Código mezclado (Code tangling):** En un mismo módulo de un sistema de software pueden simultáneamente convivir más de un requerimiento. Esta múltiple existencia de requerimientos lleva a la presencia conjunta de elementos de implementación de más de un requerimiento, resultando en un código mezclado.
- **Código diseminado (Code scattering):** Como los requerimientos están dispersos sobre varios módulos, la implementación resultante también queda diseminada sobre esos módulos.

Si la aplicación es desarrollada con código mezclado y diseminado se obtendrá un incremento en el tamaño final del mismo y en el costo de mantenimiento de la aplicación, haciendo difícil encontrar código a reutilizar y realizar cambios de forma consistente. La combinación de estos síntomas afecta tanto al diseño como a la implementación de software [DI76].

La ingeniería del software ha utilizado el principio de separación de concerns para gestionar la complejidad del desarrollo de software mediante la separación de las funcionalidades principales de la aplicación de otras partes con un propósito específico, (tales como autenticación, administración, rendimiento, gestión de memoria, logging, etc.) que son ortogonales a la funcionalidad principal. La aplicación final se construye con el código de las funcionalidades principales más el código de propósito específico. La meta principal es brindar un contexto que permita discernir claramente entre componentes y crosscutting concerns, separando componentes entre sí, crosscutting concerns entre sí, y crosscutting concerns de componentes, a través de mecanismos que hagan posible abstraerlos y componerlos para producir el sistema completo.

La base de una buena separación de concerns, es la capacidad de identificar y manejar de forma individual los distintos componentes del sistema, para lograrlo de forma adecuada se deben identificar y manejar de forma individual los distintos componentes del sistema en cada una de las fases de desarrollo del software.

2.1 Desarrollo de Software Orientado a Aspectos (DSOA)

Las técnicas orientadas a aspectos extienden técnicas tradicionales como la orientación a objetos, permitiendo a los desarrolladores de software encapsular en módulos separados la implementación de los crosscutting concerns de la aplicación. Estas técnicas no reemplazan a las técnicas de programación orientada a objetos sino que las complementan y extienden.

El Desarrollo de Software Orientado a Aspectos [AOSD] es una disciplina que constituye una alternativa para mejorar el proceso de desarrollo de software en cada una de sus etapas, superando la creciente complejidad de los sistemas de software. El principal objetivo de este

paradigma es obtener un beneficio de mejora en la modularización de los sistemas, evitando la combinación entre funcionalidad y aspectos, facilitando el mantenimiento y la evolución del código, desde una mejor separación de concerns. Para lograr este objetivo proporciona mecanismos y abstracciones para la implementación de las propiedades no funcionales o la funcionalidad secundaria de manera separada y aislada. Permite encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los concerns, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Los principales beneficios de la orientación a aspectos son [GJ04]:

- Un nivel de abstracción más alto, debido a que el desarrollador se puede centrar en propósitos concretos y de forma aislada.
- Una mayor facilidad a la hora de entender la funcionalidad de la aplicación. El código implementado no está entremezclado con código de otros concerns.
- Una mayor reusabilidad al haber un menor acoplamiento.
- Una mayor mantenibilidad del código, al ser éste menos complejo.
- Una mayor flexibilidad en la integración de componentes.
- Un incremento de la productividad en el desarrollo.
- Un mejor análisis y diseño de las funcionalidades comunes, separándolas del sistema principal.
- Centralizar la implementación de las funcionalidades comunes, separando su implementación y utilización del resto del código.
- Se obtiene una modularización del sistema más óptima, lo que facilita su reutilización.
- Los sistemas resultantes son más fáciles de mantener, siendo más fácil adaptarlos a nuevos requisitos.
- Un código menos enmarañado, más natural y más reducido.
- Se consigue que un conjunto grande de modificaciones en la definición de un ítem tenga un impacto mínimo en los otros.
- Facilita el acoplamiento/desacoplamiento de las funcionalidades.

A continuación se presentan algunas definiciones de conceptos básicos en el Desarrollo de Software Orientado a Aspectos:

Concern: *"Un concern puede ser definido de una manera genérica como algo de interés para un proceso de ingeniería"* [FI04]. Un concern es un área de interés particular para el sistema. Básicamente es todo lo que sea importante para la aplicación, ya sea código, infraestructura, requerimientos, elementos de diseño, etc.

Functional concern: *"Un componente funcional es un módulo de software que puede ser encapsulado en un procedimiento (un objeto, método, procedimiento o API). Los componentes serán unidades funcionales en las que se descompone el sistema"* [KLMMLLI97].

Crosscutting concern: *"Un crosscutting concern es un concern para el cual la implementación se esparce a través del resto de la implementación"* [FI04].

Aspecto: *"Un aspecto es una unidad modular diseñada para implementar un crosscutting concern"* [FI04].

Se puede diferenciar entre un componente funcional y un aspecto viendo al primero como aquella propiedad que se puede encapsular claramente en un procedimiento, mientras que un aspecto no se puede encapsular en un único módulo con las técnicas tradicionales. Se puede decir que con las componentes funcionales se implementa la funcionalidad principal de una aplicación (como por ejemplo, la gestión de un comercio), mientras que con los aspectos se capturan conceptos técnicos tales como persistencia, gestión de errores, sincronización o comunicación de procesos.

Puntos de enlace (Join point): *"Un Punto de Enlace es un lugar bien definido en la estructura o flujo de ejecución de un programa donde se puede anexar comportamiento adicional"* [FI04]. Para que los aspectos y los componentes se puedan combinar, deben tener fijados los puntos donde puedan hacerlo, estos puntos son denominados puntos de enlace. Ejemplos de puntos de enlace pueden ser invocaciones a métodos, acceso a campos, etc.

Tejedor de aspectos (Aspect weaver): Es el encargado de combinar los puntos de enlace, el comportamiento y los demás elementos de los aspectos a la aplicación. Se combinan los diferentes mecanismos de abstracción y composición que aparecen en los lenguajes de aspectos y componentes ayudándose de los puntos de enlace. El tejedor de aspectos permite que en ciertos puntos de la ejecución de los componentes funcionales se inserte el código de los aspectos. El proceso de tejido es realizado por un nuevo tipo compilador o intérprete que combinan los componentes de funcionalidad básica con aspectos para realizar la aplicación ejecutable [PZ03]. Según Filman el proceso de tejido o weaving, *"Es el proceso de composición de la funcionalidad central con los aspectos, en consecuencia produciendo el sistema final"* [FI04].

Como se puede ver en la Figura 2.1, en las metodologías tradicionales el proceso de generar un programa consiste en pasar el código a través de un compilador o un intérprete para así disponer de un ejecutable. En el DSOA no se tiene un único código del programa sino que está separado el código que implementa la funcionalidad básica y el código que implementa cada uno de los aspectos. Todo este código debe pasar no sólo a través del compilador, sino que debe ser tratado por el tejedor de aspectos.

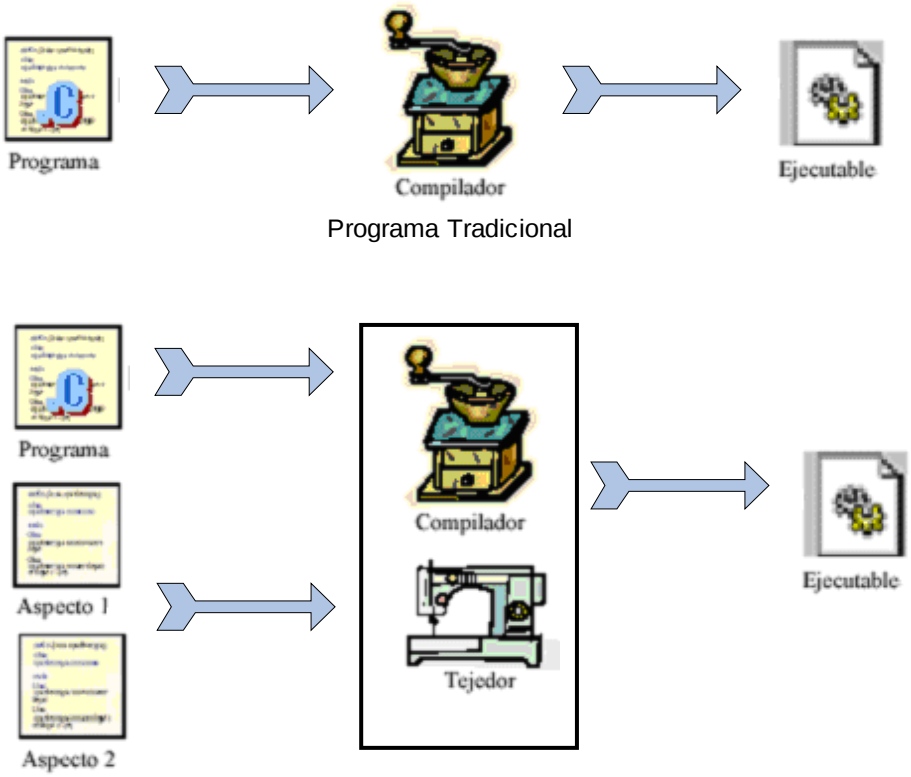

Figura 2.1: Programa tradicional vs. Programa orientado a aspectos

El tejedor de aspectos puede hacer la combinación de dos formas distintas según el momento en el que se realiza el tejido: Estática o Dinámicamente.

- **Entretejido estático:** El entretejido estático implica modificar el código fuente de una clase insertando sentencias en los puntos de enlace. La principal ventaja de esta forma de entretejido es que se evita que el nivel de abstracción que se introduce con el DSOA se derive en un impacto negativo en el rendimiento de la aplicación. Pero, por el contrario, es bastante difícil identificar los aspectos en el código una vez que éste ya se ha tejido, lo cual implica que si se desea adaptar o reemplazar los aspectos de forma dinámica en tiempo de ejecución se encuentren con un problema de eficiencia, e incluso difícil de resolver.
- **Entretejido dinámico:** Los aspectos están unidos a los componentes en tiempo de ejecución, se relacionan de forma débil. Se controla la ejecución del programa y, cada vez que se llega a un punto de unión incluido en un punto de corte de un aspecto, se ejecuta el código asociado. Este tipo de entretejido tiene problemas de performance, ya que se debe usar código extra en ejecución para acoplar y desacoplar los aspectos y las clases. Una precondición o requisito para que se pueda realizar un entretejido dinámico es que los aspectos existan de forma explícita tanto en tiempo de compilación como en tiempo de ejecución [MJVRV97].

Tejido: El tejido es el sistema final, es el resultado de la combinación entre los aspectos y el código de las funcionalidades básicas del sistema.

Conflicto: Un conflicto ocurre cuando dos o más aspectos compiten por su activación [PDM02]. En el desarrollo de una aplicación es frecuente encontrar situaciones en las que un componente de funcionalidad básica este afectado por más de un aspecto. Cada aspecto le adiciona un

comportamiento diferente. Los conflictos surgen cuando se ejecutan dos o más aspectos que están asociados a un mismo componente funcional, dado que esta situación puede provocar un comportamiento impredecible o indeseado en el sistema.

2.2 Programa orientado a aspectos

Existen situaciones en las que los lenguajes orientados a objetos no permiten modelar de forma suficientemente clara las decisiones de diseño tomadas previamente a la implementación. El sistema final se codifica entremezclando el código propio de la especificación funcional del diseño con llamadas a rutinas de diversas librerías encargadas de obtener una funcionalidad adicional (por ejemplo, distribución, persistencia o multitarea). El resultado es un código fuente excesivamente difícil de desarrollar, de entender, y por lo tanto de mantener [KLMMLLI97].

En la Figura 2.2 se puede apreciar la diferencia entre el código de un programa tradicional y el de un programa orientado a aspectos. En la parte izquierda de la Figura 2.2 puede verse la forma que tiene el código de un programa siguiendo el paradigma Orientado a Objetos (OO). Se puede observar que el código está entremezclado siendo difícil de entender, depurar y modificar. En cambio, siguiendo el DSOA, mostrado en la parte derecha de la Figura 2.2, los diferentes concerns están separados por lo tanto son más entendibles y, como consecuencia, es más sencillo trabajar con ellos.

En la Figura 2.3 se puede ver la estructura de un programa orientado a aspectos, se muestra a un programa como la combinación de distintos módulos. Unos contienen la funcionalidad básica, lo que sería el modelo de objetos, mientras que los demás contienen otro tipo de características como son la persistencia, logging, optimización de memoria, etc.

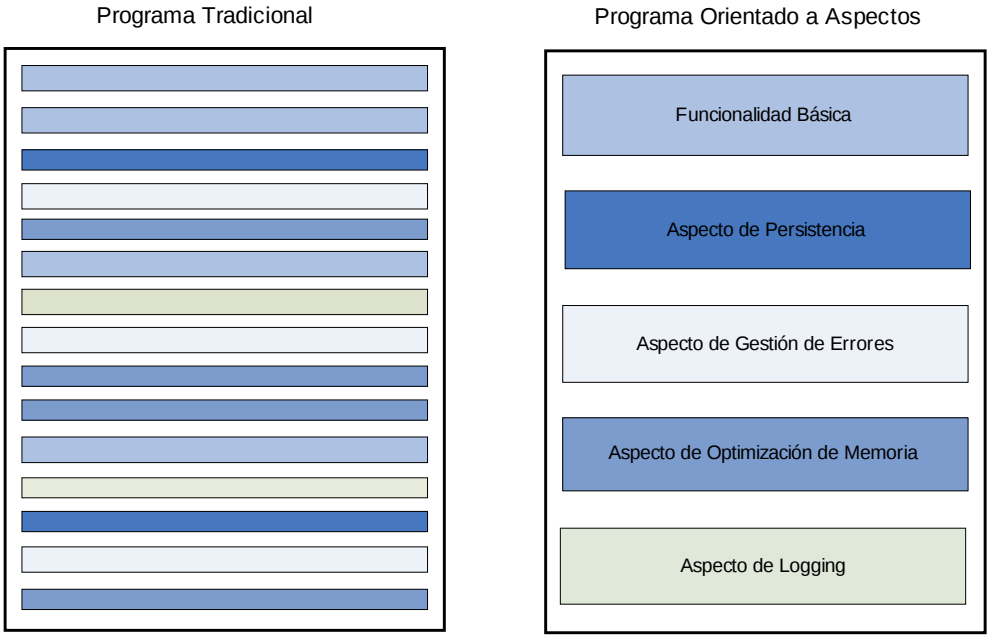

Figura 2.2: Código programa OO vs. DSOA

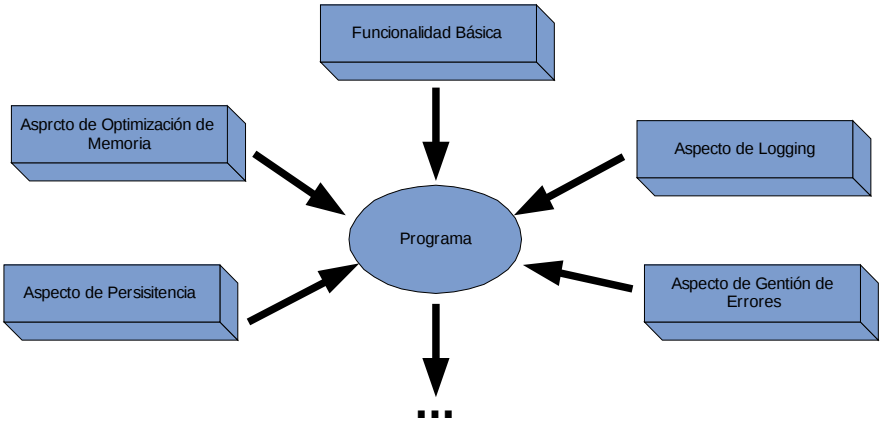

Figura 2.3: Estructura de un programa orientado a aspectos

Los tres principales requerimientos en la Programación Orientada a Aspectos son los siguientes:

- Un lenguaje para definir la funcionalidad básica, conocido como lenguaje base o componente.
- Uno o varios lenguajes de aspectos, para especificar el comportamiento de los aspectos.
- Un tejedor de aspectos, que se encargará de combinar los lenguajes.

2.3 Ingeniería de requerimientos orientada a aspectos

"Un requerimiento es una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo" [IEEE]. Los requerimientos pueden dividirse en requerimientos funcionales y requerimientos no funcionales.

- **Requerimiento funcional:** Define el comportamiento interno del software: cálculos, manipulación de datos y otras funcionalidades específicas que muestran cómo las funcionalidades serán llevadas a la práctica.
- **Requerimiento no funcional:** Es un requerimiento que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos. Son aspectos del sistema visibles para el usuario, que no están relacionados en forma directa con el comportamiento funcional del sistema. Complementan a los requerimientos funcionales.

Los requerimientos pueden ser separados de acuerdo a diferentes criterios, generalmente, se parte de los modelos de procesos del negocio y hacen la discriminación de acuerdo a las funciones del dominio del problema para así identificar los requerimientos funcionales, no funcionales y restricciones. Muchas veces se logra una separación mayor identificando reglas de negocio, requisitos de información y servicios, para determinar con mayor exactitud la arquitectura del sistema desde fases tempranas de desarrollo. También es posible lograr otro tipo de separación identificando los requerimientos que cruzan a otros, los crosscutting concerns. Comúnmente son difíciles de factorizar y soportar en módulos separados ya que se encuentran dispersos o enmarañados en diferentes concerns o requerimientos del sistema [TAA07]. Los requerimientos más comunes que tienen esta naturaleza transversal son los que representan atributos de calidad del sistema. Por ejemplo, los requerimientos asociados al concern de seguridad, la mayoría de las veces no son encontrados en etapas tempranas o son

descriptos de forma solapada en los requisitos funcionales y, consecuentemente, este concern no logra ser separado fácilmente en una clase o método en posteriores fases del ciclo de vida.

Cuando los crosscutting concerns no pueden ser detectados desde etapas tempranas del desarrollo, tienden a dispersarse en artefactos generados durante el ciclo de vida del software. A menos que estos crosscutting concerns puedan ser localizados y manejados durante el ciclo de vida puede traer serios problemas de mantenibilidad. El hecho de identificar los concerns en etapas tempranas del ciclo de desarrollo de software puede significar la reducción de costos de desarrollo, mantenimiento y evolución.

La ingeniería de requerimientos orientada a aspectos reduce el peligro de cambios no esperados en los productos de software a través de una identificación temprana de aspectos y resolución de conflictos [HDMP05]. O sea, si no se tratan los aspectos tempranamente, podrían surgir cambios en el sistema que se está desarrollando en etapas posteriores aumentando los costos de mantenimiento, desarrollo y evolución.

2.4 Enfoques para identificar aspectos en la ingeniería de requerimientos

A continuación se presentan algunos enfoques de diferentes autores para identificar y modelar aspectos en etapas tempranas del desarrollo de software.

2.4.1 Ingeniería de Requerimientos Orientada a Aspectos (AORE)

La Ingeniería de Requerimientos Orientada a Aspectos [RMA03] es un modelo para el tratamiento de concerns que enfatiza el tratamiento de crosscutting concerns. Un concern es definido como un conjunto coherente de requisitos, los crosscutting concerns son identificables en matrices donde es posible establecer cuándo algunos de sus requisitos influencian de manera positiva o afectan de manera negativa otros concerns. La AORE provee enfoques para la elicitación y especificación de los concerns y crosscutting concerns en etapas tempranas de desarrollo de software. El modelo asocia los concerns a los requerimientos para ver cuáles crosscutting concerns son requerimientos de los stakeholders y cuáles califican como aspectos candidatos. Este modelo soporta la separación de la especificación de los requerimientos funcionales de los que no y proporciona igual trato para ambos tipos de requerimientos. Reconoce que los requerimientos se entrecruzan e influyen entre sí, por este motivo también proporciona una análisis de cómo un requerimiento no funcional afecta el comportamiento de un conjunto de requerimientos funcionales. Este modelo hace posible establecer tempranamente los puntos de "*trade-off*" entre los requerimientos funcionales, por consiguiente proporciona soporte para la negociación y toma de decisiones futuras entre los stakeholders. El modelo AORE reduce el peligro de cambios no esperados a través de la identificación temprana de aspectos y resolución de conflictos. La aproximación desarrollada por el modelo AORE (Figura 2.4), es un proceso determinado por cinco actividades [TAA07]:

1- Identificar y especificar concerns: Para realizar esta actividad se utilizan técnicas de separación de requisitos como: viewpoints, casos de usos, y orientación a objetivos, entre otras., que permitan identificar desde el problema los concerns y sus requisitos.

2- Identificar las relaciones entre concerns y requerimientos de los stakeholders de granularidad gruesa: Se buscan las posibles influencias, positivas o negativas, entre los concerns. Los concerns son relacionados entre si en una matriz en cuyas intersecciones señala la influencia de un concerns con respecto a otro.

3- Especificar las proyecciones de los concerns usando reglas de composición: Se especifican las relaciones entre los concerns en base en las influencias encontradas entre los requisitos en la actividad anterior. Esto se logra a través de reglas de composición especificadas a través de una estructura XML que permite relacionar los requisitos junto con la acción u operador que lo cualifica dentro del concern.

4- Manejo de conflictos entre concerns: Esta actividad permite identificar si un concern determinado puede contribuir positiva o negativamente con otros concerns.

5- Especificación de las dimensiones de los concerns: Se decide la manera en que los concerns serán soportados en la arquitectura con respecto a las dimensiones de influencia y mapeo:

- **Influencia:** Indica en cuáles etapas del ciclo de vida afectarán los concerns. Son particulares a cada proyecto o son determinadas por los desarrolladores.
- **Mapeo:** Indica los artefactos en los cuáles serán correlacionados los concerns.

El resultado de este proceso es un conjunto de aspectos candidatos, es decir, qué concerns atraviesan múltiples requerimientos. Una concreta instanciación del modelo AORE, usando viewpoint y un lenguaje de composición basado en XML, es la herramienta llamada Arcade [CRS05].

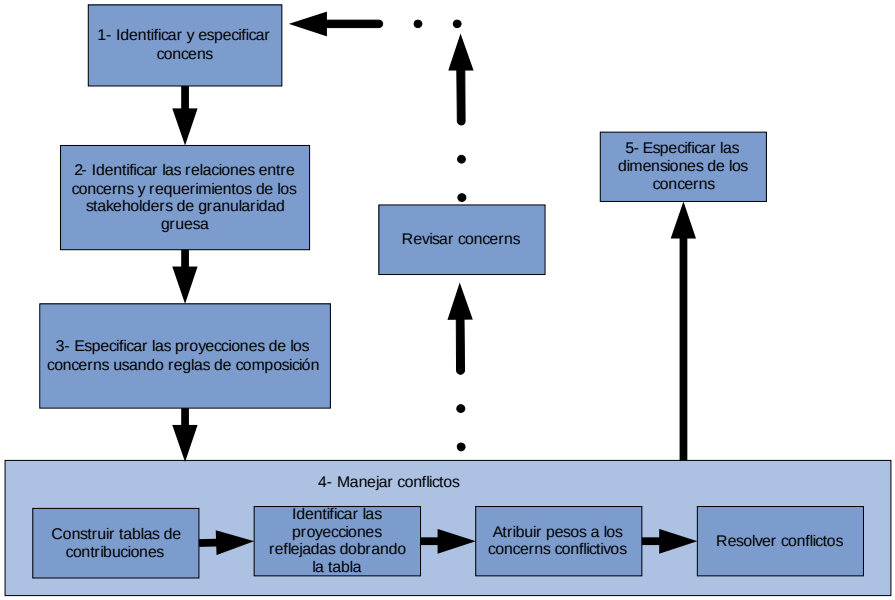

Figura 2.4: Modelo de Ingeniería de Requerimientos Orientada a Aspectos

La capacidad de AORE resalta la identificación y análisis de concerns, provee características para:

- Disminuir la complejidad de desarrollo.
- Disminuir el riesgo en el diseño e implementación a partir de la solución de conflictos.
- Controlar y soportar la trazabilidad de requisitos al poder inferir vínculos de trazado identificando su influencia y mapeo en posteriores artefactos o fases del ciclo de vida.

2.4.2 Theme/Doc

Theme/Doc es un enfoque que cubre el ciclo de vida completo usando una tecnología de separación de concerns. Theme/Doc esta basado en la noción de theme, un theme representa una característica del sistema [CW01]. El theme es un elemento de diseño, es una colección de estructuras y comportamientos. Múltiples themes pueden ser combinados para formar el sistema. El modelo tiene dos clases de themes, los theme base, los cuales pueden compartir la misma

estructura y comportamiento con otros themes mientras modelan su propia perspectiva, y los theme crosscutting, los cuales tienen un comportamiento que cubre la funcionalidad de los theme base. Los themes crosscutting son aspectos [CW01].

Cuando se necesita identificar aspectos dentro de un conjunto de requerimientos, es necesario ver los comportamientos descriptos en los requerimientos y sus relaciones con otros requerimientos. El enfoque Theme/Doc proporciona vistas de las especificaciones y las relaciones entre los comportamientos de los requerimientos [CW01]. Este enfoque es soportado por la herramienta Theme/Doc, esta herramienta opera sobre la suposición de que si dos comportamientos están descriptos en el mismo requerimiento, entonces están relacionados. Los comportamientos se pueden relacionar en tres formas: pueden estar relacionados erróneamente o coincidentemente, significando que el requerimiento puede ser re-escrito de modo que ellos no estuvieran fuertemente acoplados, ellos pueden estar relacionados jerárquicamente, en donde un comportamiento es un sub comportamiento de otro, o pueden estar relacionados por crosscutting, significando que el requerimiento describe un comportamiento como un aspecto de otro [BS06].

La herramienta Theme/Doc provee vistas que exponen qué comportamientos están colocados en los requerimientos. Estas vistas asisten al desarrollador en determinar qué clase de relaciones existen entre los comportamientos, y si aquellos comportamientos son base o aspectos [CW01]. Theme/Doc consiste en tres subprocesos [BS06]:

1- Identificación de acciones y entidades: El punto de comienzo para el desarrollador es identificar un conjunto de acciones claves en los documentos de los requerimientos. El conjunto de acciones es un conjunto del comportamiento descripto, más que una lista de todos los verbos de los documentos. Se genera una vista llamada *"action view",* la cual muestra todas las relaciones entre las acciones, si dos acciones son mencionadas en el mismo requerimiento, son linkeadas en la vista.

2- Categorización de acciones dentro de los themes: No todas las acciones son modeladas como características separadas del sistema, algunas acciones son sub-comportamientos de otras acciones. El desarrollador simplifica la vista decidiendo qué acciones son más importantes y cuáles no. Esto esencialmente señala qué acciones se convertirán en themes, y cuáles se convertirán en métodos dentro de themes.

3- Identificación de crosscutting themes: Este subproceso usa la vista "*major action view",* que es una vista que se usa para ayudar a determinar cuáles themes son base y cuáles son aspectos.

La herramienta toma como entrada la documentación de requerimientos en forma textual y palabras claves provistas por el ingeniero de sistemas que representan las acciones. Luego, se genera una matriz con los requerimientos y las acciones, y la herramienta permite identificar qué acciones afectan a qué requerimiento. Si una acción es linkeada con más de un requerimiento es reflejada como un potencial crosscutting concern. Los ingenieros de requerimientos deben revisar los links, reagrupar las palabras y los requerimientos y eliminar los links entre acciones. Este proceso en primer lugar agrupa los requerimientos en funcionalidad base por un lado y funcionalidad secundaria o aspectual por otro y los organiza de acuerdo al orden de composición. Finalmente, algunos themes son encapsulados en aspectos funcionales, mientras que otros themes en requerimientos individuales.

2.4.3 Exploración de Aspectos en Requerimientos

El enfoque Exploración de Aspectos en Requerimientos (Mining Aspects in Requirements) [SLRR05] tiene como principal objetivo determinar aspectos candidatos potenciales en documentos de requerimientos independientemente de cómo estén estructurados estos documentos utilizando técnicas NLP (Natural Language Processing) [GR97] que proveen soporte para el análisis sensible del contexto de los requerimientos. La técnica propuesta describe cómo diferentes fuentes de requerimientos pueden ser automáticamente exploradas

para ayudar a los ingenieros en requerimientos a identificar rápidamente y construir un modelo orientado a aspectos estructurado de los requerimientos del sistema [SRR05]. Esta herramienta de exploración de aspectos utiliza características provistas por otra herramienta llamada WMATRIX [RA03], que automatiza la extracción de verbos y sustantivos desde el texto. Los requerimientos pueden ser obtenidos desde diferentes fuentes y pueden estar expresados en muchas formas, desde descripciones de lenguaje natural a métodos más estructurados tales como casos de uso, view points y especificaciones formales. La herramienta permite un desarrollo acelerado porque no impone una clase de formato específico y no depende del conocimiento previo del analista sobre los requerimientos. La herramienta permite al desarrollador realizar una exploración rápida de los requerimientos y lograr un entendimiento global de ellos, por ejemplo usando características semi-automáticas, para producir un modelo intermedio usando view points.

Las principales ventajas de este enfoque son [SLRR05]:

- Se puede trabajar con cualquier clase de documento textual, ya que no se tiene en cuenta su estructura.
- La actividad consume poco tiempo, porque es un enfoque semi-automatizado.

La Figura 2.5 muestra el enfoque para identificar y representar aspectos en los documentos de los requerimientos [SLRR05]:

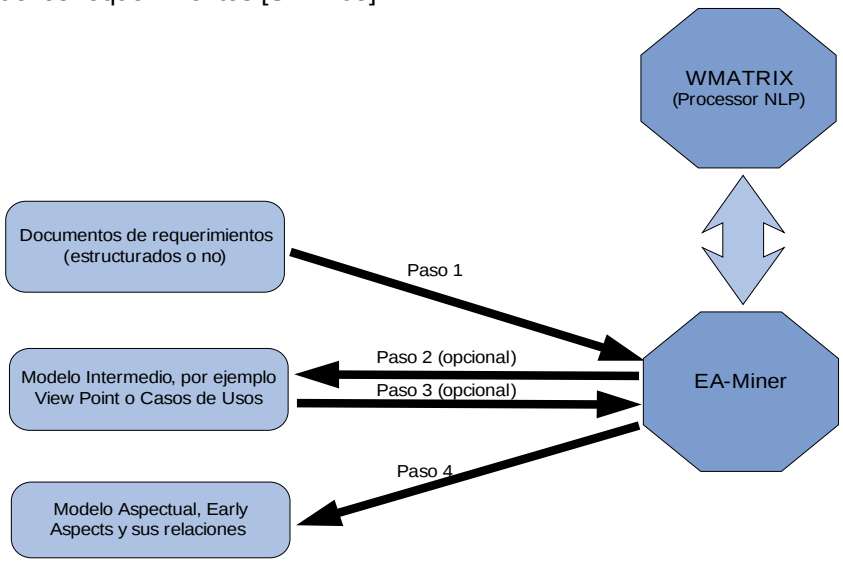

Figura 2.5: Modelo de Exploración de Aspectos en Requerimientos

El enfoque comienza analizando los documentos existentes que son fuente de requerimientos (Paso 1). La herramienta de exploración lee estos archivos y los pasa a WMATRIX (Paso 2), que puede realizar un análisis que ayude a identificar concerns y view points usando técnicas de procesamiento de lenguaje natural. La herramienta de exploración permite ajustar a medida la información que fluye desde WMATRIX, por ejemplo, mostrando sólo un subconjunto de action words (verbos identificados por WMATRIX), basados en algún criterio. Al final de este paso opcional puede ser producida como salida una especificación de requerimientos más estructurada. El siguiente paso es leer directamente desde archivos informales, o de modelos intermedios, y procesar la información a fin de buscar requerimientos crosscutting y aspectos candidatos (Paso 3). La herramienta de exploración tiene el rol de setear el criterio que será usado por WMATRIX para procesar la información. Además, la herramienta de exploración puede filtrar los resultados para identificar cuáles son los aspectos candidatos y producir un modelo que represente las relaciones entre los requerimientos. La herramienta de exploración (por medio de WMATRIX) examina la dispersión de aspectos candidatos a través de

los requerimientos (Paso 4). Si un aspecto candidato está disperso, por ejemplo, aparecerá en muchos requerimientos, entonces tal aspecto candidato es muy fuerte [SLRR05]. De este modo, el enfoque para la exploración de aspectos puede ser usado a pesar de la estructura de los documentos textuales provistos como entrada. La herramienta permite un desarrollo acelerado porque no impone una clase de formato específico y no depende del conocimiento previo del analista sobre los requerimientos.

La herramienta Early-AIM es una herramienta de soporte para ayudar a los desarrolladores a inspeccionar automáticamente y modelar los crosscutting concerns sin tener que leer previamente los documentos de los requerimientos. El principal objetivo de Early-AIM es determinar aspectos candidatos potenciales en los documentos de los requerimientos independientemente de cómo estén estructurados [SRR05]. EA-Miner es una herramienta para la identificación de aspectos en el nivel de los requerimientos, la principal motivación de EA-Miner es proveer soporte automatizado para [SR08]:

- **Explorar aspectos tempranos en el nivel de los requerimientos**: La herramienta soporta la identificación de aspectos tempranos desde varios documentos relacionados con los requerimientos. La herramienta identifica los aspectos tempranos funcionales y no funcionales, y provee mecanismos para visualizarlos.

- **Estructura diferentes modelos de AORE:** EA-Miner ayuda a estructurar los aspectos tempranos y otros conceptos tales como viewpoints en el enfoque AORE. EA-Miner posee características para editar las abstracciones identificadas (por ejemplo: add, remove, filter) tales como la generación de los requerimientos.

2.4.4 Buscando Aspectos en la Especificación de los Requerimientos

El enfoque Buscando Aspectos en la Especificación de los Requerimientos (Identifying Crosscutting Concerns in Requirements Specifications) [RO04] se enfoca en la identificación de requerimientos e influencias crosscutting en documentos de requerimientos ya existentes. Un requerimiento es una clase especial de concern. Los requerimientos que atraviesan transversalmente a otros son referidos como requerimientos crosscutting. La expresión *"influencia crosscutting"* es utilizada como un sinónimo para las relaciones entre dos requerimientos que está establecida por uno atravesando transversalmente el otro. Sin embargo, no todas las dependencias de requerimientos son de naturaleza crosscutting.

Se sugieren dos técnicas con el propósito de explorar requerimientos e influencias crosscutting a partir de las especificaciones de requerimientos: la primera es manual y la segunda es semi-automática.

- **Identificación a través de inspección:** Después que ha sido escrita una especificación de requerimientos o alguna parte de la misma deberá ser inspeccionada para identificar y documentar cualquier tipo de deficiencia, por ejemplo omisiones, inconsistencias, etc. Es recomendable comenzar desde un requerimiento no funcional dado, y probablemente crosscutting, por ejemplo tiempo de respuesta, y entonces buscar en el texto por manifestaciones mencionando alguna respuesta del sistema. Si tal manifestación ha sido encontrada, el analista lo anota. Después de que ha terminado la inspección, las manifestaciones colectadas deben ser analizadas para ver si reflejan requerimientos restringidos por el requerimiento de tiempo de respuesta. Si es así, ha sido localizada una influencia crosscutting y será documentada. Si al menos se ha encontrado una influencia crosscutting, el requerimiento de tiempo de respuesta es marcado como un requerimiento crosscutting.

- **Identificación soportada por técnicas de recuperación de información:** Es posible soportar la identificación de requerimientos e influencias crosscutting por medio de técnicas de recuperación de información. Un programa debe, por ejemplo, encontrar manifestaciones que parezcan estar afectadas por un requerimiento que el analista

asume como crosscutting. En una manera similar como la anterior, el analista debe entonces examinar cada manifestación y documento cuando una influencia crosscutting ha sido encontrada. Otra vez, registrar al menos una influencia crosscutting lleva a que la caracterización del requerimiento sea un crosscutting.

2.4.5 Aspects Extractor

Aspect Extractor [HD05] es un enfoque en el contexto de ingeniería de requerimientos, que identifica, especifica, integra y evalúa aspectos en etapas tempranas del ciclo de desarrollo del software. Para lograr estos objetivos, se automatizan las tareas que conforman el modelo de ingeniería de requerimientos. Esta automatización se realiza sobre la información de la funcionalidad del sistema brindada por el analista y se obtiene una lista de aspectos candidatos. El analista decide cuáles son los aspectos definitivos a partir de tal lista. Esta automatización contribuye al trabajo del analista, ya que no debe inspeccionar la especificación de casos de uso de un sistema en la búsqueda de aspectos. Además, una inspección manual sería un trabajo arduo en el cual muchos detalles importantes, por ejemplo crosscutting concerns, podrían ser pasados por alto por falta de tiempo, fatiga o monotonía. El modelo de ingeniería de requerimientos esta conformado por cinco tareas principales:

1- Identificar concerns: Consiste en identificar a partir de los requerimientos, los posibles concerns de un sistema. En esta tarea se genera automáticamente información de interés que se utilizará en las tareas restantes.

2- Elección de aspectos candidatos: En esta tarea se identifican los crosscutting concerns y los functional crosscutting concerns. Una vez identificados, el analista seleccionará algunos de ellos de acuerdo a su experiencia e intuición. Estos concerns pasan a ser aspectos candidatos.

3- Especificar aspectos candidatos: El analista debe describir el objetivo y la funcionalidad de los aspectos candidatos seleccionados en la tarea anterior. Además, el sistema determina con cuáles elementos del modelo está relacionado un determinado aspecto candidato.

4- Identificar conflictos: Se identifican todas las posibles situaciones conflictivas que se dan cuando un elemento del modelo es afectado por más de un aspecto candidato. Si existe dicha situación, entonces será indicada de forma que pueda ser tratada posteriormente por el analista.

5- Modelar en UML: Basados en la información obtenida en la realización de las tareas previas, se construirá un modelo visual, con el agregado de los aspectos finalmente seleccionados por el analista y las posibles situaciones conflictivas entre los aspectos candidatos.

El enfoque presentado, Aspects Extractor, es automatizado a través del desarrollo de una herramienta denominada Aspects Extractor Tool. Dicha herramienta permite especificar la funcionalidad del sistema, identificar los aspectos candidatos, y finalmente, visualizar la especificación con los aspectos candidatos identificados.

La especificación de la funcionalidad del sistema puede ser realizada de dos maneras: por medio de las interfaces gráficas que provee Aspects Extractor Tool o con alguna otra herramienta CASE al estilo Rational Rose. Los aspectos candidatos son identificados a partir de la implementación de una heurística con la utilización de la técnica de recuperación de información. La especificación del sistema con los aspectos candidatos identificados pueden visualizarse de dos maneras diferentes, al igual que para el caso de la entrada. La primera es mediante la utilización de Aspects Extractor Tool, la cual brinda la posibilidad de observar en una serie de tablas los diferentes aspectos candidatos finalmente escogidos, como así también las posibles situaciones conflictivas. La segunda, es mediante un archivo con extensión XMI generado por Aspects Extractor Tool, el cual podrá importarse utilizando una determinada herramienta CASE.

2.5 Análisis de los enfoques de identificación de aspectos presentados

Se realizó una tabla para comparar los distintos enfoques presentados (Tabla 2.1). Esta tabla esta basada en [BTA04], los criterios para formar la tabla son los siguientes:

Fase: Indica en que parte del ciclo de vida del desarrollo de software el enfoque es usado.

Objetivo: Objetivo del enfoque presentado.

Artefactos: Artefactos que son usados para lograr el objetivo del enfoque.

Heurísticas: Heurísticas o procesos que define el enfoque para lograr su objetivo.

Herramienta de soporte: Herramientas que soportan el enfoque.

	Fase	Objetivo	Artefactos	Heurística	Herramienta de Soporte
AORE	Análisis de Req.	Identificación Especificación Evaluación	Concerns Requerimientos Matriz relacionando los concerns y los requerimientos Reglas de Composición	Reglas de Composición Soporte de proceso para identificar y especificar aspectos Distingue aspectos de viewpoints	ARCADE -Aspectual Requirements Composition and Decision
Theme/Doc	Análisis de Req.	Identificación Especificación	Requerimientos textuales Acciones Entidades Crosscutting Themes Base Themes	Action View Soporte de reglas y de procesos Para identificar aspectos	Theme /Doc
Exploración de Aspectos en Req.	Análisis de Req.	Identificación	Documentos independientes de su estructura	Técnicas LNP	EA-Miner Usa características de WMATRIX
Buscando Aspectos en la esp. de los req.	Análisis de Req.	Identificación	Documentos de req. Concerns Corsscutting concerns	Inspección Recuperación de Información	(en desarrollo)
Aspect Extractor	Análisis de Req.	Identificación Especificación Integración Evaluación	Interfaz gráfica para Casos de Uso Importación de archivos XMI	Técnica de recuperación de información: Stop Words y Stemming	Aspect Extractor Tool

Tabla 2.1: Comparación de los enfoques de identificación de aspectos

Algunos puntos destacados en la comparación y análisis de los enfoques para la detección de aspectos tempranos presentados son:

- El enfoque Theme/Doc provee una herramienta semi-automática de identificación crosscutting concerns en especificaciones de requerimientos. El proceso de identificación es basado en un análisis léxico en documentos de los requerimientos, los cuales buscan en los documentos palabras claves provistas por el desarrollador. La herramienta automáticamente produce una vista gráfica de mapas de relaciones entre crosscutting concerns, los cuales pueden ayudar al desarrollador a identificar los aspectos candidatos. Un problema principal es que el analista tiene que leer todos los documentos de requerimientos para ingresar las palabras claves, lo cual consume

tiempo y no es posible de realizar en sistemas complejos o proyectos con poco tiempo [SLRR05].

- Theme/Doc es el enfoque más adecuado cuando se requiere encontrar todas las influencias crosscutting y modelar sus relaciones. Pero se deben leer los documentos de los requerimientos para el ingreso de las palabras claves.
- Si sólo se esta interesado en una identificación rápida de crosscutting concerns, entonces es más adecuado el enfoque basado en técnicas de recuperación de información.
- Los enfoques Buscando Aspectos en la Especificación de los Requerimientos y Exploración de Aspectos en Requerimientos son los únicos que no requieren los documentos de entrada en un formato específico.
- La mayoría de los enfoques de exploración de aspectos en requerimientos no proveen una aproximación efectiva cuando los requerimientos son complejos o no estructurados [SLRR05].
- Todos los enfoques tienen como objetivo la identificación de los aspectos, unos pocos tienen como objetivo la especificación y solo algunos tratan la evaluación de los aspectos.
- No existe un modelo uniforme de concerns. Aunque la noción de concern parezca ser el tema común para todos los métodos, los artefactos adoptados para identificar los aspectos son diferentes.
- Las heurísticas usadas por los enfoques difieren entre sí. Aunque la mayoría de ellos definen procesos explícitos, las heurísticas adoptadas están generalmente definidas implícitamente o dependen de la intuición del analista. Esto no sólo reduce la entendibilidad de los enfoques, sino que también hace que la interpretación de los métodos sea subjetiva.
- Mientras AORE con Arcade se centra principalmente en el alcance del tratamiento de los concerns no funcionales, Theme/Doc se centra sobre el comportamiento (principalmente funcional) del tratamiento de los concerns [CSRSS06].

2.6 Conclusión

Cuando se habla de orientación a aspectos, se usan conceptos basándose en la propia experiencia e intuición. Sin embargo, son necesarias definiciones precisas para facilitar la utilización de herramientas, tales como herramientas de minería o de identificación de crosscutting concerns. El DSOA tiene la necesidad de un framework que unifique los conceptos que determinan los crosscutting concerns.

El DSOA enfrenta la problemática de que los enfoques existentes para la identificación, especificación, integración y evaluación de los crosscutting concerns no son completamente apropiados. Existen varias implementaciones de estos enfoques, pero ninguno satisface de forma completa las principales necesidades de los ingenieros de software. Los métodos actuales poseen deficiencias tales como la falta de precisión en los resultados obtenidos, necesidad de un conocimiento previo del proyecto por parte del ingeniero de software, necesidad de documentos con algún formato específico, es tedioso el trabajo con proyectos de grandes volúmenes de información. En el DSOA es necesaria una identificación rápida y precisa de los crosscutting concerns para lograr una mejor calidad en la aplicación desarrollada, reduciendo los costos de desarrollo y mantenimiento, y evitando complicaciones que puedan ocurrir en el futuro.

Se puede concluir que los problemas que enfrenta el DSOA son:

- La falta de unificación de conceptos en los términos que representan a los crosscutting concerns en las aplicaciones de software.

- Necesidad de un enfoque que agilice la identificación, especificación, integración y evaluación de los crosscutting concerns en etapas tempranas del desarrollo de software.

De los problemas presentados para el DSOA, se puede concluir que actualmente existe la necesidad de un enfoque para la identificación de los aspectos candidatos que trabaje con términos unificados en el dominio DSOA. De esta manera, el enfoque ofrecerá resultados de forma adecuada y acertada. El nuevo enfoque es una herramienta clave para el descubrimiento de los concerns en las aplicaciones de software y presenta un papel de conciliador de conceptos en el DSOA. La definición de una ontología para aspectos sería un camino viable para resolver estos problemas.

Capítulo 3: Ontología

Una definición del término ontología es: *"Una ontología es una especificación de una conceptualización"* [GR93]. La conceptualización es una forma abstracta y simplificada de ver el mundo que se quiere representar con algún propósito. Se intenta formular un exhaustivo y riguroso esquema conceptual dentro del dominio dado con la finalidad de facilitar la comunicación y poder compartir información entre diferentes sistemas. Entonces, al realizar una ontología, lo que se intenta hacer es representar una visión del mundo con respecto a un dominio.

Como definición más detallada, una ontología es un modelo de datos que representa un conjunto de conceptos de un dominio y las relaciones entre estos conceptos. Las ontologías catalogan y definen los tipos de conceptos que existen en un cierto dominio, así como sus relaciones y propiedades. Las ontologías explícitas se pueden expresar de muchas maneras, pero como mínimo, deben incluir un vocabulario de términos, con la definición de cada uno [AB05]. El objetivo de las ontologías es el de obtener conocimiento consensuado de una manera genérica y formal para ser reutilizado y compartido.

Por ejemplo, una ontología para representar el mundo musical usará conceptos como *"Instrumento de Cuerda", "Instrumento de Metal"*, etc., y relaciones como *"Un Instrumento de Metal es un Instrumento de Viento"*, etc. (Figura 3.1), que indica que los *"Instrumentos Musicales"* pueden ser *"Instrumentos de Viento"* o *"Instrumentos de Cuerda"*, los *"Instrumentos de Metal"* e *"Instrumentos de Viento"* son una clasificación de *"Instrumento de Viento"*, y, los *"Instrumentos de Traste"*, al igual que los *"Instrumentos sin Traste"* son clasificados como *"Instrumentos de Cuerda"*.

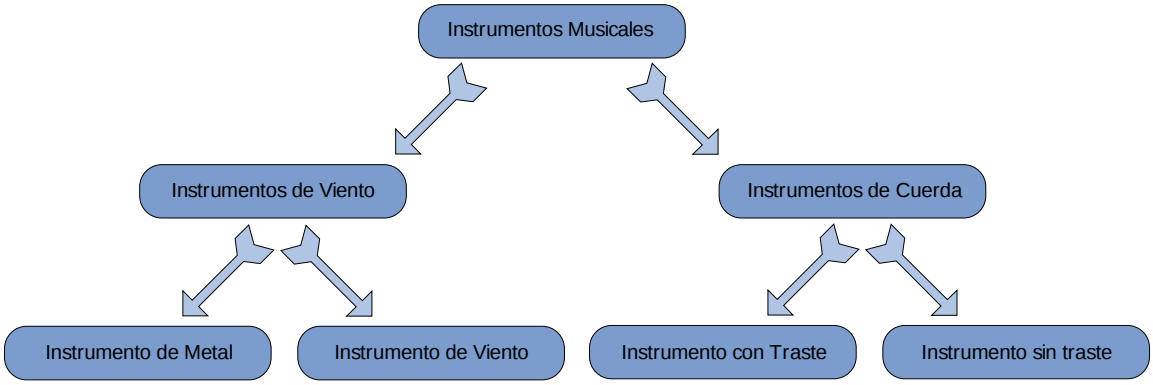

Figura 3.1: Ejemplo de ontología

Un dominio es un área de temática específica o un área de conocimiento, tal como medicina, fabricación de herramientas, bienes inmuebles, reparación automovilística, gestión financiera, etc. [WC304]. Las ontologías incluyen definiciones de conceptos básicos del dominio, y las relaciones entre ellos, que son útiles para los sistemas, codifican el conocimiento de un dominio específico y también 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.

Una ontología debe representar conocimiento de un dominio claramente especificado para así proporcionar un entendimiento compartido de dicho dominio, además de destacar la claridad, la coherencia y la extensibilidad. En los sistemas basados en el conocimiento, lo que existe es exactamente lo que se puede representar, y lo que se representa, se conoce con el

nombre de Universo de Discurso. El Universo de Discurso de una ontología es el conjunto de objetos que están representados en ella y sobre los cuales se puede hablar y razonar [PH03].

Un beneficio principal cuando una ontología es creada es:*"Compartir el entendimiento común de la estructura de información entre personas y sistemas de software"* [GR93]. 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. Las ontologías se usan para lograr la interoperabilidad entre sistemas informáticos. Dos sistemas son ínter-operables si pueden trabajar conjuntamente de una forma automática, sin esfuerzo por parte del usuario, y esto se logra traduciendo los términos usados por una aplicación a otra. A medida que aumenta el número de aplicaciones que deben ínter-operar, se hace necesario el uso de ontologías traductoras.

Las ontologías ayudan a la especificación de los sistemas de software. Como la falta de un entendimiento común conduce a dificultades en identificar los requisitos y especificaciones del sistema que se busca desarrollar, las ontologías facilitan el acuerdo entre desarrolladores y usuarios, donde todos sabrán cuál es el significado de cada término del que se esta hablando. Otra de las características sobresalientes de las ontologías, es su capacidad de reutilización. Es posible establecer correspondencias entre las clases definidas en ontologías diversas y, de este modo, poder reutilizarlas. Por medio de las ontologías se pueden clasificar los datos debido a que la información se puede abstraer en conceptos bien determinados, y gracias a una ontología desarrollada para un dominio específico, la búsqueda de información dentro de ese dominio puede arrojar resultados mucho más concretos. Las ontologías pueden emplear el poder de razonamiento automático para guiar a información que se no se encuentra directamente representada en la ontología. Por ejemplo, si se define una relación *"está_compuesto"* entre la clase *"computadora"* y la clase *"dispositivo_de_entrada"*, y además existe una relación de herencia entre la clase *"dispositivo_de_entrada"* y la clase *"teclado"*, se podrá deducir que un teclado es una parte de la computadora, de forma que se sabrá que la instancia *"teclado_inhalambrico"* de la clase *"teclado"* es, además de un dispositivo de entrada, una parte de la computadora. Los beneficios de utilizar ontologías se pueden resumir de la siguiente manera [FSZ07]:

- Proporcionan una forma de representar, compartir y reutilizar el conocimiento utilizando un vocabulario común.
- Permiten usar un formato de intercambio de conocimiento.
- Proporcionan un protocolo específico de comunicación.
- Permiten la reutilización del conocimiento.

3.1 Componentes de una ontología

Las ontologías tienen los siguientes componentes que servirán para representar el conocimiento de algún dominio [GR93]:

- **Conceptos o Clases (Concepts):** Son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento, etc. Las clases en una ontología se pueden organizar en taxonomías a las que se puede aplicar los mecanismos de herencias.
- **Relaciones (Relations):** Las relaciones representan la interacción y enlace entre los conceptos de un dominio. Se definen formalmente como cualquier subconjunto de un producto de n conjuntos: R: C1 x C2 x ... x Cn. Ejemplos de relaciones pueden ser subclase-de, conectado-a, parte-de, etc.

- **Funciones (Functions):** Son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología. Formalmente, se definen las funciones F como: F: C1 x C2 x ... x Cn-1 -> Cn. Por ejemplo, pueden aparecer funciones como: asignar-fecha, categorizar-clase, etc.

- **Instancias (Individuals):** Las instancias se utilizan para representar objetos determinados de un concepto.

- **Axiomas (Axioms):** Son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Por ejemplo: *"Si A y B son de la clase C, entonces A no es subclase de B"*, *"Para todo A que cumpla la condición C1, A es B"*, etc. Los axiomas formales sirven para modelar sentencias que son siempre ciertas. Normalmente se usan para representar conocimiento que no puede ser formalmente definido por los componentes descriptos anteriormente. Los axiomas son sentencias expresadas en lógica de primer orden, representan el conocimiento en forma declarativa y rigurosa. Mediante los axiomas se tiene mayor capacidad expresiva del dominio almacenado. Tienen dos papeles en la definición de una ontología:

  - Representan el significado total o parcial en la descripción de una ontología.

  - Dentro del alcance del conocimiento representado declarativamente, responden a las consultas sobre la capacidad de la ontología y del conocimiento creado a partir de los conceptos en la ontología.

La categorización define clases y las relaciones entre ellas. Los elementos concretos son instancias de esas clases, los conceptos de la ontología. La relación básica entre términos es la de herencia, donde una clase A, subclase, es un tipo de la clase B, superclase, por lo que posee todas sus características.

3.2 Clasificación y tipos de ontologías

Las ontologías pueden ser clasificadas según su formalidad en cuatro grupos: informales, semi-informales, semi-formales y formales.

Las ontologías informales se expresan directamente en cualquier lenguaje natural. Las semi-informales se expresan en una forma estructurada y restringida de algún lenguaje natural. Las semi-formales se expresan en lenguajes estructurados y las ontologías formales definen los términos mediante lenguajes lógico-matemáticos cuyos símbolos se definen exactamente y sin ambigüedades, permiten emplear teoremas y demostraciones. Los dos últimos tipos de ontologías permiten que las aplicaciones puedan usar las definiciones de los conceptos del dominio y sus relaciones. Así como los tres primeros tipos de ontologías pueden contener términos ambigüos o inconsistentes, las ontologías formales no los permiten.

Una ontología puede ser clasificada en diferentes tipos de acuerdo al nivel de dependencia de una tarea particular o de punto de vista [GU04]. La especialización de los tipos de ontologías se muestra en la Figura 3.2.

Ontologías de alto nivel (Top-level ontologies): Una ontología de alto nivel describe varios conceptos generales como espacio, tiempo, objeto, evento, acción, etc., los cuales son independientes de un problema o dominio particular.

Ontologías de dominio (Domain ontologies): Este tipo de ontología describe el vocabulario relacionado a un dominio particular como medicina o automóviles.

Ontologías de tarea (Task ontologies): Describe el vocabulario relacionado con una tarea o actividad particular para especializaciones de términos de una ontología de alto nivel. Se podría pensar que al implementar ontologías de este tipo, se debería tratar de representar todo el conocimiento disponible de cada clase en cuestión, pero realmente esto no es apropiado [NM01], sino que es recomendable representar sólo el conocimiento que es realmente necesario para la

aplicación, de otra manera sería más ineficiente su utilización, porque se comparte conocimiento innecesario y los procesos de inferencia son más lentos.

Ontología de aplicación (Application ontologies): Una ontología de aplicación describe conceptos que dependen de ambos: dominio particular y tarea particular.

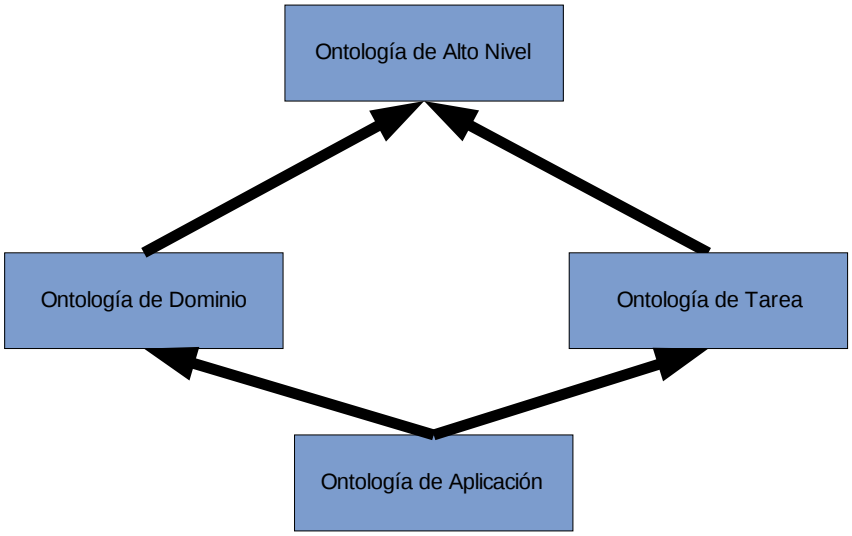

Figura 3.2: Tipos de ontologías

3.3 Razonamiento automático

Las ontologías resultan muy útiles para facilitar el razonamiento automático, es decir, sin intervención humana. Partiendo de reglas de inferencia, un motor de razonamiento puede usar los datos de las ontologías para inferir conclusiones de ellos [AB05]. Por ejemplo, se establecen las siguientes reglas: *"Todos los ríos desembocan en un mar, en un océano o en un lago"* y *"Si el curso de un río termina en una población, esa población está junto al mar, océano o lago donde desemboca"*, las máquinas pueden hacer deducciones cómo la que es mostrada en la Figura 3.3.

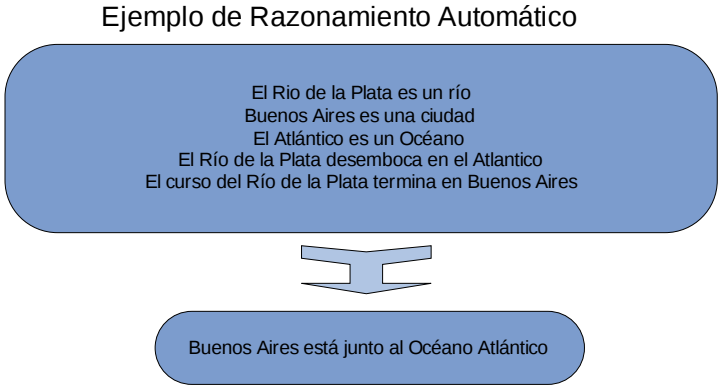

Figura 3.3: Ejemplo de razonamiento automático

Las reglas de inferencia son esquemas para construir razonamiento válido, la aplicación de una regla de inferencia es un procedimiento puramente sintáctico. Los axiomas, junto con la herencia de conceptos, permiten inferir conocimiento que no esté indicado explícitamente en la taxonomía de conceptos. Las relaciones semánticas que se definen entre las clases de una ontología pueden considerarse predicados que, junto a un conjunto de reglas de inferencia,

permiten construir software que realice razonamiento automático. Al tener la ontología construida, junto con las reglas de inferencia y los axiomas, se procede a la creación de las instancias de los conceptos que son las que generan conocimiento dentro del dominio. Una instancia es la que representa objetos determinados del dominio y permite inferir nuevo conocimiento a partir del explícito.

Utilizando un razonador o motor de reglas de inferencias es posible obtener conocimiento no explícito en el modelo original de la ontología, permitiendo además realizar consultas sobre el modelo. Algunos de los motores principales de reglas de inferencia son [ME07]:

- **Bossam:** Soporta razonamiento sobre ontologías en OWL.
- **Hoolet:** Hoolet es un razonador para ontologías en OWL-DL.
- **Pellet:** Es un razonador Java de código abierto para OWL-DL. Sus características incluyen múltiples interfaces de acceso, razonamiento sobre tipos, respuesta conjuntiva, soporte para reglas, análisis y reparación de ontologías, depurado de ontologías y razonamiento incremental.
- **KAON2:** Infraestructura de razonamiento para ontologías OWL-DL, es bastante veloz y comparable a razonadores como Fact y Racer. Este razonador proporciona una API para su utilización y permite su uso de modo distribuido mediante RMI.
- **RACER:** Es un sistema razonador de Web Semántica, ofrece una interfaz HTTP. No es de código abierto.
- **Jena:** Es un entorno para la Web Semántica de código libre programado en Java. Proporciona una API para extraer y escribir datos de un grafo RDF. Los modelos pueden ser consultados mediante SPARQL (lenguaje de consultas para RDF). Se puede utilizar OWL con Jena. Proporciona varios razonadores internos y se pueden añadir otros. El razonador Pellet se puede conectar directamente y así superar las limitaciones de velocidad. El conjunto de razonadores básico que Jena incluye es OWL Reasoner, DAML Reasoner, RDF Rule Reasoner, Generic Rule Reasoner.
- **FaCT:** El razonador Fact es un clasificador de Lógica Descriptiva (DL).
- **FaCT++:** Versión avanzada del razonador anterior.
- **SweetRules:** Es un conjunto integrado de herramientas para gestionar reglas de Web Semántica y ontologías.
- **Protégé:** Es un entorno Java para OWL. Este razonador permite exportar a otros formatos y puede extenderse mediante una API. Esta concebido como una plataforma básica a la que se le pueden añadir múltiples funcionalidades en forma de plugins.
- **Sesame:** Es una base de datos de código abierto para RDF. Sesame puede ser implementado sobre una gran variedad de sistemas de almacenamiento, incluyendo bases de datos relacionales, memoria, sistemas de archivos, índices por palabras clave, etc.

El motor de inferencia se encarga de ampliar la información del contexto explicitando la información que se encuentra implícita. Para hacer esto hace uso tanto de las relaciones semánticas de la ontología como de las reglas diseñadas especialmente para el dominio.

3.4 Problemas semánticos

Los sistemas informáticos pueden utilizar una ontología para una variedad de propósitos, incluyendo el razonamiento inductivo, la clasificación, y una variedad de técnicas de resolución

de problemas, pero, en cuanto dos sistemas de información intentan comunicarse, aparecen problemas semánticos que dificultan o imposibilitan la comunicación entre ellos. Los problemas semánticos son de dos tipos [AB05]:

- **Problemas de dominio:** Este tipo de problema aparece cuando conceptos similares en cuanto a significado, pero no idénticos, se representan en distintos dominios.
- **Problemas de nombre:** Los problemas de nombre pueden ser de dos tipos: Sinónimos u Homónimos.

Los sinónimos ocurren cuando los sistemas usan distintos nombres para referirse al mismo concepto. Por ejemplo, como se muestra en la Figura 3.4, el Sistema 1 usa un nombre distinto al Sistema 2 para referirse a un mismo concepto del mundo real. En ese caso, se podría usar una ontología que definiera como idénticos los dos términos. Así, las aplicaciones que manejaran esas bases de datos sabrían cómo llevar datos de una a otra.

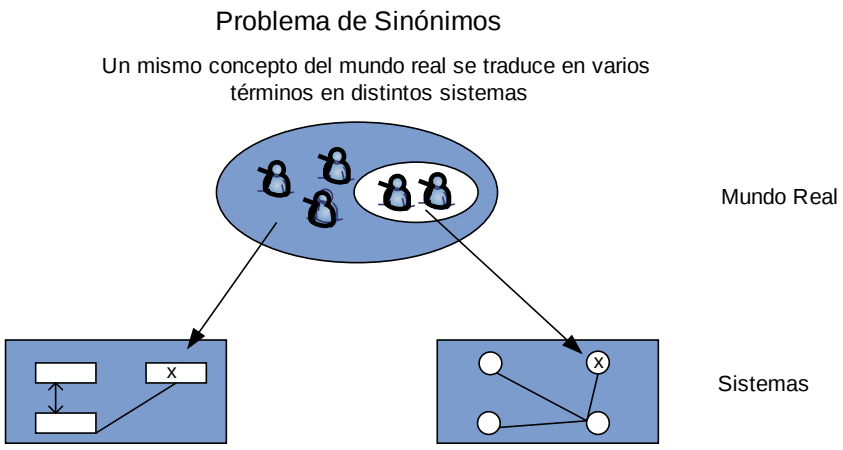

Figura 3.4: Ejemplo de problema por nombre – sinónimo

Los homónimos surgen cuando los sistemas usan el mismo nombre para representar conceptos distintos.

3.5 Diseño de una ontología

Es conveniente que en el diseño de la ontología se tengan en cuentan los siguientes principios [AB04]:

- **Claridad y objetividad:** Una ontología debe poder comunicar de manera efectiva el significado de sus términos. Debe proporcionar al usuario el significado de los términos de forma objetiva y deben explicarse también en lenguaje natural.
- **Coherencia:** Una ontología debe permitir hacer inferencias que sean consistentes con las definiciones.
- **Extensibilidad:** Deben anticiparse nuevos usos para así poder permitir extensiones y especializaciones. Éstas deben poder incluirse en la ontología sin necesidad de revisar las definiciones ya existentes.
- **Especificidad:** Los conceptos se deben especificar a nivel de conocimiento, sin que dependa de una codificación particular a nivel de símbolo.
- **Precisión:** Debe hacerse la menor cantidad de suposiciones acerca del mundo modelado.

- **Completitud:** Las definiciones deben expresarse en términos necesarios y suficientes.
- **Principio de distinción ontológica:** Las clases de una ontología deben ser disjuntas. El criterio usado para aislar el conjunto de propiedades invariantes en una instancia de una clase se denomina criterio de identidad.
- **Diversificación:** Se deben diversificar las jerarquías incluidas para aumentar la potencia de los mecanismos de herencia múltiple.
- **Estandarización:** Se debe intentar usar un vocabulario lo más universal posible.
- **Minimización de la distancia semántica entre conceptos emparentados:** Los conceptos similares estarán agrupados y representados utilizando las mismas primitivas.
- **Hacer explícitos los supuestos de un dominio por medio de la inferencia:** Capacidad de inferir conocimiento implícito.

Uno de los aspectos que debe quedar claro a la hora de comenzar a desarrollar y usar una ontología es su necesidad y finalidad. El desarrollo o en su defecto, la adaptación de una ontología ya existente, debe responder a las necesidades básicas del problema a tratar.

3.6 Lenguajes de implementación de ontologías

Al implementar una ontología es importante decidir primero las necesidades en términos de expresividad y servicios de inferencia, porque no todos los lenguajes permiten representar los mismos componentes de la misma forma. Es frecuente que las traducciones entre lenguajes no sean lo suficientemente precisas por lo que puede perderse información en el proceso de traducción [TA07].

Los primeros lenguajes, estaban basados en lógicas de primer orden y lógicas descriptivas. Estos lenguajes son los más expresivos. Ejemplos: KIF (Knowledge Interchange Format), Ontolingua, Loom, OCML (Operational Conceptual Modelling Language) y FLogic. Los lenguajes más modernos están basados en lenguajes tipo markup y no son tan expresivos como los anteriores. Ejemplos: SHOE (Simple HTML Ontology Extensión), XML (eXtensible Markup Language), XOL (Ontology Exchange Language), RDF (Resource Description Framework), RDFS (RDF Schema), OIL (Ontology Inference Layer), DAML (DARPA Agent Mark-Up Language) + OIL, OWL (Ontology Web Language) [TA07].

Estos lenguajes tienen una semántica bien definida y permiten la manipulación de taxonomías complejas y de relaciones lógicas entre entidades en la Web. Todos ellos pueden “traducirse” al lenguaje RDF y a su complemento RDFS. Actualmente, existen editores que permiten generar el código de una ontología en diferentes formatos, como OILed o Protégé. No todos los lenguajes permiten el mismo nivel de expresividad a la ontología construida ni tampoco ofrecen las mismas funcionalidades [AB04].

A la hora de elegir un lenguaje para la definición de una ontología deben tenerse en cuenta los siguientes aspectos:

- El lenguaje debe poseer una sintáxis bien definida para poder leer con facilidad la ontología creada.
- Debe tener una semántica bien definida para comprender perfectamente el funcionamiento de la ontología.
- Debe tener suficiente expresividad para poder capturar varias ontologías.
- Debe ser fácilmente mapeable desde/hacia otros lenguajes ontológicos.
- Debe ser eficiente a la hora de realizar razonamiento.

En la Figura 3.5 se muestra la estructura de los lenguajes de implementación de ontologías según la expresividad que poseen para definir los conceptos y sus relaciones. El lenguaje XML es el más restrictivo, posee una sintáxis muy superficial, los siguientes lenguajes se hacen cada vez más expresivos. XML Schema permite trabajar con tipos de datos, RDF proporciona la posibilidad de agregar semántica a los documentos y RDF Schema permite la descripción de propiedades y clases. A continuación se describe brevemente cada uno de ellos.

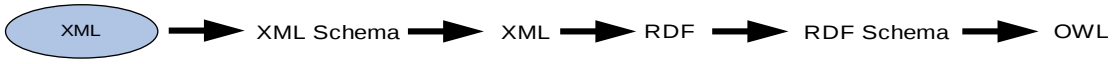

Figura 3.5: Estructura de los lenguajes

- **XML (Extensible Markup Language):** Es una sintáxis superficial para documentos semiestructurados. XML es un estándar de representación que proporciona la sintáxis elemental para expresar metadatos sobre cualquier recurso. Define una estructura en árbol para un documento de manera que las hojas del mismo contienen la información. La estructura y la semántica de un documento XML están entrelazadas.

- **XML Schema:** XML Schema es un lenguaje que restringe la estructura de XML. Además, le proporciona la capacidad de manejar tipos de datos.

- **RDF (Resource Description Framework):** El RDF provee un medio de agregar semántica a un documento sin referirse a su estructura. RDF es una aplicación XML recomendada como estándar por la W3C. Este lenguaje sirve para etiquetar metadatos, la interpretación de este lenguaje es semántica. En el caso de RDF es fundamental utilizar palabras que transmitan un significado inequívoco con el fin de que las aplicaciones entiendan el enunciado para un procesamiento correcto. Este significado se expresa a través de un esquema. Se puede ver a un esquema como una especie de diccionario que define los términos que se utilizarán en una declaración o sentencia RDF para otorgarle significados específicos. Con RDF se pueden utilizar una gran variedad de formas de esquema, incluyendo la definida en RDF Schema que posee características especiales para automatizar tareas utilizando RDF.

- **RDFS (RDF Schemas):** Provee un vocabulario para la descripción de propiedades y clases de recursos RDF. Este lenguaje cuenta con semántica para la generalización de jerarquías de las propiedades de las clases. RDFS permite modelar metadatos con una representación explícita de su semántica y permite especificar restricciones de tipos de datos para los sujetos y objetos de RDF, introduciendo primitivas de modelado orientado a objetos: rdfs: Class, rdfs: Property, rdfs: subClassOf. RDFS ofrece un entramado en el cual las comunidades independientes pueden desarrollar vocabularios que se adapten a sus necesidades específicas. Para compartir vocabularios, el significado de los términos debe describirse con detalle. A las descripciones de estos conjuntos de vocabularios se les llaman RDF Schemas. Un schema define el significado, características y relaciones de un conjunto de propiedades. El lenguaje RDF permite que cada documento que contiene metadatos, sea clarificado con el vocabulario empleado asignando a cada vocabulario una dirección Web.

- **OWL (Ontology Web Language):** Es un lenguaje web para ontologías compatible con la World Wide Web. Se usa cuando la información de los documentos requiere ser procesada por aplicaciones, no para presentarlos. Puede representar el significado de los términos en vocabularios y las relaciones entre ellos. OWL posee más funcionalidades para expresar el significado y semántica que XML, RDF, y RDFS, pero OWL va más allá que estos lenguajes pues ofrece la posibilidad de representar contenido de la Web interpretable por máquina. Este lenguaje es una revisión del lenguaje de ontologías web DAML+OIL que incorpora lecciones aprendidas desde el diseño y aplicaciones de DAML+OIL. OWL tiene capacidad de ser distribuidas a través de varios sistemas, es escalable a las necesidades de la Web, compatible con los estándares Web de accesibilidad e internacionalización, es un lenguaje abierto y

extensible. Provee vocabulario para la descripción de propiedades y clases, por ejemplo: relaciones entre clases, cardinalidad, equivalencia, características de las propiedades.

OWL se diferencia de otros leguajes 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), no fueron definidos para ser compatibles con la arquitectura de la World Wide Web en general, y la Web Semántica en particular. OWL posee varias extensiones con respecto a RDF [WC304]:

- Los recursos para limitar las propiedades de clases con respecto a número y tipo.
- Los recursos para inferir qué elementos que tienen varias propiedades son miembros de una clase en particular.
- Los recursos para determinar si todos los miembros de una clase tendrán una propiedad en particular, o si puede ser que sólo algunos la tengan.
- Los recursos para distinguir entre relaciones uno a uno, varios a uno o uno a varios, permitiendo que las claves externas de las bases de datos puedan representarse en una ontología.
- Los recursos para expresar relaciones entre clases definidas en diferentes documentos en la Web.
- Los recursos para construir nuevas clases a partir de uniones, intersecciones y complementos de otras.
- Los recursos para restringir rangos y dominios para especificar combinaciones de clases y propiedades.

3.7 Herramientas de edición de ontologías

Entre las diversas herramientas informáticas empleadas actualmente en el desarrollo de ontologías, se destacan las siguientes [CO05]:

- **Protégé:** Desarrollada por el grupo SMI (Stanford Medical Informatics) de la Universidad de Stanford. Protégé [PROTEGE] es una herramienta que permite definir los conceptos y sus relaciones. Posee una interfaz muy amigable, permite la creación de gráficos, tablas, diagramas, y diferentes componentes de animación para acceder a la base de conocimiento. Permite importar, incluir y configurar un proyecto desde un archivo de texto, tablas de base de datos y archivos RDF. Otra posibilidad que ofrece esta herramienta es la capacidad de incluir un proyecto existente en un proyecto actual, lo cual, permite construir un proyecto grande de unos o más proyectos más pequeños. Protégé permite construir ontologías sobre RDFS, OWL y XML Schema.
- **KAON:** Desarrollada por la Universidad de Karlsruhe. La herramienta KAON [KAON] es un gestor de ontologías, incluye un conjunto de herramientas para crear y gestionar ontologías y provee un framework para construir aplicaciones basadas en ontologías. La característica más destacada de KOAN es que posee un razonamiento eficiente y escalable con las ontologías.
- **WebOnto:** Desarrollado por el Knowledge Media Institute (Kmi) de la Open University (Reino Unido). La herramienta WebOnto [WEBONTO] permite crear, editar, compartir y navegar modelos de conocimiento a través de la Web. Usando WebOnto se pueden ver las clases creadas en forma gráfica. También permite la edición de métodos de resolución de problemas.

- **OntoEdit:** Esta herramienta es desarrollada inicialmente por el Instituto AIFB en la Universidad de Karlsruhe, actualmente está comercializada por Ontoprise GmbH. OntoEdit [ONTOEDIT] es una herramienta de edición de ontologías que apoya el desarrollo y mantenimiento de las mismas utilizando medios gráficos en un entorno Web. Permite la representación semántica de lenguajes conceptuales y estructuras mediante conceptos, jerarquías de conceptos, relaciones y axiomas. OntoEdit ofrece interfaces gráficas ricas en funcionalidades para editar, integrar y traducir ontologías, además permite almacenarlas y posteriormente manipularlas en una base de datos relacional directamente por el editor como si fuera un Sistema de Gestión de Base de Datos (SGBD). Aunque OntoEdit está pensado para un entorno Web, también puede aplicarse a una Intranet. De esta forma se preserva la seguridad de la información que contiene.

- **Ontolingüa:** Ontolingüa es desarrollada por el Knowledge Systems Laboratory (KSL) de la Universidad de Stanford. El principal objetivo de Ontolingüa [ONTILIGUA] es facilitar el desarrollo colaborativo de ontologías y proporcionar un repositorio de las mismas. Ontolingua provee, en un entorno colaborativo, un buscador, generador y modificador de ontologías. Esta herramienta es un servicio basado en Web que ofrece una plataforma común donde las ontologías desarrolladas por diferentes grupos se pueden compartir.

- **OntoSaurus:** Desarrollada por el Information Sciences Institute (ISI) de la Universidad de Southern California para la creación de ontologías con el lenguaje Loom [LOOM]. Consiste en dos módulos: un servidor de ontologías, que usa Loom como sistema de representación de conocimiento, y un servidor de exploración de ontologías que crea páginas HTML dinámicamente para mostrar la jerarquía de ontologías. La ontología puede ser editada mediante formularios HTML. Además, existen traductores de Loom a Ontolingua, KIF, KRSS y C++. En Ontosaurus, los usuarios cooperan para construir la misma ontología, de forma que el bloqueo es necesario, aunque proporciona la facilidad de enviar mails a otros usuarios.

3.8 Conclusión

Las ontologías se presentan como una forma de representar el conocimiento compartido acerca de un dominio específico y son comúnmente asociadas a la idea de contenedores semánticos [SA03]. Al usar una ontología toda la información se convierte en conocimiento, porque los conceptos que se modelan tienen su significado semántico asociado.

Una ontología proporciona un marco para entender, unificar y extraer términos de los cuales se puede crear una abstracción de la realidad. Una ontología construida es la representación del conocimiento compartido que se tiene sobre un dominio, este conocimiento es descompuesto a través de conceptos, por lo que la representación del conocimiento es abordada mediante la representación de conceptos que, de alguna forma, están interrelacionados y generan dicho conocimiento o idea sobre el dominio. Una ontología se utiliza para definir vocabularios que las máquinas puedan entender y que sean especificados con la suficiente precisión como para permitir diferenciar términos y referenciarlos de manera precisa. Las ontologías son utilizadas por las personas, las bases de datos, y las aplicaciones que necesitan compartir un dominio de información. Las ontologías no sólo codifican el conocimiento de un dominio, sino que también definen el conocimiento que extiende el dominio, en este sentido, hacen el conocimiento reutilizable [WC304]. Las ontologías han surgido como una manera de estandarizar los términos de acuerdo a un dominio para facilitar la interoperabilidad de los sistemas actuales. Entonces, en cada dominio particular que es construida y usada una ontología todos los términos o conceptos que formen parte de este dominio serán conceptos estandarizados, unificados y tendrán su significado semántico asociado.

Se puede decir entonces que las ontologías no sólo toman un papel clave en la resolución de interoperabilidad semántica entre sistemas de información y su uso dentro del área

de aplicación, sino que también cumplen la función de conciliadora de términos dentro del dominio o área de aplicación que se esta modelando.

Una ontología es una herramienta viable para representar los términos que representan aspectos en el área de conocimiento del DSOA para lograr la desambigüación entre éstos. Al construir una ontología de aspectos para el DSOA se estará estableciendo un conocimiento consensuado dentro del área para ser reutilizado y compartido.

Capítulo 4: Aspects Extractor Tool

Aspect Extractor Tool representa un enfoque en el contexto de ingeniería de requerimientos, denominado Aspects Extractor [HD05]. Esta herramienta identifica, especifica, integra y evalúa aspectos en etapas tempranas del ciclo de desarrollo del software utilizando una serie de tareas automatizadas. La funcionalidad de esta herramienta permite especificar la funcionalidad del sistema, identificar los aspectos candidatos, y visualizar la especificación con los aspectos candidatos identificados. Aspect Extractor Tool ha sido desarrollada para tratar aspectos tempranamente, evitando de esta forma que surjan cambios en el sistema que se está desarrollando.

4.1 Modelo de Aspect Extractor

El modelo de ingeniería de requerimientos, Aspects Extractor (Figura 4.1), esta conformado por cinco tareas principales: Identificar concerns, Elección de los aspectos candidatos, Especificar aspectos candidatos, Identificar conflictos y Modelar en UML.

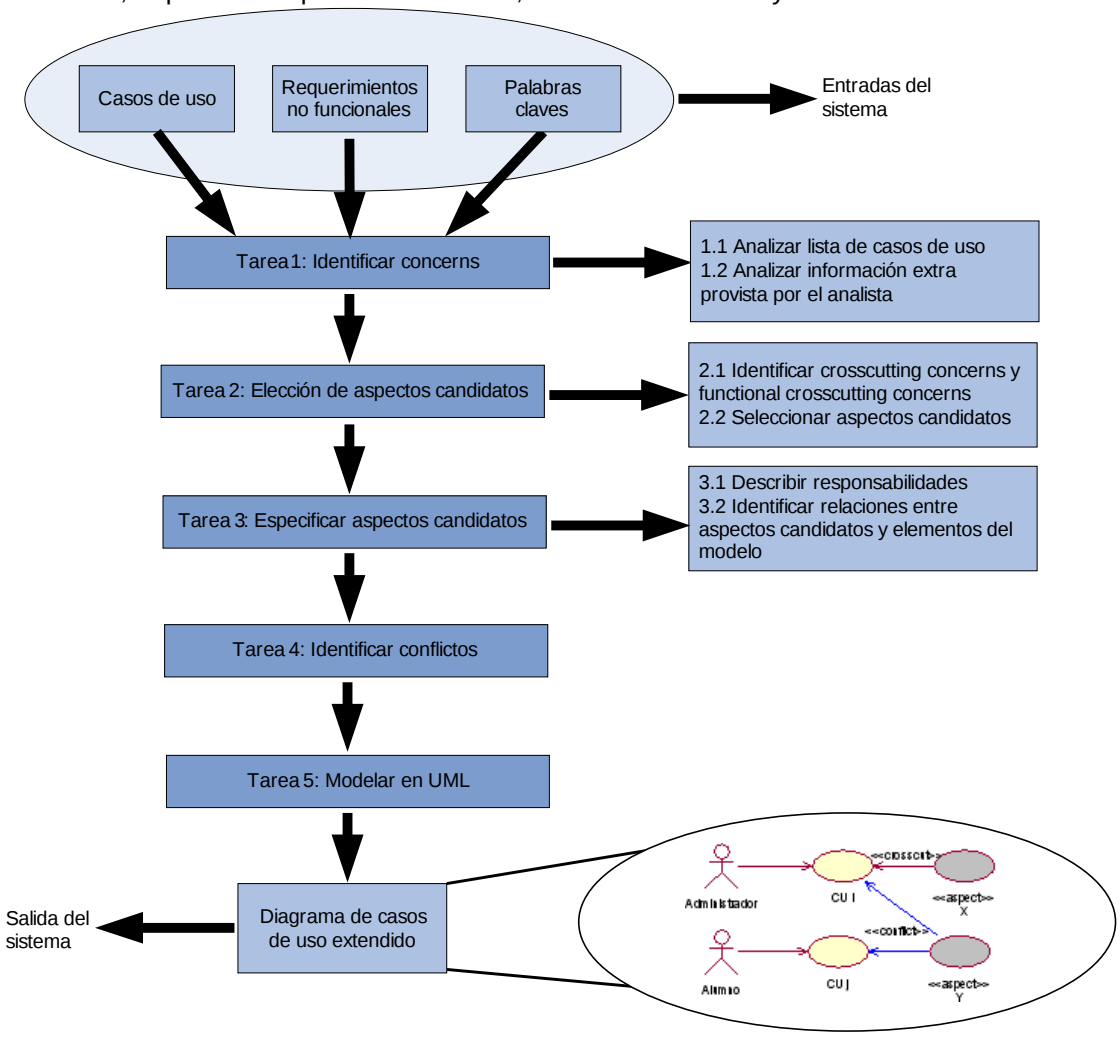

Figura 4.1: Tareas de Aspect Extractor Tool

4.1.1 Entrada de Aspect Extractor Tool

La entrada para la herramienta se compone de: casos de uso, requerimientos no funcionales y palabras claves. La especificación de los casos de uso puede ser realizado de dos formas: por medio de las interfaces gráficas que provee Aspects Extractor Tool o por medio de la importación de diagramas generados con Rational Rose 2003, en un archivo XMI. El ingreso de las palabras claves se hace a través del menú principal de la herramienta.

4.1.2 Salida de Aspect Extractor Tool

La salida que provee esta herramienta son los aspectos candidatos identificados. Estos pueden visualizarse de dos maneras diferentes, al igual que para el caso de la entrada. La primera es mediante la utilización de Aspects Extractor Tool, la cual brinda la posibilidad de observar en una serie de tablas los diferentes aspectos candidatos finalmente escogidos, y las posibles situaciones conflictivas. La segunda, es mediante un archivo con extensión XMI generado por la herramienta, el cual podrá importarse utilizando Rational Rose 2003.

4.1.3 Tareas de Aspect Extrator Tool

A continuación se presentan las tareas que forman la funcionalidad de Aspects Extractor Tool:

- **Tarea 1 - Identificar concerns:** Consiste en identificar a partir de los requerimientos, los posibles concerns de un sistema. Dicha tarea está dividida en dos sub-tareas: analizar la lista de casos de uso y analizar información extra provista por el analista. En ambas sub-tareas se genera automáticamente información de interés que se utilizará en las tareas restantes.

  - **Sub-tarea 1.1 - Analizar la lista de casos de uso:** Se analizan los casos de uso y se produce automáticamente una lista de palabras relevantes que representan al caso de uso, utilizando la técnica de recuperación de información para estandarizar estos datos. La recuperación de información es el conjunto de tareas o técnicas mediante las cuales el usuario localiza y accede a los recursos de información que son pertinentes para la resolución del problema planteado. Los casos de uso del sistema a desarrollar son representados por medio de palabras claves que van a permitir realizar la identificación de los concerns. Son usadas dos técnicas: Stop Words y Stemming [SN]. La primer técnica de recuperación de información usada elimina las palabras que, desde el punto de vista no lingüístico, no contienen información relevante, las llamadas palabras stop words, por ejemplo los artículos, preposiciones, etc. Como resultado se obtiene una lista de palabras que representan el caso de uso. En esta lista de palabras obtenidas, se emplea la técnica de Stemming, que transforma cada palabra en su palabra raíz. El resultado es una lista de palabras que se las clasifica en verbo o sustantivo para lo cual se utiliza un diccionario de verbos. Entonces, un caso de uso queda representado por dos listas, una lista de verbos (raíces de los verbos), y otra lista de sustantivos (raíces de sustantivos), lo que facilita la búsqueda de palabras similares en el interior de los distintos casos de uso.

  - **Sub-tarea 1.2 - Analizar información extra provista por el analista:** El analista suministrará ciertas palabras claves y requerimientos no funcionales con el propósito de enriquecer la información almacenada en el sistema para la obtención de los concerns. Las palabras claves pueden representar un aspecto en el sistema ya sea funcional o no funcional. El sistema comienza con un conjunto de palabras predefinidas al cual el analista le podrá agregar nuevas o eliminar existentes. La librería de palabras claves irá aumentando su tamaño a medida que el analista ingrese más palabras. Cuando el analista ingresa una palabra, se comprueba que no exista, se la reduce a su raíz y se almacena la palabra original y su raíz en la base de datos. Esto se debe a que

cuando se realiza matching entre los casos de uso y las palabras claves, ya sean funcionales o no, se debe guardar una referencia a la palabra original. El otro tipo de información que el analista tendrá posibilidad de suministrar son los requerimientos no funcionales del sistema. Para identificar los concerns a partir de estos, se toma un requerimiento no funcional por vez y se analiza sus influencias crosscutting sobre los casos de uso. El análisis de sus influencias crosscutting estará dado por la búsqueda de ocurrencias de las palabras que representan el requerimiento no funcional y las palabras que representen al caso de uso.

- **Tarea 2 - Elección de aspectos candidatos:** Se subdivide en dos sub-tareas: Identificar crosscutting concerns y elección de aspectos candidatos por parte del analista.

  - **Sub-tarea 2.1 - Identificar crosscutting concerns:** Existen dos formas principales de identificar crosscutting concerns. La primera de ellas se da cuando un concern corta transversalmente más de un caso de uso. La segunda forma se da cuando se identifica un caso de uso que cuenta con un requerimiento especial, no funcional, en su especificación. Dicho requerimiento no funcional será un aspecto candidato. Una forma adicional de detección de crosscutting concerns, ocurre solamente si el sistema fue usado previamente y se identificaron aspectos candidatos. Dichos aspectos se almacenan en una base de datos y serán reutilizados a futuro. Entonces, se busca concordancia de los concerns con los mismos. Si existe una concordancia, entonces el concern será candidato.

  - **Sub-tarea 2.2 - Elección de aspectos candidatos por parte del analista:** El analista seleccionará de una lista de concerns presentados en una tabla, los aspectos candidatos. Para ello deberá utilizar su experiencia e intuición. Se dice que son aspectos candidatos, dado que en etapas posteriores del ciclo de desarrollo de software podrían no ser vistos como aspectos, si es que así se decidiera.

- **Tarea 3 - Especificar aspectos candidatos:** Se subdivide en dos sub-tareas: identificar responsabilidades e identificar las relaciones entre aspectos para que los conflictos puedan ser detectados.

  - **Sub-tarea 3.1 - Identificar responsabilidades:** El analista deberá describir el objetivo y la funcionalidad del aspecto candidato.

  - **Sub-tarea 3.2 - Identificar las relaciones entre aspectos:** El sistema determinará con cuáles casos de uso está relacionado un determinado aspecto candidato.

- **Tarea 4 - Identificar conflictos:** Se identifican todas las posibles situaciones conflictivas entre aspectos que están relacionados al caso de uso. Si un caso de uso está afectado por más de un aspecto, entonces existe una situación conflictiva entre los aspectos involucrados. Si existe dicha situación, entonces será indicada de forma que pueda ser tratada posteriormente por el analista.

- **Tarea 5 - Modelar en UML:** Basados en la información obtenida en la realización de las tareas previas, se construirá un modelo visual, el cual será una extensión de un diagrama de casos de uso, con el agregado de los aspectos finalmente seleccionados por el analista y las posibles situaciones conflictivas entre los aspectos candidatos.

4.2 Identificación de aspectos candidatos utilizando el mecanismo actual

El mecanismo que Aspects Extractor Tool utiliza para la identificación de aspectos candidatos consiste en tomar la lista de casos de uso ingresados y representarlos por un texto que se compone de la concatenación del nombre, la descripción, el trigger, la suposición, el fluyo básico y el flujo alternativo del caso de uso. Luego recorre esta lista de palabras eliminando las palabras no relevantes denominadas stop words, al resto de las palabras se le aplica la técnica de Stemming [AAE99] y se la clasifica en sustantivo o verbos. La técnica de Stemming calcula la palabra raíz de cada palabra. Cada caso de uso en el proyecto abierto estará representado por dos listas, una lista de palabras sustantivos relevantes acompañadas de su palabra raíz y otra lista de verbos relevantes y sus raíces.

El mecanismo genera una tabla a partir de los verbos relevantes que se repiten en más de un caso de uso. Es decir, por cada caso de uso, se toman sus verbos relevantes. Si alguno de sus verbos se repite en algún otro caso de uso, se almacena en la tabla proponiéndolo como crosscutting concern y se almacena la referencia a los casos de uso involucrados. A medida que se recorren los casos de uso, se verifica si tienen requerimientos especiales en su especificación. Si así es, son propuestos como crosscutting concerns. Como el proyecto puede contar con requerimientos no funcionales, se verifica si existe una relación entre los casos de uso y dichos requerimientos no funcionales. Para verificarlo se toma un requerimiento no funcional por vez y se buscan las ocurrencias de palabras que representan el requerimiento no funcional y las palabras que representen al caso de uso. Si es que existe una relación, el requerimiento no funcional es propuesto como crosscutting concern y se almacena una referencia a los casos de uso relacionados. La tabla también se va generando a medida que se realiza un matching con palabras claves no funcionales, y las palabras relevantes de cada caso de uso. Es decir, si un caso de uso tiene en sus sustantivos relevantes una ocurrencia con las palabras no funcionales de la biblioteca del sistema, dichas palabras son propuestas como crosscutting concerns y se almacena una referencia al caso de uso afectado. La última actividad para la generación de la tabla del sistema se corresponde con la reutilización de aspectos detectados anteriormente en el análisis de otros proyectos. Es decir, si la herramienta hubiese sido utilizada previamente, los aspectos que se identificaron en tales ocasiones, se almacenaron debidamente en la base de datos. Entonces, para el nuevo proyecto, se verifica si no existe matching entre los casos de uso y dichos aspectos. De este modo se permite reutilizar el conocimiento adquirido en el uso de Aspects Extractor Tool.

A continuación se presentan algunos ejemplos utilizados en Aspect Extractor Tool que muestran el funcionamiento del algoritmo básico usado actualmente.

4.2.1 Caso de estudio 1: Gestión de alquiler de autos

El ejemplo presenta un proyecto de gestión de alquiler de autos. El sistema tiene dos tipos de actores: los clientes y la secretaria. Para que un cliente pueda alquilar un auto debe haber realizado la reserva previamente, y para poder alquilarlo, debe presentar el número de la reserva realiza. La secretaria será quien se encargue de realizar la incorporación de nuevos autos para alquiler en la agencia y de darlos de baja si es que fuera necesario.

Listado de casos de uso:

1. Reservar un auto.
2. Alquilar un auto.
3. Cancelar la reserva de un auto.
4. Dar de alta un auto para alquiler.
5. Eliminar un auto de alquiler.
6. Devolver un auto alquilado.

Para mostrar el funcionamiento del algoritmo usado por Aspect Extractor se especifican dos casos de uso, los casos de uso que se consideran más apropiados.

Caso de Uso Nº 1

Nombre	Reservar un auto.
Descripción	Reserva un auto para un cliente en una locación específica para las *fechas* dadas.
Actor	Cliente
Flujo Básico	1. El caso de uso comienza cuando un cliente desea hacer una reserva de *alquiler* de un auto. 2. El sistema *solicita* la localidad de salida y las *fechas* en las que el auto permanecerá alquilado. 3. El sistema *verifica* los autos disponibles para las *fechas* solicitadas en la localidad deseada por el cliente. 4. El sistema *muestra* los autos disponibles. 5. El sistema *solicita* al cliente la elección del auto deseado. 6. El sistema informa el costo del auto que el cliente seleccionó. 7. El sistema *solicita* al cliente la confirmación de la reserva. 8. El sistema *almacena* la información de la reserva del auto en las *fechas* y localidad elegidos por el cliente. 9. El sistema *muestra* al cliente el número de reserva.
Flujo Alternativo	1. En el paso 5, si el cliente no encuentra el auto deseado puede volver a *realizar* la búsqueda volviendo al paso 2, en caso contrario el caso de uso *termina*. 2. En el paso 7, si el cliente no confirma la reserva, puede volver a realizar la búsqueda volviendo al paso 2, en caso contrario el caso de uso *termina*.
Poscondición	Se almacenó la información de la reserva de un auto.

Tabla 4.1: Reservar un auto - Gestión de alquiler de autos

Caso de Uso Nº 2

Nombre	*Alquilar* un auto.
Descripción	*Realiza* el alquiler de un auto para un cliente.
Actor	Cliente
Suposición	Debe existir la reserva del auto *realizada* por el cliente.
Flujo Básico	1. El caso de uso comienza cuando un cliente desea *alquilar* un auto. 2. El sistema *solicita* el número de reserva del auto. 3. El sistema *verifica* el número de reserva. 4. El sistema *muestra* los datos de la reserva. 5. El sistema *verifica* las fechas de la reserva con la fecha de alquiler del auto. 6. El sistema *almacena* la información del alquiler del auto.
Flujo Alternativo	1. En el punto 2, si el cliente no ha realizado la reserva del auto el caso de uso *termina*. 2. En el punto 5, si las *fechas* de la reserva no coinciden con la *fecha* de alquiler se informa de la situación al cliente y el caso de uso *termina*.

Poscondición	Se almacenó la información del alquiler de un auto.

Tabla 4.2: Alquilar un auto - Gestión de alquiler de autos

Inicialmente el sistema representa a cada caso de uso con dos listas de palabras relevantes, una lista de sustantivos y otra de verbos, cada palabra esta acompañada por su palabra raíz, que va a ser utilizada para la comparación entre palabras. Los casos de uso anteriores quedan representados de la siguiente manera:

Representación Caso de Uso Nº 1:

Sustantivos → [aut, reserv, aut, client, locacion, especif, cas, comienz, client, dese, reserv, aut, sistem, local, aut, sistem, aut, dispon, local, client, sistem, aut, dispon, sistem, client, eleccion, aut, sistem, inform, aut, client, seleccion, sistem, client, confirm, reserv, sistem, almacen, inform, reserv, aut, local, client, sistem, client, numer, reserv, pas, 5, client, encuentr, aut, busqued, pas, 2, cas, contrari, cas, pas, 7, client, confirm, reserv, busqued, pas, 2, cas, contrari, cas, reserv, aut]

Verbos → [reserv, fech, dad, alquil, solicit, sal, fech, permanec, alquil, verif, fech, solicit, des, muestr, solicit, des, cost, solicit, fech, eleg, muestr, des, volv, realiz, volv, termin, volv, realiz, volv, termin, cre]

Representación Caso de Uso Nº 2:

Sustantivos → [aut, realiz, aut, client, deb, reserv, aut, client, cas, comienz, client, dese, aut, sistem, numer, reserv, aut, sistem, numer, reserv, sistem, dat, reserv, sistem, reserv, fech, aut, sistem, almacen, inform, aut, punt, 2, client, reserv, aut, cas, punt, 5, reserv, fech, inform, situacion, client, cas, alquil, aut]

Verbos → [alquil, alquil, exist, realiz, alquil, solicit, verif, muestr, verif, fech, alquil, alquil, realiz, termin, fech, coincid, alquil, termin]

Para realizar la división entre palabras sustantivos y verbos, la herramienta utiliza un diccionario de verbos. En la representación de los casos de uso se puede observar que el diccionario de verbos utilizado no es apropiado, ya que las listas no son correctas. En la lista de verbos pueden encontrarse palabras que no son verbos, por ejemplo la palabra fech (palabras raíz de la palabra fecha). En la lista de sustantivos aparecen palabras que son verbos, por ejemplo la palabra dese (palabra raíz de la palabra desea). Esta situación impacta de forma negativa en los resultados mostrados por Aspect Extractor Tool.

Luego de tener los casos de uso representados por las listas de verbos y de sustantivos, el sistema recorre cada caso de uso buscando los verbos que están en más de uno, la comparación de los verbos puede verse en la Tabla 4.3.

Verbos Caso de Uso Nº 1	Verbos Caso de Uso Nº 2
alquil	*alquil*
dad	exist
realiz	*realiz*
solicit	*solicit*
muestr	*muestr*
verif	*verif*
fech	*fech*
sal	coincid
termin	*termin*
permanec	

des	
cre	
volv	
eleg	
cost	
reserv	

Tabla 4.3: Comparación de verbos - Gestión de alquiler de autos

De esta tarea surgen las primeras palabras que serán presentadas como aspectos candidatos. El listado obtenido de esta tarea es el siguiente:

Aspectos Candidatos: alquilar, realizada, solicita, verifica, muestra, fechas, termina.

Otro concern candidato que la herramienta presenta es el concern persistencia. El concern candidato persistencia es mostrado como tal debido a que la palabra *"persistencia"* es una palabra sinónima de *"almacenar"* de acuerdo al diccionario de sinónimos con el trabaja Aspect Extractor Tool. La palabra *"almacena"* se encuentra en los flujos básicos de los casos de uso del ejemplo, para unificar el vocabulario se reemplaza la palabra *"almacena"* por su sinónimo *"persistencia"* , como esta palabra hace matching con la palabra clave no funcional del sistema es propuesta como concern candidato.

El listado de aspectos candidatos presentado por la herramienta al usuario se muestra en la Figura 4.2:

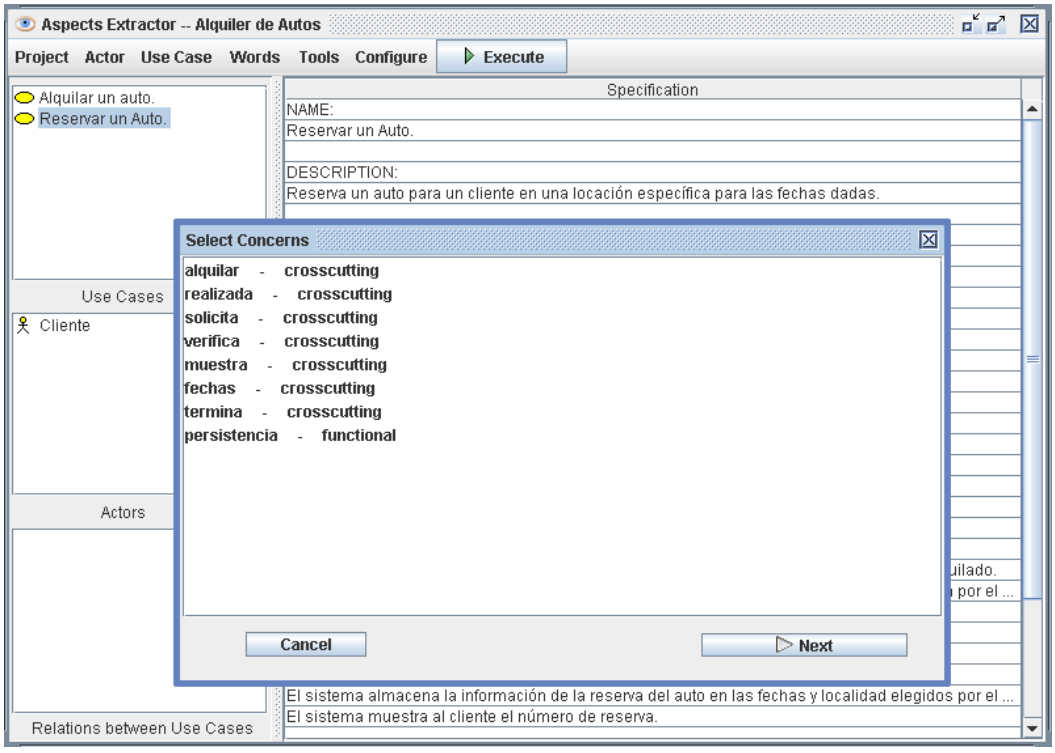

Figura 4.2: Aspectos candidatos propuestos – Gestión de alquiler de autos

Los aspectos candidatos propuestos: *"alquilar"*, *"realizada"*, *"solicita"*, *"verifica"*, *"muestra"*, *"fechas"* y *"termina"* no tienen un valor semántico válido como posible aspecto en el ejemplo presentado. No son resultados correctos retornados por la herramienta. En este ejemplo

la herramienta presento ocho aspectos candidatos, de los cuales sólo uno puede ser considerado como posible aspecto candidato, *"persistencia"*. En este caso el listado de aspectos candidatos presentado por Aspect Extractor Tool es considerado muy impreciso.

4.2.2 Caso de estudio 2: Administración de cuentas de profesores

El ejemplo 2 presenta a un sistema de gestión de cuentas de profesores para un colegio. El sistema cuenta con un sólo un tipo de actor: los profesores. Un profesor puede realizar altas de cuentas de otros profesores, pero estas cuentas sólo podrán ser eliminadas por el profesor que la creo. Un profesor puede enviar, recibir, leer y eliminar un mensaje. El sistema permite también enviar y recibir archivos.

Listado de casos de uso:

1. Crear una cuenta.
2. Borrar una cuenta.
3. Leer un mensaje.
4. Enviar un mensaje.
5. Borrar un mensaje.
6. Recibir un archivo.
7. Enviar un archivo.

Para mostrar el funcionamiento del algoritmo usado por Aspect Extractor Tools se especifican dos casos de uso, el caso de uso Crear una cuenta y Borrar una cuenta.

Caso de Uso Nº 1

Nombre	Crear una cuenta.
Descripción	El profesor *crea* una nueva cuenta. Esa cuenta sólo podrá ser *borrada* por el profesor que la *creó*.
Actor	Profesor
Suposición	El profesor debe estar *logueado* en el sistema.
Precondición	No debe *existir* en el sistema una cuenta con el mismo nombre.
Flujo Básico	1. El caso de uso comienza cuando un profesor desea *crear* una cuenta. 2. El sistema *solicita* el nombre de la cuenta. 3. El sistema comprueba que no *exista* una cuenta con el mismo nombre. 4. El sistema *solicita* que el profesor ingrese la clave. 5. El sistema *solicita* el reingreso de la clave. 6. El sistema *verifica* que ambas claves sean iguales. 7. El sistema valida la clave ingresada por el profesor. 8. El sistema almacena la nueva cuenta de profesor.
Flujo Alternativo	1. Si en paso 3 el sistema verifica que *existe* una cuenta con el mismo nombre, se informa al profesor y se vuelve al paso 2. 2. Si en el paso 6 el sistema *verifica* que las claves no son iguales, informa de la situación y se vuelve al paso 4. 3. Si en el paso 7 el sistema comprueba que la clave no es válida porque no cumple con el formato requerido de mínimo 6 caracteres, informa de la situación y se vuelve al paso 4.
Poscondición	Se almacenó información de la nueva cuenta de profesor.

Tabla 4.4: Crear una cuenta – Administración de cuentas de profesores

Caso de Uso Nº 2

Nombre	*Borrar* una cuenta.
Descripción	Se *borra* una cuenta de profesor. Sólo la puede *borrar* aquel profesor que la *creó*.
Actor	Profesor
Suposición	El profesor debe estar *logueado* en el sistema.
Precondición	La cuenta de profesor que se quiere *borrar* debe *existir* en el sistema.
Flujo Básico	1. El caso de uso comienza cuando el profesor desea *borrar* una cuenta. 2. El sistema *solicita* el nombre de la cuenta de profesor. 3. El sistema *verifica* que la cuenta de profesor *exista*. 4. El sistema comprueba si el profesor que desea *borrar* la cuenta es el profesor que la *creó*. 5. El sistema *borra* la cuenta del profesor.
Flujo Alternativo	1. Si en paso 3 el sistema comprueba que la cuenta del profesor no existe se informa de la situación y se vuelve al paso 2. 2. Si en el paso 4 el sistema comprueba que el profesor que intenta eliminar la cuenta de profesor no es el mismo profesor que la *creó* informa de la situación al profesor y termina el caso de uso.
Poscondición	Se *borró* la información de la cuenta especificada.

Tabla 4.5: Borrar una cuenta – Administración de cuentas de profesores

Representación Caso de Uso Nº 1:

Sustantivos → [cuent, profesor, nuev, cuent, cuent, solo, podr, profesor, profesor, deb, sistem, deb, sistem, cuent, nombr, cas, comienz, profesor, dese, nuev, cuent, sistem, nombr, cuent, sistem, comprueb, exist, cuent, nombr, sistem, profesor, acces, clav, sistem, reingres, clav, sistem, ambas, sean, sistem, clav, profesor, sistem, persistent, nuev, cuent, profesor, pas, 3, sistem, exist, cuent, nombr, inform, profesor, vuelv, pas, 2, pas, 6, sistem, son, inform, situacion, vuelv, pas, 4, pas, 7, sistem, comprueb, clav, autent, cumpl, format, minim, 6, inform, situacion, vuelv, pas, 4, persistent, inform, nuev, cuent, profesor]

Verbos → [cre, cre, borr, cre, log, exist, cre, solicit, solicit, solicit, verif, clav, igual, val, acces, verif, verif, clav, igual, requer, caracter]

Representación Caso de Uso Nº 2:

Sustantivos → [cuent, borr, cuent, profesor, solo, profesor, profesor, deb, sistem, cuent, profesor, quier, deb, sistem, cas, comienz, profesor, dese, cuent, sistem, nombr, cuent, profesor, sistem, cuent, profesor, exist, sistem, comprueb, profesor, dese, cuent, profesor, sistem, borr, cuent, profesor, pas, 3, sistem, comprueb, cuent, profesor, exist, inform, situacion, vuelv, pas, 2, pas, 4, sistem, comprueb, profesor, cuent, profesor, profesor, inform, situacion, profesor, cas, borr, inform, cuent]

Verbos → [borr, borr, cre, log, borr, exist, borr, solicit, verif, borr, cre, elimin, cre, termin, especific]

La comparación de verbos entre casos de uso que la herramienta realiza puede verse en la Tabla 4.6.

Verbos Caso de Uso Nº 1	Verbos Caso de Uso Nº 2
cre	*cre*
borr	*borr*
exist	*exist*
solicit	*solicit*
log	*log*
verif	*verif*
clav	elimin
igual	termin
val	especific
acces	
requer	
caracter	

Tabla 4.6: Comparación de verbos - Administración de cuentas de profesores

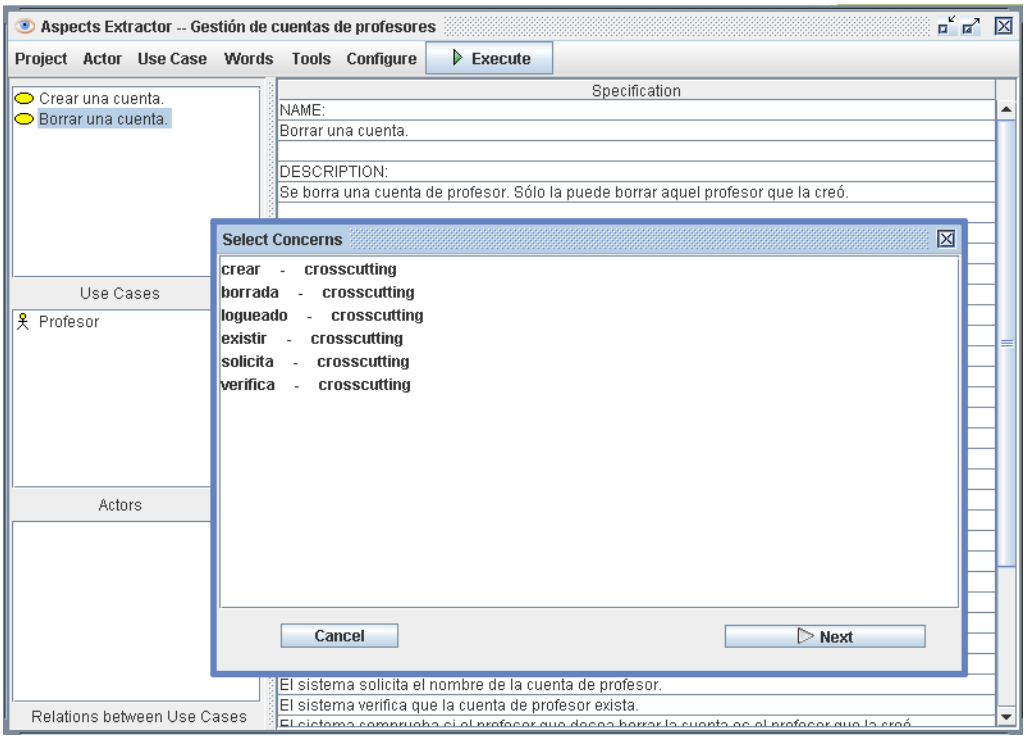

Figura 4.3: Aspectos candidatos propuestos – Administración de cuentas de profesores

Los aspectos candidatos que la herramienta presenta como resultado de la ejecución con el ejemplo anterior es el mostrado en la Figura 4.3. Este listado es muy impreciso, los crosscutting propuestos *"crear", "borrada", "existir" y "solicita"* no son palabras que en el ejemplo tengan un significado que representen a posibles aspectos. El crosscutting concern propuesto *"logueado"* y *"verifica"* se consideran resultados correctos para el ejemplo presentado.

4.2.3 Caso de estudio 3: Administración de habitaciones de un hotel

El ejemplo presenta a un sistema de administración de habitaciones de un hotel. El sistema cuenta con dos tipos de actores: los pasajeros y las agencias de tarjetas de crédito. Un pasajero puede tomar una habitación sólo en el caso que haya realizado una reserva previamente. Para cancelar la deuda que se creó con el hotel, el pasajero puede usar dos opciones: pagar con dinero en efectivo o pagar con una tarjeta de crédito.

Listado de casos de uso:

1. Realizar reserva
2. Asignar habitación.
3. Cancelar reserva.
4. Entregar habitación.
5. Inhabilitar habitación.

Se especifican dos casos de uso para mostrar el funcionamiento de la herramienta: Asignar habitación y Realizar reserva.

Caso de Uso Nº 1

Nombre	*Realizar* reserva.
Descripción	Registra una reserva de un cliente en el sistema**.**
Actor	Pasajero
Flujo Básico	1. El caso de uso comienza cuando el pasajero desea *realizar* una reserva en el hotel. 2. El sistema *solicita* las fechas de la reserva y tipo de habitación al cliente. 3. El sistema *verifica* disponibilidad para las fechas dadas por el cliente. 4. Si existe disponibilidad el sistema registra la información de la reserva. 5. El sistema genera un número de reserva que es mostrado al cliente.
Flujo Alternativo	1. En el paso 4, si no si existe disponibilidad el sistema informa la situación al cliente, ir al paso 2.
Poscondición	Se registro la información de la reserva de una habitación.

Tabla 4.7: Realizar reserva – Administración de habitaciones de un hotel

Caso de Uso Nº 2

Nombre	Asignar habitación.
Descripción	Asigna una habitación a un cliente que ha hecho una reservación previa**.**
Actor	Pasajero / Agencia de tarjeta de crédito
Suposición	El pasajero debe haber *realizado* una reserva.
Flujo Básico	1. El caso de uso comienza cuando el pasajero desea que se le asigne una habitación del hotel. 2. El sistema *solicita* el código de reserva. 3. El sistema *verifica* sí la reservación es válida.

	4. El sistema muestra los datos de la reserva y el precio de la habitación. 5. Si el cliente va a cancelar con tarjeta de crédito. a) El sistema *solicita* número de tarjeta de crédito b) El sistema *verifica* la tarjeta de crédito con la agencia. c) La agencia de tarjetas de crédito informa la aprobación de la tarjeta. 6. Si el cliente va a cancelar en efectivo el sistema cancela el valor de la habitación. 7. El sistema almacena la información de la cuenta del cliente en el hotel. 8. El sistema muestra el número de habitación asignada. 9. El sistema almacena la información de la ocupación de la habitación.
Flujo Alternativo	1. En el paso 3 si no es válida la reservación o el cliente no la tiene se termina el caso de uso. 2. En el paso 5 c) si la agencia de tarjetas de crédito rechaza la tarjeta el cliente deberá pagar en efectivo, ir al paso 6. 3. En el paso 6 si el cliente no desea cancelar en efectivo se cancela el proceso y termina el caso de uso.
Poscondición	Se almaceno información de la ocupación de la habitación.

Tabla 4.8: Asignar habitación – Administración de habitaciones de un hotel

A continuación se muestra la representación de los casos de uso con que los que la herramienta realiza al algoritmo para la identificación de los aspectos candidatos:

Representación Caso de Uso Nº 1:

Sustantivos → [reserv, registr, reserv, client, sistem, persistent, inform, reserv, habit, cas, comienz, pasajer, dese, reserv, hotel, sistem, reserv, tip, habit, client, sistem, disponibil, client, exist, disponibil, sistem, registr, reserv, sistem, gener, numer, reserv, client, sistem, persistent, inform, reserv, pas, 4, exist, disponibil, sistem, inform, situacion, client, pas, 2]

Verbos → [realiz, realiz, solicit, fech, verif, fech, dad, mostr]

Representación Caso de Uso Nº 2:

Sustantivos → [habit, asign, habit, client, hech, reserv, previ, pasajer, reserv, cas, comienz, pasajer, dese, asign, habit, hotel, sistem, codig, reserv, sistem, reserv, autent, sistem, dat, reserv, preci, habit, client, tarjet, credit, sistem, numer, tarjet, credit, b, sistem, tarjet, credit, agenci, c, agenci, credit, inform, autent, tarjet, client, real, sistem, cancel, habit, sistem, alamcen, inform, cuent, client, hotel, sistem, numer, habit, sistem, persistent, inform, habit, pas, 3, autent, reserv, o, client, cas, pas, 5, c, agenci, credit, tarjet, client, real, pas, 6, pas, 6, client, dese, real, cancel, proces, cas, persistent, inform, habit]

Verbos → [asign, deb, hab, realiz, solicit, verif, muestr, cancel, solicit, verif, tarjet, cancel, muestr, asign, ocup, termin, tarjet, rechaz, deb, pag, cancel, termin, ocup]

Las palabras que se repiten en ambos casos de uso y que el sistema propone como crosscutting concerns candidatos son las mostradas en la Figura 4.4.

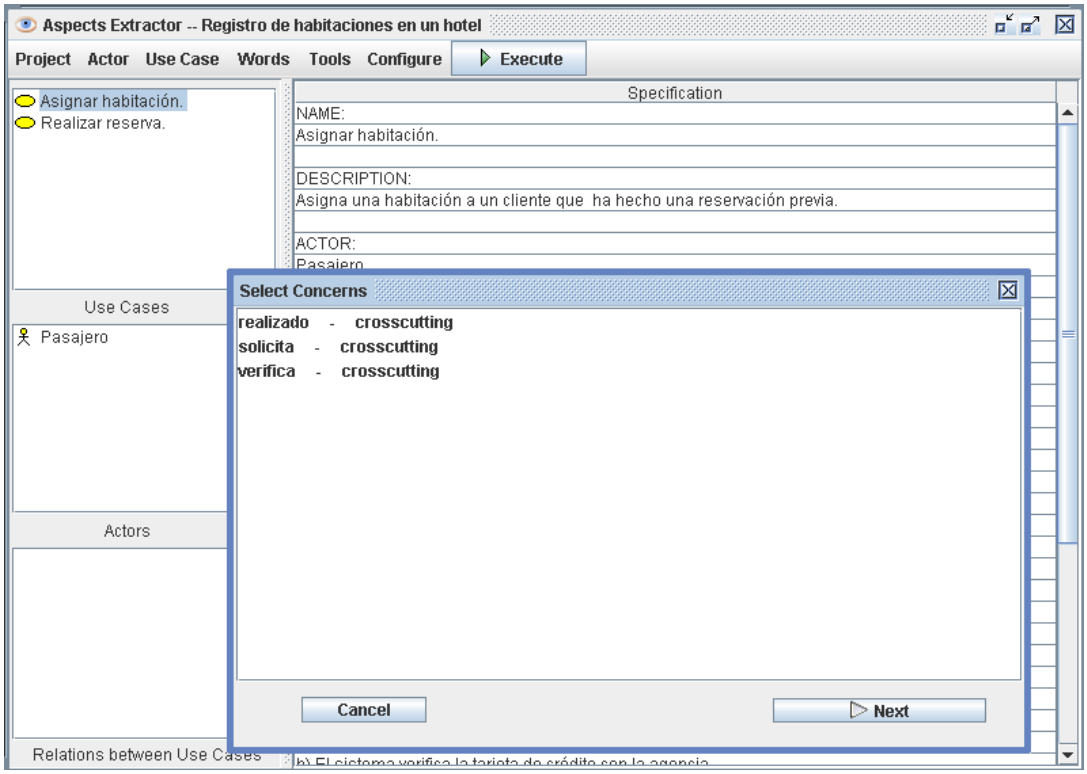

Figura 4.4: Aspectos candidatos propuestos – Administración de habitaciones de un hotel

Los crosscutting concerns candidatos propuestos por la herramienta *"solicita", "verifica" y "realizado",* no son potenciales crosscutting concerns. Carecen de significado semántico en este ejemplo. El total de los aspectos candidatos presentados por la herramienta para el ejemplo es incorrecto.

Al observar los casos de uso se puede ver que el caso de uso Realizar Reserva utiliza la palabra *"almacena"* para realizar el registro de información en la base de datos, el otro caso de uso especificado, Asignar Habitación, utiliza la palabra *"registra"* para la misma actividad. Esta actividad es un aspecto candidato válido en el ejemplo, pero la herramienta no lo detecta. Esto se debe a que el diccionario de sinónimos que Aspect Extractor utiliza no es completo. Las palabras registra y almacena no son detectadas como sinónimos y por lo tanto el algoritmo no lo identifica como aspecto candidato.

4.3 Conclusión del algoritmo actual usado por Aspect Extractor Tool

Luego de analizados los resultados ofrecidos por la herramienta en cada ejemplo presentado, se puede decir que el algoritmo utilizado actualmente ofrece gran cantidad de crosscutting concerns candidatos, pero poco de estos son potenciales crosscutting concerns. Una gran parte de los resultados ofrecidos deberán ser rechazados, el analista que realice esta operación deberá tener conocimiento del proyecto que esta desarrollando para poder llevar a cabo esta tarea. Esto se produce debido a que las palabras que representan a los crosscutting concerns candidatos no son estandarizadas, y en la mayoría de los casos, no representan claramente qué responsabilidad tienen.

El algoritmo utilizado para la identificación de aspectos candidatos por Aspect Extractor Tool posee algunas deficiencias tales como:

- Presenta una lista grande de aspectos candidatos que el analista deberá revisar para seleccionar cuáles considera apropiados.

- El analista debe tener conocimiento previo sobre el proyecto para poder analizar la lista de aspectos candidatos, ya que las palabras que la herramienta propone no son palabras estandarizadas que representen a los aspectos candidatos.
- Identifica una gran cantidad de aspectos candidatos que no son potenciales aspectos para los proyectos.
- A medida que los proyectos son más grandes o las especificaciones más detalladas la herramienta encuentra una lista de crosscutting concerns iniciales más grande y con menos precisión.
- El diccionario de verbos con el que la herramienta trabaja actualmente no es totalmente apropiado, se puede observar que los listados de palabras verbos y sustantivos que representan a los casos de uso no son completamente correctos y en muchos casos, esto afecta de manera negativa los resultados obtenidos. En algunos casos propone aspectos candidatos incorrectos y en otros omite aspectos candidatos válidos.
- El diccionario de sinónimos que la herramienta usa no es completo. Lo que tiene como consecuencia la omisión de aspectos candidatos válidos para algunos proyectos.

Se puede concluir que el algoritmo con el cual trabaja la herramienta actualmente no es muy apropiado, se necesita otro mecanismo de identificación de aspectos candidatos que trabaje sobre conceptos estandarizados y unificados de los términos que representan los aspectos en área de conocimiento del DSOA. De forma tal de agilizar la tarea del analista y lograr que los resultados obtenidos sean más acertados.

4.4 Ontología de aspectos para Aspect Extractor

Se propone la incorporación de un mecanismo para la detección de aspectos candidatos de manera tal de resolver los problemas encontrados utilizando la primer técnica. Se propone la incorporación de un mecanismo que, a diferencia del anterior, trabaje sobre una ontología la cual representa a los conceptos intervinientes en el dominio de aspectos para la identificación de crosscutting concerns. Al crear una ontología para el área de aplicación de DSOA se estandarizan y unifican los conceptos que representan crosscutting concern para el DSOA. La ontología permite que las búsquedas realizadas sobre los conceptos que se modelan en ella arrojen resultados más acertados. Por estos motivos se incorpora a Aspect Extractor un mecanismo que trabaja con una ontología, este mecanismo ayudará a resolver las deficiencias encontradas en el primer algoritmo de Aspect Extractor Tool.

El nuevo mecanismo se incorpora para que la lista de concerns candidatos obtenidos sea más precisa que la obtenida con la técnica anterior, aprovechando los beneficios obtenidos de la utilización de la ontología. Cuando se usa una ontología se esta trabajando sobre un modelo de concerns uniforme, es decir, todos los términos que representen a aspectos en el DSOA están unificados en la ontología, esto se produce porque por medio de una ontología se pueden clasificar y abstraer conceptos de un dominio especifico, en este caso el DSOA. De esta manera también se esta logrando facilitar el acuerdo entre los stakeholders, porque todos sabrán cuál es el significado de cada término del que se esta hablando.

La ontología puede ser utilizada cuando la herramienta realice la sub-tarea de Identificar Aspectos Candidatos de la tarea Elección de Aspectos Candidatos (sub-tarea 1 de la tarea 2). En el mecanismo utilizado hasta el momento existen dos formas principales de identificar crosscutting concerns. La primera forma se da cuando un concern corta transversalmente más de un caso de uso. La segunda forma se da cuando se identifica un caso de uso que cuenta con un requerimiento especial, no funcional, en su especificación. Dicho requerimiento no funcional será un aspecto candidato. Con el mecanismo que utiliza la ontología se trabaja sobre una lista de conceptos estandarizados, se buscan ocurrencias de palabras de los casos de uso sobre los conceptos definidos en la ontología. Si una palabra de la lista que representa un requerimiento, a

través de su semántica puede derivar en un concern, éste va a ser presentado al usuario como un concern candidato.

4.5 Cambios en Aspect Extractor Tool

Con la incorporación del algoritmo que usa una ontología el usuario de la herramienta podrá elegir qué mecanismo desea utilizar para la identificación de aspectos candidatos de su sistema e indicárselo a Aspect Extractor Tool a través del menú de configuración que se agregó en el menú principal de la herramienta. La herramienta utilizará el mecanismo que corresponda de acuerdo a la configuración que tenga indicada. La Figura 4.5 muestra que la herramienta puede ser utilizada usando una u otra heurística para la identificación de aspectos candidatos.

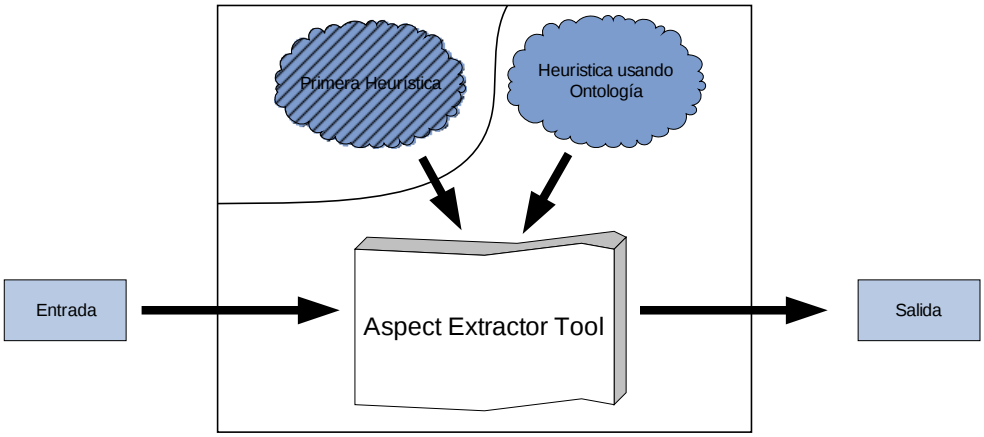

Figura 4.5: Elección de heurísticas

En el menú principal de la herramienta Aspects Extractor Tool se agregó un menú de configuración. Desde este menú el analista puede seleccionar cuál es el algoritmo que desea utilizar para la identificación de los aspectos candidatos. Las opciones disponibles son: usar la heurística actual con la que cuenta la herramienta, o usar la nueva heurística que usa la ontología. Esta funcionalidad se puede observar en la Figura 4.6.

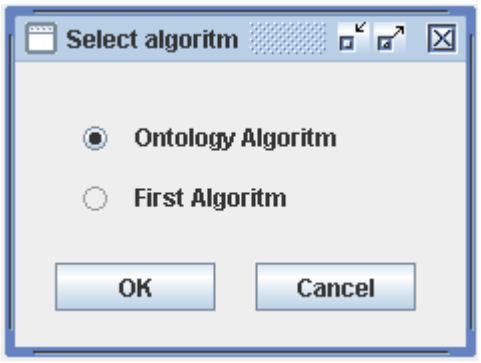

Figura 4.6: Pantalla de selección de un algoritmo en Aspect Extractor Tool

El menú de configuración también provee las opciones de configuración del camino donde se encuentra el archivo que contiene las reglas de inferencia, el archivo de la ontología en el idioma español, y el archivo de la ontología en el idioma inglés (Figura 4.7). Según el idioma del proyecto que se esta analizando será la ontología que el nuevo algoritmo utilice para la identificación de los aspectos candidatos. Si es un proyecto en español, la herramienta usa la

ontología *OntologyAspectSpanish.owl*, si es un proyecto en inglés, la ontología usada es *OntologyAspectInglish.owl*.

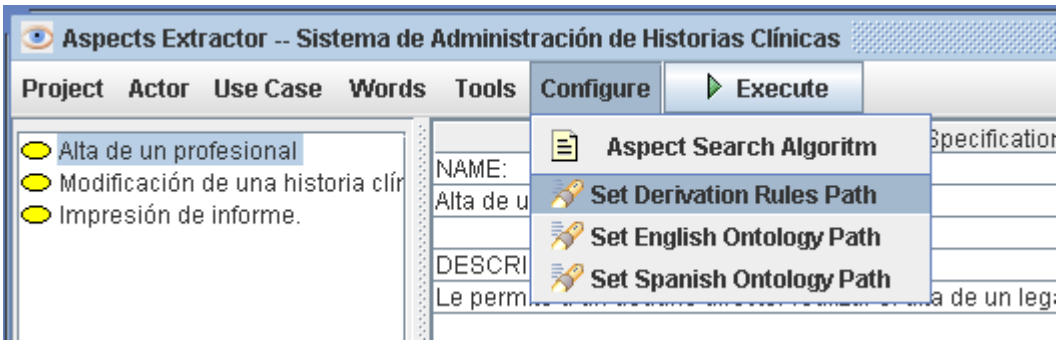

Figura 4.7: Opciones del menú Configure de Aspect Extractor Tool

Los datos que son configurados desde este menú son guardados en un archivo de propiedades, el archivo llamado earlyAspect.properties. Cuando se ejecuta la aplicación se lee desde este archivo y se trabaja con los valores que tenga almacenado.

La incorporación del nuevo mecanismo no modifica la estructura actual de la herramienta. Las cinco tareas principales mostradas en la Figura 4.1: Identificar concerns, Elección de los aspectos candidatos, Especificar aspectos candidatos, Identificar conflictos y Modelar en UML seguirán marcando la funcionalidad que esta posee.

El cambio se realizó en la tarea número dos: Elección de aspectos candidatos, según el mecanismo que la herramienta tenga configurado, debe decidir si la sub-tarea de identificar crosscutting concerns se realiza utilizando la técnica anterior o se realiza utilizando la ontología construida para el dominio DSOA.

El cambio que se produce en Aspect Extractor Tool puede verse en la Figura 4.8, se mantiene la estructura de las tareas que forman el enfoque Aspect Extractor. El cambio sólo afecta a la sub-tarea Identificar crosscutting concerns de la tarea Elección de aspectos candidatos, donde ahora el analista podrá elegir qué heurística desea ejecutar.

El cambio realizado afecta sólo al paquete principal Aspects Identification, que es el paquete relacionado a la identificación, especificación y evaluación de aspectos. Este paquete está conformado por otros tres paquetes, *Proyecto*, *Heurística* y *Stemming*. El paquete *Heurística* es el paquete que contiene la funcionalidad para la detección de los aspectos candidatos usando el método actual. Para la incorporación del nuevo algoritmo utilizando la ontología se agregaron dos paquetes. El paquete *algoritmoAspectExtractor* es un paquete que contiene las clases necesarias para poder ejecutar uno de los algoritmos de detección de aspectos que la herramienta posee. El otro paquete incorporado es el paquete *logicaOntologia*, este paquete comprende la funcionalidad básica para la detección de aspectos candidatos a través del uso de una ontología.

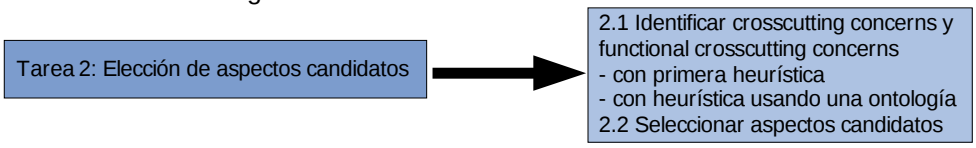

Figura 4.8: Modificación en la Tarea 2 de Aspect Extractor Tool

El paquete *algoritmoAspectExtractor* implementa el patrón de diseño *Strategy* [GVJH95]. Este patrón permite mantener los dos algoritmos para la detención de aspectos implementados hasta el momento, también permite incorporar nuevos algoritmos de forma sencilla y rápida si surgiera la necesidad, porque permite un número ilimitado de estrategias (Figura 4.9). Se podrá

seleccionar el algoritmo conveniente, entre los algoritmos disponibles, e intercambiarlo según se considere el algoritmo apropiado en cada situación.

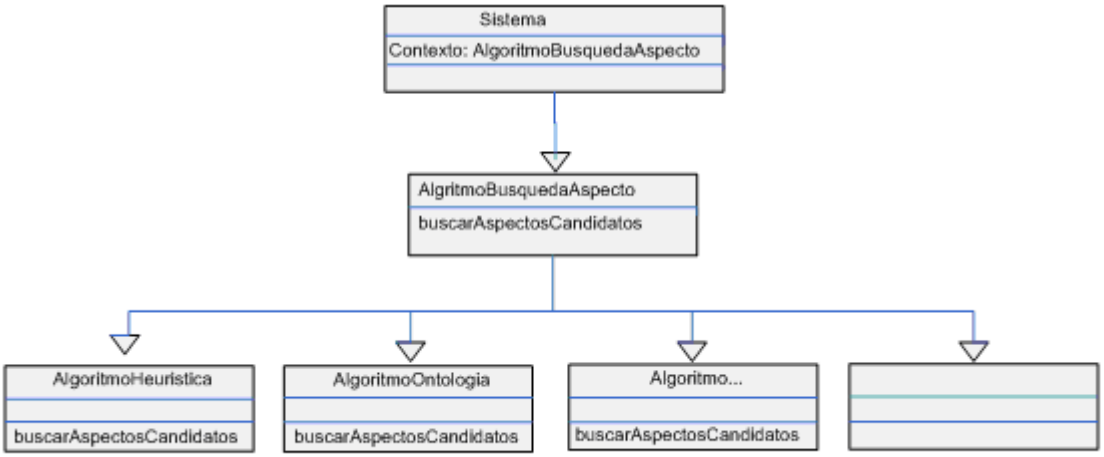

Figura 4.9: Patrón de diseño Strategy

4.6 Conclusión

Se analizaron los resultados que Aspect Extractor Tool presenta con el primer algoritmo mediante distintos ejemplos. Del análisis presentado se concluye que este algoritmo presenta algunas deficiencias tales como:

- Presenta una lista grande de aspectos candidatos, la cual el analista deberá analizar para seleccionar cuáles considera apropiados.
- Identifica una gran cantidad de aspectos candidatos que no son correctos.
- El diccionario de verbos y de sinónimos con los que trabaja no son apropiados, lo que produce resultados erróneos u omisión de resultados correctos.
- El analista debe tener conocimiento del proyecto para poder analizar la lista de aspectos candidatos presentada, esto es porque las palabras que la herramienta propone no son palabras estandarizadas que representen a los aspectos candidatos.

Se propone la incorporación de un mecanismo para la detección de aspectos candidatos de manera tal de resolver estas deficiencias. Se propone la incorporación de un mecanismo que trabaje sobre una ontología la cual representa a los conceptos intervinientes en el dominio de aspectos para la identificación de crosscutting concerns. Al crear una ontología para el área de aplicación de DSOA se estandarizan y unifican los conceptos que representan crosscutting concern para el DSOA. Cuando se usa una ontología se esta trabajando sobre un modelo de concerns uniforme, es decir, todos los términos que representen a aspectos en el DSOA están unificados en la ontología, esto se produce porque por medio de una ontología se pueden clasificar y abstraer conceptos de un dominio especifico, en este caso el DSOA. De esta manera también se esta logrando facilitar el acuerdo entre los stakeholders del proyecto, porque todos sabrán cuál es el significado de cada término del que se esta hablando.

Capítulo 5: Ontología para Aspect Extractor Tool

En esta sección se presentan los detalles de la ontología construida que es utilizada por la herramienta para la identificación de aspectos candidatos.

5.1 Ontología para el dominio Desarrollo de Software Orientado a Aspectos

La ontología que se presenta es una ontología de tipo dominio, describe el vocabulario relacionado a un dominio particular. El dominio sobre el que la ontología es construida es el dominio Desarrollo de Software Orientado a Aspectos. La ontología es creada para cubrir la necesidad presentada de falta de unificación de conceptos en los términos que representan a los aspectos en las aplicaciones de software y mejorar el proceso de identificación de aspectos. La ontología proporciona un marco para organizar, entender, unificar y extraer términos en el campo de trabajo del DSOA de los cuales es posible crear una abstracción para identificarlos de manera unificada y adecuada. La ontología es utilizada por la herramienta Aspect Extractor Tool para la Identificación de Aspectos en la Ingeniería de Requerimientos. Se crearon dos ontologías, una para trabajar con proyectos en el idioma español y otra en el idioma inglés. La Figura 5.1 muestra la ontología creada en el idioma español.

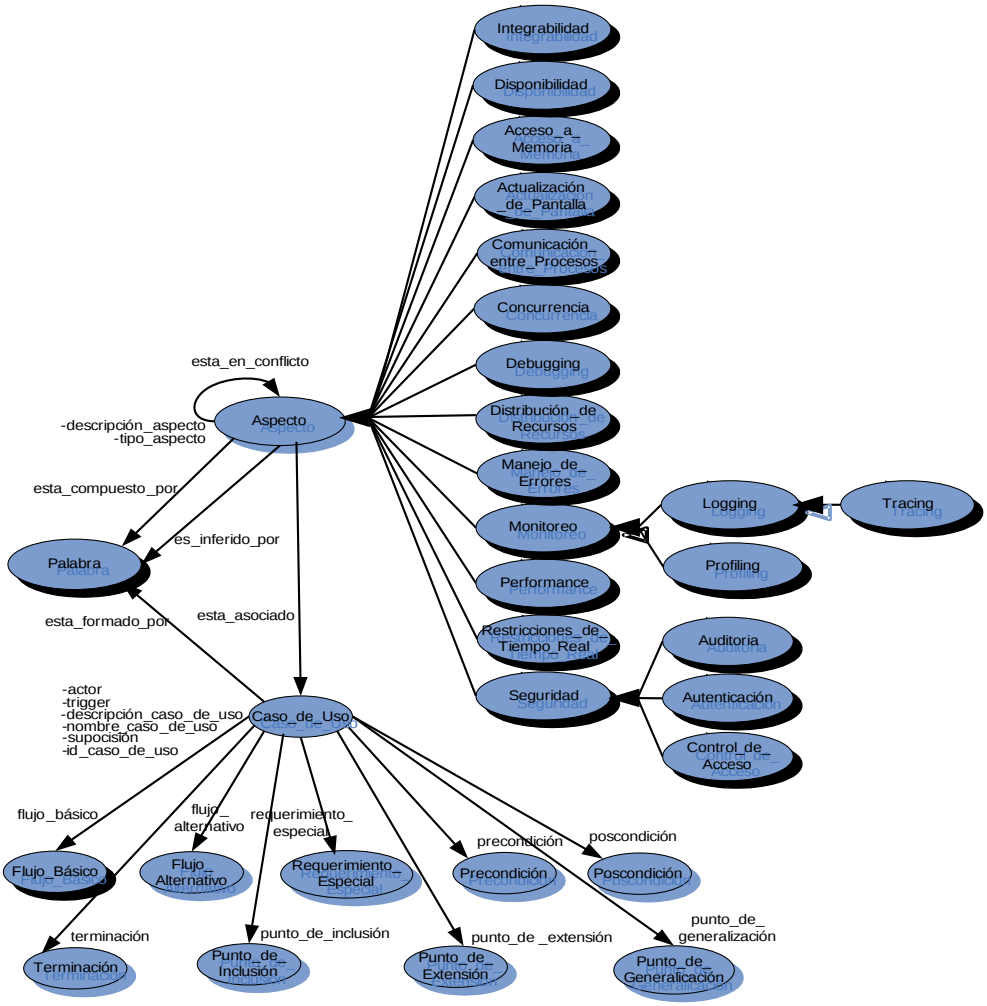

Figura 5.1: Ontología en español para el DSOA

Los términos que representan a los aspectos en el DSOA también fueron modelados en el idioma inglés, de forma tal que la herramienta Aspect Extractor Tool pueda usar este mecanismo incorporado para proyectos en el idioma inglés, tal y como lo hace con el mecanismo con el que trabaja actualmente, la ontología en el idioma inglés puede verse en la Figura 5.2:

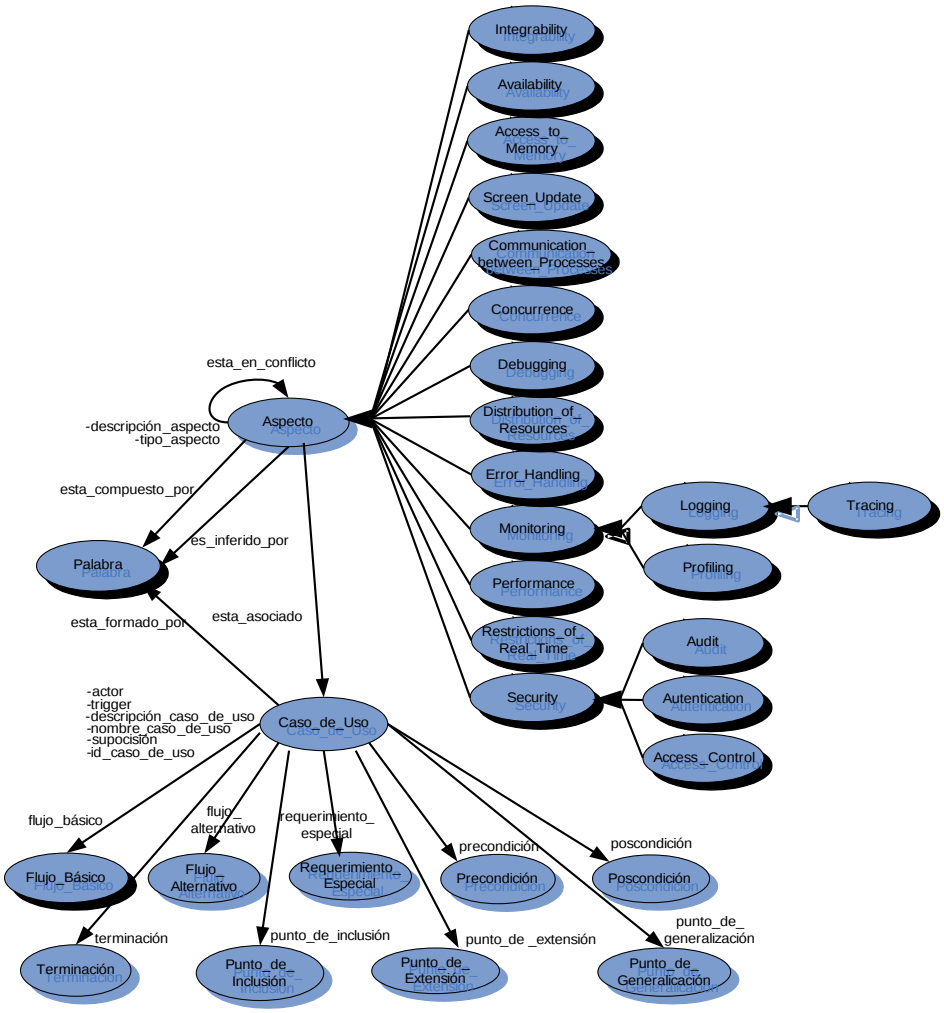

Figura 5.2: Ontología en inglés para el DSOA

En la ontología que se presenta se muestran cada uno de los términos que representan a aspectos candidatos en el DSOA. Todos los términos que representan a aspectos candidatos son subclase de la clase *"Aspecto"*. Los aspectos candidatos que la ontología modela son: *"Acceso a Memoria", "Actualización de Pantalla", "Comunicación entre Procesos", "Concurrencia", "Debugging", "Distribución de Recursos", "Manejo de Errores", "Monitoreo", "Logging", "Tracing", "Profiling", "Performance", "Restricciones de Tiempo Real", "Seguridad", "Auditoria", "Autenticación", "Control de Acceso", "Disponibilidad" e "Integrabilidad".* En el concepto *"Aspecto"* se modelan propiedades de datos (Datatype Properties), que son datos que modelan propiedades pertenecientes a los conceptos, y propiedades de objetos (Object Properties), que modelan la relación con otros conceptos:

Propiedades de dato de la clase Aspecto:

- **descripción_aspecto:** Guarda la descripción de las responsabilidades del aspecto candidato.
- **tipo_aspecto:** Indica el tipo del aspecto candidato modelado.

Propiedades de objeto de la clase Aspecto:

- **esta_asociado:** Esta propiedad modela la relación que existe entre un aspecto y un caso de uso. Indica en cuáles casos de uso el aspecto debe adicionar su comportamiento (Figura 5.3):

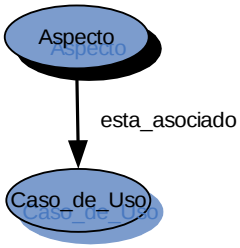

Figura 5.3: Propiedad del concepto Aspecto: esta_asociado

- **esta_en_conflicto:** Cuando un caso de uso es afectado por más de un aspecto se produce una situación de conflicto entre los aspectos, los aspectos compiten entre sí por su activación. El conflicto existe dado que esta situación puede provocar un comportamiento impredecible o indeseado en la aplicación. Esta propiedad modela una situación de conflicto entre dos aspectos (Figura 5.4).

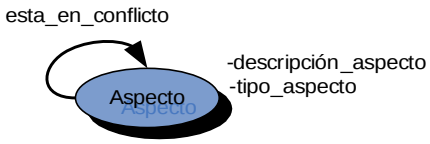

Figura 5.4: Propiedad del concepto Aspecto: esta_en_conflicto

- **esta_compuesto_por (Figura 5.5):** Un aspecto candidato esta compuesto por una o más palabras relevantes. Para formar el conjunto de palabras relevantes de un aspecto candidato se quitan las palabras que desde el punto de vista lingüístico, no contienen información relevante, estas palabras son las palabras stop words, para lograrlo se usa la técnica Stop Words [SN]. A la lista de palabras obtenidas, se emplea la técnica de Stemming [SN], que transforma cada palabra en su palabra raíz. Por ejemplo, el aspecto candidato *"Distribución de Recursos"* esta formado por dos palabras relevantes: *"Distribución"* y *"Recursos"*, las palabras raíces de estas palabras son: *"Distribu"* y *"Recur"* respectivamente. Al conjunto de palabras obtenido se agregan las raíces de las palabras sinónimo de las palabras relevantes. Los sinónimos junto con sus palabras raíces se encuentran almacenados en la base de datos de la aplicación.

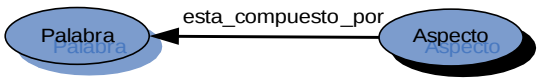

Figura 5.5: Propiedad del concepto Aspecto: esta_compuesto_por

- **es_inferido_por:** Un aspecto es propuesto como un aspecto candidato si el aspecto es influenciado desde un caso de uso por las palabras que componen el caso

de uso y las que componen el aspecto. Esta propiedad que involucra a la clase Aspecto y la clase Palabra modela a las palabras de los casos de uso que hicieron que se infiera el aspecto candidato propuesto (Figura 5.6).

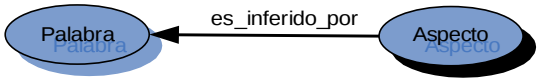

Figura 5.6: Propiedad del concepto Aspecto: es_inferido_por

Los casos de uso son la entrada para Aspect Extractor Tool, estos casos de uso son modelados en la ontología según la forma que la herramienta requiere (Figura 5.7). De esta manera el concepto *"Caso de Uso"* esta formado por las siguientes propiedades:

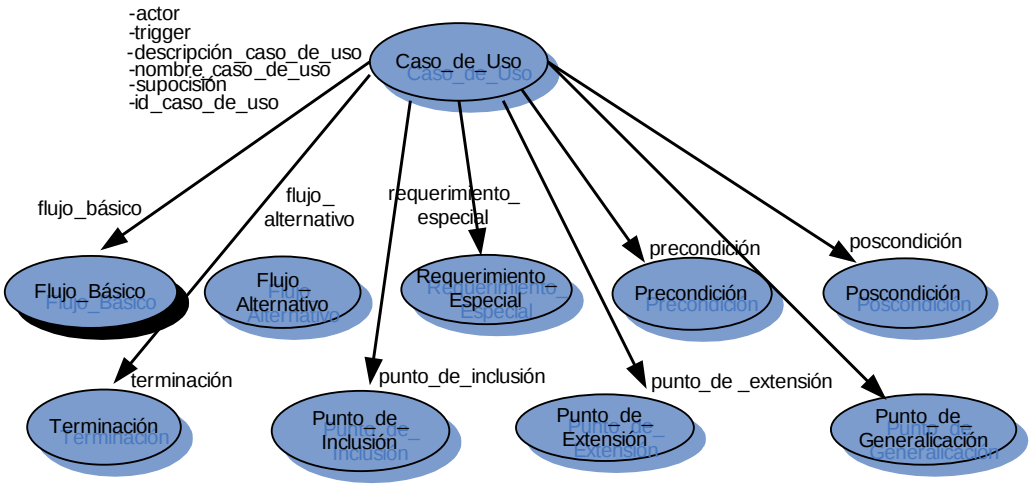

Figura 5.7: Propiedades del concepto Caso de Uso

Propiedades de dato de la clase Caso_de_Uso:

- **id_caso_de_uso:** Esta propiedad modela el número identificador de un caso de uso.
- **nombre_caso_de_uso:** Nombre del caso de uso.
- **actor_caso_de_uso:** Propiedad que modela los actores que intervienen en el caso de uso.
- **suposición_caso_de_uso:** Suposiciones en el caso de uso.
- **trigger_caso_de_uso:** Trigger del caso de uso.

Propiedades de objeto de la clase Caso_de_Uso:

- **poscondición:** Indica que cosas tienen que ser verdad el finalizar el caso de uso.
- **precondición:** Indica que cosas tienen que ser verdad para el inicio del caso de uso.
- **punto_de_generalización:** Esta propiedad indica los puntos de generalización del caso de uso.
- **punto_de_terminación:** Listado de puntos donde todas las actividades del caso de uso están realizadas.

- **punto_de_inclusión:** Indica los puntos de inclusión en el caso de uso.
- **punto_de_extensión:** Puntos de extensión del caso de uso.
- **flujo_básico:** Esta propiedad indica cuáles son los pasos del flujo básico.
- **flujo_alternativo:** Indica cuáles son los pasos del flujo alternativo del caso de uso.
- **requerimiento_especial:** Describe el conjunto de requerimientos especiales de un caso de uso.
- **esta_formado_por:** Esta propiedad modela las palabras por las que esta formado el caso de uso. Para encontrar el conjunto de palabras se aplican las técnicas de Stop Words y Stemming [SN] al conjunto de palabras que forman el caso de uso.

5.1.1 Herramientas utilizadas en la construcción de la ontología

La ontología se ha construido y modelado usando las herramientas Protégé [PROTEGE] y GraphViz [GRAPHVIZ]. El lenguaje seleccionado para su representación es OWL (Ontology Web Language), porque permite la definición de los conceptos, relaciones y propiedades. OWL es un lenguaje muy expresivo que permite establecer la semántica adecuada y es posible usarlo con varios frameworks de manipulación de ontologías.

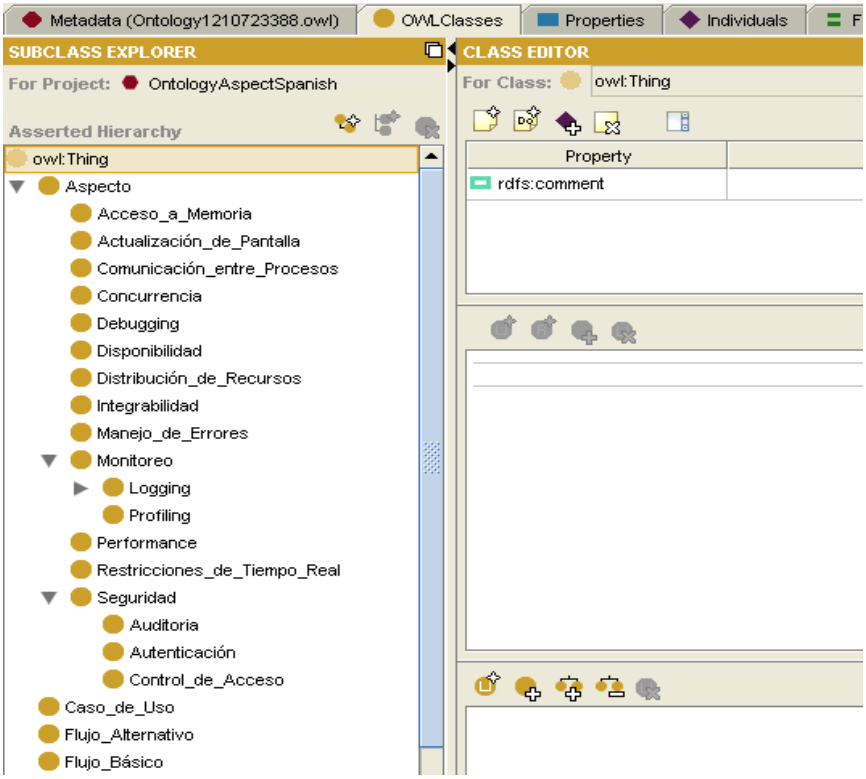

Figura 5.8: Vista de clases con Protégé

Protégé versión 3.3.1: Esta herramienta es un editor de ontologías libre con licencia pública [PROTEGE], por medio de la cual se generaron los archivos *OntologyAspectSpanish.owl* y *OntologyAspectInglish.owl* que son las ontologías para el DSOA en el idioma español e inglés

respectivamente. Una vista de clases de la ontología en español desde esta herramienta se muestra en la Figura 5.8.

Una vista de propiedades de la ontología en español con Protégé se muestra en la Figura 5.9. En la herramienta Protégé, las propiedades de objetos son de color azul y las propiedades de dato son de color verde.

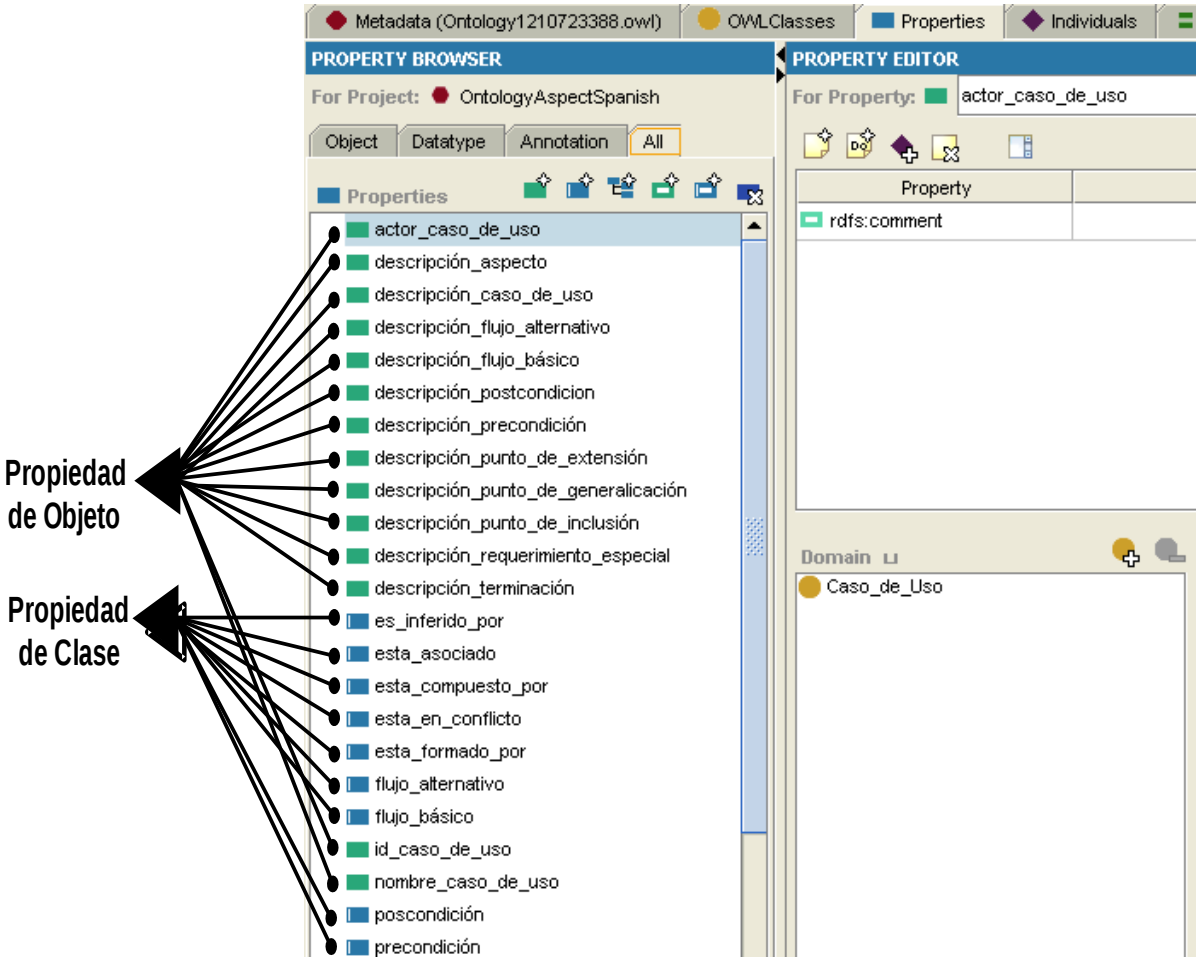

Figura 5.9: Vista de propiedades con Protégé

GraphViz versión 2.16.1: Es un software libre con licencia pública para la visualización de grafos [GRAPHVIZ]. Los tipos de visualizaciones son altamente configurables e incluyen la selección de un conjunto de clases o de instancias para visualizar una parte de una ontología, una visualización creada usando esta herramienta se muestra en la Figura 5.10.

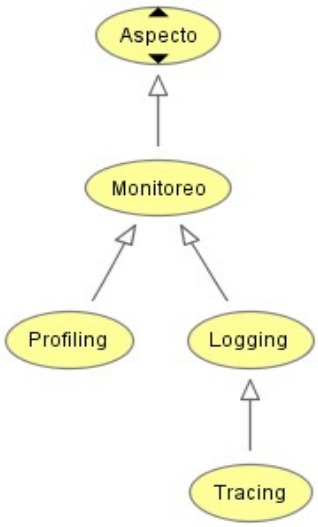

Figura 5.10: Visualización de un conjunto de clases con GraphViz

5.1.2 Framework utilizado para la manipulación de la ontología

Para manipular las ontologías en la herramienta Aspect Extractor Tool se trabajó con el framework de desarrollo Jena [JENA]. Jena es un API para Java, un entorno para el desarrollo de aplicaciones en el lenguaje de programación Java. Jena permite el uso y la realización de búsquedas en la ontología. Provee la capacidad de contar con un motor de inferencia basado en reglas. Este framework permite gestionar todo tipo de ontologías, agregar hechos, borrarlos, editarlos, almacenarlos y realizar consultas sobre ellas. Estas propiedades serán usadas en el algoritmo del nuevo mecanismo incorporado a la herramienta para la identificación de aspectos candidatos. Jena soporta RDF, DAML y OWL, y es independiente del lenguaje. Jena cuenta con el siguiente conjunto de APIs que se enumeran a continuación:

- APIs de RDF.
- API de OWL.
- API ARQ para manejo de Sparql [SPARQL], es un lenguaje utilizado para realizar consultas a una ontología.
- Soporte de persistencia.

5.2 Proceso de identificación

El proceso de identificación de aspectos candidatos usando la ontología trabaja de forma similar con proyectos en el idioma español e inglés. Cuando se necesita trabajar con la ontología, según el idioma del proyecto con el que se esta trabajando, se usa la ontología adecuada para el proyecto. Cuando se aplican las técnicas de Stops Word y Stemming, estas técnicas también van a trabajar con el idioma del proyecto con el cual se esta trabajando. La forma de identificar los aspectos candidatos, inferir el nombre completo para un aspecto candidato, inferir los conflictos entre aspectos e inferir las palabras que determinan a los aspectos, trabajan de forma indistinta para proyectos de diferentes idiomas.

El proceso de identificación de aspectos candidatos usando la ontología es dividido en cinco actividades principales, la descripción de cada actividad es realizada a continuación:

- **Actividad 1 - Verificar consistencia de la ontología:** Esta actividad consiste en verificar si la ontología no contiene hechos contradictorios. Para verificar la consistencia se hace un chequeo global a través del esquema y las instancias buscando

inconsistencias. Si se encuentra alguna, se informa de la situación, en caso contrario se comienza con la ejecución de las operaciones para la identificación de los aspectos candidatos.

- **Actividad 2 - Representar los casos de uso del proyecto:** Esta tarea consiste en representar a cada caso de uso del proyecto con una lista de palabras. Esta lista se obtiene de formar una cadena de palabras compuesta por el nombre, la descripción, el trigger, la suposición, las precondiciones, las poscondiciones, el flujo básico, el flujo alternativo, las terminaciones, los puntos de inclusión, los puntos de exclusión, los puntos de generalización y los requerimientos especiales del caso de uso. A esta cadena se le aplica la técnica de Stop Words, que elimina las palabras de la cadena que no son relevantes desde el punto de vista lingüístico. A la lista de palabras que se obtiene del paso anterior se aplica la técnica de Stemming, esta técnica convierte a cada palabra en su palabra raíz. De esta manera cada caso de uso estará representado por una lista que contiene las palabras que lo representan y sus palabras raíces, que son usadas en pasos posteriores para buscar coincidencias con las palabras que forman los conceptos que representan a los aspectos candidatos en la ontología.

- **Actividad 3 - Buscar aspectos candidatos en los casos de uso:** La siguiente tarea es recorrer la ontología buscando los posibles aspectos candidatos que se puedan inferir por cada uno de los casos de uso. La ontología es recorrida como un grafo dirigido con un recorrido en profundidad [TA72].

  - **Sub-actividad 3.1 – Buscar nodo inicial:** Para buscar los aspectos candidatos que se infieren mediante un caso de uso la primera tarea a realizar es buscar el nodo de comienzo del recorrido, el nodo raíz. En la ontología construida el nodo raíz es el nodo *"Aspecto"*. Desde este nodo se comienza el recorrido en profundidad, cada vez que se encuentre un nodo, este nodo va a estar representando un concepto de la ontología (un aspecto en el DSOA).

  - **Sub-actividad 3.2 – Recorrer ontología:** Se recorre la ontología buscando coincidencias entre las palabras que representan al caso de uso analizado y las palabras que representan a los aspectos candidatos en la ontología. Si en las palabras del caso de uso se encuentra una palabra que sea igual a alguna de las palabras que representan el aspecto o a las palabras sinónimos de estas palabras, se considera que se ha encontrado una coincidencia. Se cuenta la cantidad de ocurrencias que existe de palabras del caso de uso en el conjunto de palabras que forman el concepto. Luego de encontrar todas las coincidencias se guarda una referencia al posible aspecto candidato junto con la cantidad de coincidencias, y se continúa analizando el resto de los aspectos. La comparación de palabras se realiza por medio de su raíz. Las palabras sinónimo de las palabras que representan un aspecto candidato y sus raíces se encuentran almacenadas en la base de datos de la aplicación.

  - **Sub-actividad 3.3 – Obtener aspectos candidatos:** Cada vez que se termina de analizar un caso de uso se recorre la lista de aspectos candidatos encontrados para el caso de uso. Si la cantidad de coincidencias encontradas es igual a la cantidad de palabras relevantes en el aspecto, se considera que se ha encontrado un aspecto candidato. Este aspecto candidato es guardado en una lista de aspectos candidatos del proyecto junto con una referencia al caso de uso que lo infirió. Si el aspecto candidato no se encuentra en la lista de aspectos candidatos del proyecto se agrega, en caso contrario se agrega la referencia al caso de uso. Al final del recorrido por todos los casos de uso del proyecto, se obtiene una lista de aspectos candidatos, conteniendo una referencia a los casos de uso que lo infieren, esta información es presentada al analista y será utilizada en las siguientes tareas del enfoque Aspects Extractor.

Si el concepto tiene n cantidad de palabras relevantes que lo componen, se buscan n o más coincidencias con el caso de uso, si se encuentra n o más cantidad de coincidencias se considera que el caso de uso analizado infiere al aspecto en cuestión.

Por ejemplo, si es analizado el concepto: “Comunicación_entre_Procesos”, luego de aplicarse las técnicas de Stop Word y Stemming, el concepto queda representado por el siguiente conjunto de palabras: [comunicación, procesos], entonces, la cantidad de coincidencias buscadas para considerar este concepto como aspecto candidato es mayor o igual a dos. Se compara cada una de las palabras raíces que representan el caso de uso con las palabras raíces del conjunto de palabras que representan el concepto.

- **Actividad 4 - Buscar aspectos candidatos en los requerimientos especiales del proyecto:** En este punto del algoritmo se analizaron todos los casos de uso, restan analizar los requerimientos especiales que el proyecto tiene.

  - **Sub-actividad 4.1 – Representar requerimientos especiales del proyecto:** Se le aplican las técnicas de Stop Words y Stemming a los requerimientos especiales del proyecto para que queden representados por las palabras relevantes que lo componen.

  - **Sub-actividad 4.2 – Recorrer ontología:** La siguiente tarea es recorrer una vez más la ontología para buscar aspectos candidatos con el listado de palabras que representan a los requerimientos especiales.

  - **Sub-actividad 4.3 – Obtener aspectos candidatos:** Si se encuentra algún aspecto candidato, se agrega a la lista de aspectos candidatos del proyecto referenciado a cada uno de los casos de uso del proyecto. Esto se debe a que el requerimiento especial afecta a todo el proyecto.

- **Actividad 5 - Inferir nombres completos de los aspectos candidatos:** Esta actividad permite inferir los nombres completos para los aspectos candidatos identificados en las tareas anteriores desde la ontología. Se va a obtener conocimiento que se encuentra explicito en la ontología, para realizar esta tarea se usa un motor de razonamiento que, usando los datos que se encuentran en la ontología, permite obtener conclusiones de ellos.

Como puede observarse en la figura que muestra la ontología en el idioma español (Figura 5.1), se modelan aspectos que son especializaciones de otros. Por ejemplo, los aspectos Tracing y Profiling son tipos especializados de aspecto Monitoreo, el analista que usa la herramienta debe acceder a esta información. Cuando se identifican los aspectos candidatos esta información no es obtenida, aunque se encuentra implícita en la ontología. Para compartir esta información con el analista se va a usar un razonador para inferirla desde los datos en el modelo.

Para inferir esta información no es necesario definir reglas de inferencia, se usa la herencia de conceptos definida en la ontología. Se usa un razonador para el lenguaje OWL interno que provee Jena, el razonador OWL Reasoner.

Un componente esencial que se utiliza para hacer consultas sobre un modelo es una tripleta. Una tripleta esta formada por objeto-atributo-valor. Un objeto O tiene un atributo A con valor V: representado matemáticamente como A(O, V). Otra forma de ver esta relación es como un enlace anotado entre dos nodos: [O] - A - [V]. Esta notación es útil porque permite intercambiar objetos y valores. Así, cualquier objeto en una tripleta puede jugar el papel de valor en otra, formándose una red de nodos interconectados [DDAGMMS05].

Para inferir el conocimiento que se busca, se recorre la lista de aspectos candidatos obtenidos para el proyecto. Por cada aspecto candidato se crea un objeto O con el nombre del aspecto. También es necesario crear un atributo, se crea el atributo A con la propiedad que se quiere buscar, en todos los casos el atributo que se quiere buscar es la propiedad “subClassOf”. Con el objeto O y el atributo A se tienen los datos necesarios para que el razonador realice las consulta sobre la ontología buscando todos los valores V posibles para formar las tripletas. Por ejemplo, si se quiere saber cuál es el nombre completo del aspecto candidato detectado Control de Acceso la consulta que se realiza sobre la ontología es: subClassOf(Control de Acceso, V), donde V son todos los posibles valores para la superclase de la clase Control de Acceso. El

conjunto resultado obtenido es: {Seguridad}, lo que no esta indicando que el nombre completo del aspecto candidato Control de Acceso es Control de Acceso (Seguridad).

En todos los casos se obtienen también como resultado las clases Thing, Resource y Aspecto. Todos los conceptos modelados en una ontología son subclases de la clase Thing, todos los aspectos modelados son subclase de la clase Aspecto y todos los conceptos modelados son subclase de la clase Resourse. Por las razones mencionadas, estos resultados se consideran válidos pero no son información útil para el conocimiento que se quiere inferir.

5.3 Inferencia de conocimiento desde la ontología

Para aprovechar los beneficios que provee la ontología se verifican los aspectos candidatos propuestos por la herramienta desde las palabras que componen a los casos de uso y a los aspectos candidatos y se calculan los conflictos entre estos.

Se crea un nuevo modelo ontológico con las instancias de las palabras que representan a los aspectos candidatos detectados y los casos de usos que éstos afectan. Con los datos que se tienen de un aspecto candidato se pueden cargar en el modelo las instancias de las propiedades: *"tipo_aspecto"* y *"esta_compuesto_por"*. Se recorre la lista de aspectos candidatos propuestos para el proyecto cargando éstas propiedades. La propiedad "*esta_compuesto_por"* también se instancia con las palabras sinónimos de las palabras que componen el aspecto candidato, ya que las palabras sinónimo también intervienen en la inferencia de un aspecto. Se recorre la lista de los casos de uso a los que el aspecto candidato afecta cargando las instancias de las propiedades: *"nombre_caso_de_uso", "descripcion_caso_de_uso"*, *"id_caso_de_uso"*, *"esta_formado_por"* y *"esta_asociado"*. La propiedad *"esta_formado_por"* se carga con todas las palabras que forman el caso de uso (nombre, descripción, trigger, suposición, precondiciones, poscondiciones, flujo básico, flujo alternativo, terminaciones, puntos de inclusión, puntos de exclusión, puntos de generalización y requerimientos especiales). Las propiedades *"esta_compuesto_por"* y *"esta_formado_por"* se cargan con el conjunto de palabras relevantes que forman al caso de uso y al aspecto candidato respectivamente, aplicando las técnicas de Stop Words y Stemming. De esta manera se crea un muevo modelo ontológico para el proyecto que se esta analizando.

Antes de comenzar con el proceso de inferir conocimiento se verifica si el modelo creado es consistente, de ser así se comienza con la tarea. Para inferir el conocimiento buscado se usa un razonador genérico interno de los que provee Jena, el razonador Generic Rule Reasoner. Con este razonador se usan reglas de inferencia definidas para buscar cuáles son los conflictos existentes entre aspectos candidatos y cuáles son las palabras de los casos de uso que infirieron a los aspectos candidatos. El razonador chequea cada una de las reglas para ver si los datos satisfacen las premisas de alguna de las reglas. Si una regla es satisfecha, es ejecutada derivando nuevos hechos que pueden ser utilizados por otras reglas para derivar hechos adicionales. Las reglas definidas están registradas en el archivo *myRules.rules*.

La siguiente regla ayuda a derivar la existencia de conflictos entre aspectos. Se deriva qué aspecto candidato está en situación de conflicto y con cuáles aspectos es el conflicto existente. Esta información se deduce también en la tarea Identificar conflictos (Tarea 4) del enfoque Aspect Extractor.

[estaEnConflictoCon:
(?aspecto1 cc:esta_asociado ?casoUso1),
(?aspecto2 cc:esta_asociado ?casoUso2),
equal(?casoUso1,?casoUso2),
notEqual(?aspecto1,?aspecto2)
→ (?aspecto1 cc:esta_en_conflicto ?aspecto2)]

Esta regla de inferencia nos esta indicando que el *aspecto1* se encuentra en situación de conflicto con el *aspecto2* si el *aspecto1* afecta al *casoUso1*, el *aspecto2* afecta al *casoUso2*, estos casos de uso son el mismo caso de uso, y *aspecto1* y *aspecto2* son distintos.

Las consultas que se realizan en el modelo se muestran en forma de tripletas, donde V son todos los posibles aspectos con el cual el objeto O se encuentra en situación de conflicto: esta_en_conflicto(O, V). Esta consulta devuelve los resultados en pares, cada par representa a dos aspectos que se encuentran en conflicto.

La siguiente regla de inferencia permite determinar cuáles son las palabras del proyecto que intervienen en la derivación de un aspecto candidato. Es regla nos indica que un aspecto es inferido por una palabra si el nombre de *aspecto* esta compuesto por *palabra1* y la descripción de *casoUso* esta formado por *palabra2,* y *palabra1* y *palabra2* son palabras iguales.

[palabraComponeAspectoCandidato:
(?aspecto cc:esta_compuesto_por ?palabra1),
(?casoUso cc:esta_formado_por ?palabra2),
equal(?palabra1,?palabra2)
→ (?aspecto cc:es_inferido_por ?palabra1)]

Las consultas realizadas en el modelo se muestran en forma de tripletas donde O es el conjunto de aspectos candidatos y V son los valores formados por las palabras raíces que infieren el aspecto: es_inferido_por(O, V). El conjunto de resultados es una lista de palabras que intervienen en la identificación del aspecto candidato O.

5.4 Proceso de razonamiento

El proceso de razonamiento en un sistema basado en reglas es una progresión desde un conjunto inicial de afirmaciones y reglas hacia una solución, respuesta o conclusión [HMCQ04]. Algo que es verdadero o conocido es un hecho [JENA], los datos representados en la ontología son hechos.

Los razonadores son los encargados de inferir conocimiento en una ontología. El proceso de inferencia es deducir información adicional que se encuentra implícita. Los razonadores que se utilizaron en el trabajo son soporte de inferencia de Jena (Figura 5.11) [JENA]:

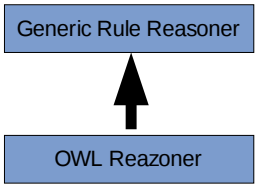

Figura 5.11: Soporte de inferencia de Jena

Generic Rule Reasoner es un razonador de inferencia basada en reglas. Cada regla es una lista de premisas y conclusiones T1, T2,….,Tn → T0 donde los T son tripletas tales como (x? Rdf:type C?). Si T1, … Tn matchea en el conjunto de datos, entonces T0 puede ser inducido [JENA]. Generic Rule Reasoner tiene dos motores de inferencia, se pueden usar los dos motores a la vez: Encadenamiento Progresivo y Encadenamiento Regresivo. Para configurar el motor de inferencia que se desea utilizar se usan las propiedades del razonador, si no se le define ninguno, el razonador trabaja por defecto con un motor de inferencia hibrido, que es una combinación de ambos motores.

5.4.1 Encadenamiento progresivo (Fordward chiang engine)

Se puede partir considerando todos los datos conocidos y luego ir progresivamente avanzando hacia la solución [HMCQ04].

- Se chequea cada una de las reglas para ver si los datos satisfacen las premisas de alguna de las reglas.
- Si una regla es satisfecha es ejecutada derivando nuevos hechos que pueden ser utilizados por otras reglas para derivar hechos adicionales.

En el caso del razonamiento progresivo, se comienza a partir de un conjunto de datos y se evoluciona hacia una conclusión. Se chequea cada una de las reglas para ver si los datos observados satisfacen las premisas de alguna de las reglas. Si una regla es satisfecha, es ejecutada derivando nuevos hechos que pueden ser utilizados por otras reglas para derivar hechos adicionales. Este proceso de chequear reglas para ver si pueden ser satisfechas se denomina interpretación de reglas. Esta progresión hace que se vayan conociendo nuevos hechos o descubriendo nuevas afirmaciones, a medida que va guiando hacia la solución del problema.

5.4.2 Encadenamiento regresivo (Backward chiang engine)

Se puede seleccionar una posible solución y tratar de probar su validez buscando evidencia que la apoye [HMCQ04]:

- Parte de los objetivos y trata de cumplir condiciones necesarias para llegar a ellos.
- Cuando el modelo inferido es consultado, transforma la consulta en un objetivo.
- Ejecución similar a motores Prolog.

El mecanismo de inferencia, o intérprete de reglas para el razonamiento regresivo, difiere significativamente del mecanismo de razonamiento progresivo. El razonamiento regresivo empieza con la conclusión deseada y decide si los hechos que existen pueden dar lugar a la obtención de un valor para esta conclusión. El razonamiento regresivo sigue un proceso muy similar a la búsqueda primero en profundidad.

El sistema empieza con un conjunto de hechos conocidos que típicamente está vacío. Se proporciona una lista ordenada de objetivos (o conclusiones), para las cuales el sistema trata de derivar valores. El proceso de razonamiento regresivo utiliza esta lista de objetivos para coordinar su búsqueda a través de las reglas de la base de conocimiento.

5.4.3 Ejemplo de inferencia

A continuación se presenta un ejemplo de inferencia [JENA] con los motores de inferencia de Encadenamiento Progresivo y Encadenamiento Regresivo.

Objetivo: A is C?

Datos: A is B, B is C

Reglas de Inferencia:

is1 = (?A is ?B) (?B is ?C) (?A is ?C)

is2 = (?A is ?B) (?A is ?C)

Inferencia con Fordward chaining Engine:

- Aplica la regla is2 y no llega a cumplir el objetivo.

- Aplica la regla is1 y llega a cumplir el objetivo.

Inferencia con Backward chaining Engine:

- El objetivo se matchea con el objetivo de is2, luego a partir de los hechos no se cumplen la premisas.
- El objetivo se matchea con el objetivo de is1, luego a partir de los hechos se cumplen la premisas.

5.5 Conclusión

En este capítulo se presento la ontología que fue construida para la incorporación a Aspect Extractor Tool del algoritmo de identificación de aspectos candidatos usando una ontología. Se presento el modelo en forma gráfica, cada uno de los conceptos que se modelan con sus propiedades y cuáles fueron las herramientas que se usaron para su creación y edición.

Se describieron las actividades que se realizan para la identificación de aspectos candidatos en la sub-tarea Identificar Crosscutting concerns y functional crosscutting concerns de la tarea Elección de aspectos candidatos en Aspect Extractor. Las actividades son: Verificar consistencia de la ontología, Representar los casos de uso del proyecto, Buscar aspectos candidatos en los casos de uso, Buscar aspectos candidatos en los requerimientos especiales del proyecto e Inferir nombres completos de los aspectos candidatos.

Se describieron las reglas de inferencia que se usan para inferir conocimiento implícito dentro del modelo de la ontología. Una de las reglas de inferencia que se presentan permite obtener cuales son los aspectos candidatos que están en situación de conflicto y con cuáles aspectos se produce el conflicto. La otra regla de inferencia permite saber cuáles son las palabras de los casos de uso que intervienen en la derivación de un aspecto candidato.

Capítulo 6: Aspect Extractor Tool con ontología

En esta sección se presenta el funcionamiento de Aspect Extractor Tool con la ontología definida para identificar los aspectos candidatos mediante diferentes ejemplos. Se presentan tres casos de estudio. El Sistema de administración de historias clínicas en el idioma español muestra el funcionamiento del algoritmo que usa la ontología para proyectos en español. El Sistema de administración de historias clínicas en el idioma inglés muestra que el funcionamiento del algoritmo es igual para proyectos en ambos idiomas y el Sistema de administración de artículos que muestra los resultados obtenidos en un ejemplo completo.

6.1 Caso de estudio 1: Sistema de administración de historias clínicas (español)

El ejemplo presenta a un sistema para la administración de historias clínicas en un hospital. Para este sistema de gestión existen tres tipos de usuarios: los profesionales, los directores y el administrador. Cada uno de los pacientes del hospital tiene su historia clínica registrada. Si se atiende un paciente en el hospital por primera vez, el profesional que lo atiende deberá crear la historia clínica para el nuevo paciente. Cada vez que un profesional atiende la consulta de un paciente deberá registrar la consulta realizada en la historia clínica. Los usuarios de tipo director tienen permisos para realizar las operaciones de alta, baja o modificación de los profesionales del hospital. El sistema permite la impresión de informes respecto de los profesionales y las consultas atendidas, sólo algunos usuarios directores tendrán accesos a los diferentes tipos de informes.

Listado de casos de uso:

1. Dar de alta un profesional.
2. Modificar una historia clínica.
3. Asignar usuario y clave para un profesional.
4. Imprimir un informe.
5. Login de un usuario.
6. Modificar un profesional.
7. Dar de baja un profesional.
8. Dar de alta una historia clínica.
9. Dar de baja una historia clínica.
10. Consultar una historia clínica.
11. Gestionar permisos para los informes

Requerimiento especial del proyecto: Gestión adecuada de errores, de forma tal que el sistema sea amigable para el usuario y la navegabilidad sea intuitiva.

Se especifican sólo los casos de uso que se consideran de mayor importancia para mostrar el funcionamiento de Aspect Extractor Tool usando la ontología.

6.1.1 Caso de uso Nº 1

En la Tabla 6.1 se muestra la especificación del caso de uso Dar de alta un profesional que describe la funcionalidad para que un director realice el alta de un profesional en el hospital:

Nombre	Dar de alta un profesional
Descripción	Le permite a un usuario director realizar el alta de un legajo de un profesional.

Actor	Primario: Director / Secundario: Profesional
Precondición	No debe existir un legajo para el profesional.
Suposición	El director debe estar autenticado en el sistema.
Flujo Básico	1. El caso de uso comienza cuando un director desea dar de alta un legajo para un profesional. 2. El sistema comprueba si el director tiene permiso para la operación validando los permisos registrados. 3. El sistema solicita el número de documento del profesional. 4. El sistema comprueba que no exista un legajo con el mismo número de documento. 5. El sistema pide el ingreso de datos: nombre completo, fecha de nacimiento, número de matrícula, especialidad en la cual va a ejercer, títulos obtenidos y referencias laborales, necesarios para crear el legajo 6. El sistema registra los datos del legajo del profesional.
Flujo Alternativo	1. Si en paso 3 el sistema verifica que existe un legajo con el mismo número de documento, se informa al director y se vuelve al paso 2. 2. Si en el paso 4 el sistema comprueba que el director no tiene el permiso para realizar el alta del legajo, se informa de la situación y el caso de uso termina.
Poscondición	Se almacenó la información del nuevo legajo del profesional.

Tabla 6.1: Dar de alta un profesional – Administración de historias clínicas (español)

6.1.2 **Caso de uso Nº 2:**

En la siguiente tabla, Tabla 6.2, se puede ver la especificación del caso de uso donde se describen las actividades para modificar una historia clínica de un paciente:

Nombre	Modificar una historia clínica.
Descripción	Permite a un profesional del hospital registrar la consulta realizada a un paciente.
Actor	Profesional
Precondición	La historia clínica del paciente debe estar registrada.
Suposición	El profesional debe estar autenticado en el sistema.
Flujo Básico	1. El caso de uso comienza cuando un profesional desea modificar una historia clínica de un paciente. 2. El sistema solicita el número de documento del paciente. 3. El sistema comprueba que la historia clínica exista. 4. El sistema muestra los datos de la historia clínica del paciente. 5. El sistema solicita que el profesional ingrese los datos actualizados de la historia clínica.

	6. El sistema almacena los datos actualizados de la historia clínica del paciente.
Flujo Alternativo	1. Si en paso 3 el sistema verifica que no existe una historia clínica que se corresponda con el número de documento ingresado se informa al profesional de la situación y se vuelve al paso 2.
Poscondición	Se almacenó la información actualiza de la historia clínica del paciente.

Tabla 6.2: Modificar una historia clínica – Administración de historias clínicas (español)

6.1.3 **Caso de uso Nº 3**

La especificación del caso de uso Imprimir un informe puede verse en la siguiente tabla (Tabla 6.3):

Nombre	Imprimir un informe.
Descripción	Permite que un director autorizado para la operación realice la impresión de un informe.
Actor	Director
Suposición	El director debe estar autenticado en el sistema.
Flujo Básico	1. El caso de uso comienza cuando un director desea realizar un informe. 2. El sistema solicita el tipo de informe a generar. 3. El sistema comprueba si el director tiene permiso para la generación del tipo de informe seleccionado. 4. El sistema pide el ingreso de los datos para crear el informe según el tipo de informe seleccionado. 5. El sistema busca los datos necesarios para la confección del informe según los datos ingresados. 6. El sistema realiza la impresión del informe.
Flujo Alternativo	1. Si en el paso 3 el sistema comprueba que el director no tiene el permiso para realizar la impresión del tipo del informe, se informa de la situación y se vuelve al paso 2.
Poscondición	Se realizó la impresión del informe.
Requerimiento Especial	La impresión de los informes debe realizarse de forma segura, ninguna persona no autorizada para la tarea debe tener acceso a ésta.

Tabla 6.3: Imprimir un informe – Administración de historias clínicas (español)

6.1.4 **Identificación de aspectos candidatos**

La primera actividad que se realiza para la identificación de aspectos candidatos es representar a cada caso de uso por sus palabras relevantes.

6.1.4.1 Representación interna de los casos de uso

Los casos de uso del ejemplo estudiado quedan representados de la siguiente manera:

Representación caso de uso Nº 1:

Palabras : [alta, profesional, permite, usuario, director, realizar, legajo, debe, autenticado, sistema, existir, caso, comienza, desea, dar, solicita, número, documento, comprueba, exista, permiso, operación, validando, permisos, registrados, pide, ingreso, datos, nombre, completo, fecha, nacimiento, matrícula, especialidad, ejercer, títulos, obtenidos, referencias, laborales, necesarios, crear, registra, almacenó, información, nuevo, permite, realizar, autenticado, existir, dar, solicita, autenticación, registrados, ejercer, obtenidos, laborales, crear]

Raíces : [alta, profesional, permit, usuari, director, realiz, legaj, deb, autentic, sistem, exist, cas, comienz, dese, dar, solicit, numer, document, comprueb, exist, permis, oper, valid, permis, registr, pid, ingres, dat, nombr, complet, fech, nacimient, matricul, especial, ejerc, titul, obten, referent, laboral, necesari, cre, registr, almacen, inform, nuev, permit, realiz, autentic, exist, dar, solicit, autent, registr, ejerc, obten, laboral, cre]

Representación caso de uso Nº 2:

Palabras : [modificación, historia, clínica, permite, profesional, hospital, registrar, consulta, realizada, paciente, debe, autenticado, sistema, registrada, caso, comienza, desea, modificar, solicita, número, documento, comprueba, exista, muestra, datos, ingrese, actualizados, almacena, paso, 3, verifica, existe, corresponda, ingresado, informa, situación, vuelve, 2, almacenó, información, actualiza, permite, registrar, realizada, autenticado, registrada, modificar, solicita, muestra, actualizados, verifica, acceso]

Raíces : [modif, histori, clinic, permit, profesional, hospital, registr, consult, realiz, pacient, deb, autentic, sistem, registr, cas, comienz, dese, modific, solicit, numer, document, comprueb, exist, muestr, dat, ingres, actualiz, almacen, pas, 3, verif, exist, correspond, ingres, inform, situacion, vuelv, 2, almacen, inform, actualiz, permit, registr, realiz, autentic, registr, modific, solicit, muestr, actualiz, verif, acces]

Representación caso de uso Nº 3:

Palabras : [impresión, informe, permite, director, autorizado, operación, realice, debe, autenticado, sistema, caso, comienza, desea, realizar, solicita, tipo, generar, comprueba, permiso, generación, seleccionado, pide, ingreso, datos, crear, según, busca, necesarios, confección, ingresados, realiza, paso, 3, informa, situación, vuelve, 2, realizó, informes, realizarse, forma, segura, ninguna, persona, autorizada, tarea, acceso, ésta, permite, autorizado, autenticado, realizar, solicita, generar, seleccionado, crear, busca, acceso, informes, realizarse, autorizada]

Raíces : [impresion, inform, permit, director, autoriz, oper, realic, deb, autentic, sistem, cas, comienz, dese, realiz, solicit, tip, gener, comprueb, permis, gener, seleccion, pid, ingres, dat, cre, segun, busc, necesari, confeccion, ingres, realiz, pas, 3, inform, situacion, vuelv, 2, realiz, inform, realizar, form, segur, ningun, person, autoriz, tare, acces, esta, permit, autoriz, autentic, realiz, solicit, gener, seleccion, cre, busc, acces, inform, realizar, autoriz]

6.1.4.2 Búsqueda aspectos candidatos

El siguiente paso es recorrer la ontología buscando los aspectos candidatos que se puedan inferir por cada uno de los casos de uso. A continuación se muestra una tabla (Tabla 6.4) con la cantidad de coincidencias encontradas para cada concepto en el caso de uso Nº 1.

Aspecto candidato	Coincidencias buscadas	Coincidencias encontradas	Palabras que infieren
Comunicación entre Procesos	2	0	

Concurrencia	1	0	
Monitoreo	1	0	
Profiling	1	0	
Logging	1	0	
Tracing	1	0	
Manejo de Errores	2	0	
Performance	1	0	
Seguridad	1	0	
Control de Acceso	2	4	ingreso, permiso, permisos, almacenó
Auditoria	1	0	
Autenticación	1	2	validando, autenticado
Debugging	1	0	
Acceso a Memoria	2	2	almacenó, información
Actualización de Pantalla	2	0	
Distribución de Recursos	2	0	
Restricciones de Tiempo Real	3	0	
Disponibilidad	1	0	
Integrabilidad	1	0	

Tabla 6.4: Coincidencias para Dar de alta un profesional (español)

Al recorrer la lista de coincidencias encontradas para el caso de uso Nº 1 se va estudiando cuales son los posibles aspectos candidatos se han inferido. Se compara si la cantidad buscada de coincidencias es mayor o igual a la cantidad obtenida, de ser así el aspecto pasa a formar parte de la lista de aspectos candidatos del proyecto. De esta forma la lista de aspectos candidatos del proyecto formada por analizar el caso de uso Nº 1 es la siguiente: *"Control de Acceso", "Autenticación"* y *"Acceso a Memoria".*

El siguiente caso de uso a analizar es el caso de uso Nº 2, la tabla con la cantidad de coincidencias para cada concepto es mostrada a continuación (Tabla 6.5):

Aspecto candidato	**Coincidencias buscadas**	**Coincidencias encontradas**	**Palabras que infieren**
Comunicación entre Procesos	2	0	
Concurrencia	1	0	
Monitoreo	1	0	
Profiling	1	0	
Logging	1	0	
Tracing	1	0	
Manejo de Errores	2	0	
Performance	1	0	
Seguridad	1	0	
Control de Acceso	2	5	ingresado, ingrese, consulta, almacena,

			almacenó
Auditoria	1	0	
Autenticación	1	0	
Debugging	1	0	
Acceso a Memoria	2	6	almacena, datos, almacenó, informa, consulta, información
Actualización de Pantalla	2	0	
Distribución de Recursos	2	0	
Restricciones de Tiempo Real	3	0	
Disponibilidad	1	0	
Integrabilidad	1	0	

Tabla 6.5: Coincidencias para Modificar una historia clínica (español)

A la lista de aspectos candidatos del proyecto no se agrega ningún aspecto. Se agrega la referencia al caso de uso analizado en los aspectos candidatos *"Control de Acceso"* y *"Acceso a Memoria"*. El caso de uso Nº 3 es el siguiente caso de uso que se analiza, la tabla con la cantidad de coincidencias encontradas para cada concepto se muestra en la Tabla 6.6:

Aspecto candidato	**Coincidencias buscadas**	**Coincidencias encontradas**	**Palabras que infieren**
Comunicación entre Procesos	2	0	
Concurrencia	1	0	
Monitoreo	1	0	
Profiling	1	0	
Logging	1	0	
Tracing	1	0	
Manejo de Errores	2	0	
Performance	1	0	
Seguridad	1	1	segura
Control de Acceso	2	5	ingreso, permiso, ingresados, segura, acceso
Auditoria	1	0	
Autenticación	1	0	
Debugging	1	0	
Acceso a Memoria	2	6	acceso, informe, ingreso, permiso, ingresados, datos
Actualización de Pantalla	2	0	
Distribución de Recursos	2	0	
Restricciones de Tiempo Real	3	0	
Disponibilidad	1	0	
Integrabilidad	1	0	

Tabla 6.6: Coincidencias para Imprimir un informe (español)

El caso de uso Nº 3 agrega a la lista de aspectos candidatos del proyecto el aspecto candidato *"Seguridad"* con una referencia al caso de uso, y se agrega la referencia al caso de uso Nº 3 a los aspecto candidatos *"Control de Acceso",* y *"Acceso a Memoria".*

6.1.4.3 Representación del requerimiento especial

Los requerimientos especiales que el proyecto tiene son representados de igual forma que un caso de uso, por una lista de palabras relevante y sus raíces.

Palabras → [gestión, adecuada, errores, forma, sistema, sea, amigable, usuario, navegabilidad, sea, intuitiva]

Raíces → [gestion, adecu, error, form, sistem, sea, amig, usuari, naveg, sea, intuitiv]

Al recorre una vez más la ontología para buscar aspectos candidatos en el requerimiento especial, se agrega al proyecto el aspecto candidato *"Manejo de Errores"*, la lista de aspectos candidatos del proyecto queda forma como se muestra en la Tabla 6.7. La lista de resultados retornados por Aspect Extractor Tool es la que se puede ver en la Figura 6.1.

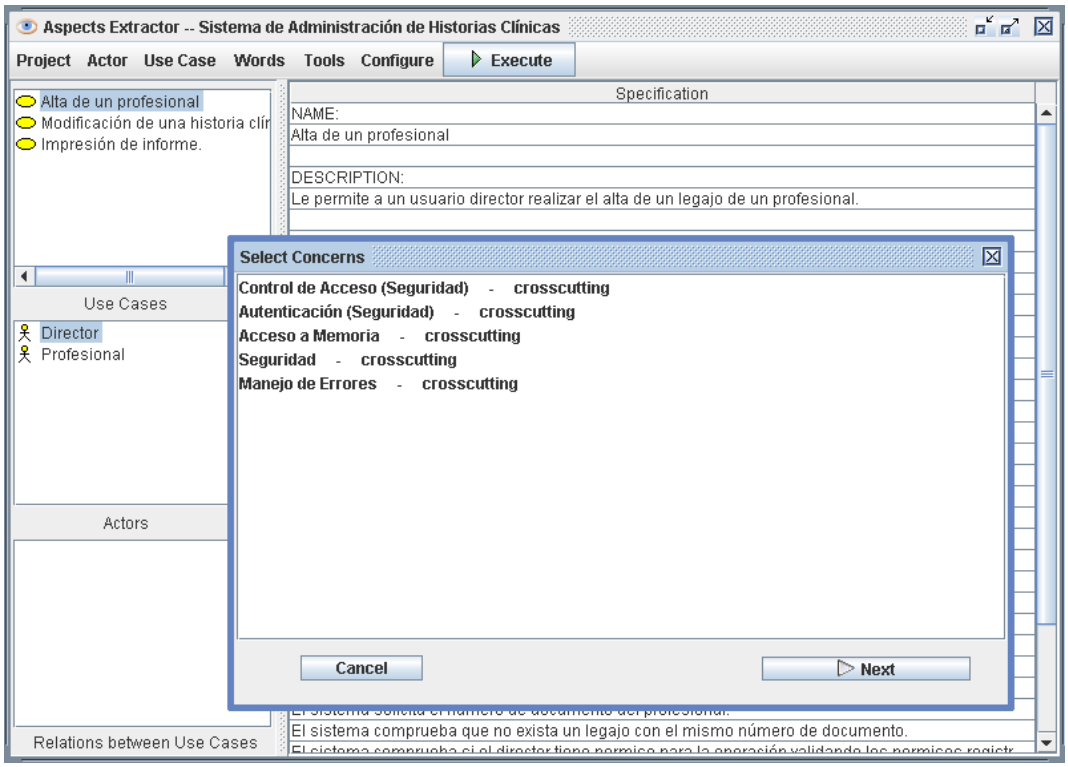

Figura 6.1: Aspectos candidatos para Administración de historias clínicas (español)

Aspecto Candidato	CU N° 1	CU N° 2	CU N° 3
Control de Acceso	X	X	X
Autenticación	X		
Acceso a Memoria	X	X	X
Seguridad			X

Manejo de Errores	X	X	X

Tabla 6.7: Aspectos candidatos para Administración de historias clínicas (español)

La lista de resultados retornados por Aspect Extractor Tool muestra resultados precisos, cada uno de los aspectos candidatos propuestos son aspectos candidatos válidos para el proyecto. También se puede ver que los nombres de los aspectos candidatos mostrados no confunden a los stakeholders del proyecto, dan una idea clara de lo que se quiere decir, esto es porque son conceptos estandarizados en la ontología con la que se trabaja.

6.1.5 Inferencia de nombres

Para inferir los nombres completos para los aspectos candidatos identificados se realizan las siguientes consultas sobre la ontología, donde V son todos los posibles valores para la superclase del objeto en cuestión:

- subClassOf(Control de Acceso, V) □ {Seguridad}
- subClassOf(Autenticación, V) □ {Seguridad}
- subClassOf(Acceso a Memoria, V) □ {}
- subClassOf(Seguridad, V) □ {}
- subClassOf(Manejo de Errores, V) □ {}

El conocimiento semántico que se infiere es que los aspectos Autenticación y Control de Acceso son un tipo especializado del aspecto Seguridad. El nombre completo del aspecto Autenticación es Autenticación (Seguridad) y del aspecto Control de Acceso es Control de Acceso (Seguridad) como se puede observar en la Figura 6.1.

6.1.6 Creación del modelo para la inferencia

Para verificar los aspectos candidatos propuestos por la herramienta desde las palabras que componen a los casos de uso y a los aspectos candidatos y obtener las situaciones de conflicto producidas es necesario crear un modelo de instancias. Este modelo se crea con las propiedades de los aspectos y de los casos de uso tales como tipo_aspecto, esta_compuesto_por, esta_formado_por, esta_asociado, etc. El modelo de instancias generado para este ejemplo se adjunta en el documento: *Ontología de Instancias de Aspectos Candidatos y Casos de Uso (ingles).doc*. A continuación se muestran las propiedades del aspecto candidato Control de Acceso:

- tipo_aspecto(Control de Acceso, Control de Acceso)
- esta_compuesto_por(Control de Acceso, acces)
- esta_compuesto_por(Control de Acceso, permis)
- esta_compuesto_por(Control de Acceso, handl)
- esta_compuesto_por(Control de Acceso, organiz)
- esta_compuesto_por(Control de Acceso, consult)
- esta_compuesto_por(Control de Acceso, entrad)
- esta_compuesto_por(Control de Acceso, ingres)
- esta_compuesto_por(Control de Acceso, administr)
- esta_compuesto_por(Control de Acceso, administers)
- esta_compuesto_por(Control de Acceso, manag)
- esta_compuesto_por(Control de Acceso, gestion)
- esta_compuesto_por(Control de Acceso, manten)
- esta_compuesto_por(Control de Acceso, direccion)

- esta_compuesto_por(Control de Acceso, controls)
- esta_compuesto_por(Control de Acceso, manej)
- esta_compuesto_por(Control de Acceso, control)
- esta_compuesto_por(Control de Acceso, almacen)
- esta_compuesto_por(Control de Acceso, commands)
- esta_compuesto_por(Control de Acceso, mand)
- esta_compuesto_por(Control de Acceso, maintain)

6.1.7 Identificación de conflictos

Para derivar la existencia de conflictos entre aspectos se realizan las siguientes consultas, donde V son todos los posibles aspectos con el cual el objeto O se encuentra en situación de conflicto:

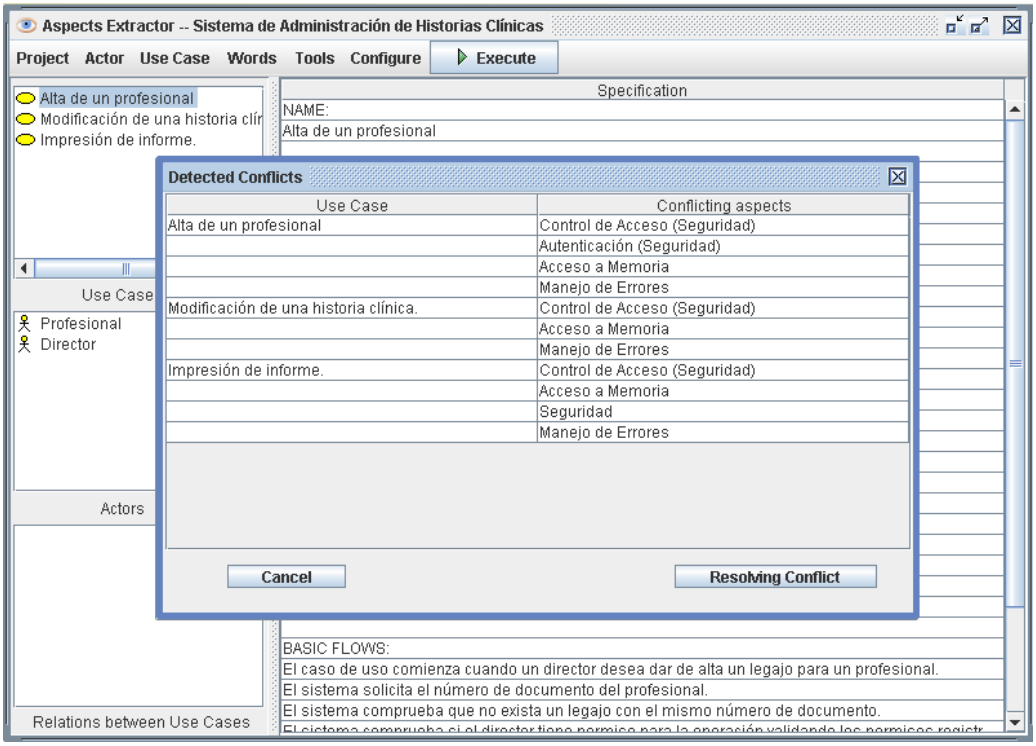

Figura 6.2: Situaciones de conflicto para Administración de historias clínicas (español)

esta_en_conlficto(O, V) □ {Acceso a Memoria, Autenticación (Seguridad)}
□ {Acceso a Memoria, Seguridad}
□ {Acceso a Memoria, Control de Acceso (Seguridad)}
□ {Acceso a Memoria, Manejo de Errores}
□ {Seguridad, Acceso a Memoria}
□ {Seguridad, Manejo de Errores}
□ {Seguridad, Control de Acceso (Seguridad)}
□ {Manejo de Errores, Acceso a Memoria}
□ {Manejo de Errores, Autenticación (Seguridad)}
□ {Manejo de Errores, Control de Acceso (Seguridad)}
□ {Manejo de Errores, Seguridad}
□ {Autenticación (Seguridad), Control de Acceso (Seguridad)}
□ {Autenticación (Seguridad), Acceso a Memoria}

- {Autenticación (Seguridad), Manejo de Errores}
- {Control de Acceso (Seguridad), Acceso a Memoria}
- {Control de Acceso (Seguridad), Seguridad}
- {Control de Acceso (Seguridad), Manejo de Errores}
- {Control de Acceso (Seguridad), Autenticación (Seguridad)}

En la Figura 6.2 se muestran las situaciones de conflicto encontradas por Aspect Extrator Tool en la tarea de Identificar Conflictos. Puede verse que las situaciones de conflicto encontradas con el razonador son las mismas que las encontradas con Aspect Extractor Tool.

6.1.8 Inferencia de palabras

Para determinar cuáles son las palabras del proyecto que intervienen en la derivación de los aspectos candidatos identificados, la consulta y el conjunto de resultados obtenidos del modelo es el mostrado a continuación, donde O es el conjunto de aspectos candidatos y V son los valores formados por las palabras raíces que infieren el aspecto. Los valores que se muestran del lado derecho de la tripleta están indicando en que parte del proyecto esta la palabra que intervino en la inferencia del aspecto candidato.

es_inferido_por(O, V)
- {Manejo de Errores, gestion} (Req. Especial)
- {Manejo de Errores, error} (Req. especial)
- {Autenticación (Seguridad), segur} (CU3)
- {Autenticación (Seguridad), valid} (CU1)
- {Autenticación (Seguridad), autent} (CU 1, CU 2, CU3)
- {Acceso a Memoria, permis} (CU 1, CU3)
- {Acceso a Memoria, almacen} (CU 1, CU2)
- {Acceso a Memoria, dat} (CU 1, CU 2, CU3)
- {Acceso a Memoria, acces} (CU3)
- {Acceso a Memoria, inform} (CU 1, CU2, CU3)
- {Acceso a Memoria, consult} (CU2)
- {Control de Acceso (Seguridad), gestion} (Req. especial)
- {Control de Acceso (Seguridad), acces} (CU3)
- {Control de Acceso (Seguridad), consult} (CU2)
- {Control de Acceso (Seguridad), ingres} (CU 1, CU 2, CU3)
- {Control de Acceso (Seguridad), almacen} (CU 1, CU2)
- {Control de Acceso (Seguridad), permis} (CU 1, CU3)
- {Control de Acceso (Seguridad), segur} (CU3)
- {Seguridad, segur} (CU3)

6.2 Caso de estudio 2: Sistema de administración de historias clínicas (inglés)

Este caso de estudio presenta al Sistema de administración de historias clínicas en el idioma ingles.

Projects special requirements: Appropriate error management, in order to have a friendly, intuitive and easy for system's users to navigate.

6.2.1 Caso de uso Nº 1

La especificación del caso de uso N° 1 en el idioma inglés se muestra en la siguiente tabla, Tabla 6.8:

Name	Create Professional

Description	Allows a director user to create a professional personal file with personal information.
Actor	Director, Professional
Precondition	A professional personal file does not exist.
Assumption	Director must be authenticated in the system.
Basic Flow	1. Use case starts when a director creates a professional docket. 2. System checks if director has permission for this operation validating his registered permissions. 3. The system requires the Director to enter a professional identification number DNI. 4. The system checks that there is no a professional personal file with same DNI number. 5. System asks to enter data to create professional personal file: full name, born date, enrolment number, field work, obtained degrees, work references. 6. The system registers the professional personal file data.
Alternative Flow	1. In step 3, if system verifies that a professional personal file already exists with same DNI number, director is advised and go to step 2. 2. In step 4, if system verifies that director has no right permissions to create a professional file, director is advised and use case end.
Poscondition	New professional file information was stored.

Tabla 6.8: Create professional - Administración de historias clínicas (inglés)

6.2.2 **Caso de uso Nº 2**

La especificación del caso de uso Medical Record update se muestra en la Tabla 6.9:

Name	Medical Record update.
Description	Allows a hospital's professional to register a patient's medical record.
Actor	Professional
Precondition	Medical record must be already registered.
Assumption	Professional user must be authenticated into the system.
Basic Flow	1. Use case starts when a professional modifies a patient's medical record. 2. The system asks for patient's DNI. 3. The system checks that medical record exists. 4. The system shows patient's medical record data. 5. The system asks the professional to enter patient's updated medical record data. 6. System stores patient's updated medical record data.

Alternative Flow	1. In step 3, if the system verifies that there is no a patient's medical record that matches entered DNI number then professional is advised and goes to 2.
Poscondition	Patient's updated medical record was stored.

Tabla 6.9: Medical record update - Administración de historias clínicas (inglés)

6.2.3 Caso de uso Nº 3

En la Tabla 6.10 se puede ver la especificación del caso de uso Report printing.

Name	Report printing.
Description	Allows an authorized director to realise a report printing.
Actor	Director
Assumption	Director must be authenticated into the system.
Basic Flow	1. Use case starts when a director wants to print a report. 2. System asks for report type to be generated. 3. The system checks if director has right permissions to realise a report printing. 4. System asks for entering data to print selected report type. 5. System recovers data from repository to print selected report type. 6. System prints report.
Alternative Flow	1. In step 5, if system checks that director has no rights permissions to do an operation, director is advised and goes to 2.
Poscondition	Report was successfully printed.
Special Requirements	Report printing must be done in a secure way, non authorized users could not execute unauthorized tasks.

Tabla 6.10: Report printing - Administración de historias clínicas (inglés)

6.2.4 Identificación de aspectos candidatos

La primera actividad a realizar para la obtención de los aspectos candidatos es representar a cada caso de uso por una lista de sus palabras relevantes y sus raíces:

6.2.4.1 Representación interna de los casos de uso

Luego de recorrer los casos de uso aplicando las técnicas de Stops Word y Stemming, los casos de uso del proyecto en inglés quedan representados de la siguiente manera:

Representación caso de uso Nº 1:

Palabras [create, professional, allows, director, user, personal, file, information, authenticated, system, exist, use, case, starts, creates, docket, requires, enter, identification, number, dni, checks, permission, operation, validating, registered, permissions, asks, data, full, name, born,

date, enrolment, field, work, obtained, degrees, references, registers, step, 3, verifies, already, exists, advised, 2, 4, right, end, stored, checks, registered, registers, verifies]

Raíces → [creat, profession, allow, director, user, person, file, inform, authent, system, exist, us, case, start, creat, docket, requir, enter, identif, number, dni, check, permiss, oper, valid, regist, permiss, ask, data, full, name, born, date, enrol, field, work, obtain, degre, refer, regist, step, 3, verifi, alreadi, exist, advis, 2, 4, right, end, store, check, regist, regist, verifi]

Representación caso de uso Nº 2:

Palabras → [medical, record, update, allows, hospital's, professional, register, patient's, user, authenticated, system, already, registered, use, case, starts, modifies, asks, dni, checks, exists, shows, data, enter, updated, stores, step, 3, verifies, matches, entered, number, advised, 2, stored, register, registered, checks, shows, verifies]

Raíces → [[medic, record, updat, allow, hospital', profession, regist, patient', user, authent, system, alreadi, regist, us, case, start, modifi, ask, dni, check, exist, show, data, enter, updat, store, step, 3, verifi, match, enter, number, advis, 2, store, regist, regist, check, show, verifi]

Representación caso de uso Nº 3:

Palabras → [report, printing, allows, authorized, director, realise, authenticated, system, use, case, starts, wants, print, asks, type, generated, checks, right, permissions, entering, data, selected, recovers, repository, prints, step, 5, rights, operation, advised, 2, successfully, printed, done, secure, non, users, execute, unauthorized, tasks, checks]

Raíces → [report, print, allow, author, director, realis, authent, system, us, case, start, want, print, ask, type, gener, check, right, permiss, enter, data, select, recov, repositori, print, step, 5, right, oper, advis, 2, success, print, done, secur, non, user, execut, unauthor, task, check]

6.2.4.2 Búsqueda aspectos candidatos

La siguiente tarea es recorrer la ontología una vez por cada caso de uso buscando los aspectos candidatos. Al realizar el recorrido de la ontología para el caso de uso N° 1 se identifican los aspectos candidatos *"Access Control", "Autentication"* y *"Access to Memory",* la tabla de coincidencias encontradas es la tabla que se muestra a continuación (Tabla 6.11):

Aspecto candidato	Coincidencias buscadas	Coincidencias encontradas	Palabras que infieren
Communication between Processes	2	0	
Concurrence	1	0	
Monitoring	1	0	
Profiling	1	0	
Logging	1	0	
Tracing	1	0	
Error Handling	2	0	
Performance	1	0	
Security	1	0	
Access Control	2	2	asks, enter
Audit	1	0	
Autentication	1	2	validating, authenticated
Debugging	1	0	
Access to Memory	2	2	asks, enter

Screen Update	2	0	
Distribution of Resources	2	0	
Restrictions of Real Time	3	0	
Availability	1	0	
Integrability	1	0	

Tabla 6.11: Coincidencias para Create professional (inglés)

Se recorre en profundidad la ontología buscando aspectos candidatos para el caso de uso N° 2. En este caso se identifican los aspectos candidatos *"Access Control", "Autentication" y "Access to Memory"*. Estos aspectos candidatos ya se encontraban en la lista de aspectos candidatos del proyecto, se agregan las referencias al caso de uso N° 2. La tabla de coincidencias encontradas se muestra a continuación (Tabla 6.12):

Aspecto candidato	**Coincidencias buscadas**	**Coincidencias encontradas**	**Palabras que infieren**
Communication between Processes	2	0	
Concurrence	1	0	
Monitoring	1	0	
Profiling	1	0	
Logging	1	0	
Tracing	1	0	
Error Handling	2	0	
Performance	1	0	
Security	1	0	
Access Control	2	3	asks, enter, entered
Audit	1	0	
Autentication	1	1	authenticated
Debugging	1	0	
Access to Memory	2	3	enter, entered, asks
Screen Update	2	0	
Distribution of Resources	2	0	
Restrictions of Real Time	3	0	
Availability	1	0	
Integrability	1	0	

Tabla 6.12: Coincidencias para Create professional (inglés)

Se recorre una vez más la ontología buscando los aspectos candidatos para el caso de uso N° 3, son identificados los aspectos candidatos *"Security", "Access Control", "Access to Memory"* y *"Autentication"*. Para los aspectos que ya se encontraban en la lista de aspectos candidatos del proyecto, se almacena una referencia al caso de uso y se incorpora *"Security"* junto a la referencia al caso de uso. La tabla de coincidencias encontradas para este caso se muestra en la Tabla 6.13.

Aspecto candidato	Coincidencias buscadas	Coincidencias encontradas	Palabras que infieren
Communication between Processes	2	0	
Concurrence	1	0	
Monitoring	1	0	
Profiling	1	0	
Logging	1	0	
Tracing	1	0	
Error Handling	2	0	
Performance	1	0	
Security	1	1	secure
Access Control	2	2	asks, secure
Audit	1	0	
Autentication	1	2	secure, authenticated
Debugging	1	0	
Access to Memory	2	2	asks, entering
Screen Update	2	0	
Distribution of Resources	2	0	
Restrictions of Real Time	3	0	
Availability	1	0	
Integrability	1	0	

Tabla 6.13: Coincidencias para Report printing (inglés)

6.2.4.3 Representación del requerimiento especial

Un vez que se terminaron de analizar los casos de uso que forman parte del proyecto Administración de historias clínicas en el idioma inglés, falta analizar el requerimiento especial. El requerimiento especial, se representa por sus palabras relevantes, del mismo modo que con los casos de uso. Las palabras que representan al requerimiento especial del proyecto se pueden ver a continuación:

Palabras → [appropriate, error, management, order, friendly, intuitive, easy, systems, users, navigate]

Raíces → [appropri, error, manag, order, friend, intuit, easi, system, user, navig]

Se realiza otro recorrido de la ontología para buscar aspectos candidatos en el requerimiento especial del proyecto, éste aporta un aspecto candidato a la lista de aspectos candidatos del proyecto, el aspecto *"Error Handling"*, este aspecto por provenir de un requerimiento especial del proyecto afecta a todos los casos de usos. Se agrega el aspecto detectado a la lista de aspectos candidatos del proyecto junto a las referencias a los casos de uso. La lista resultado de aspectos candidatos del proyecto queda formada como se muestra en la Tabla 6.14. Los resultados identificados en Aspect Extractor Tool son mostrados en Figura 6.3.

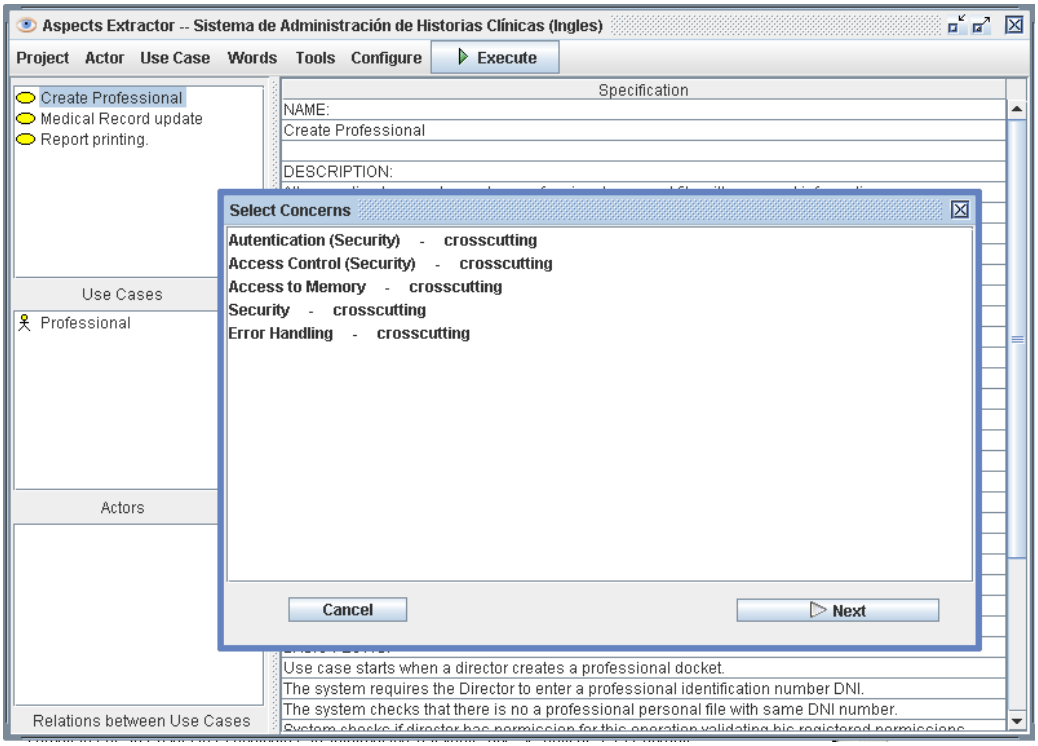

Figura 6.3: Aspectos candidatos para Administración de historias clínicas (inglés)

Aspecto Candidato	CU N° 1	CU N° 2	CU N° 3
Access Control	X	X	X
Autentication	X	X	
Access to Memory	X	X	X
Security			X
Error Handling	X	X	X

Tabla 6.14: Aspectos candidatos para Administración de historias clínicas (inglés)

Como se puede observar la lista de aspectos candidatos obtenidos para el proyecto en inglés es similar a la lista obtenida para el proyecto en español, esta situación puede verse en las figuras Figura 6.1 y Figura 6.3. Esto sucede porque el algoritmo trabaja de la misma forma para todos los idiomas.

6.2.5 Inferencia de nombres

Cuando se usa el razonador genérico OWL Reasoner para obtener los nombres completos de los aspectos candidatos, la información relevante que se obtiene es el nombre completo para los aspectos *"Autentication"* y *"Access Control"*. Al ejecutar las consultas sobre la ontología se obtienen los siguientes resultados:

- subClassOf(Access Control, V) □ {Security}
- subClassOf(Autentication, V) □ {Security}
- subClassOf(Access to Memory, V) □ {}
- subClassOf(Security, V) □ {}

- subClassOf(Error Handling, V) □ {}

6.2.6 **Creación del modelo para la inferencia**

Para buscar cuáles son las palabras del proyecto que infieren los aspectos candidatos propuestos por la herramienta y establecer las situaciones de conflicto existentes, es necesario crear un modelo con instancias de las propiedades de los aspectos candidatos propuestos y de los casos de uso a los que estos aspectos afectan. El modelo instanciado generado para este ejemplo se adjunta en el documento: *Ontología de Instancias de Aspectos Candidatos y Casos de Uso (ingles).doc*.

6.2.7 **Identificación de conflictos**

Las consultas realizadas en el modelo se muestran en forma de tripletas, donde V son todos los posibles aspectos con el cual el objeto O se encuentra en situación de conflicto:

esta_en_conlficto(O, V) □ {Security, Autentication (Security)}
□ {Security, Access to Memory}
□ {Security, Error Handling}
□ {Security, Access Control (Security)}
□ {Autentication (Security), Security}
□ {Autentication (Security), Access Control (Security)}
□ {Autentication (Security), Access to Memory}
□ {Autentication (Security, Error Handling}
□ {Error Handling, Security}
□ {Error Handling, Access to Memory}
□ {Error Handling, Access Control}
□ {Error Handling, Autentication (Security)}
□ {Access Control (Security), Error Handling}
□ {Access Control (Security), Access to Memory}
□ {Access Control (Security), Security}
□ {Access Control (Security), Autentication (Security)}
□ {Access to Memory, Error Handling}
□ {Access to Memory, Access Control}
□ {Access to Memory, Autentication (Security)}
□ {Access to Memory, Security}

En la Figura 6.4 se muestran las situaciones de conflicto encontradas por la herramienta para el ejemplo. Puede verse que los conflictos encontrados por el razonador y por Aspect Extractor Tool son iguales.

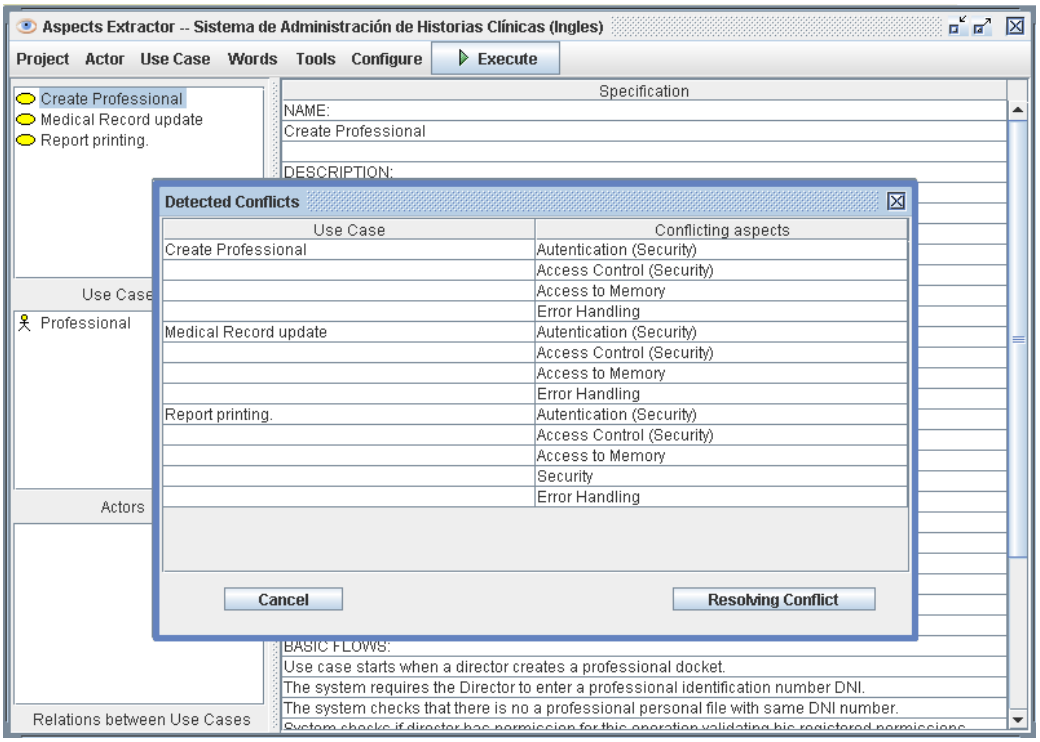

Figura 6.4: Situaciones de conflicto para Administración de historias clínicas (inglés)

6.2.8 Inferencia de palabras

El siguiente conjunto de datos que se obtienen son todas las palabras que infieren a los aspectos candidatos. La consulta y el conjunto de resultados obtenidos del modelo es el mostrado a continuación, donde O es el conjunto de aspectos candidatos y V son los valores formados por las palabras raíces que infieren el aspecto. Los valores que se muestran del lado derecho de la tripleta están indicando en que parte del proyecto esta la palabra que intervino en la inferencia del aspecto candidato:

es_inferido_por(O, V) □ {Autentication (Security), secur} (CU3)
□ {Autentication (Security), valid} (CU1)
□ {Autentication (Security), authent} (CU1, CU2, CU3)
□ {Access to Memory, enter} (CU3)
□ {Security, secur} (CU3)
□ {Access Control (Security), ask} (CU3)
□ {Error Handling, error} (Req. Especial)
□ {Error Handling, manag} (Req. Especial)
□ {Access to Memory, ask} (CU3)
□ {Access Control (Security), manag} (Req. Especial)
→ {Access Control (Security), enter} (CU3)

6.3 Caso de estudio 3: Administración de artículos

En esta sección se analizan los resultados de un caso de estudio completo. Se analizan los aspectos candidatos propuestos por los dos algoritmos de identificación de aspectos que Aspect Extractor Tool posee.

Listado de casos de uso:

1. Consult Review
2. Register Article
3. Register Preference
4. Review Article
5. Maintain User Profile
6. View Statistical Data
7. View Log
8. Login

Requerimiento especial del proyecto:

1. Data Persistency: All the data generated and used by the Automatic Review System shall be saved using a persistence System. This probably will be a database.
2. Secure Login: All the users must login before using the System, and this login must be secure. Additionally it must be allowed to use more than one type of secure login for the verification.
3. Visualization: The user interface shall be designed according the type of user that is using it, and it must be easy to change it.
4. Availability: The Automatic Review System shall be available 24 hours a day, 7 days a week. There shall be no more than 10% down time.
5. Internal Database Access Response Time: The system must provide access to the internal database with no more than 2 second latency.
6. External Database Access Response Time: The system must provide access to the Article Database System with no more than 10 second latency.
7. Article Database System: The system shall integrate with the existing Article Database System, and it must comply it's communication standards, security policies, etc.

6.3.1 **Caso de uso Nº 1**

En la Tabla 6.15 se presenta la especificación del caso de uso Consult Review, se describe la funcionalidad para la consulta de articulos en diferentes vistas y listados.

Name	Consult Review
Description	This use case allows a Writer to obtain and view his article reviews. The Writer may list all the reviews sorted by date, reviewer, score, or any combination of those. The main actor of this use case is the Writer. The Article Database System is an actor within the use case.
Actor	Writer
Precondition	
Assumption	
Basic Flow	Consult An Article Review 1. The System retrieves the list of articles the Writer published. 2. The System display the list of his articles, showing the name and the last date the article was reviewed (if any). 3. The Writer selects the article to consult, and specifies the sort method required for the presentation of the consult. That includes

	any combination of: a. Date. b. Reviewer. c. Overall recommendation. d. Average evaluations criteria grade. e. Particular evaluation criteria grade. 4. The System retrieves the article interacting with the Article Database System. 5. The System format the consult according to the sort method specified. 6. The System shows the article and the result of the consult sorted. 7. When the Writer has finished viewing the reviews the use case ends.
Alternative Flow	No Articles Found If in the "Consult An Article Review" basic-flow the System retrieves zero articles published, an error message is displayed saying "No articles found". The Writer acknowledges the error and the use case finishes at this point. No Reviews Found If in the "Consult An Article Review" basic-flow the System didn't found any reviews made to that article, an error message is displayed saying "No reviews found". The Writer acknowledges the error and the use case finishes at this point. Article Database System Unavailable If the System is unavailable to establish a communication with the Article Database System after a specified lapse of time, the System will display an error message saying "Article Database System unavailable". The Writer acknowledges the error and the use case finishes at this point.
Poscondition	
Special Requirements	Information Log: All the action performed in the flow of the use case must be logged into the System, including transactions made with the Article Database System, errors occurred, consults made, etc. Communication Security: The communication established with the Article Database System must be done in agreement with the security standards of this system.

Tabla 6.15: Consult Review - Administración de artículos

6.3.2 **Caso de uso Nº 2**

La especificación del caso de uso para registrar los articulos se puede ver en Tabla 6.16:

Name	Register Article
Description	This use case allows a Writer to register an article in the Article Database System. The Writer can also modify or delete articles already published in the Article Database System. And an additional feature of the Automatic Review System, present in this use case, is that it will allow a Writer to save a draft of the article in the System, before he has to publish it in the Article Database System. The main actor of this use case is the Writer. The Article Database

	System is an actor within the use case.
Actor	Writer
Precondition	Login: Before this use case begins the Writer has logged into the system.
Assumption	
Trigger	The use case begins when the Writer selects the “Register article” activity.
Basic Flow	Add Article 1. The Writer selects the “Add article” activity. 2. The System displays a blank form. 3. The Writer writes the article filling the blank form. The information provided is: a. Article title. b. Article body. c. Article topics. 4. Once the Writer has finished, it selects the “Publish article” activity. 5. The “Operate With Article Database System” alternative-flow is performed at this step. 6. The System adds the article.
Alternative Flow	Modify Article 1. The Writer presses the “Modify article” activity. 2. The System retrieves and displays a list to all the writer’s articles, interacting with the Article Database System, excepting the article drafts, which are read from the Automatic Review System. 3. The Writer selects the article he/she want to modify. 4. The System retrieves and displays in a form the article selected, interacting with the Article Database System. 5. The Writer can then modify the article, including the article title, body and topics. Once the edits are complete the Writer selects the “Publish modified article” activity. 6. The “Operate With Article Database System” alternative-flow is performed at this step. 7. The System updates the article. Delete Article 8. The Writer presses the “Delete article” activity. 9. The System retrieves and displays a list to all the writer’s articles, interacting with the Article Database System. 10. The Writer selects the article he/she want to delete. 11. The System retrieves and displays in a form the article selected, interacting with the Article Database System. 12. The Writer selects the “Delete article” article. 13. The System asks for a delete confirmation, before proceeding to the deletion. 14. If the Writer confirms the use case continues, otherwise the use case goes to step 2. 15. The “Operate With Article Database System” alternative-flow is performed at this step. 16. The System deletes the article. Save Article Draft At any point, the Writer may choose to save a draft without submitting

	it by selecting the “Save draft” activity. The current article is saved, but the article is not added to the Article Database System. The System marks the article as “Draft”, and the use case ends. Operate With Article Database System The System verifies that the article has the necessary prerequisites to be submitted to the Article Database System. The System submits the article operation (add, modify or delete) to the Article Database System and the article is marked as “In article database system”. Article Database System Reject Article If in the “Operate With Article Database System” alternative-flow the System determines that the article operation either hasn’t got the necessary prerequisites or the Article Database System doesn’t accept the transaction, an error message is displayed saying “Article rejected”. The Writer can either modify the article or cancel the operation, at which point the use case is restarted. Article Not Found If in the “Modify Article” or “Delete Article” alternative-flow the System is unable to retrieve the article from the Article Database System, an error message is displayed saying “Article not found”. The Writer acknowledges the error message and the use case finishes. Article Database System Unavailable If the System is unavailable to establish a communication with the Article Database System after a specified lapse of time, the System will display an error message saying “Article Database System unavailable”. The Writer acknowledges the error and the use case finishes at this point. Cancel Operation At any point in the “Add Article” basic-flow or the “Modify Article” or “Delete Article” alternative-flow the Writer can cancel the operation, at which point the use case ends.
Poscondition	
Special Requirements	Information Log: All the action performed in the flow of the use case must be logged into the System, including users added, modified, and deleted, as well as any error occurred during the flow of the use case. Communication Security: The communication established with the Article Database System must be done in agreement with the security standards of this system.

Tabla 6.16: Register Article - Administración de artículos

6.3.3 **Caso de uso Nº 3**

La especificación del caso de uso que permite registrar las preferencias de los usuarios se muestran en la Tabla 6.17:

Name	Register Preference
Description	This use case allows a Reviewer to add, modify and remove his article’s preferences for the articles he/she is interest and wishes to review. The main actor starting this use case is the Reviewer.
Actor	Reviewer

Precondition	Login: Before this use case begins the Reviewer has logged into the system.
Assumption	
Trigger	The use case begins when a Reviewer selects the “Register preference” activity.
Basic Flow	Add Preference 1. The Reviewer selects the “Add preference” activity. 2. The System displays a blank form. 3. The Reviewer fills the form, specifying the following information: a. Preference name b. For each criteria added: i. Type of filter ii. Filter value/s 4. A preference name, and a list of criterias. For each criteria added she/he specify the type of filter (Writer, Title, Topic) and the corresponding filter value. 5. The Reviewer selects “Continue”. 6. The System displays a formatted view of the preference specified and asks for confirmation. 7. If the Reviewer confirms the use case continues, otherwise the use case goes to step 2. 8. The System validates that the preference specified is properly defined (for each criteria contained in the preference). 9. The System verifies that there aren’t conflicts between the new preference and the already existing ones. 10. The System saves the preference in the reviewer’s profile. 11. Steps 2-9 are repeated for each preference added to the reviewer. When the Reviewer is finished adding preferences, he/she selects the ”No more preferences to add” activity and the use case ends.
Alternative Flow	Modify Preference 1. The Reviewer selects the “Modify preference” activity. 2. The System displays a list containing the reviewer’s preferences. 3. The Reviewer selects the preference from the list that he/she wants to modify. 4. The System retrieves the preference and displays a form filled with the preference information. 5. The Reviewer modifies one or more of the preference information fields: a. Preference name b. Criteria c. Etc… 6. When the Reviewer finishes modifying that preference, he/she selects the “Save changes” activity. 7. The System updates the preference in the reviewer’s profile. 8. Steps 2-7 are repeated for each preference the Reviewer wants to modify. When edits are complete, the Reviewer selects the “No more preferences to modify” activity and the use case ends. Delete Preference 1. The Reviewer selects the “Delete preference” activity. 2. The System displays a list containing the reviewer’s preferences. 3. The Reviewer selects the preference from the list that he/she wants to delete. 4. The Reviewer selects the “Delete” activity.

	5. The System displays a delete verification dialog confirming the deletion. 6. If the Reviewer confirms the use case continues, otherwise the use case goes to step 2. 7. The System deletes the preference from the reviewer's profile. 8. Steps 2-7 are repeated for each user the Reviewer wants to delete. When the Reviewer is finished deleting preferences, he/she selects the "No more preferences to delete" activity and the use case ends. Incorrectly Defined Preference If in the "Add Preference" basic-flow or the "Modify Preference" alternative-flow the System determinates that the preference is not correctly defined, an error message will be displayed saying "Preference not valid". The Reviewer can either change the preference or cancel the operation, at which point the use case ends. Conflicts Between Preferences If in the "Add Preference" basic-flow or the "Modify Preference" alternative-flow the System finds a conflict between preferences, an error message will be displayed saying "Preferences conflict" and indicating information (if possible) about the conflict occurred. The Reviewer can either resolve the preference conflict (i.e., by modifying the preference) or cancel the operation, in which case any changes will be lost and the use case ends. Cancel Operation At any point in the "Add Preference" basic-flow or the "Modify Preference" or "Delete Preference" alternative-flow the Reviewer can cancel the operation, at which point the use case ends.
Poscondition	
Special Requirements	Information Log: All the action performed in the flow of the use case must be logged into the System, including preferences added, modified, and deleted, as well as any error occurred during the flow of the use case.

Tabla 6.17: Register Preference - Administración de artículos

6.3.4 **Caso de uso Nº 4**

La especificación del caso de uso Review Article se muestra en la Tabla 6.18:

Name	Review Article
Description	This use case allows a Reviewer to submit reviews for his preferred articles, determined according to his registered preferences. The system retrieves the articles interacting with the Article Database System and after the Reviewer reads the article, he/she writes the corresponding review. The main actor in this use case is the Reviewer. The Article Database System is an actor within the use case.
Actor	Reviewer
Precondition	Login: Before this use case begins the Reviewer has logged into the system.

Assumption	
Trigger	The use case begins when the Reviewer selects the “Review article” activity.
Basic Flow	1. The System retrieves the list of preferred articles using the preferences of the Reviewer. 2. The Reviewer selects the article to review and selects the “Review article” activity. 3. The System retrieves the article interacting with the Article Database System, and displays it in a form. 4. The System displays a blank form, beneath the article form, to write a new review. 5. The Reviewer writes the review, entering an overall recommendation for the article and grading the article regards to evaluation criteria on a scale of 1 (weakest) to 5 (strongest). The overall recommendation includes a brief impression of the article to the reviewer, and a selection of one of this items as recommendation: i. Accept the article as it is. ii. Accept subject to minor changes. iii. Resubmit with major changes. iv. Reject. The grading with regards to evaluations criteria are: Technical Content i. Clarity of motivation behind the work. ii. Clearly defined technical objectives. iii. Soundness of the proposed approach. iv. Novelty of the proposed approach. v. Technical depth of the proposed approach. vi. Clear relationship with related work. vii. Clarity and strength of the conclusion. Structure And Style i. Appropriateness of the title, abstract and keywords. ii. Style of English. iii. Clarity of presentation. iv. Structure of the discussion. v. Style and suitability of the references. vi. Coherence of argument. Evaluation Of The Proposed i. Validation of the proposed concepts through implementation, case studies, industrial applications, etc. ii. Soundness of the evaluation approach and its results. iii. Relevance to he congress. Each point in each evaluation criteria must be filled with a grade form 1 to 5. It also will be saved the date the review is made. 6. The Reviewer selects the “Submit review” activity. 7. The System saves the review.
Alternative Flow	No Articles To Review If in the “Submit Review” basic-flow, the Reviewer hasn’t have any articles to review, the System displays an error message saying “No article to review found” and the use case ends. Article Database System Unavailable If the System is unavailable to establish a communication with the

	Article Database System after a specified lapse of time, the System will display an error message saying “Article Database System unavailable”. The Reviewer acknowledges the error and the use case finishes at this point.
Poscondition	
Special Requirements	Information Log: All the action performed in the flow of the use case must be logged into the System, including reviews submitted, as well as any error occurred during the flow of the use case. Communication Security: The communication established with the Article Database System must be done in agreement with the security standards of this system. Performance: Because the Article Database System performance is rather poor, the accesses made to the Article Database System must be the minimum possible.

Tabla 6.18: Review Article - Administración de artículos

6.3.5 **Caso de uso Nº 5**

La funcionalidad correspondiente al caso de uso Maintain User Profile se describe en la Tabla 6.19:

Name	Maintain User Profile
Description	This use case allows the Administrator to maintain user’s information in the Automatic Review System. This includes adding, modifying and deleting Writers, Reviewers or Administrators from the system. The main actor of this use case is the Administrator.
Actor	Administrator
Precondition	Login: Before this use case begins the Administrator has logged into the system.
Assumption	
Trigger	The use case begins when the Administrator selects the “Maintain user profile” activity.
Basic Flow	Add User 1. The Administrator selects the “Add user” activity. 2. The System displays a blank user form. 3. The Administrator fills the blank form specifying the following information: a. Username and Password. b. First and Last name. c. Address and City. d. Province / State and Postal / Zip code. e. Country. f. Phone and Fax number. g. E-mail address. h. Type of user (Writer, Reviewer or Administrator) And according to the type of user: i. If the user to be added it’s a Writer:

	i. Username in the Article Database System. ii. Password in the Article Database System. iii. Company j. If the user to be added it's a Reviewer: i. Etc… k. If the user to be added it's a Administrator: i. Etc… 4. The Administrator selects "Continue". 5. The System displays a formatted view of the profile specified and asks for confirmation. 6. If the Administrator confirms the use case continues, otherwise the use case goes to step 2. 7. The System validates the data to insure that the profile specified complies with the proper format of the Automatic Review System, and searches for conflicts of that profile with another user profile already registered in the system. If the data is valid the system creates a new user, saving his/her profile in the system and assigning a unique system-generated id-number. 8. Steps 2-7 are repeated for each user added to the system. When the Administrator is finished adding users to the system selects the "No more users to add" activity and the use case ends.
Alternative Flow	Modify User 1. The Administrator selects the "Modify user" activity. 2. The System displays a blank user form and an input dialog for entering the user id number. 3. The Administrator types in the user id number he/she wants to modify. 4. The System retrieves the user profile and uses it to fill the user form. 5. The Administrator modifies one or more of the user information fields. 6. When the Administrator finishes modifying that user profile, selects "Save changes". 7. The System updates the user profile. 8. Steps 2-7 are repeated for each user the Administrator wants to modify. When edits are complete, the Administrator selects the "No more users to modify" activity and the use case ends. Delete User 1. The Administrator selects the "Delete user" activity. 2. The System displays a blank user form and an input dialog for entering the user id number. 3. The Administrator types in the user id number he/she wants to delete. 4. The System retrieves the user profile and uses it to fill the user form. 5. The Administrator selects "Delete". 6. The System displays a delete verification dialog confirming the deletion. 7. If the Administrator confirms the use case continues, otherwise the use case goes to step 2. 8. The System deletes the user from the system. 9. Steps 2-8 are repeated for each user the Administrator wants to delete. When the Administrator is finished deleting users to the system selects the "No more users to delete" activity and the use case ends.

	User Already Exists If in the “Add User” basic-flow the System finds an existing user with the same name an error message is displayed saying “User already exists”. The Administrator can either change the name or cancel the operation at which point the use case ends. User Not Found If in the “Modify User” or “Delete User” alternative-flow the user id number is not located, the System displays an error message saying “User not found”. The Administrator can then type in a different id number or cancel the operation at which point the use case ends. Cancel Operation At any point in the “Add User” basic-flow or the “Modify User” or “Delete User” alternative-flow the Administrator can cancel the operation, at which point the use case ends.
Poscondition	
Special Requirements	Information Log: All the action performed in the flow of the use case must be logged into the System, including users added, modified, and deleted, as well as any error occurred during the flow of the use case.

Tabla 6.19: Maintain User Profile - Administración de artículos

6.3.6 **Caso de uso Nº 6**

La funcionalidad del caso de uso View Statistical Data se presenta en la Tabla 6.20:

Name	View Statistical Data
Description	This use case allows the Administrator to obtain and view statistical data from the Automatic Review System. The Administrator may list information about a specific writer or reviewer, or also about the system itself. The main actor of this use case is the Administrator. The Article Database System, according to the sub-flow, may be an actor within the use case.
Actor	Administrator
Precondition	Login: Before this use case begins the Administrator has logged into the system.
Assumption	
Trigger	The use case begins when the Administrator selects the “View statistical data” activity.
Basic Flow	View Reviewer Statistical Data 1. The Administrator selects the “View reviewer statistical data” activity. 2. The System displays an input dialog for entering the reviewer id number. 3. The Administrator types in the reviewer id number. 4. The System retrieves the reviewer profile. 5. The System calculates the statistical data about that reviewer and displays it. That statistical data includes:

	a. Number of reviews made. b. Number of preferences. c. Percentage of reviews with an average grade greater or equal than 7. d. Percentage of reviews with an average grade greater or equal than 4 and less than 7. e. Percentage of reviews with an average grade less than 4. 6. When the Administrator has finished viewing the reviewer statistical data the use case ends.
Alternative Flow	1. View Writer Statistical Data 2. The Administrator selects the "View writer statistical data" activity. 3. The System displays an input dialog for entering the writer id number. 4. The Administrator types in the writer id number. 5. The System retrieves the writer profile. 6. The System calculates the statistical data about that writer and displays it. That statistical data includes: a. Number of articles published (retrieved form the Article Database System). b. Number of reviews made to his articles. c. Percentage of reviews per article. d. Percentage of reviews with an average grade greater or equal than 7. e. Percentage of reviews with an average grade greater or equal than 4 and less than 7. f. Percentage of reviews with an average grade less than 4. 7. When the Administrator has finished viewing the writer statistical data the use case ends. View System Statistical Data 1. The Administrator selects the "View system statistical data" activity. 2. The System calculates the statistical data about the system and displays it. That statistical data includes: a. Number of articles published (retrieved form the Article Database System). b. Number of reviews made. c. Percentage of reviews per article. d. Percentage of reviews with an average grade greater or equal than 7. e. Percentage of reviews with an average grade greater or equal than 4 and less than 7. f. Percentage of reviews with an average grade less than 4. 3. When the Administrator has finished viewing the writer statistical data the use case ends. User Not Found If in the "View Reviewer Statistical Data" basic-flow or the "View Reviewer Statistical Data" alternative-flow the user id number is not located, the System displays an error message saying "User not found". The Administrator can then type in a different id number or cancel the operation at which point the use case ends. Article Database System Unavailable If the System is unavailable to establish a communication with the Article Database System after a specified lapse of time, the System will display an error message. The Writer acknowledges the error and the use case finishes at this point.

Poscondition	
Special Requirements	Communication Security: The communication established with the Article Database System must be done in agreement with the security standards of this system.

Tabla 6.20: View Statistical Data - Administración de artículos

6.3.7 **Caso de uso Nº 7**

La funcionalidad del caso de Nº 7, View Log, se presenta en la Tabla 6.21:

Name	View Log
Description	This use case allows the Administrator to obtain and view the Automatic Review System log. The Administrator may list all the log entries sorted by date, time, user id number, type, or any combination of those. Also the Administrator can specify filters, in order to remove entries that aren't of interest. The main actor of this use case is the Administrator.
Actor	Administrator
Precondition	Login: Before this use case begins the Administrator has logged into the system.
Assumption	
Trigger	The use case begins when the Administrator selects the "View log" activity.
Basic Flow	View System Log 1. The Administrator selects the sort method and filters required for the presentation of the log entries. 2. The System retrieves the Automatic Review System log entries. 3. The System applies the filters and format the log entries according to the sort method specified. 4. The System displays the resulting log entries. 5. When the Administrator has finished viewing the log entries the use case ends.
Alternative Flow	No Entries Found If in the "View System Log" basic-flow the System didn't found any entries (either because there aren't any, or because all of the entries where filtered), an error message is displayed saying "No log entries to display". The Administrator acknowledges the error and the use case finishes at this point.
Poscondition	
Special Requirements	

Tabla 6.21: View Log - Administración de artículos

6.3.8 **Caso de uso Nº 8**

La funcionalidad para realizar el login de un usuario se presenta en la Tabla 6.22:

Name	Login
Description	This use case describes how a User logs into the Automatic Review System. The main actors starting this use case are Writer, Reviewer and Administrator.
Actor	User
Precondition	Login: Before this use case begins the Administrator has logged into the system.
Assumption	
Trigger	The use cases begins when a User selects the “Login” activity when there isn’t any user already logged, or when a already logged user selects the “Login as another user” activity.
Basic Flow	1. The System displays the login form. 2. The User inserts his username and password, and selects “Ok”. 3. The System validates the user’s password and logs him/her into the Automatic Review System. 4. The System displays the “Main” form according to the type of user (Writer, Reviewer or Administrator), and the use case ends.
Alternative Flow	Invalid Name / Password If in the basic-flow the System cannot find the name or the password is invalid, an error message is displayed saying “Invalid username or password”. The User can type in another username and password or choose to cancel the operation, at which point the use case ends.
Poscondition	
Special Requirements	Information Log: All the action performed in the flow of the use case must be logged into the System, including log-ins, log-outs, as well as any error occurred during the flow of the use case.

Tabla 6.22: Login - Administración de artículos

Como los resultados obtenidos para el ejemplo presentado con el primer algoritmo son muy extensos, no se logran apreciar completos desde la captura de una pantalla, los resultados obtenidos por ambos algoritmos se van a mostrar en una tabla (Tabla 6.23). En la tabla se presentan los resultados que se encontraron con el primer algoritmo enfrentados con el resultado similar que se obtuvo con al algoritmo que usa la ontología.

Algoritmo con ontología	**Primer Algoritmo**
Security	Communication Security: The communication established with the Article Database System must be done in agreement with the security standards of this system. - crosscutting
Performance	Performance: Because the Article Database System performance is rather poor, the accesses made to the Article

	Database System must be the minimum possible. - crosscutting
Logging (Monitoring)	Information Log: All the action performed in the flow of the use case must be logged into the System, including transactions made with the Article Database System, errors occurred, consults made, etc. - crosscutting
	Information Log: All the action performed in the flow of the use case must be logged into the System, including users added, modified, and deleted, as well as any error occurred during the flow of the use case. - crosscutting
	Information Log: All the action performed in the flow of the use case must be logged into the System, including preferences added, modified, and deleted, as well as any error occurred during the flow of the use case. - crosscutting
	Information Log: All the action performed in the flow of the use case must be logged into the System, including log-ins, log-outs, as well as any error occurred during the flow of the use case. - crosscutting
	Information Log: All the action performed in the flow of the use case must be logged into the System, including reviews submitted, as well as any error occurred during the flow of the use case. - crosscutting
Screen Update	3. Visualization: The user interface shall be designed according the type of user that is using it, and it must be easy to change it. - crosscutting
	display - crosscutting
Autentication (Security)	verifies - crosscutting
Access Control (Security)	Secure Login: All the users must login before using the System, and this login must be secure. Additionally it must be allowed to use more than one type of secure login for the verification. - crosscutting
Access to Memory	Data Persistency: All the data generated and used by the Automatic Review System shall be saved using a persistence System. This probably will be a database. – crosscutting
	register - crosscutting
Integrability	7. Article Database System: The system shall integrate with the existing Article Database System, and it must comply it's communication standards, security policies, etc. - crosscutting
Restrictions of Real Time	5. Internal Database Access Response Time: The system must provide access to the internal database with no more than 2 second latency. - crosscutting
	6. External Database Access Response Time: The system must provide access to the Article Database System with no more than 10 second latency. - crosscutting
Availability	Availability: The Automatic Review System shall be available 24 hours a day, 7 days a week. There shall be no more than 10% down time. - crosscutting
Profiling (Monitoring)	
Error Handling	

Tabla 6.23: Aspectos candidatos para Administración de artículos

En este caso de estudio se puede notar que los resultados obtenidos con el algoritmo que usa la ontología son más acertados. Todos los resultados que se obtuvieron con este algoritmo son aspectos candidatos válidos para el ejemplo. Los nombres de los aspectos candidatos presentados son estandarizados dando al usuario una idea clara de la funcionalidad que el aspecto debe contener.

Existe un aspecto candidato en el ejemplo, el aspecto candidato comunicación entre procesos, que no es detectado por ninguno de los dos algoritmos. Es una omisión para ambos algoritmos. El primer algoritmo omitió este aspecto por la palabra Communications es identificada como un sustantivo, y el algoritmo busca la repetición de palabras verbos en dos o más casos de uso. El algoritmo que usa la ontología omite el aspecto candidato porque no encuentra la cantidad de coincidencias buscadas, en el aspecto Communication between Processes son necesarias dos coincidencias. Encuentra una coincidencia con la palabra Communications en varios casos de uso y en los requerimientos especiales del proyecto, pero no encuentra una coincidencia con la palabra Processes, ya que en el proyecto se indica el nombre de la aplicación con la que se debe realizar la comunicación.

El primer algoritmo omite los aspectos candidatos administración de errores y administración de perfiles de usuarios, estos aspectos son potenciales aspectos candidatos en el proyecto.

Al observar los resultados que presenta el primer algoritmo se puede ver que en algunos casos se han identificado varios aspectos candidatos que en realidad representan el mismo crosscutting concern. Tal es el caso de:

- Information Log: All the action performed in the flow of the use case must be logged into the System, including transactions made with the Article Database System, errors occurred, consults made, etc. – crosscutting
- Information Log: All the action performed in the flow of the use case must be logged into the System, including users added, modified, and deleted, as well as any error occurred during the flow of the use case. – crosscutting
- Information Log: All the action performed in the flow of the use case must be logged into the System, including preferences added, modified, and deleted, as well as any error occurred during the flow of the use case. – crosscutting
- Information Log: All the action performed in the flow of the use case must be logged into the System, including log-ins, log-outs, as well as any error occurred during the flow of the use case. – crosscutting
- Information Log: All the action performed in the flow of the use case must be logged into the System, including reviews submitted, as well as any error occurred during the flow of the use case. - crosscutting

El primer algoritmo propone todos los puntos anteriores como aspectos candidatos. El algoritmo que usa la ontología propone uno solo: Logging (Monitoring). Al proponer todos los puntos como aspectos candidatos distintos se obtiene una idea clara de la funcionalidad que van a contener, pero también puede llevar a una confusión. El aspecto candidato que abarca la funcionalidad es: Logging, si se desea ver con mayor detalle cuál es la responsabilidad que debe tener el aspecto candidato se puede leer la especificación del proyecto.

6.4 Conclusiones

En el capítulo presentado se muestra la forma de trabajo del algoritmo que usa la ontología. Se puede apreciar que los resultados obtenidos con este algoritmo se presentan en una lista corta y precisa. Cada uno de los aspectos que la herramienta presenta son crosscutting válidos para el proyecto que se esta analizando. Los nombres de los aspectos candidatos

presentados son conceptos estandarizados en la ontología, y dan al analista una idea clara de la responsabilidad que cada uno representa.

Al obtener resultados con estas características se puede decir que se logra satisfacer los problemas presentados en DSOA en el capítulo II. Con los resultados que se obtienen al usar Aspect Extractor Tool con el algoritmo que usa la ontología se consigue un enfoque para la identificación de los aspectos candidatos que trabaje con términos unificados en el dominio DSOA, ofreciendo resultados adecuados y acertados.

Capítulo 7: Validación de la propuesta

Para validar la propuesta presentada se va a realizar una comparación entre los dos algoritmos con los que Aspect Extractor Tool trabaja. Se identifican aspectos candidatos con ambos algoritmos para el mismo ejemplo de forma de comparar los resultados y obtener conclusiones de ellos.

7.1 Caso de comparación N° 1

Para realizar una comparación entre los dos algoritmos con los que Aspect Extractor Tool trabaja se van a usar los ejemplos que fueron utilizados en el capítulo IV para describir la funcionalidad de Aspect Extractor Tool con la heurística que busca palabras que se repiten en más de un caso de uso.

Ejemplo 1: Presenta un proyecto de gestión de alquiler de autos. Las especificaciones de estos casos de uso pueden verse en las tablas Tabla 4.1 y Tabla 4.2 del Capitulo IV.

Los resultados arrojados por la herramienta al utilizar al algoritmo que usa la ontología se presentan en la Figura 7.1.

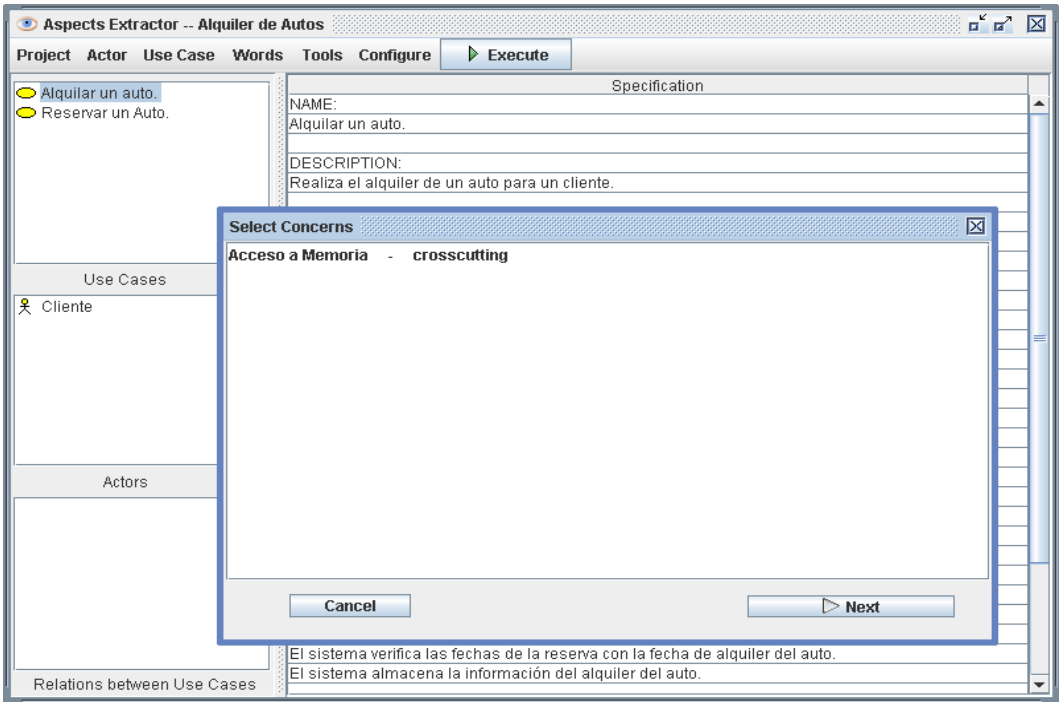

Figura 7.1: Aspectos candidatos para Gestión de alquiler de autos (algoritmo con ontología)

Los resultados obtenidos por Aspect Extractor Tool al utilizar el algoritmo previo pueden verse en la Figura 7.2. Los aspectos candidatos propuestos por la herramienta usando el algoritmo previo: *"alquilar", "realizada", "solicita", "verifica", "muestra", "fechas" y "termina"* no son aspectos candidatos válidos en el ejemplo. El aspecto candidato válido que presenta es *"persistencia"*. Aspect Extractor Tool usando ontología presenta sólo un aspecto candidato candidato: *"Acceso a Memoria"*. Persistencia y Acceso a Memoria están indicando el mismo comportamiento. En este caso se puede ver claramente que los resultados retornados por el

algoritmo que usa la ontología presenta un listado más corto a analizar y con resultados más precisos.

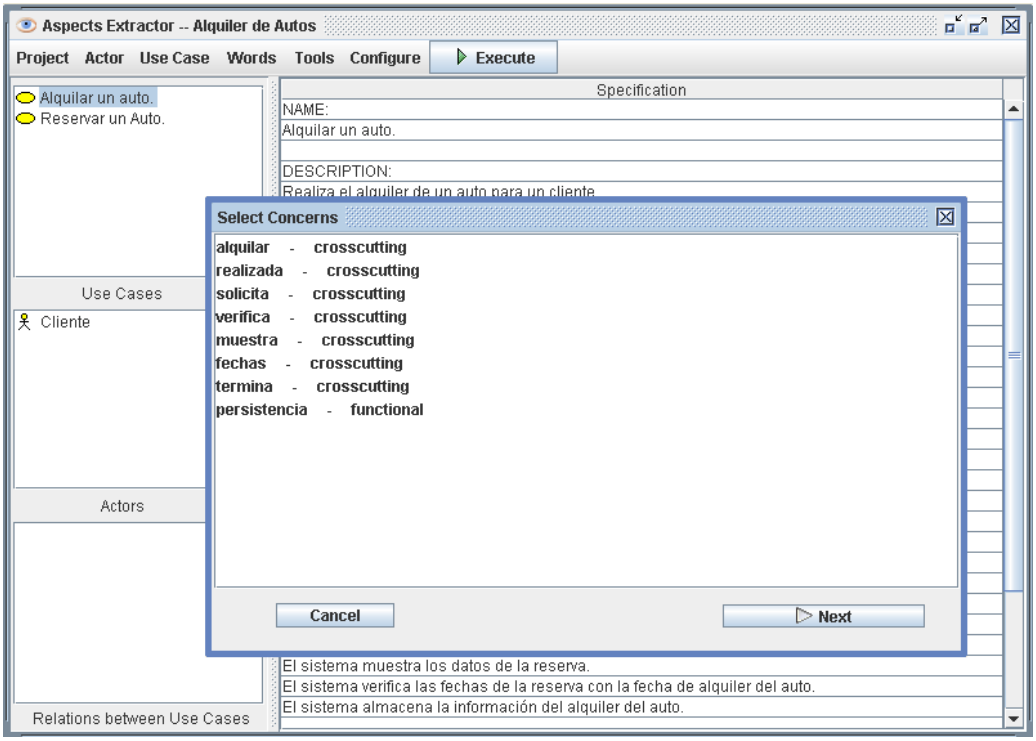

Figura 7.2: Aspectos candidatos para Gestión de alquiler de autos (primer algoritmo)

Ejemplo 2: Es un sistema de gestión de cuentas de profesores para un colegio. Las especificaciones de los casos de uso utilizadas pueden verse en las tablas Tabla 4.4 y Tabla 4.5 del Capitulo IV.

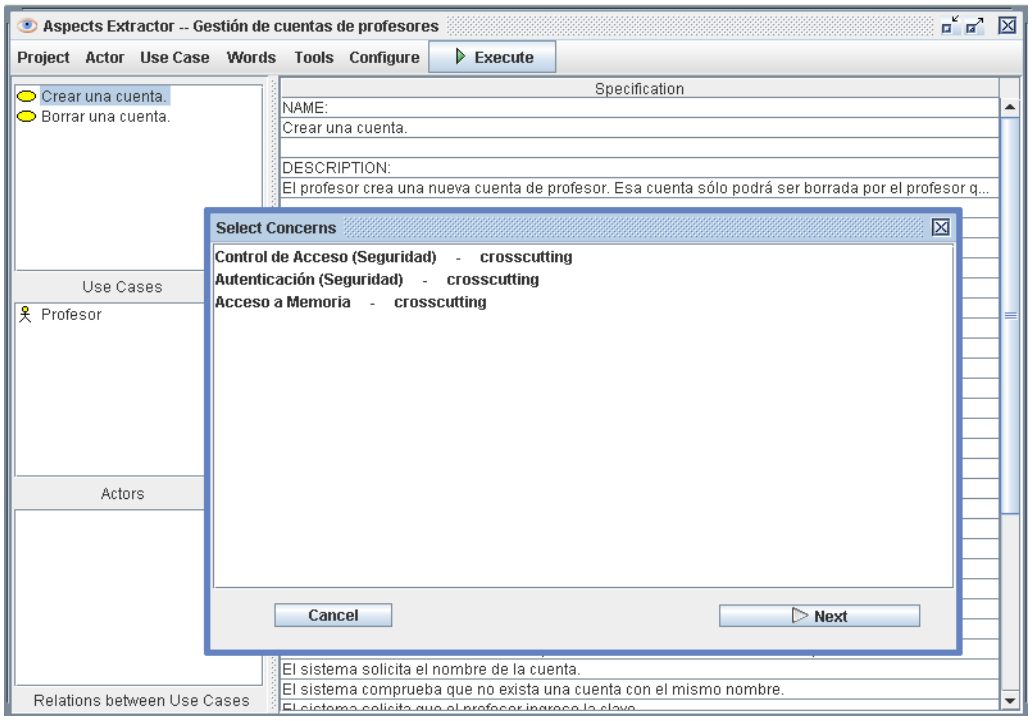

Figura 7.3: Aspectos candidatos para Gestión de cuentas de profesores (algoritmo con ontología)

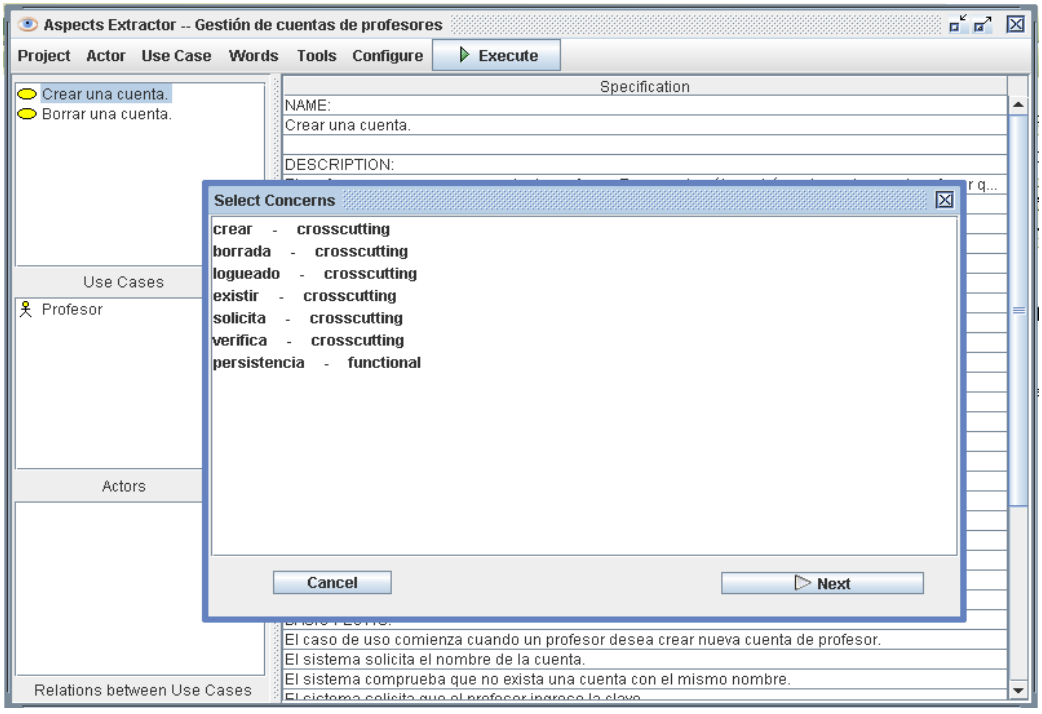

Figura 7.4: Aspectos candidatos para Gestión de cuentas de profesores (primer algoritmo)

Los resultados obtenidos por la herramienta al utilizar al algoritmo que usa la ontología se presentan en la Figura 7.3. En la Figura 7.4 se presentan los resultados obtenidos con el primer algoritmo.

Los resultados retornados por la herramienta al usar el algoritmo actual, buscando las palabras que se repiten en más de un caso de uso es el siguiente: *"crear", "borrada", "existir", "solicita" y "acceso".* Estos aspectos candidatos propuestos no son palabras que en el ejemplo tengan un significado que representen a posibles aspectos. Los aspectos candidatos propuesto *"persistencia", "logeado"* y *"verifica"* son considerados resultados correctos. El algoritmo que usa la ontología identifica los aspectos candidatos: *"Control de Acceso (Seguridad)", "Autenticación (Seguridad)" y "Acceso a Memoria",* éstos son resultados correctos para el ejemplo.

Ejemplo 3: Este ejemplo presenta a un sistema de administración de habitaciones de un hotel. Las especificaciones utilizadas pueden observarse en las tablas Tabla 4.7 y Tabla 4.8 del Capitulo IV.

Los resultados obtenidos por la herramienta Aspect Extractor Tool con ontología se presentan en la Figura 7.5. Los aspectos candidatos detectados por la herramienta al utilizar el primer algoritmo son mostrados en la Figura 7.6.

Los crosscutting concerns candidatos propuestos por la herramienta utilizando el algoritmo actual son: *"solicita", "verifica" y "realizado".* Éstos no son potenciales crosscutting concerns, porque no tienen un significado semántico asociado para el ejemplo. El total de los aspectos candidatos presentados por la herramienta para el ejemplo es incorrecto. *"Autenticación (Seguridad)"* y *"Acceso a Memoria"* son los resultados que presenta Aspect Extractor Tool utilizando la ontología. Acceso a Memoria es un aspecto candidato válido. Autenticación (Seguridad) no es un resultado correcto para este caso, porque el aspecto candidato es influenciado por la palabra verifica, que un sinónimo de la palabra autenticación. En el ejemplo analizado la palabra verifica se refiere a verificar disponibilidad de fechas o si una tarjeta de crédito es válida para su uso, y el aspecto se refiere a validación de la identidad de un entidad en el sistema.

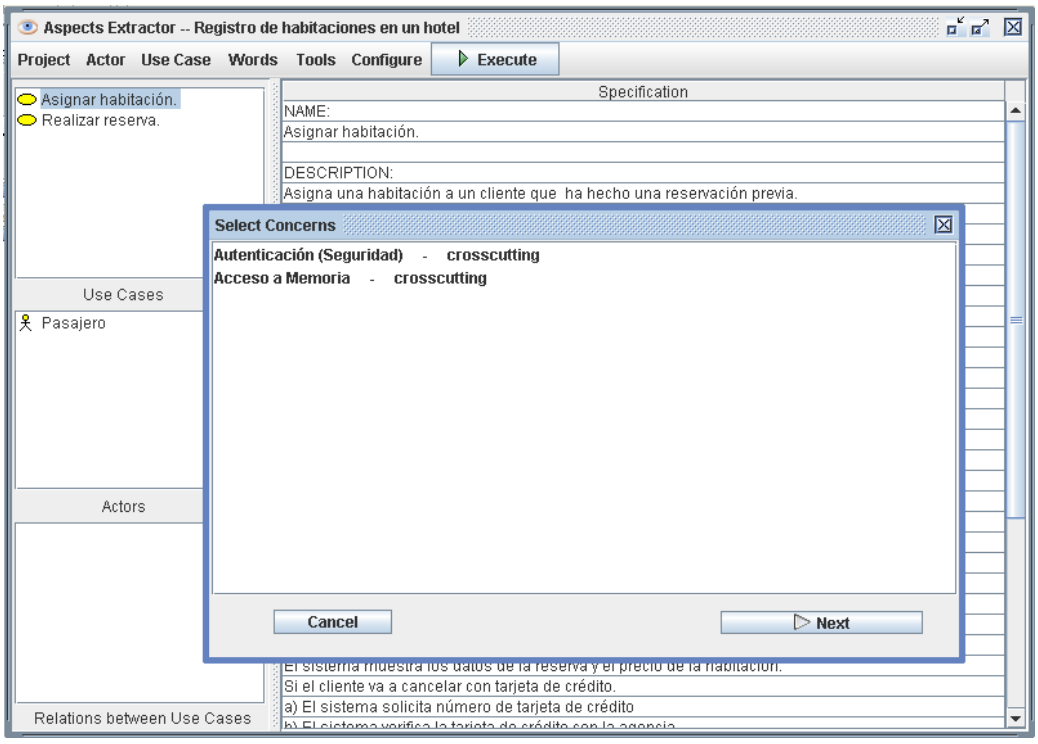

Figura 7.5: Aspectos candidatos para Administración de habitaciones de un hotel (algoritmo con ontología)

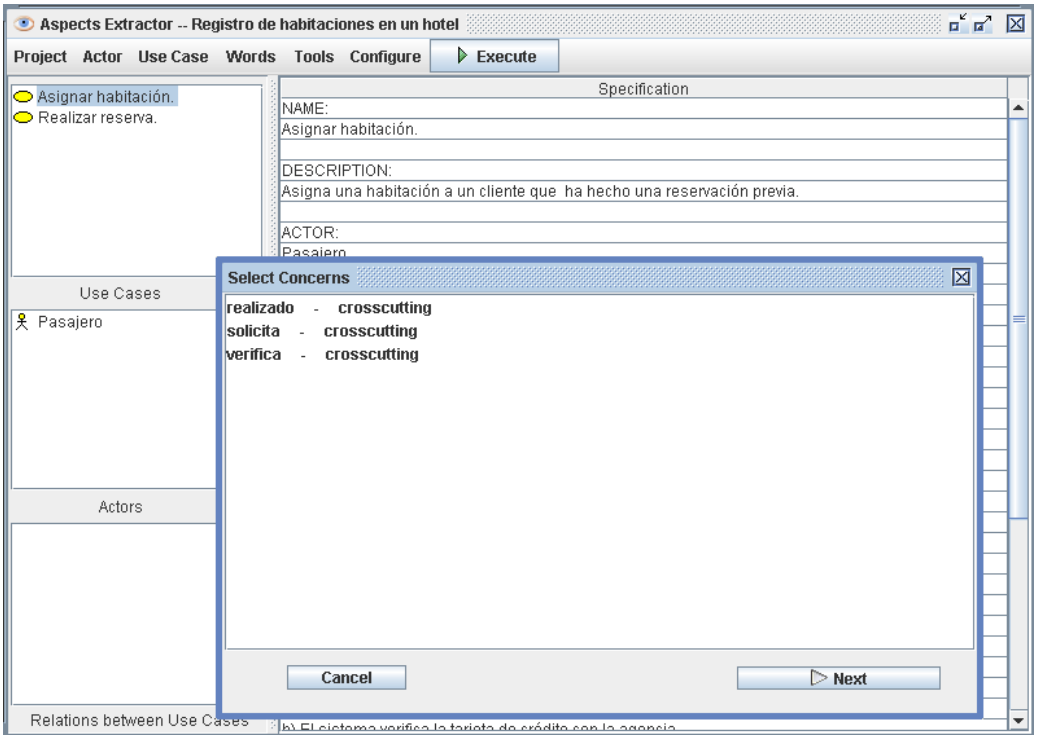

Figura 7.6: Aspectos candidatos para Administración de habitaciones de un hotel (primer algoritmo)

7.2 Caso de comparación N° 2

El primer ejemplo que se va analizar es el caso de estudio presentado anteriormente, el Sistema de Administración de Historias Clínicas en el idioma español. Los casos de uso que fueron especificados para este proyecto son los casos de uso Dar de alta, Modificar una historia clínica e Imprimir un informe que se presentan en las tablas Tabla 6.1, Tabla 6.2 y Tabla 6.3 respectivamente. Los resultados que la herramienta presenta usando el algoritmo que usa la ontología se muestran en la Figura 7.7, y los resultados presentados al utilizar el primer algoritmo se muestran en la Figura 7.8.

A continuación se presentan algunos puntos de comparación entre los dos resultados:

- Los aspectos candidatos *"permite", "realizar", "solicita", "registrada"* y *"crear"* son palabras que se repiten en más de un caso de uso, no dan idea de lo que representan, y no se consideran resultados válidos retornados por la herramienta.
- Los aspectos candidatos propuestos *"autentificado", "verifica*" y *"acceso"* pueden ser considerados resultados válidos.
- El aspecto candidato *"autentificado"* da la idea de que sólo pueden acceder al sistema o a cierta funcionalidad de ésta los usuarios que tienen permiso. Es un aspecto candidato válido, pero, comparado con el nombre que arrojó la herramienta con el algoritmo usando la ontología para esta misma responsabilidad: *"Autenticación (Seguridad)"*, se puede decir que ambos resultados son correctos, pero el nombre Autenticación (Seguridad) da una idea más representativa de la responsabilidad que el aspecto candidato deberá tener.
- El aspecto candidato propuesto "*acceso"* da la idea de un acceso controlado a algún tipo de información. Esta responsabilidad es la misma que se intenta representar con del aspecto candidato *"Control de acceso (Seguridad)"*.

Los resultados obtenidos con los dos algoritmos son resultados diferentes. Como se mencionó anteriormente, el listado de aspectos candidatos presentado al usar la ontología es

más preciso y cada uno de los aspectos candidatos propuestos dan al analista la idea de lo que representan, o de la funcionalidad que van a tener. No se puede decir lo mismo del listado presentado por el primer algoritmo.

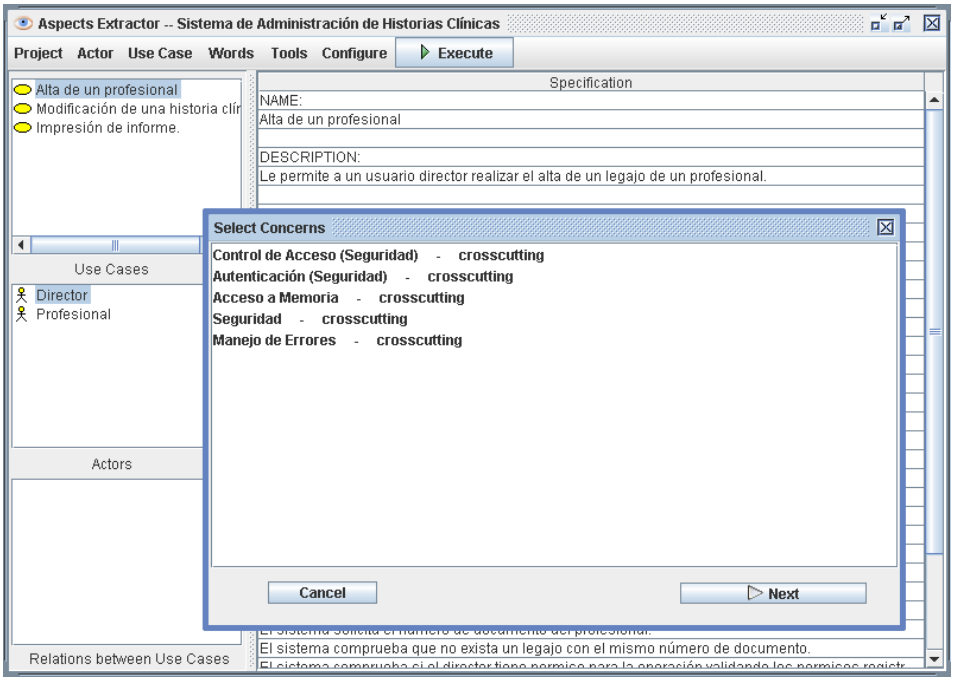

Figura 7.7: Aspectos candidatos para Administración de historias clínicas (algoritmo con ontología)

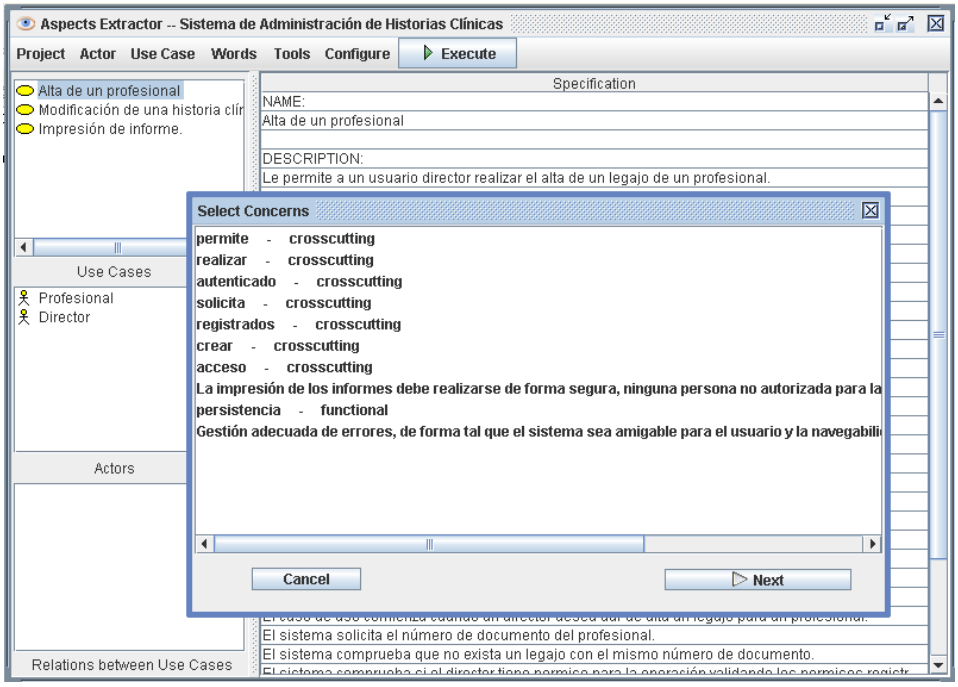

Figura 7.8: Aspectos candidatos para Administración de historias clínicas (primer algoritmo)

7.3 Caso de comparación N° 3

En la presentación de Aspect Extractor Tool [HD05] se utiliza un caso de estudio con el que se describe el funcionamiento de la heurística utilizada. El caso de estudio es un sistema de gestión de alumnos, se presenta a continuación [HD05]:

Sistema de Gestión de alumnos

El ejemplo se trata de un sistema de gestión para alumnos de una facultad, donde existen tres tipos de usuarios: alumnos, profesores y administradores (en este caso secretarias/os). Los alumnos podrán darse de alta, inscribirse para cursar una materia, rendir un final, observar sus notas, etc. Los profesores podrán ingresar nuevas notas a un determinado alumno. Los administradores podrán visualizar estadísticas respecto de los alumnos. Para realizar cada tipo de operación, los usuarios deberán contar con los permisos correspondientes.

Listado de casos de uso

1. Login.
2. Inscribir en una materia.
3. Cancelar inscripción a materia.
4. Inscribir para rendir final.
5. Cancelar inscripción para rendir final.
6. Observar notas.
7. Alta alumno.
8. Baja alumno.
9. Cargar nota de una materia para un alumno.
10. Visualizar estadística respecto de los alumnos.

Caso de Uso N° 2: En la siguiente tabla se presenta la especificación del caso de uso (Tabla 7.1).

Nombre	Inscribir en una materia.
Descripción	Le permite a un alumno inscribirse para cursar una materia, si es que cuenta con los permisos necesarios. Esta actividad debe ser realizada en un tiempo razonable.
Actor	Alumno
Suposición	El alumno debe estar logeado.
Trigger	El alumno selecciona la actividad de inscripción.
Flujo Básico	1. El sistema verifica que el alumno este autorizado para realizar tal operación. 2. El sistema analiza las materias cursadas por el alumno e identifica aquellas que puede cursar dependiendo de las correlativas. 3. El sistema muestra al alumno las materias en las cuales puede anotarse. 4. El alumno selecciona la materia. 5. El sistema almacena la información de la inscripción.
Flujo Alternativo	1. Si la cuenta está inactiva, o el alumno no posee los permisos necesarios, el sistema muestra un mensaje de error. 2. El alumno recibe el error. 3. El sistema reinicia el caso de uso.
Requerimiento	Conexión segura

Especial	

Tabla 7.1: Inscribir en una materia - Sistema de gestión de alumnos

Caso de Uso N° 4: La especificación del caso de uso es mostrada en la Tabla 7.2.

Nombre	Inscribir para rendir un final.
Descripción	Le permite a un alumno inscribirse para rendir un final, si es que cuenta con los permisos necesarios. Esta actividad debe ser realizada en un tiempo razonable.
Actor	Alumno
Suposición	El alumno debe estar logeado.
Trigger	El alumno selecciona la actividad de inscripción para rendir un final.
Flujo Básico	1. El sistema verifica que el alumno este autorizado para realizar tal operación. 2. El sistema muestra un menú con las posibles materias a rendir por parte del alumno. 3. El alumno selecciona la materia a rendir. 4. El sistema almacena la información de la inscripción al final.
Flujo Alternativo	1. Si el alumno no posee las materias correlativas aprobadas, el sistema muestra el error. 2. El alumno recibe el error. 3. El sistema reinicia el caso de uso.
Requerimiento Especial	Conexión segura.

Tabla 7.2: Inscribir para rendir un final Sistema de gestión de alumnos

Caso de Uso N° 10: La especificación del caso de uso N° 10 es mostrada en la Tabla 7.3.

Nombre	Visualizar una estadística respecto de los alumnos.
Descripción	Le permite al administrador visualizar estadísticas, de acuerdo a datos extraídos de los alumnos.
Actor	Administrador
Suposición	El administrador debe estar logeado.
Trigger	El administrador selecciona la actividad de “estadísticas”.
Flujo Básico	1. El sistema verifica que el administrador esté autorizado para realizar tal operación. 2. El sistema muestra un menú para que el administrador séte los filtros deseados.

	3. El sistema visualiza el diagrama con las estadísticas.
Flujo Alternativo	1. Si la cuenta está inactiva, o el administrador no posee los permisos necesarios, el sistema muestra un mensaje de error. 2. El administrador recibe el error. 3. El sistema reinicia el caso de uso.
Punto de Extensión	El sistema visualiza los resultados obtenidos llamando al caso de uso "Un administrador observa una estadística".

Tabla 7.3: Visualizar una estadística respecto de los alumnos - Sistema de gestión de alumnos

En la Figura 7.9 se muestran los resultados obtenidos con Aspect Extractor Tool usando el algoritmo que usa la ontología, en la Figura 7.10 se presentan los resultados obtenidos usando la primer heurística.

Los resultados obtenidos por la herramienta usando la ontología se identificaron por diferentes palabras del proyecto que hacen matching con palabras de la ontología. El aspecto candidato *"Manejo de Errores"* fue inferido por las palabras: administrador, error y control. *"Autenticación (Seguridad)"* fue inferido por el conjunto de palabras: segura, autorizado y aprobadas. Al aspecto candidato *"Acceso a Memoria"* lo infieren las palabras: acceso, información, permisos y datos. *"Control de Acceso"* es inferido por: control, segura, acceso, administrador, permiso y almacena. El aspecto candidato *"Performance"* es inferido por la palabra performance.

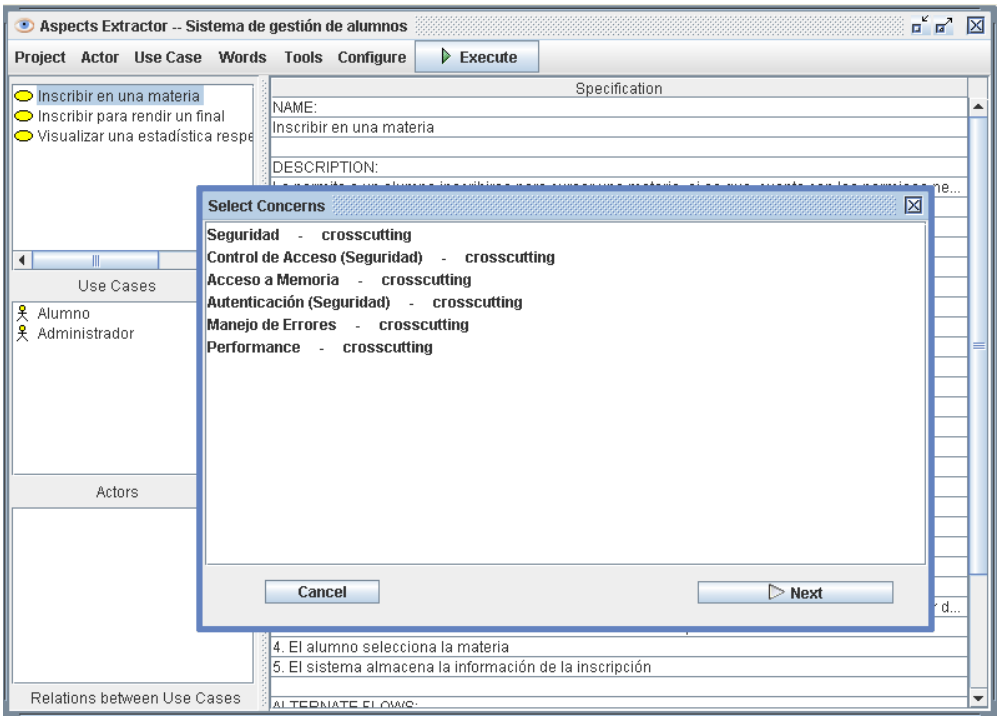

Figura 7.9: Aspectos candidatos para Gestión de alumnos (algoritmo con ontología)

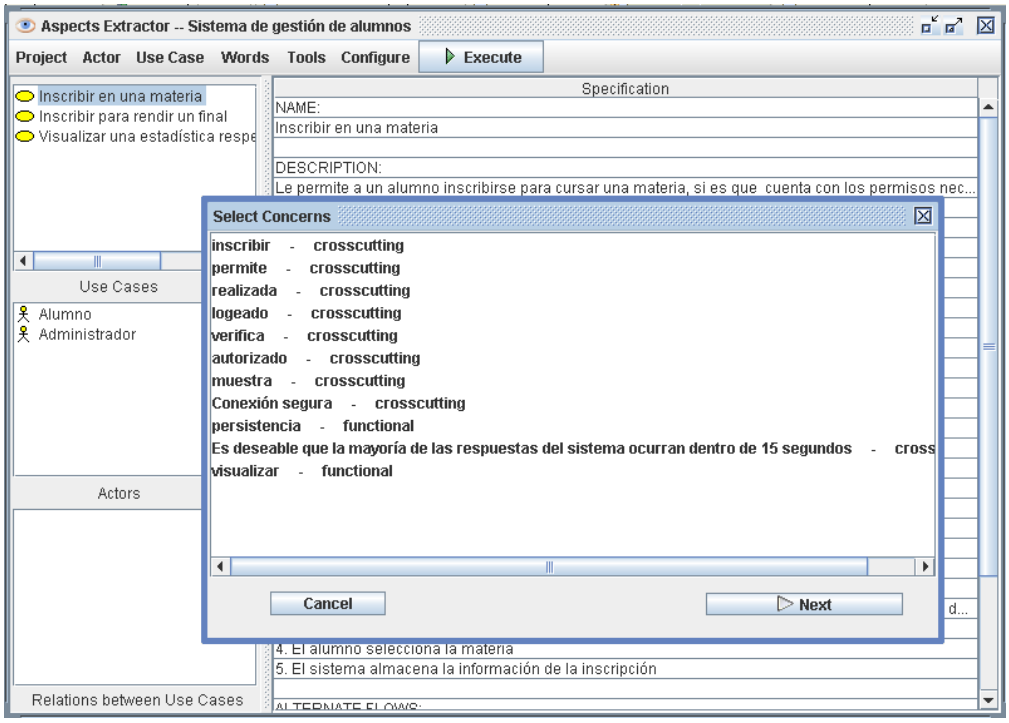

Figura 7.10: Aspectos candidatos para Gestión de alumnos (primer algoritmo)

Los concerns mostrados al ejecutar la herramienta con el primer algoritmo fueron identificados por diferentes causas. Por ejemplo, los verbos "*inscribir*", "*permite*", "*realizada*", "*logeado*", "*verifica*", "*autorizado*" y "*muestra*" se detectaron porque aparecen en más de un caso de uso. El verbo "*visualizar*" fue identificado porque hace matching con la palabra clave funcional *visualización* que se encuentra en la base de datos. El requerimiento no funcional del sistema tiene una correspondencia con todos los casos de uso ingresados dado que en su descripción aparece la frase "*Esta actividad debe ser realizada en un tiempo razonable*". Además, los casos de uso "*Inscribir en una materia* e *Inscribir para rendir una final*" tienen ambos un requerimiento especial, "*Conexión segura*". Por lo tanto, dicho requerimiento especial es un concern candidato. Por último, la presencia del concern "*Persistencia*" se debe a que tal palabra es sinónima de "*almacenar*", de acuerdo al diccionario de sinónimos. Observar que la raíz de *almacenar*, *almacen,* se encentra en los flujos básicos de los casos de uso "*Inscribir en una materia* e *Inscribir para rendir una final* en la frase: *El sistema almacena la información de la inscripción*". Por lo tanto, se reemplaza dicha palabra por su sinónimo, *persistencia*, unificando el vocabulario. Esta palabra es propuesta porque hace matching con la palabra clave no funcional *persistencia* [HD05]. La elección de aspectos candidatos realizada finalmente se debe al conocimiento sobre el dominio y la intuición del analista. Los aspectos candidatos seleccionados son: "*logeado*", "*autorizado*", "*Conexión segura*", "*persistencia*", "*Es deseable que la mayoría de las respuestas del sistema ocurran dentro de 15 segundos*" *y* "*visualizar*" [HD05].

Comparando los aspectos candidatos seleccionados en ambos casos, se puede ver que los resultados son parecidos. Los aspectos candidatos *"logueado", "autorizado", "conexión segura", "persistencia"* y *"es deseable que la mayoría de las respuestas del sistema ocurran dentro de 15 segundos"* pueden ser comparados con: *"Autenticación (Seguridad)", "Control de Acceso (Seguridad)", "Seguridad", "Acceso a Memoria"* y *"Performance"* respectivamente. Todos ellos pueden ser considerados como aspectos candidatos válidos para el proyecto. Al usar el algoritmo sin la ontología también se propone el aspecto funcional visualizar, y según [HD05] se considera como un aspecto candidato válido, pero este aspecto candidato no es detectado con el algoritmo usando la ontología. Esto no se considera un error al ejecutar la herramienta con el algoritmo que usa la ontología, porque en la ontología se modelan sólo los aspectos candidatos

no funcionales. Por otro lado, al usar la herramienta con el algoritmo que usa la ontología se encuentra el aspecto *"Manejo de Errores"*, se considera un aspecto candidato válido en el proyecto, pero al usar la herramienta con el primer algoritmo no lo detecta. En este caso es un error que la herramienta al ejecutar el primer algoritmo no lo detecte, porque en los flujos alternativos de los tres casos de uso especificados (Tabla 7.1, Tabla 7.2 y Tabla 7.3) aparecen frases que hacen que sea correcto que el aspecto candidato propuesto *"Manejo de Errores"* es válido en el proyecto.

Es este ejemplo se puede ver claramente que al usar la herramienta para un mismo proyecto con ambos algoritmos con los que cuenta, se obtienen resultados más precisos, una lista de resultado más corta y nombres de aspectos candidatos propuestos más apropiados con el algoritmo que usa la ontología.

7.4 Comparación de algoritmos de identificación a través de métricas

Para completar la comparación entre los dos algoritmos que Aspect Extractor Tool tiene incorporados para realizar la identificación de los aspectos candidatos se tomaron métricas en los ejemplos estudiados. Las métricas que se tuvieron en cuenta se describen a continuación:

Aspectos positivos: Son aquellos aspectos identificados por el algoritmo que son considerados resultados correctos.

Aspectos negativos: Son los aspectos identificados por el algoritmo que son considerados resultados incorrectos.

Aspectos identificados: Cantidad total de aspectos identificados por el algoritmo (suma de aspectos positivos y aspectos negativos).

Aspectos en el sistema: Cantidad deseable de aspectos que se deberían identificar en el sistema que se esta analizando.

Efectividad: La cantidad de aspectos positivos dividido el total de aspectos en el sistema.

Ruido: La cantidad de aspectos negativos sobre el total de los aspectos identificados.

Tiempo: Tiempo para ejecutar el algoritmo en el sistema. Se mide en milisegundos.

A continuación se presentan las medidas que se tomaron para cada proyecto:

Sistema de Administración de Historias Clínicas (HC)

- Algoritmo con ontología
  - Aspectos positivos: 5
  - Aspectos negativos: 0
  - Aspectos identificados: 5
  - Aspectos en el sistema: 5
  - Efectividad: 0/5 = 1
  - Ruido: 0/5 = 0
  - Tiempo: 12468 ms.
- Primer Algoritmo
  - Aspectos positivos: 5
  - Aspectos negativos: 5
  - Aspectos identificados: 10
  - Aspectos en el sistema: 5
  - Efectividad: 5/5 = 1
  - Ruido: 5/10 = 0.5
  - Tiempo: 469 ms.

Sistema de Gestión de alumnos (GA)

- Algoritmo con ontología
  - Aspectos positivos: 6
  - Aspectos negativos: 0
  - Aspectos identificados: 6
  - Aspectos en el sistema: 6
  - Efectividad: 6/6 =1
  - Ruido: 0/6 = 0
  - Tiempo: 7516 ms.
- Primer Algoritmo
  - Aspectos positivos: 6
  - Aspectos negativos: 5
  - Aspectos identificados: 11
  - Aspectos en el sistema: 6
  - Efectividad: 6/6 = 1
  - Ruido: 5/11 = 0.45
  - Tiempo: 407 ms.

Sistema de alquiler de autos (AA)

- Algoritmo con ontología
  - Aspectos positivos: 1
  - Aspectos negativos: 0
  - Aspectos identificados: 1
  - Aspectos en el sistema: 1
  - Efectividad: 1/1 =1
  - Ruido: 0/1 = 0
  - Tiempo: 4994 ms.
- Primer Algoritmo
  - Aspectos positivos: 1
  - Aspectos negativos: 7
  - Aspectos identificados: 8
  - Aspectos en el sistema: 1
  - Efectividad: 1/1 = 1
  - Ruido: 7/8 = 0.875
  - Tiempo: 328 ms.

Sistema de gestión de cuentas de profesores (CP)

- Algoritmo con ontología
  - Aspectos positivos: 3
  - Aspectos negativos: 0
  - Aspectos identificados: 3
  - Aspectos en el sistema: 3
  - Efectividad: 3/3 =1
  - Ruido: 0/3 = 0
  - Tiempo: 4407 ms.
- Primer Algoritmo
  - Aspectos positivos: 3
  - Aspectos negativos: 4
  - Aspectos identificados: 7
  - Aspectos en el sistema: 3
  - Efectividad: 3/3 = 1

- Ruido: 4/7 = 0.57
- Tiempo: 390 ms.

Sistema de administración de habitaciones de un hotel (HH)

- Algoritmo con ontología
  - Aspectos positivos: 1
  - Aspectos negativos: 1
  - Aspectos identificados: 2
  - Aspectos en el sistema: 1
  - Efectividad: 1/1 =1
  - Ruido: 1/2 = 0.5
  - Tiempo: 4453 ms.
- Primer Algoritmo
  - Aspectos positivos: 0
  - Aspectos negativos: 3
  - Aspectos identificados: 3
  - Aspectos en el sistema: 1
  - Efectividad: 0/3 = 0
  - Ruido: 3/3 = 1
  - Tiempo: 359 ms.

Sistema de administración de artículos (AA)

- Algoritmo con ontología
  - Aspectos positivos: 12
  - Aspectos negativos: 0
  - Aspectos identificados: 12
  - Aspectos en el sistema: 13
  - Efectividad: 12/13 =1
  - Ruido: 0/12 = 0
  - Tiempo: 43765 ms.
- Primer Algoritmo
  - Aspectos positivos: 12
  - Aspectos negativos: 6
  - Aspectos identificados: 18
  - Aspectos en el sistema: 13
  - Efectividad: 12/13 = 0.92
  - Ruido: 6/18 = 0.33
  - Tiempo: 4625 ms.

Para comparar la efectividad obtenida en cada proyecto por cada uno de los algoritmos se presentan los resultados en forma gráfica. En el gráfico que presenta los resultados obtenidos para la efectividad con ambos algoritmos (Figura 7.11) se puede observar que la efectividad para el algoritmo que usa la ontología es una línea recta sobre el valor 1. Este valor es el resultado óptimo para esta métrica, nos esta indicando que la cantidad de aspectos candidatos detectados por el algoritmo es la cantidad esperada. Si se compara este resultado con el obtenido para el primer algoritmo se puede ver que en los ejemplos analizados también son detectados la cantidad de aspectos esperada, con un pequeño desvío dado por los resultados de un proyecto. Comparando la efectividad entre los algoritmos se concluye que ambos tienen una efectividad muy buena.

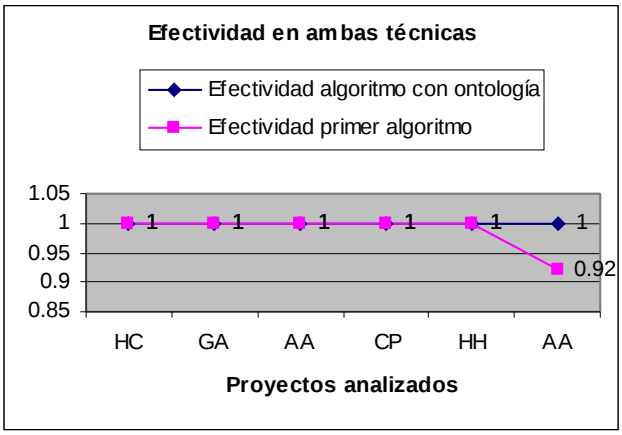

Figura 7.11: Efectividad en ambas técnicas

Los resultados obtenidos para el ruido en ambos algoritmos se pueden observar en la Figura 7.12.

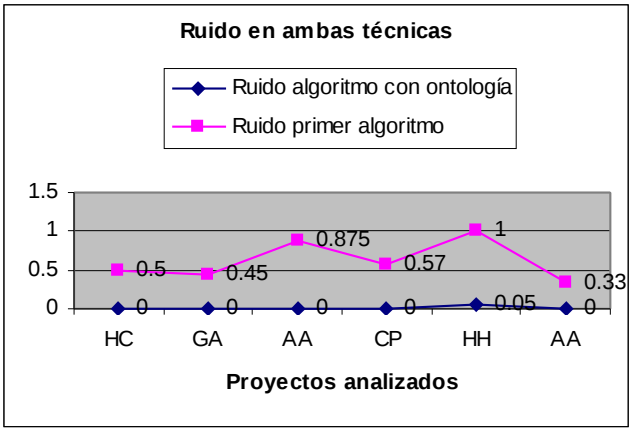

Figura 7.12: Ruido en ambas técnicas

Entre los valores obtenidos para el ruido en el algoritmo que usa la ontología se puede notar que la cantidad de resultados negativos sobre los esperados es nula o muy aproximada a nula. Con este algoritmo en muy pocas ocasiones se identifican aspectos candidatos que no lo son para el ejemplo analizado. No se puede decir lo mismo del primer algoritmo, ya que en todos los proyectos analizados se identifican algunos aspectos candidatos que no lo son. Incluso hay un proyecto, el proyecto de administración de habitaciones de un hotel, en el que todos los resultados obtenidos por al algoritmo son incorrectos. Comparando el ruido entre los dos algoritmos de identificación de aspectos se concluye que los valores para el algoritmo que usa la ontología son casi ideales. Hay mucho ruido en los resultados que presenta Aspect Extractor al usar el primer algoritmo.

Otra forma de comparar los algoritmos de identificación de aspectos es mediante gráficos que indican la cantidad total de aspectos identificados, la cantidad de resultados correctos y la cantidad de resultados erróneos. En la Figura 7.13 se muestran los resultados de éstas métricas para el algoritmo que usa la ontología. Se puede observar que los resultados obtenidos para los aspectos identificados y los aspectos positivos están a la misma altura, lo que indica que la cantidad de aspectos positivos identificados es casi igual a la cantidad de aspectos identificados. Los resultados que Aspect Extractor presenta al usar el algoritmo que usa la ontología son correctos en casi todos los proyectos analizados. Otro punto muy notable en la

Figura 7.13 es que las barras que indica a los aspectos incorrectos detectados por el algoritmo esta en casi todos los casos en el valor cero, lo que indica que el algoritmo no presenta resultados incorrectos en casi todos los proyectos analizados.

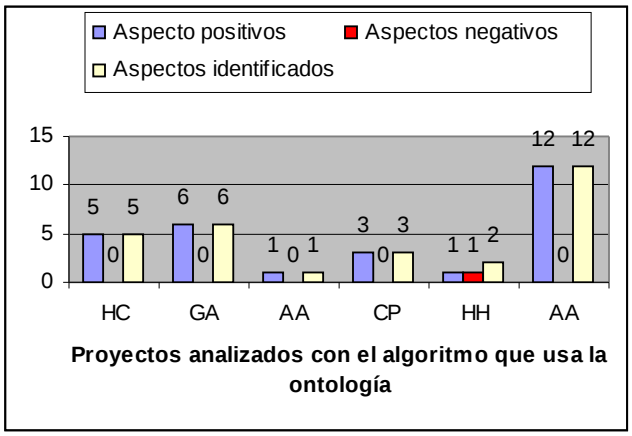

Figura 7.13: Aspectos identificados con el algoritmo con ontología

En la Figura 7.14 se muestran los resultados de las métricas aspectos positivos, aspectos negativos y aspectos identificados para el primer algoritmo. Se puede observar que en casi todo momento las barras que representan a los aspectos negativos son más altas que las que representan a los aspectos positivos, lo que indica que entre los resultados que presenta la herramienta hay más resultados incorrectos que correctos. Si bien se vio que la efectividad para este algoritmo resulto con un valor muy bueno, este algoritmo presenta una cantidad de resultados grande, entre los que se encuentran los resultados esperados y otros resultados incorrectos. Distinto es el caso para el algoritmo que usa la ontología, que la efectividad presentada es muy buena, pero casi no presenta resultados incorrectos.

Otra forma de comparar los algoritmos de identificación de aspectos es mediante el tiempo que se necesita para llevar a cabo la tarea (Figura 7.15). Se puede notar que el algoritmo que usa la ontología utiliza tiempos mucho más grandes que el primer algoritmo. Esto se debe que la estructura de la ontología es recorrida una vez por cada caso de uso, y una vez más por los requerimientos especiales del proyecto, y este recorrido a través de la ontología es muy costoso. Si se compara la métrica tiempo entre los dos algoritmos es más beneficiado el primer algoritmo. Si se tiene en cuenta que el algoritmo será ejecutado solo una vez proyecto, el tiempo que tardan los algoritmos no es de gran importancia.

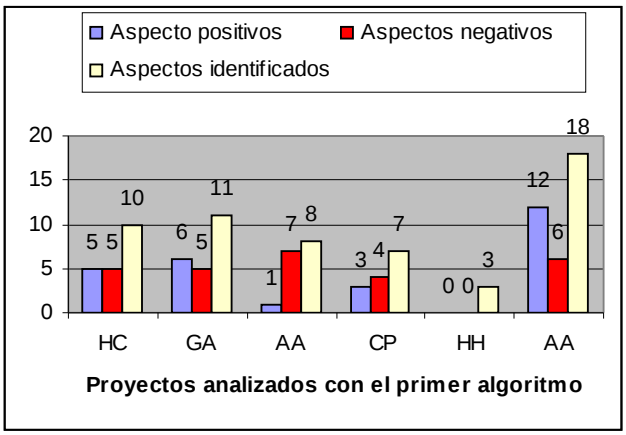

Figura 7.14: Aspectos identificados con el primer algoritmo

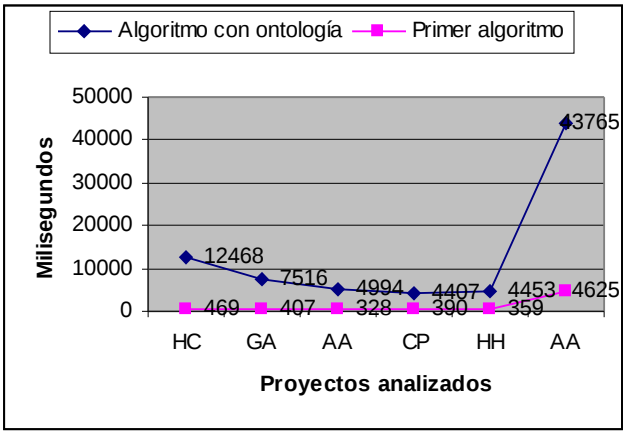

Figura 7.15: Velocidad en ambas técnicas

7.5 Conclusión

Al comparar los resultados obtenidos con Aspect Extractor Tool usando el primer algoritmo y el algoritmo que usa la ontología se pueden destacar los siguientes puntos:

- El listado de aspectos candidatos obtenido con el algoritmo que usa la ontología es más corto y preciso que el obtenido con el primer algoritmo. Esto facilita la tarea del analista al momento de realizar la tarea de Elección de Aspectos Candidatos del enfoque Aspect Extractor.
- Los nombres de los aspectos candidatos retornados por Aspect Extractor Tool con ontología son conceptos estandarizados. Este conocimiento estandarizado puede ser entendido y compartido por todas las personas que forman parte del proyecto.
- Los nombres de los aspectos candidatos identificados con el algoritmo que usa la ontología dan una idea clara de la responsabilidad del aspecto al que representan. De esta manera también se ayuda al analista en la tarea de Elección de Aspectos Candidatos del enfoque Aspect Extractor.
- Los resultados obtenidos con el algoritmo que usa la ontología tienen una efectividad muy buena y casi no se presenta resultados incorrectos.

Capítulo 8: Conclusiones

En este trabajo se realizó un análisis de algunos de los mecanismos existentes para la identificación de aspectos candidatos en etapas tempranas del ciclo de desarrollo de software. Del análisis presentado se concluye que el Desarrollo de Software Orientado a Aspectos (DSOA) actualmente presenta dos problemas principales. Uno de los problemas es la falta de la unificación de conceptos en los términos que representan a los crosscutting concerns en las aplicaciones de software. El segundo problema es la necesidad de un enfoque que agilice la identificación, especificación, integración y evaluación de los crosscutting concerns en etapas tempranas del desarrollo de software. Actualmente, existen varios enfoques para la identificación de aspectos candidatos, pero ninguno cubre completamente las necesidades de los analistas: la identificación, especificación, integración y evaluación de los crosscutting concerns en etapas tempranas del desarrollo de software de forma ágil, eficiente y que presente una unificación de conceptos en términos que representan a los crosscutting concerns identificados con semántica clara.

Como solución a los problemas que el DSOA enfrenta se presenta una ontología. Una ontología es una forma de representar el conocimiento compartido de un dominio, proporciona un marco para entender, unificar y extraer términos de la realidad. Una ontología permite realizar consultas sobre los conceptos modelados en ella e inferir conocimiento que no esté indicado explícitamente por medio del uso de razonadores, éste es uno de los beneficios más interesantes a la hora de decidir trabajar con una ontología. La ontología que se creó es una ontología de dominio, el dominio es el DSOA. Al construir una ontología para el área de conocimiento del DSOA se unificaron los términos que representan a los crosscutting concerns. La ontología creada se incorporó a la herramienta Aspect Extractor Tool para mejorar el proceso Aspect Extractor en la identificación de aspectos candidatos. Se crearon dos ontologías: una en el idioma español y otra en el idioma inglés, de forma tal que la herramienta pueda continuar trabajando con proyectos en inglés y español como lo hace actualmente.

La ontología definida se integró a la herramienta Aspect Extractor Tool, la cual es una automatización del enfoque Aspect Extractor. El enfoque esta divido en cinco tareas principales: Identificar concerns, Elección de los aspectos candidatos, Especificar aspectos candidatos, Identificar conflictos y Modelar en UML. La extensión que se presentó es otra forma de realizar la sub-tarea Identificar Crosscutting Concerns de la tarea Elección de Aspectos Candidatos. El algoritmo que realizaba esta tarea la hacia de dos formas principales: la primera de ellas se da cuando un concern corta transversalmente más de un caso de uso, la segunda forma se da cuando se identifica un caso de uso que cuenta con un requerimiento especial, no funcional, en su especificación. El mecanismo que se extiende incorpora el uso de una ontología. Se recorre la ontología con las palabras relevantes de los casos de uso buscando concordancias de estas palabras con las palabras que representan a los aspectos en la ontología. Si se encuentran coincidencias, el nombre del aspecto modelado es mostrado al analista como aspecto candidato, los nombres presentados para los aspectos candidatos son estandarizados y dan al analista una idea clara de la responsabilidad que este aspecto candidato posee. Actualmente, el analista puede configurar con cual de los dos mecanismos desea que la tarea de Identificar Crosscutting Concerns sea realizada, esta decisión se recuerda para ejecuciones futuras de la herramienta.

Se analizaron algunos ejemplos para comparar los dos algoritmos de identificación de aspectos candidatos que Aspect Extractor Tool posee. Se puede concluir que al usar la ontología se obtiene una lista de aspectos candidatos más corta, más precisa y con nombres estandarizados para los aspectos candidatos que presenta. De esta manera se facilita la tarea Elección de Aspectos Candidatos que debe realizar el analista de forma manual y se ayuda en la comunicación entre los stakeholders del proyecto, ya que todos sabrán el significado de cada término del que se este hablando.

8.1 Beneficios y limitaciones

La herramienta Aspect Extractor Tool al usar la ontología para la identificación de los crosscutting concern presenta varios beneficios sobre el método que busca la repetición de palabras en los casos de uso. Presenta al analista una lista más corta y precisa de los aspectos candidatos propuestos, donde cada uno de éstos da al analista una idea clara de la responsabilidad que debe contener. La lista de aspectos candidatos propuestos presentada es una lista estandarizada de los conceptos que representan a los aspectos en el DSOA, permitiendo de esta forma una mejor comunicación y mayor entendimiento entre los stakeholders del proyecto.

Este mecanismo de identificación también presenta una limitación con respecto al mecanismo anterior: no identifica a aspectos candidatos funcionales, limita la identificación sólo a la identificación de aspectos candidatos no funcionales.

8.2 Trabajos futuros

A continuación se detallan trabajos futuros que se desprenden del trabajo presentado:

- Usar la ontología para trabajar en otras tareas del enfoque Aspect Extractor. En el trabajo presentado se usa la ontología en la sub-tarea Identificar crosscutting concern de la tarea Elección de Aspecto candidatos. La ontología también es usada para la obtención de las palabras de los casos de uso que infieren a los aspectos candidatos y el cálculo de los conflictos existentes entre éstos mediante el uso de razonadores. Se extenderá el uso de la ontología en las tareas restantes del enfoque Aspect Extractor tal como identificar las relaciones entre los aspectos y realizar el cálculo y la resolución de conflictos.
- Extender el uso de la ontología de forma de trabajar con el lugar semántico que ocupan las palabras dentro de los casos de uso, de esta forma se podrán obtener resultados más precisos.
- Extender la ontología de manera de no sólo identificar crosscutting concern, sino también identificar concern funcionales en los proyectos.

Bibliografía

[BM06] Lorena S. Baigorria, Germán Montejano - Métricas aplicadas a la Programación Orientada a Aspectos (Workshop de Investigadores en Ciencias de la Computación WICC-2006).

[DI76] Dijkstra, E.W. - A Discipline of Programming (Prentice-Hall Series in Automatic Computation), 1976.

[NPME04] A. Navasa, K. Palma, J.M. Murillo, Y. Eterovic - Dos modelos arquitectónicos para el DSOA (Esponsorizado por AOSD-Europe: European Network of Excellence on Aspect-Oriented Software Development. En Colaboración con IX Jornadas de Ingeniería de Software y Bases de Datos, Málaga), 2004.

[BCC05] Klaas van den Berg, José María Conejero, Ruzanna Chitchyan - AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation. (AOSD-Europe), 2005.

[WC304] OWL Web Ontology Language - Use Cases and Requirements (W3C Recommendation - http://www.w3.org/TR/webont-req/), 2004.

[HDMP05] Betina Haak, Miguel Díaz, Claudia Marcos, Jane Prior – Identificación Temprana de Aspectos (ISISTAN Instituto de Sistemas Tandil), 2005.

[RA03] P. Rayson - Matrix: A statistical method and software tool for linguistic analysis through corpus comparison (Computing Department, Lancaster University), 2003.

[HD05] Betina C. Haak, Miguel A. Díaz – Aspect Extractor: Identificación de Aspectos en la Ingeniería de Requerimientos (Tesis de Grado presentada en la Facultad de Ciencias Exactas de la UNCPBA), 2005.

[RO04] Lars Rosenhainer - Identifying Crosscutting Concerns in Requirements Specifications (In Proceedings of OOPSLA Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop. October 2004, Vancouver, Canada.).

[SRR05] Américo Sampaio, Awais Rashid and Paul Rayson - Early-AIM: An Approach for Identifying Aspects in Requirements (International Conference on Requirements Engineering (RE), IEEE), 2004.

[SLRR05] Américo Sampaio, Neil Loughran, Awais Rashid and Paul Rayson - Mining Aspects in Requirements (presented at Early Aspects 2005: Aspect-Oriented Requirements Engineering and Architecture Design Workshop (held with AOSD 2005), Chicago, Illinois, USA).

[BA04] Elisa Baniassad, Siobh´an Clarke - Theme: An Approach for Aspect-Oriented Analysis and Design (Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on).

[TAA07] Marta S. Tabares, Raquel Anaya, Fernando Aranjo - Aspect - Oriented Software Engineering: an experience of application in a help desk system (Aspect-Oriented Requirements Engineering and Architecture Design, 2007. Early Aspects at ICSE: Workshops in).

[CLMC05] Isi Castillo, Francisca Losavio, Alfredo Matteo, Rafael Caldera - La Fiabilidad y los Requisitos de Software bajo la Perspectiva de la Orientación a Aspectos (X Workshops Iberoamericano de Ingeniería de Requisitos y Ambientes de Software), 2005.

[LHB06] Fernando Aranjo, Raquel Anaya Hernández, Marta Silvia Tabares Betancur – Una revisión de modelos y semánticas para la trazabilidad de requisitos (Revista EIA, ISSN 1794-1237 Número 6, p. 33-42. Diciembre 2006 Escuela de Ingeniería de Antioquia, Medellín (Colombia)).

[FI04] Filman, R. - Aspect-Oriented Software Development. (Addison-Wesley. Research Institute for Advanced Computer Science NASA Ames Research Center Moffett Field, California), 2004.

[KLMMLLI97] Kiczales, G.; Lamping, J.; Mendhekar, A.; Maeda, C.; Lopes, C. V.; Loingtier, J.-M.; Irwin, J. - Aspect- Oriented Programming. (European Conference on Object- Oriented Programming, ECOOP, Jyväskylä, Finland, June 9-13, 1997).

[PZ03] Pipeta E. Zancanela L. – Aspect Weaving Strategies (Journal of Universal Computer Science), 2003

[MJVRV 97] F. Matthijs, W. Joosen, B. Vanhaute, B. Robben, P. Verbaeten. - Aspects should not die (Object-Oriented Technology: ECOOP'98 Workshop Reader), 1997.

[PDM02] Prior J., Diaz Pace A, Campo M - Reflection on Separation of concerns (RITA. Vol.9. Num.1. 2002).

[PM03] Prior j, Marcos C. - Solving Conflict in Aspect – Oriented Aplications (In Proceedings of the Fourth Argentine Symposium on Software Engineering (ASSE´2003), 32 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), ISSN 1666-1141, Volumen 32 Buenos Aires, Argentina. September 2003).

[HDPM05] Betina Haak, Miguel Díaz, Jane Pryor, Claudia Marcos - Un Enfoque Automatizado para la Identificación Temprana de Aspectos (Workshop Chileno de Ingeniería de Software. Valparaiso Chile. Noviembre 2005).

[HE01] Lizka Johany Herrera – Ingeniería de Requerimientos, 2001.

[BTA05] Jethro Bakker, Bedir Tekinerdoğan, Mehmet Aksit - Characterization of Early Aspects Approaches (Proceedings of the Early Aspects Workshop at AOSD, 2005).

[SSKRWK02] Schauerhuber, W. Shinier, E. Kapsammer, W. Retschitzegger, M. Wimmer1, and G. Kappel - A Survey on Aspect-Oriented Modeling Approaches (This research has been partly funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.).

[WSZP06] Victor Winter, Harvey Siy, Mansour Zand, Prasanna R. Aryal - Aspect Traceability through Invertible Weaving (In Early Aspects Workshop at. AOSD'06, Bonn, Germany).

[CF05] Carine Courbis, Anthony Finkelstein - Towards Aspect Weaving Applications (Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on).

[AOSD] Aspect-Oriented Software Development, (http://aosd.net), 2004.

[EA] Aspect-Oriented Requirements Engineering and Architecture Design (http://www.early-aspects.net/)

[GR93] Gruber, T. R. A translation approach to portable ontology specifications (Knowl. Acquis., Vol. 5, No. 2. (June 1993), pp. 199-220.).

[BH06] Sue Black, Mark Harman - Aspect Oriented Software Development: Towards A Philosophical Basis (Technical Report TR-06-01, Department of Computer Science, King's College London), 2006.

[RMA03] Rashid, A., Moreira, A., and Araújo, J. 2003. Modularisation and composition of aspectual requirements (In 2nd Int'l Conf. Aspect-Oriented Software Development (AOSD 2003), Boston)

[RSMA02] Rashid, A., Sawyer, P., Moreira, A., and Araujo, J. 2002. Early aspects: A model for aspect-oriented requirements engineering (In Joint Int'l Conf. Requirements Engineering (RE), (Essen, Germany). IEEE, 199202.), 2002.

[RA02] Ramnivas Laddad - I want my AOP (Part 1, 2, 3 from Java World 2002)

[IEEE] IEEE Standard Glossary of Software Engineering Terminology (http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html)

[BTA04] Jethro Bakker, Bedir Tekinerdoğan, Mehmet Aksit - Characterization of Early Aspects Approaches (Proceedings of the Early Aspects Workshop at AOSD, 2005)

[CW01] S. Clarke and R. J. Walker - Composition patterns: An approach to designing reusable aspects. In International (IEEE Computer Society Washington, DC, USA, ICSE 2001)

[SRG00] P. Sawyer, P. Rayson, and R. Garside. - REVERE: support for requirements synthesis from documents (Information Systems Frontiers Journal. Volume 4, issue 3, Kluwer, Netherlands, pp. 343 – 353) , 2000.

[SR08] Americo Sampaio and Awais Rashid - Requirements-Level Aspect Identification Tool (EA-Miner) (AOSD-Europe-ULANC-41)

[AB04] María Teresa Abad – Inteligencia Artificial (FIB-UPC Ontologías), 2004.

[GA97] R. Garside et al. Corpus Annotation: Linguistic Information from Computer Text Corpora (London & New York: Longman), 1997.

[CSRSS06] Ruzanna Chitchyan, Americo Sampaio, Awais Rashid, Pete Sawyer, Safoora Shakil Khan (University of Lancaster, UK, Initial Version of Aspect-Oriented Requirements Engineering Model), 2006.

[CHRS05] Ruzanna Chitchyan, Awais Rashid, Peter Sawyer - Comparing Requirements Engineering Approaches for Handling Crosscutting Concerns (Workshop on Requirements Engineering (held with CAiSE), Porto, Portugal, June 12-14), 2005.

[GJ04] Pedro Antonio Gómez Sánchez y Jesús Enrique Villalobos Jiménez - Introducción a la Programación Orientada a Aspectos (Escuela Politécnica Superior de Albacete – Universidad de Castilla la Mancha), 2004.

[GI06] Guadalupe E., Ibargüengoitia G. - Desarrollo de Software Orientad a Aspectos (La SG '06 Conferencia y Expo, congreso técnico sobre desarrollo de software enfocado a profesionales de TI en México y Latinoamérica), 2006.

[PH02] M. Chantal Pérez Hernández - Explotación de los conceptos textuales informatizados para la creación de bases de datos terminológicas basadas en el conocimiento (VII Simposio – Lisboa, 2002)

[GR93] Gruber T. - A Translation Approach to Portable Ontology Specifications (Knowledge Acquisition, 5 (2), 199-220, 1993).

[TA07] Marcelo Tallarico – Uso de ontologías en tareas de recupero de información (Tesis de Licenciatura), 2007.

[GU04] Nicola Guarino - Formal Ontology and Information Systems (Laboratory for Applied Ontology, ISTC-CNR, Italy)

[ME07] Morfeo-EzWeb - Técnicas de descubrimiento de recursos en función del contexto (Morfeo Proyect), 2007.

[NM01] Natalya F. Noy and Deborah L. McGuinness - Ontology Development 101: A Guide to Creating Your First Ontology (Stanford University, Stanford, CA, 94305), 2001.

[FJ07] Jorge Fox, Jan Jurjens - A Framework for Analyzing Composition of Security Aspects (CAiSE 20067)

[HR05] Pavel Hruby - Domain-Driven Development with Ontologies and Aspects (ECOOP 2005 workshop on Views)

[GR05] Thomas R. Gruber - Toward Principles for the Design of Ontologies Used for Knowledge Sharing (Domain Specific Modeling Workshop, OOPSLA 2005)

[CO05] Jesús Contreras -Tutorial Ontologías (Universidad Complutense de Madrid), 2005.

[SA03] Shamsfard, M.; Abdollahzadeh Barforoush, A. The state of the art in ontology learning: a framework for comparison (The Knowledge Engineering Review archive Volume 18 , Issue 4 (December 2003))

[AB05] Miguel Ángel Abián – La Web Semántica Hoy.

[GVJH95] Erich Gamma, John Vlissides, Ralph Johnson, Richard Helm - Design Patterns: Elements of Reusable Object-Oriented, 1995.

[AAE99] Abu-Salem, H., Al-Omari, M., and Evens, M. W. Stemming methodologies over individual queries words for an arabian information retrieval system (JASIS, 50(6):524–529, 1999)

[LOOM] http://www.isi.edu/isd/LOOM/LOOM-HOME.html

[PROTEGE] http://protege.semanticweb.org

[KAON] http://kaon.semanticweb.org/

[WEBONTO] http://kmi.open.ac.uk./projects/webonto

[ONTOEDIT] http://www.ontoknowledge.org/tools/ontoedit.shtml

[ONTILIGUA] http://www.ksl.stanford.edu/software/ontolingua/

[SN] Snowball http://snowball.tartarus.org/

[GRAPHVIZ] http://www.graphviz.org/

[DDAGMMS05] Chabela Dragoevich, Jorge Diezhandino, Salvador Aragón, José Jesús García Rueda, Elisa Mena, Jaime Moreno, Mariano Sanz - Herramientas avanzadas para la investigación, la docencia y la gestión del conocimiento (Grupo de expertos de Campusred; Informe Diciembre 2005; Chabela Dragoevich - Fundación Telefónica (Directora))

[BS04] Elisa Baniassad and Siobh´an Clarke - Finding Aspects In Requirements with Theme/Doc (Workshop on Early Aspects, held with AOSD 2004)

[TA72] R. E. Tarjan - Depth First Search an Linear Graph Algorithms, 1972.

[SPARQL] http://www.w3.org/TR/rdf-sparql-query/ - SPARQL Query Language for RDF 2008

[JENA] http://jena.sourceforge.net/

[LHB06] Luis Fernando Londoño, Raquel Anaya Hernandez, Marta Silvia Tabares Betancur - Un Análisis de la Aplicación de la Ingeniería de Requisitos Orientada por Aspectos desde la Óptica de la Industria del Software, 2006.

[HMCQ04] Hernández Marín Gilma Lissette, Castro Quiñónez Geraldina Altagracia - Razonamiento con Incertidumbre (Universidad Don Bosco, Facultad de Ingeniería, Escuela de computación.), 2004.