Sistema de gestión de *KOS*: vocabularios controlados y Light ontologies

Universidad Carlos III de Madrid

Trabajo de Fin de Grado - Ingeniería Informática

Alumno: Borja López Gómez

Tutora: Anabel Fraga Vázquez

Septiembre 2012

Resumen del proyecto

Alumno	Borja López Gómez
Título	Sistema de gestión de KOS: Vocabularios controlados y Light ontologies
Tutora	Anabel Fraga Vázquez
Descripción	El objetivo del proyecto es desarrollar un módulo de gestión de un sistema de organización del conocimiento (KOS) basado en la administración de un vocabulario controlado y ontologías ligeras, con el fin de buscar la máxima calidad de los requisitos escritos a posteriori por compañías de desarrollo software. Se realiza un estudio previo de las necesidades del cliente y las posibles soluciones actuales del mercado. Además, se describe la teoría necesaria sobre sistemas de organización del conocimiento. A continuación, se desarrolla un análisis y diseño detallado del módulo a desarrollar, a ser completado en un tiempo y coste definidos previamente. Este módulo es parte de *knowledgeManager*, un software completo de gestión de KOS.
Summary	The project is focused in developing a little module which manages a Knowledge Organization System (KOS) based-on controlled vocabularies and light ontologies. The main goal is to provide quality to the requirements written by a software developer company. First of all, not only has this document a previous study of the client needs, but also it has a review of the possible solutions in the market which nowadays may solve these needs. Later on, this document contains a deep analysis and a detailed design of the solution for the module, trying to be on time to reach the deadlines of planning and budget defined previously. This module is a part of *knowledgeManager*, a complete KOS-management system.

Agradecimientos:

Quiero agradecer a todos los que estuvieron, y los que están, por el apoyo recibido durante estos años, sin ellos seguramente no hubiera sido lo mismo; y a los que estarán, porque seguro que nos esperan muchas sorpresas en el futuro. En especial, destacar el apoyo que mis padres me han brindado siempre que lo he necesitado en general, y más en particular este último año, muchas gracias, sin vosotros hubiera sido misión imposible. Además de toda mi familia, se agradece de corazón el ánimo recibido. Y cómo no, a mis amigos, los amigos de verdad que nunca fallan cuando más lo necesitas, para qué decir nombres si ellos ya lo saben. A todos los que algún día confiaron en mí, y más importante aún los que lo harán en el futuro. A mis tutores y la gente del labo, es un verdadero placer estar aprendiendo tantas cosas con vosotros.

Índice de contenido

Índice de tablas

Índice de ilustraciones

1 Estado del arte

Introducción al desarrollo de software

1.1.1 Contexto histórico

El mundo del *software* tal y como lo conocemos hoy día ha cambiado mucho a lo largo de la historia. Como es lógico, no siempre han existido convenios y estándares de análisis, diseño, programación o pruebas. Hemos avanzado desde los primeros desarrollos científicos y militares durante las décadas de los 60 y 70 hasta los complejos sistemas de intercambio y procesamiento de la información basados en *cloud computing* de hoy día, pasando por la globalización de la informática que se produjo a finales de los años 80 y toda la década de los 90, con la aparición de los primeros *personal-computers*.

La evolución que ha vivido el mundo de la informática durante los últimos 20 años se debe, en gran medida, tanto al empuje de los gobiernos y los fines militares como a la necesidad empresarial que se vive históricamente en los mercados para conseguir diferenciar tu producto por encima de la competencia. No importa cuál sea el tamaño de una empresa, las más pequeñas utilizarán algún software de gestión de sus datos mientras que las más grandes necesitarán invertir en desarrollo de aplicaciones comerciales que contribuyan a su negocio.

En los comienzos de la era tecnológica, la mayor parte del foco de desarrollo estaba puesto sobre los componentes *hardware*. Estamos hablando de enormes y pesadas máquinas con muy poca capacidad de computación, donde un pequeño avance en cuanto a rendimiento de la máquina suponía posicionarse por delante del resto.

Como hemos comentado previamente, durante la década de los noventa se produjo un importante desarrollo tecnológico. Ya no sólo era interesante el uso de ordenadores para los gobiernos más pudientes o las grandes compañías, sino que la informática comenzó a globalizarse y empezó a definirse lo que hoy día conocemos como *ordenadores personales*. Así aparecieron los primeros sistemas operativos destinados a aportar mayor facilidad de uso, comenzando a alejarse de las líneas de comando, muy complejas y tediosas de cara al usuario final.

Desde entonces, durante estos veinte años el mundo de la informática ha evolucionado en todos los sentidos. Por un lado ha seguido mejorándose la capacidad de computación y rendimiento de las máquinas, lo que permite ejecutar pesados y complejos programas. Pero por otro lado, más importante aún, ha evolucionado enormemente la forma en que se desarrolla software, ya sea para sistemas empotrados, ordenadores personales, *tablets*, dispositivos móviles, lavadoras, coches, aviones, y un largo etc.

1.1.2 Evolución del desarrollo de software

Los métodos y técnicas utilizadas para desarrollar un producto software evolucionan constantemente año tras año. La forma de desarrollar aplicaciones ya no sólo es radicalmente distinta a cómo se hacía en los años 80 por ejemplo, sino que en algunos organizaciones es bastante diferente a como se ha estado haciendo hasta hace relativamente pocos años. Han aparecido nuevos lenguajes de programación que nos permiten abstraer información del mundo real frente a la programación estructural, han evolucionado los entornos de desarrollo facilitando enormemente el desarrollo de software, han surgido herramientas que permiten automatizar las pruebas de software, etc.

Pero al margen de la evolución a nivel técnica y tecnológica, cabe destacar que ha habido (y sigue habiendo) un gran cambio en la forma de afrontar un proyecto de desarrollo. Han surgido nuevos estándares con el fin de globalizar los procesos de desarrollo, metodologías de desarrollo (tanto tradicionales como ágiles) que nos permiten conocer con exactitud qué es lo que se va a desarrollar, cómo se debe hacerlo y en algunos casos cuánto esfuerzo va a suponer, procesos de calidad con el fin de obtener un producto elegante y fácil de mantener, etc. En pocas palabras, ha aparecido, ha evolucionado y se ha asentado una disciplina conocida como **Ingeniería del Software**.

1.1.3 ¿Por qué es importante la ingeniería del software?

A estas alturas se trata de una pregunta muy fácil de responder. Como hemos comentado durante el apartado anterior, el desarrollo de *software* en sus comienzos podría considerarse un “arte” que muy pocos eran capaces de conseguir. [23] Como tal, no existía ninguna metodología ni proceso de control, ya que la programación se desarrollaba prácticamente a medida para cada aplicación.

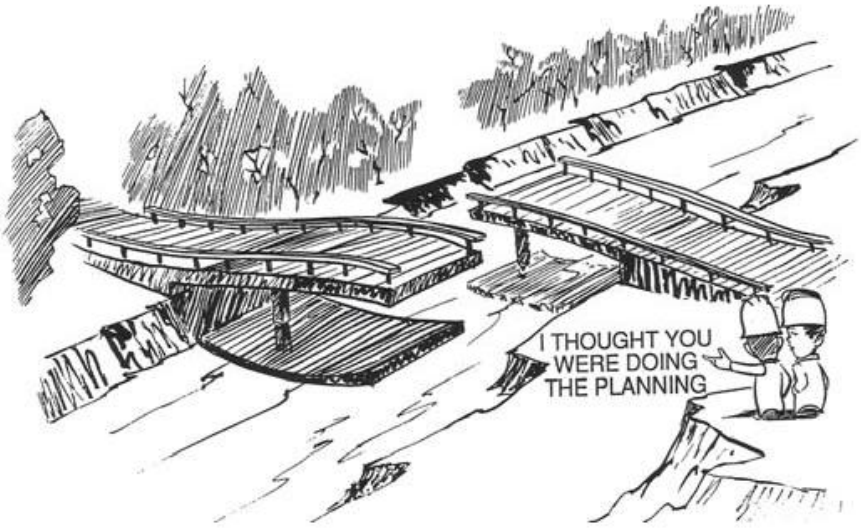

Ilustración 1: La importancia de la ingeniería

Como consecuencia, el mundo de la informática entró en lo que históricamente se conoce como **la crisis del software**. Los productos desarrollados eran en la mayoría de los casos bastante pobres en cuanto a calidad se refiere, hasta que finalmente, los inversores y compradores de *software* decidieron que no se podía continuar de aquella manera [24]. Así comenzaron a aparecer los ciclos de vida, las metodologías de desarrollo, y procesos de planificación, seguimiento y control, entre otros.

1.1.4 Éxito o fracaso en un proyecto de desarrollo

Aunque pueda dar la impresión de que se haya definido la ingeniería del software como una ciencia exacta, no lo es ni mucho menos. A pesar de que ésta disciplina ha evolucionado mucho durante los últimos años, aún queda mucho camino por recorrer. Como ejemplo, el **Informe del Caos** publicado en el año 2009 por la organización *Standish Group* [1] muestra la proporción de proyectos que se llevan a cabo con éxito, los que son cancelados y los que han salido adelante pero con dificultades de plazos o presupuesto en el mundo de *IT*.

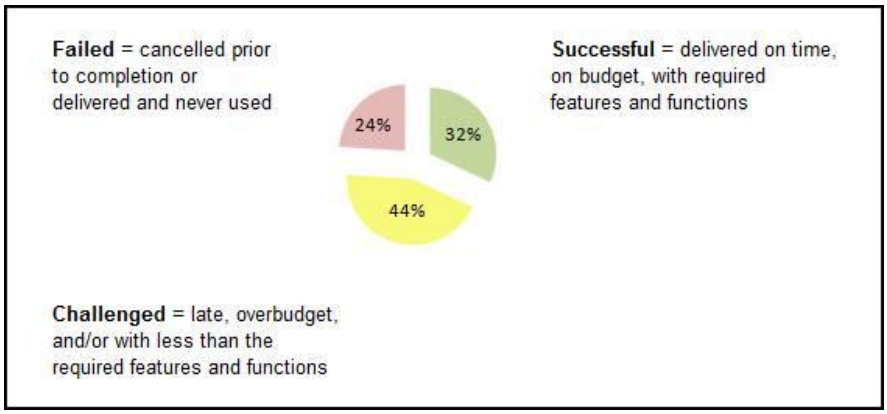

Ilustración 2: Probabilidades de éxito o fracaso de un proyecto

Como se puede ver, el informe [1] muestra que un proyecto software hoy día tiene aproximadamente un 32% de probabilidad de éxito (*Successful)*. Esto es, que se cumplan los plazos de entrega, que no se exceda el presupuesto planificado y que el sistema desarrollado satisfaga los requisitos del cliente. Por otro lado, existe un 24% de probabilidad de que el proyecto sea totalmente cancelado (*Failed)* y nunca llegue a ver la luz, mientras que en un 44% de las ocasiones el proyecto es finalizado pero o bien fuera de plazo, sobrepasando el presupuesto estimado, o con menos funcionalidad de la especificada por el cliente en un principio (*Challenged)*.

	Year 2009	Year 2006	Year 2004	Year 2002	Year 2000	Year 1998	Year 1996	Year 1994
Successful	32%	35%	29%	34%	28%	26%	27%	16%
Challenged	44%	19%	53%	15%	23%	28%	40%	31%
Failed	24%	46%	18%	51%	49%	46%	33%	53%

Ilustración 3: Evolución histórica del éxito/fracaso de un proyecto

Analizando el informe más en profundidad, se aprecia un 32% de éxito en la actualidad frente a un 35% en 2006. Por un lado, cada año hay más experiencia en gestionar proyectos, hay más personal cualificado y certificado, mejor entrenamiento, y mejores técnicas y herramientas. ¿Qué ocurre entonces? Que por otra parte, el ámbito y la complejidad de los proyectos son cada vez mayores, a la vez que el tiempo de desarrollo se ha reducido.

1.1.5 Soluciones al fracaso y búsqueda del éxito

¿Cuál es la mejor solución ante la elevada proporción de proyectos *IT* fallidos? No es tarea fácil dar con la solución, si no probablemente ya se hubiera encontrado. Como dijimos antes, la ingeniería de software no se trata de una

ciencia matemática, sino de un proceso experimental en el que se evoluciona como profesional a la vez que se va aprendiendo de los errores cometidos.

Como hemos visto en el *informe del caos* presentado en el apartado anterior, el hecho de contar con más ingenieros de software y personal más especializado en procesos de calidad en un equipo de desarrollo no es sinónimo de éxito. Aunque cierto es que reduce considerablemente la probabilidad de que el proyecto acabe siendo cancelado.

En la mayoría de los casos el fracaso viene dado por un cúmulo de causas. Es posible que los objetivos del proyecto estén poco claros, que no se haya realizado una correcta planificación, que el equipo de desarrollo se enfrente a una tecnología nueva sin una evaluación correcta de los riesgos que ello supone, que no se utilicen metodologías de gestión y seguimiento de proyectos, o que los recursos humanos sean insuficientes, entre otros.

A continuación se va a exponer lo que sería la ***gestión del conocimiento***, un punto de vista mediante el que se pueden mejorar los procesos de ingeniería y construcción de un proyecto software, que es en lo que se basa el objetivo de este trabajo de fin de grado.

1.1.5.1 Introducción a la gestión del conocimiento

Uno de los objetivos de un buen ingeniero de software debe ser la aplicación de procesos de calidad sobre todo proceso que lleve a cabo. Si se han hecho bien las cosas, es decir, si el desarrollo de un producto se determina viable tanto en tiempo como en coste (planificación), y además se ha aplicado una metodología efectiva sobre el desarrollo de un producto (procesos integrales de ingeniería de software), la pregunta es: "*¿Por qué no va a ser un proyecto exitoso?*"

Además de los distintos procesos estándares de calidad, el módulo desarrollado durante este trabajo de fin de grado tiene como objetivo gestionar el conocimiento de un determinado dominio dentro de una compañía de desarrollo software.

¿Qué se entiende por gestionar el conocimiento? A lo largo de esta memoria, se analizan las principales estructuras destinadas a organizar el conocimiento, también llamadas ***KOS*** (*Knowledge Organization System)*. Además, se focaliza en el estudio de las **ontologías**, un tipo específico de *KOS*, y se plantea el desarrollo de un *software* capaz de abstraer y dar forma a toda la información que se considera relevante a la hora de gestionar un *KOS* mediante un vocabulario controlado, una ontología ligera, grupos semánticos y sugerencias.

Gestión del conocimiento

1.1.6 ¿Por qué es importante gestionar el conocimiento?

A día de hoy hay que tener en cuenta tanto los aspectos prácticos como los teóricos para afrontar un proyecto de desarrollo. No es suficiente con tener unos buenos programadores que conozcan a la perfección el lenguaje en que se va a desarrollar, sino que también se necesita un buen estudio previo para aclarar **qué** se va a construir y **cómo** se va a llevar a cabo.

Una fase muy importante de todo proyecto es la definición del problema y la especificación de los requisitos, donde entran en juego distintos roles como el jefe de proyecto, los analistas o el responsable de calidad, en caso de que exista.

Cuanto mejores sean los analistas de una empresa de desarrollo, mejor y más clara será la especificación del sistema. ¿O no? La lógica humana puede llevarnos a apoyar este razonamiento, pero si ceñimos a los datos estadísticos que se publican año tras año, como por ejemplo el *informe del caos* del que hablamos antes, algo sigue fallando.

Si se analiza un posible escenario real, un grupo de analistas de una importante compañía aeronáutica está diseñando un nuevo sistema de propulsión para la próxima lanzadera espacial. Son profesionales muy elevada reputación que cuentan con una gran experiencia, pero su conocimiento sobre mecánica de aviones es bastante reducido, y como ya sabemos, el cliente no siempre pide realmente lo que necesita, sino lo que cree que puede necesitar.

Supongamos, por tanto, que existen un conjunto de términos específicos dentro del dominio de la aeronáutica, y que interaccionan entre ellos a través de relaciones concretas. Uno de los analistas que se encargan de entender y diseñar el funcionamiento de un módulo del propulsor no es un experto aeronáutico, y para resolver un determinado problema, decide crear un modelo en el que el objeto A está relacionado con el objeto B a través de la relación R1.

Este analista no era consciente de que A sólo puede estar relacionado con B a través de R2, ya que a posteriori su funcionalidad es reutilizada por otro módulo del propulsor cuya restricción es que sólo funciona con relaciones del tipo R2. El diseño de este modelo sigue adelante, el error cae en el olvido, se construye el sistema y tras lanzar las pruebas, el equipo de *testing* comienza a reportar fallos. Finalmente, es necesario encontrar el error previo de diseño, cuyo coste puede ser infinitas veces mayor que si se hubiera detectado durante su fase de diseño.

En conclusión, hubiera tenido un menor coste para esta compañía haber contado con un **experto del dominio**, que hubiera almacenado de alguna forma dicha restricción en un **sistema de organización del conocimiento**, de

forma que hubiera sido posible representar la información del mundo real del dominio en cuestión de una forma mucho más fiable.

1.1.7 Definición de roles

En el apartado anterior se describe un ejemplo de uso en el que incluir *knowledgeManager* en los procesos de análisis y diseño supone ventajas con respecto a la escritura tradicional de requisitos por parte de los analistas. Se habló de distintos roles, pero sin llegar a definir la función de cada uno de ellos.

Como se especifica en las tesis doctorales de Anabel Fraga [22] y Sonia Sánchez-Cuadrado [21], cuando se habla de gestión de dominios se definen los siguientes roles con el fin de optimizar los procesos y obtener los mejores resultados posibles:

- **Analista**: es el encargado de realizar, delegar y supervisar los procesos de análisis y diseño de un sistema de información. Para escribir unos requisitos de calidad puede utilizar el software *Requirements Authoring Tool* [11], bajo licencia de *The Reuse Company*, que se apoya en la ontología gestionada por *knowledgeManager*. Además, si se usa dicha herramienta, el analista será capaz de enviar sugerencias de términos o relaciones al encargado de gestionar la ontología.
- **Experto del dominio**: como su propio nombre indica, es la persona que tiene un conocimiento total sobre el dominio que se va a gestionar. Es quien usa *knowledgeManager*, y en concreto en el módulo de este TFG, es el encargado de gestionar el vocabulario controlado, las ontologías ligeras y las sugerencias de términos y relaciones.
- **Responsable de calidad**: gestiona parámetros caja negra para este proyecto, que determinan el comportamiento de algunos módulos de *knowledgeManager*. En concreto, el responsable de calidad puede utilizar el software *Requirements Quality Analizer* [12], bajo licencia de *The Reuse Company*, de forma que puede modificar parámetros de configuración de la base de datos utilizada para gestionar una determinada ontología.

1.1.8 KOS hoy día: Pros y contras.

Durante este capítulo hemos mencionado varias veces el término *KOS*, sin centrarnos realmente en él. Si queremos gestionar cualquier cosa, el conocimiento en este caso, ¿qué mejor forma que **organizar** dicho **conocimiento** mediante un **sistema** o esquema determinado, con el fin de mejorar el proceso de recuperación de la información? En tan sólo una frase hemos definido, brevemente, cuál es el objetivo final de un *Knowledge Organization System*, o también llamado *KOS*.

Utilizando un lenguaje más práctico, un *KOS* [2] puede verse como un puente entre las necesidades de información de un usuario y un determinado material disponible en una colección. Utilizando sistemas de este tipo, un usuario es

capaz de identificar un objeto de su interés sin tener, a priori, conocimiento de su existencia. Un claro ejemplo son las librerías digitales, donde se usa uno o más *KOS* que se encargan de proporcionar una amplia visión del contenido de la colección y mejoran la experiencia del usuario en cuanto a **recuperación de la información**, más conocido como *Information Retrieval* en el mundo *IT* globalizado.

Existen varios tipos de *KOS*. En general suelen agruparse en tres categorías generales [28]:

- **Listas de términos**: se centran en la enumeración de términos, en ocasiones junto con sus definiciones.
- **Clasificaciones y categorías**: se centran en la creación de conjuntos de elementos.
- **Listas de relaciones**: se centran en crear conexiones entre términos y conceptos.

A continuación se procede a explicar con más detalle en qué consiste cada tipo de *KOS* [2], [5].

1.1.8.1 Listas de términos

- **Ficheros de autoridades**. Se trata de listas de términos que son usadas para gestionar los distintos nombres de una entidad o del valor de un campo particular. Por ejemplo, nombres de países, ciudades u organizaciones. No presenta una compleja estructura ni organización, pero presenta algunas limitaciones en cuanto a su presentación, ya que suele ser alfabética u organizada siguiendo algún esquema de clasificación simple.
- **Glosarios**. Un glosario es una lista de términos, normalmente junto con sus definiciones. Dichos términos pueden ser de un campo específico o de un ámbito más extenso, aunque raramente incluyen distintos significados.
- **Diccionarios**. Son listas alfabéticas de palabras y definiciones. Por lo general, suelen aplicar distintos significados y son utilizados en un ámbito más extenso que los glosarios. Además pueden añadir información sobre el origen de una palabra, pronunciación y morfología. Pueden proporcionar información acerca de sinónimos o términos relacionados, pero no presenta explícitamente una estructura jerárquica que permita agruparlos.
- **Gazetteers**. Son listas de nombres de lugares. Tradicionalmente han sido publicados como libros o índices en atlas. Actualmente se usan mediante coordenadas geoespaciales que indican dónde está un determinado lugar sobre la superficie de la tierra.

1.1.8.2 Clasificaciones y categorías

- **Encabezados**: Proporcionan un conjunto de términos controlados para representar los temas de una determinada colección. Estas listas pueden ser muy extensas y cubrir un amplio rango de temas. Sin embargo, la estructura de este tipo de listas es muy pobre, debido a su limitada estructura jerárquica. En la práctica, estas listas suelen tener reglas que determinan cómo sus elementos pueden ser combinados para proporcionar conceptos más específicos.
- **Esquemas de clasificación, taxonomías y esquemas de categorización**. Estos tipos de *KOS* proporcionan formas de separar entidades en temas más amplios. Por ejemplo existen estructuras jerárquicas que representan una notación alfabética o numérica para representar temas extensos. Aunque pierden la posibilidad de definir relaciones explícitamente como en un tesauro, las taxonomías son cada vez más usadas en el diseño orientado a objetos para mostrar agrupamiento de elementos basado en una determinada característica.

1.1.8.3 Listas de relaciones

- **Tesauros**. Están basados en términos y relaciones entre ellos. Estas relaciones suelen incluir jerarquía, equivalencia (sinonimia), y asociación, y normalmente suelen estar representadas por las notaciones BT (broader term), NT (narrower term), SY (synonym) y RT (related term). Además existen estándares para el desarrollo de tesauros que incluyen reglas de creación de relaciones entre términos (por ejemplo la prohibición de relaciones transitivas), aunque muchos tesauros no sigan todas las normas al pie de la letra. Por contra, los tesauros suelen ser muy grandes, llegando a superar en algunas ocasiones los 50000 términos. Esto es debido a que no existe un tesauro genérico que englobe a todos, sino que un tesauro se corresponde con una determinada disciplina o producto.
- **Redes semánticas**. En los últimos años se han producido avances en este tipo de estructuras aprovechando técnicas de procesamiento del lenguaje natural. No representan estructuras jerárquicas sino redes, donde los términos son representados mediante nodos y las relaciones cuelgan de ellos. Dichas relaciones siguen el estándar BT, NT, SY y RT comentado previamente, y suelen ser de tipo *todo-parte*, *causa-efecto*, o *padre-hijo*. El principal ejemplo de este tipo de *KOS* es *Wordnet*, un sistema de organización de términos sinónimos utilizado hoy día en los módulos de expansión de consultas de muchos motores de búsqueda.
- **Ontologías**. Son la técnica más novedosa en cuanto a sistemas de organización del conocimiento se refiere. Actualmente se están desarrollando ontologías como modelos de temas específicos. Son capaces de representar complejas relaciones entre objetos e incluir distintas reglas y axiomas, característica que no está presente en las redes semánticas. En general, una ontología que describe el

conocimiento en un área específica, suele estar generalmente conectada con sistemas de procesamiento de datos y gestión del conocimiento basados en *indexing* e *information retrieval*.

1.1.9 Ontologías. Definición.

El término *ontología* [3] ha sido utilizado históricamente en el campo de la filosofía, haciendo referencia a la rama de la metafísica que trata sobre la existencia. Pero desde hace relativamente pocos años, ha comenzado a surgir como un nuevo sistema de organización del conocimiento.

Es complicado hoy día definir exactamente qué es una ontología. Existen distintas definiciones, pero las más extendidas son la de ***Gruber*** [27], por un lado, para quien una ontología es la ***conceptualización del conocimiento de un dominio*** [16], y la de ***Guarino*** [26], quien define una ontología como **un producto de ingeniería formado por un vocabulario específico** usado para describir una realidad más un conjunto de asunciones relacionadas con el significado del vocabulario [16].

En otras palabras, se trata de un vocabulario controlado que describe objetos y relaciones entre ellos de una manera formal, y que además contiene una gramática que permite utilizar los términos del vocabulario para expresar algo con sentido dentro del dominio de interés.

En general, se dice que el vocabulario de una ontología es utilizado para realizar preguntas y obtener respuestas [4], de ahí que cada vez se usan más ontologías en sistemas de *indexing* y *retrieval*.

Dejando a un lado el significado filosófico y centrándonos en la especificación del conocimiento de un dominio, lo que existe es exactamente lo que se puede representar, lo que se conoce con el nombre de *Universo del Discurso*.

Existen tres tipos principales de ontologías [5], según el ámbito del conocimiento al que se apliquen:

- **Ontologías de un dominio**: representan el conocimiento de un determinado dominio, como puede ser la medicina, la antropología, la aviónica, etc.
- **Ontologías genéricas**: representan conceptos generales del conocimiento, como el espacio, el tiempo, la materia, etc.
- **Ontologías específicas**: son ontologías especializadas que describen los conceptos para un campo limitado del conocimiento o una determinada aplicación. También se conocen como *meta-ontologías*.

Durante este TFG, una ontología va a estar asociada a un único dominio, por lo que no se profundiza en el estudio de *ontologías genéricas* ni *meta-ontologías*. Un ejemplo general puede ser el siguiente: un portal de venta de libros recibe ofertas de distintas editoriales. Se desea construir un sistema de información

que automatice el proceso de importación de datos, por ejemplo, en formato *csv*. Sin embargo, una editorial utiliza el término "autor", mientras que otra utiliza "creador". ¿Solución? Para conseguir una integración total, se debe añadir una restricción que indique que la relación de tipo "autor" es la misma que la de tipo "creador". Esta restricción extra es, en otras palabras, un caso muy simple de ontología.

1.1.9.1 Gestores de ontologías en el mercado.

En el mercado hoy día existen muchos gestores de ontologías. Consultando varios de ellos, la mayoría están basados en OWL, un lenguaje que es ampliamente utilizado en la actualidad para compartir datos a través de ontologías en la web.

Estos sistemas utilizan XML como forma de enviar la información a través de la web, ya que se utiliza RDF, una descripción conceptual para representar metadatos en web. No se profundiza más en estos aspectos ni se entra en más detalle, ya que no se desea construir el sistema sobre una plataforma web.

Por otro lado, analizando las aplicaciones de escritorio encontradas, la primera conclusión a la que se llega es que hay bastantes enfocadas hacia la gestión de tesauros. Pero realmente no se ha encontrado ninguna herramienta que permita llevar un control total sobre un vocabulario controlado y ontologías ligeras, objetivo principal del módulo a desarrollar.

A pesar de no encontrar ninguna herramienta que realice la funcionalidad requerida, se exponen las características de dos aproximaciones:

- **Swoop**: [13] gestor de ontologías enfocado a la gestión de tesauros. Esta herramienta permite crear, editar y testear ontologías OWL. Fue construido en la Universidad de Maryland, pero actualmente es un proyecto *open-source* con una gran comunidad detrás suyo.

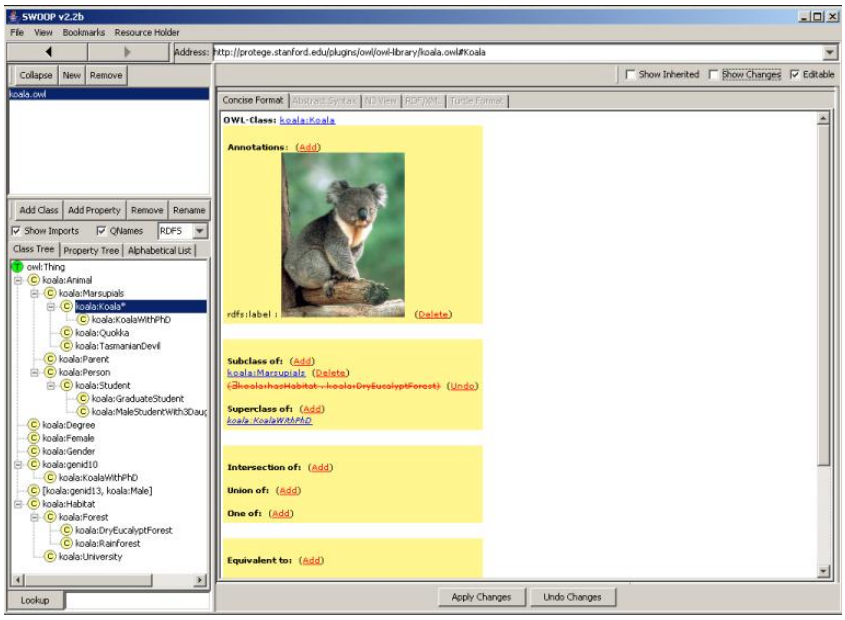

Ilustración 4: Tesauro gestionado por la herramienta *Swoop*

- **Dome**: [14] gestor de ontologías enfocado a la gestión de tesauros. Proporciona un conjunto de aplicaciones que se basan en su simplicidad, completitud y reutilización.

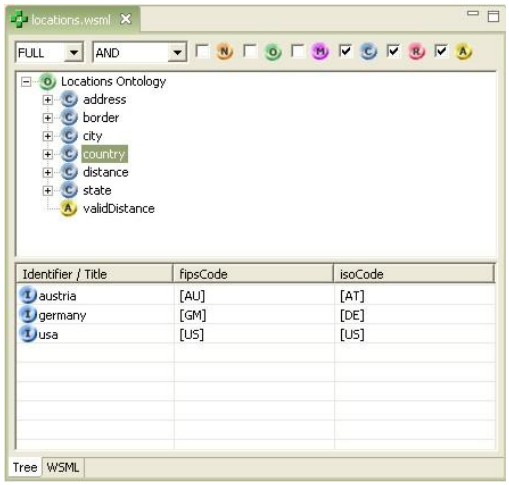

Ilustración 5: Tesauro gestionado por la herramienta *DOME*

- **Protege**: [17] es un gestor y editor de ontologías de código abierto. Está basado en lenguaje *Java*, y proporciona un entorno fácil de usar que lo hace muy flexible para construir prototipos rápidos y desarrollo de aplicaciones. Su plataforma soporta dos formas distintas de modelar ontologías. Por un lado, a través de su propio frame al que denominan *Protégé-Frame*. Por otro lado, a través de editores OWL. Además, las ontologías creadas con esta herramienta pueden ser exportadas en distintos formatos, incluyendo RDF, OWL y XML. Hoy día está siendo

utilizado en distintas áreas como biomedicina, defensa o modelado corporativo, debido a que tiene una gran comunidad de desarrolladores y académicos tras él.

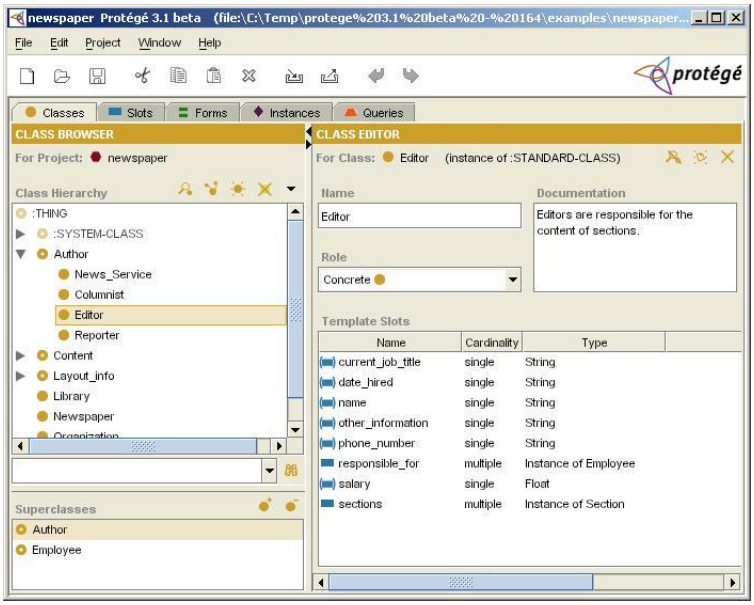

Ilustración 6: Jerarquía de clases en *Protégé*

- **TemaTres**: es un gestor de ontologías basado en web. Su principal uso es la gestión de lenguajes documentales, y está orientado especialmente al desarrollo de tesauros jerárquicos. Contiene más de un millón de términos y presenta un API para ser utilizado a través de *webservices*. Permite exportar en todos los formatos. Presenta una gran cantidad de vocabularios controlados y tesauros.

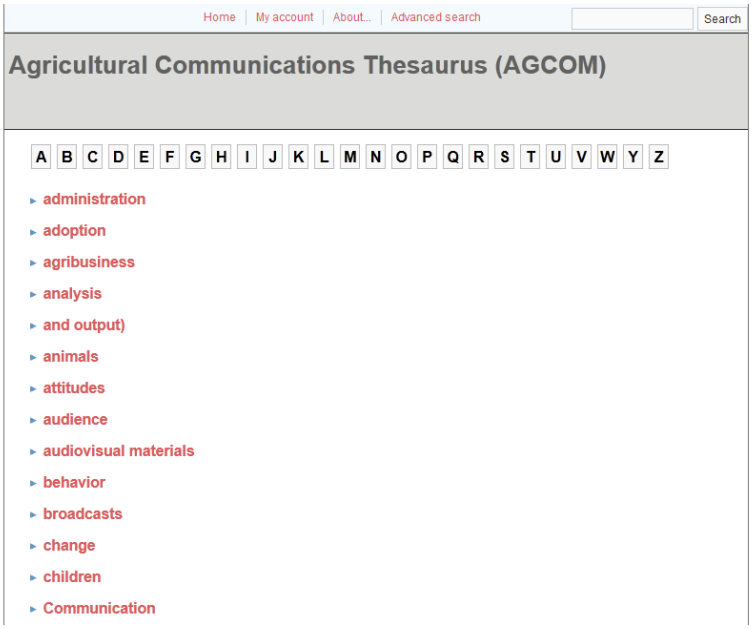

Ilustración 7: Interfaz web de *TemaTres*

A continuación se realiza una comparativa de las aplicaciones analizadas, en función de la información que cada herramienta proporciona al público.

	Swoop	Dome	Protégé	TemaTres
Aplicación de escritorio	Sí	Sí	Sí	No
Gestiona vocabulario controlado	Sí	Sí	Sí	Sí
Gestiona light-ontology	No	No	Sí	No
Gestiona sugerencias del tesauro	No	No	No	No

Tabla 1: Comparativa de gestores de ontologías en el mercado

Como vemos, cumplen algunas de las necesidades especificadas como la gestión del vocabulario controlado, para la elaboración posterior de un tesauro, pero no realizan una gestión íntegra de la *light ontology*, requisito indispensable para solucionar el problema planteado.

1.1.9.2 Vocabularios controlados y ontologías ligeras: knowledgeManager

Ya se ha comentado alguna vez a lo largo de esta memoria que el objetivo final es construir un módulo capaz de llevar el control sobre un vocabulario controlado y gestionar la ontología ligera [25].

¿Qué se entiende por gestionar un **vocabulario controlado**? Administrar una lista de términos, de forma que se pueden llevar a cabo altas, bajas, modificaciones y consultas sobre ellos. Además, cada término contiene información sobre su impacto en el dominio específico y sobre su significado sintáctico y semántico. Por ejemplo, puede existir un mismo término ***coche*** con dos semánticas totalmente distintas como ***racing*** y ***turismo***. De esta forma, el impacto del término *coche* con semántica *racing* puede tener mayor relevancia sobre el dominio actual que el mismo término con semántica *turismo*.

Por otro lado, es necesario llevar a cabo un control total de la **ontología ligera**, ya que es un punto clave ya no sólo del módulo a construir, sino del *knowlegeManager* completo.

Como ya hemos visto, una ontología puede tomar diversas representaciones, pero en cualquiera de ellas deberá tener un vocabulario de términos y algún tipo de especificación de sus significados, es decir, indicar de qué forma los términos están relacionados entre sí. [18] Se denomina ontología ligera a dichas definiciones y representaciones que tienen un menor contenido semántico. Si un modelo conceptual es considerado *ontología pesada* debido a la gran importancia que toma la semántica, las toxonomías y los tesauros son considerados *ontología ligera*.

De esta forma, se busca llevar a cabo una gestión de semánticas y etiquetas sintácticas, con el fin de dotar de un ligero significado sintáctico y semántico a la lista de términos definida previamente.

Herramientas de desarrollo utilizadas

A continuación se presentan algunas de las herramientas más relevantes utilizadas durante el desarrollo del sistema de información.

1.1.10 Visual Studio 2010

Entorno de desarrollo de Microsoft para desarrollar en multitud de lenguajes de programación, y que en concreto incluye C#, el lenguaje en que se desarrollará este proyecto, de acuerdo a lo definido en el Estudio de Viabilidad.

Además de ser una herramienta muy potente por la posibilidad de desarrollo muy de código y usabilidad, permite incorporar herramientas externas, como por ejemplo componentes de control de versiones o de pruebas unitarias. De esta forma se pueden integrar multitud de actividades dentro de un mismo *framework* de desarrollo

1.1.11 **Subversion** (Tortoise SVN)

Para desarrollar el proyecto se ha utilizado un repositorio propio de control de versiones en la nube. Está localizado en una máquina servidor encendida 24 horas al día, los 7 días de la semana. Por lo tanto el repositorio (que incluye tanto código como documentación) está siempre accesible para su uso, excepto en caso de caída del servidor, para lo cual se cuenta con copias de seguridad locales.

Un sistema de control de versiones es especialmente útil para proyectos en los que trabaja mucha gente al mismo tiempo. Aunque el presente proyecto ha sido desarrollado por una sola persona, ya que se trata de un TFG de la universidad, se ha decidido utilizar control de versiones ya que es mucho más fácil desarrollar código desde varias máquinas sin necesidad de transportar los ficheros de código fuente en algún sistema de almacenamiento de datos.

Además los sistemas basados en S*ubversión* se caracterizan por versionar las diferentes subidas que se hacen durante el ciclo de vida. Así, es fácilmente recuperable una versión estable anterior que se necesite recuperar en caso de haber detectado fallos en la funcionalidad por cualquier motivo.

Además, existen componentes que permiten integrar el entorno de desarrollo con el sistema de control de versiones. Es el caso de **Ankh SVN**, utilizado durante este proyecto para gestionar el control de versiones directamente desde Visual Studio.

1.1.12 NUnit

Una de las partes principales de un sistema de información es el desarrollo de las pruebas. Un sistema no puede ser realmente fiable sin tener una buena batería de pruebas que, al menos, certifique que sus distintas funcionalidades funcionan correctamente. Aunque es prácticamente imposible certificar que un sistema es 100% fiable mediante una batería de pruebas, personalmente considero la fase de *testing* como una parte fundamental en un proyecto.

Se ha utilizado *NUnit* como *framework* para llevar a cabo las pruebas unitarias del sistema. Presenta una interfaz de usuario intuitiva para ejecutar las pruebas, pero una vez más, incluye la opción de ser integrada en el entorno de desarrollo, por lo que se ha utilizado *NUnit* directamente desde Visual Studio.

1.1.13 Microsoft Office 2010

Sin duda alguna, el producto estrella de la documentación. Existen gran cantidad de herramientas similares en el mercado, como *iWok* de Apple o la suite *OpenOffice* de código abierto, pero bajo mi punto de vista la potencia que ofrece la suite Microsoft Office no la ofrece ningún otro. Se trata de un conjunto de herramientas ofimáticas desarrollado por Microsoft, destinadas hoy tanto al mundo de negocios como al mundo académico o del día a día. Los programas de *Office* más usados han sido:

- Microsoft **Word** como procesador de textos.
- Microsoft **Excel** como plantilla de cálculos numéricos.
- Microsoft **PowerPoint** como gestor de contenido multimedia, como por ejemplo diapositivas o algunas de las figuras utilizadas a lo largo de esta memoria.

1.1.14 Sql Server Management Studio 2008

Desarrollado por Microsoft, se trata del gestor de bases de datos relacionales basadas en Sql Server. La principal utilidad que se le ha dado a la herramienta es realizar consultas sobre la base de datos para procesos de depuración del código. Además, es interesante para obtener diagramas de diseño de tablas, o ver de forma gráfica las restricciones de integridad entre tablas.

Otro software similar considerado en un principio es *HeidiSql*, una herramienta bastante potente para realizar consultas sobre bases de datos relacionales. Es mucho más ligero en cuanto a necesidades hardware de la máquina, pero ofrece mucha menos potencia que Management Studio en cuanto a temas de visualización de diagramas y otras tareas.

1.1.15 StarUML

Software utilizado para construir diagramas UML. En un principio se pensó en utilizar *Visual Paradigm,* una aplicación muy similar pero que sólo está de prueba durante un tiempo, requiriendo después la compra de licencia.

Por el contrario, StarUML es *open-source* y posee la misma, o incluso una mayor potencia que el descrito anteriormente. Todos los diagramas UML de esta memoria fueron diseñados con este programa, como por ejemplo los casos de uso, la arquitectura del sistema o el diagrama de clases, entre otros.

2 Estudio de viabilidad

Identificación del Alcance del Sistema

El sistema a desarrollar consta de un ciclo de vida completo en el mundo de la ingeniería del software.

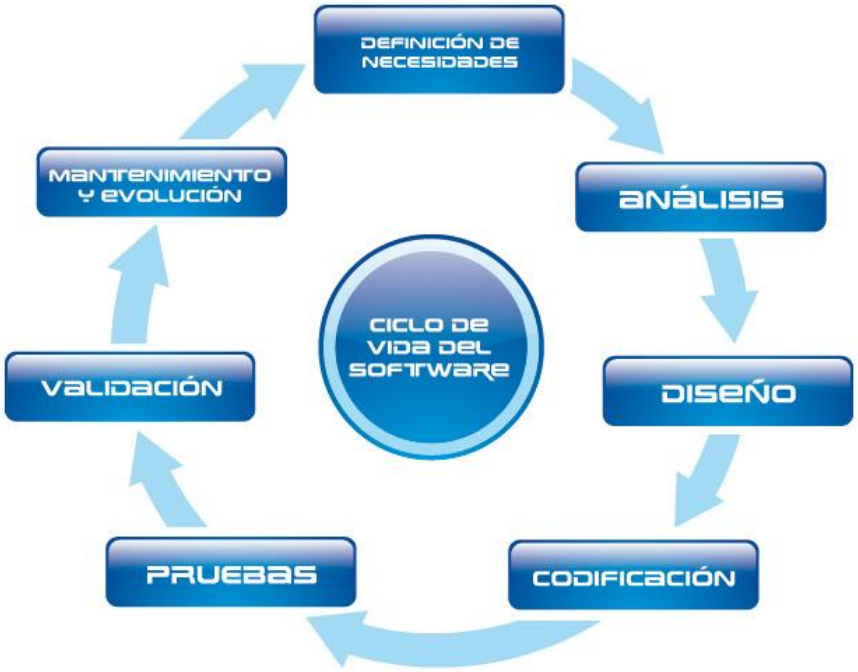

Ilustración 8: Ciclo de vida de un proceso de desarrollo software [6]

En primer lugar se ha realizado un estudio previo de las necesidades del cliente, en el que se estudia, de forma teórica, cuál es el problema que hay que afrontar y cómo se pretende solucionar, o dicho con otras palabras, **dónde estamos y a dónde queremos llegar**. Además, se ha realizado un estudio de mercado, de forma que se dan a conocer cuáles son las posibles soluciones actuales aplicables al problema propuesto, resaltando las **fortalezas y características que diferencian nuestra solución** con respecto al resto.

Por otra parte se ha realizado una **planificación** del trabajo a desarrollar durante la duración de este trabajo de fin de grado. En ella se contemplan todas las tareas que se deben llevar a cabo para lograr que sea un proyecto exitoso, incluyendo aspectos de secuenciación y esfuerzo estimado para llevarlas a cabo. Además se complementa dicha planificación con un **seguimiento** del proyecto, con el fin de facilitar la obtención de conclusiones finales tras el fin del trabajo y detectar puntos de mejora de cara al futuro.

A continuación se ha elaborado un estudio en el cuál se comparan las distintas **alternativas de diseño** que se consideran aplicables al problema propuesto. Se exponen las ventajas e inconvenientes de cada alternativa, para finalmente elegir de forma justificada una solución final.

Una vez se conoce cuál es el objetivo y cómo se ha de llegar a él, se ha realizado un **análisis** y **diseño** del sistema. Hasta ahora se habían llevado a cabo estudios previos del problema y las posibles soluciones, mientras que a en esta fase se ha realizado un estudio exhaustivo del sistema completo a desarrollar, justificando así la alternativa de diseño elegida durante la fase anterior. Además, el uso de metodologías ágiles de desarrollo durante este proyecto implica que se trabaje en paralelo en la **codificación** y **pruebas** del sistema, de forma que se crean procesos iterativos en los que una necesidad de codificación puede suponer un cambio en el diseño de un componente sin ningún problema.

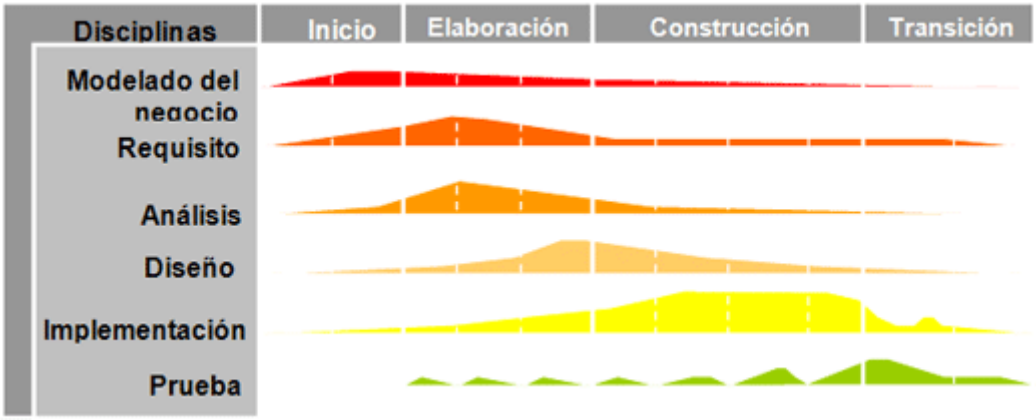

Ilustración 9: Ejemplo de dedicación de esfuerzos [7] a las distintas fases de un proyecto utilizando una metodología de desarrollo ágil

Estudio de alternativas de solución

Tras el estudio previo de la solicitud y los requisitos del sistema expuestos en el capítulo de *Análisis*, se estudian y proponen distintas alternativas de solución con el fin de seleccionar la solución óptima con respecto a las necesidades planteadas por el usuario.

Las distintas alternativas son planteadas desde un punto de vista tecnológico, enfocadas a gestionar la arquitectura de la aplicación. Se tendrán en cuenta los siguientes aspectos:

- **Sistema operativo**: capa intermedia entre el desarrollador y las aplicaciones de desarrollo. El objetivo principal es optimizar los recursos para lograr un desarrollo eficiente.
    - GNU/Linux RedHat: distribución Linux orientada al mundo empresarial.
    - Mac OS Lion 10.7: última versión del sistema operativo creado por Apple.

  - Microsoft Windows 7 Ultimate: última versión del sistema operativo creado por Microsoft.

- **Lenguaje de programación**: lenguaje utilizado durante el desarrollo de la aplicación. Se plantean los lenguajes más utilizados y/o que cuentan con un mayor crecimiento hoy día [8].
  - Java
  - .NET Framework
  - C
  - C++
- **Entorno de programación**: *framework* utilizado junto con el lenguaje de programación seleccionado.
  - Eclipse: entorno de desarrollo *open-source*.
  - Microsoft Visual Studio 2010 Professional: entorno de desarrollo de Microsoft.
- **Sistema gestor de bases de datos:** constituye la capa de control, administración y gestión de la base de datos accedida por la aplicación.
  - Access: sistema de Microsoft que permite una simple gestión de bases de datos relacionales.
  - MySQL: software libre que permite la gestión de bases de datos relacionales.
  - Microsoft SQL Server: sistema de Microsoft que permite la gestión completa de bases de datos relacionales.

Con el fin de obtener la mejor alternativa final, se realiza en primer lugar una **breve exposición** de las características de cada una, para posteriormente llevar a cabo una **valoración más detallada** de sus pros y contras.

2.1.1 Exposición de las alternativas

Alternativa 1	
Sistema operativo	GNU/Linux RedHat
Lenguaje de programación	Java
Entorno de programación	Eclipse
Sistema gestor de bases de datos	MySQL

Tabla 2: Alternativa de viabilidad 1

Todo lo que se plantea en esta alternativa es *open-source* o de código abierto, lo que supone que en un principio sea la alternativa **más económica**.

Alternativa 2	
Sistema operativo	Mac OS Lion 10.7
Lenguaje de programación	Java
Entorno de programación	Eclipse
Sistema gestor de bases de datos	MySQL

Tabla 3: Alternativa de viabilidad 2

Esta alternativa está basada en el uso de *MacOS* como sistema operativo. Se trata de un sistema que funciona exclusivamente sobre un computador *Macintosh*, por lo que a priori se trata de la alternativa **menos económica** ya que existe la necesidad de adquirir nuevas máquinas.

Alternativa 3	
Sistema operativo	Microsoft Windows 7
Lenguaje de programación	.NET
Entorno de programación	Visual Studio
Sistema gestor de bases de datos	Microsoft SQL Server

Tabla 4: Alternativa de viabilidad 3

A priori se trata de una alternativa poco atractiva, ya que está basada en el uso de software bajo licencia de *Microsoft*. Pero al tratarse de un proyecto académico y al formar parte aún de la Universidad Carlos III de Madrid, está disponible el programa **MSDN Academic Alliance**, que permite el uso gratuito de licencias de *Microsoft*.

Alternativa 4	
Sistema operativo	Microsoft Windows 7
Lenguaje de programación	Java
Entorno de programación	Eclipse
Sistema gestor de bases de datos	MySQL

Tabla 5: Alternativa de viabilidad 4

Se trata de una alternativa bastante similar a la *Alternativa 3*, con la diferencia de que se utiliza *Java* como lenguaje de programación. El hecho de cuál esté por encima de la otra puede depender del grado de experiencia del equipo de desarrollo con cada plataforma, que será analizado en la valoración posterior de las alternativas.

Alternativa 5	
Sistema operativo	Microsoft Windows 7
Lenguaje de programación	C
Entorno de programación	Eclipse
Sistema gestor de bases de datos	MySQL

Tabla 6: Alternativa de viabilidad 5

Al igual que en las dos alternativas anteriores, se propone *C* como lenguaje de programación, posiblemente menos modularizable y orientado a objetos que los ya comentados previamente.

Alternativa 6	
Sistema operativo	Microsoft Windows 7
Lenguaje de programación	C++
Entorno de programación	Eclipse
Sistema gestor de bases de datos	Access

Tabla 7: Alternativa de viabilidad 6

En este caso se propone la utilización del lenguaje C++ para el desarrollo del sistema de información. Además, se utiliza *Access* como gestor de bases de datos, lo que puede suponer diferencias con respecto a *MySQL* y *SQL Server*, los gestores más utilizados a nivel comercial.

Alternativa 7	
Sistema operativo	GNU/Linux Red Hat
Lenguaje de programación	C
Entorno de programación	Eclipse
Sistema gestor de bases de datos	MySQL

Tabla 8: Alternativa de viabilidad 7

Para esta alternativa se propone el uso del sistema operativo de código abierto *Red Hat* y *C* como lenguaje de programación a utilizar.

2.1.2 Valoración de las alternativas

Para obtener una **valoración numérica** sobre 100 puntos totales, se considera que el sistema operativo y el lenguaje de programación utilizado son las decisiones más importantes, por lo que cada una tiene un valor del 35% sobre el total. Tanto el entorno de desarrollo como el gestor de base de datos puntúan 15% cada uno.

Se tienen en cuenta los valores de 'Experiencia' (experiencia del equipo de desarrollo durante su carrera) y 'Rendimiento' (en qué medida se cree que se puede exprimir el componente para obtener su máximo potencial). Cada uno de ellos tiene tres posibles valores: **alto** (3)**, medio** (2) y **bajo** (1).

Por ejemplo, si el componente C1, equivalente al 35% del total de la valoración numérica, presenta experiencia 'Media (2)' y rendimiento 'Alto (3)' (5 puntos), dicho componente suma (35*5)/6, es decir 29.1 puntos de 35 posibles para la valoración final.

Además, en el caso de que una solución suponga coste monetario alguno, se restará al final de la evaluación un 5% por cada 1000 euros. Así, si una alternativa tiene un coste de 4000 euros, se le resta un 20% al total de la valoración.

Alternativa 1				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Linux Red Hat	Libre	0 €	Baja	Medio
Java	Libre	0 €	Media	Alto
Eclipse	Libre	0 €	Media	Alto
MySQL	Libre	0 €	Media	Alto
Valoración total	71.7 / 100			

Tabla 9: Valoración de la alternativa 1

Valoración de riesgos: frente al nulo coste que supone, el principal inconveniente de esta alternativa es la falta de experiencia con respecto a todos los componentes utilizados. Cabe destacar la baja experiencia con respecto al sistema operativo empleado (GNU/Linux Red Hat), lo que puede suponer un incremento de problemas durante cualquier fase del proyecto.

Alternativa 2				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Mac OS Lion	Propietaria	4300 €	Baja	Alto
Java	Libre	0 €	Media	Alto
Eclipse	Libre	0 €	Media	Alto
MySQL	Libre	0 €	Media	Alto
Valoración total	62 / 100			

Tabla 10: Valoración de la alternativa 2

Valoración de riesgos: es posible exprimir al máximo el sistema operativo empleado en esta alternativa, pero hay que destacar la **falta de experiencia**

con él. Además de ello, el **elevado coste** que supone es otro principal inconveniente.

Obtiene una valoración inicial de 77.5 puntos, que tras aplicarle las restricciones de coste comentadas previamente (-20%), se queda en 62 puntos.

Alternativa 3				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Microsoft Windows 7	Propietaria	0 €	Alta	Alto
.NET	Propietaria	0 €	Alta	Alto
Visual Studio	Propietaria	0 €	Alta	Alto
SQL Server	Propietaria	0 €	Alta	Alto
Valoración total	100 / 100			

Tabla 11: Valoración de la alternativa 3

Valoración de riesgos: aunque tanto el sistema operativo como el resto de módulos utilizados tienen licencia de pago de *Microsoft*, recordamos que al tratarse de un proyecto académico se puede utilizar sin ningún problema el programa *MSDN Academic Alliance* de *Microsoft*, lo que reduce el coste a 0. Además, tanto la experiencia utilizando dichos módulos como el rendimiento que se puede sacar de ellos es muy alto. Por lo tanto esta alternativa es buena en cuanto a costes, experiencia y rendimiento esperado de los módulos utilizados.

Alternativa 4				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Microsoft Windows 7	Propietaria	0 €	Alta	Alto
Java	Libre	0 €	Media	Alto
Eclipse	Libre	0 €	Media	Alto
MySQL	Libre	0 €	Media	Alto
Valoración total	89.1 / 100			

Tabla 12: Valoración de la alternativa 4

Valoración de riesgos: es muy similar a la *Alternativa 3*, pero el hecho de cambiar el lenguaje de programación utilizado supone una **falta de experiencia**, a pesar de que el rendimiento que se puede conseguir es muy alto.

Alternativa 5				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Microsoft Windows 7	Propietaria	0 €	Alta	Alto
C	Libre	0 €	Media	Bajo
Eclipse	Libre	0 €	Media	Alto
MySQL	Libre	0 €	Media	Alto
Valoración total	77.5 / 100			

Tabla 13: Valoración de la alternativa 5

Valoración de riesgos: el hecho de utilizar C como lenguaje de programación supone un gran riesgo para el éxito del proyecto, ya que se piensa que su **potencia** no es suficiente para desarrollar el sistema de información requerido por el cliente. Además, también se suman los problemas de **falta de experiencia** con este lenguaje.

Alternativa 6				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Microsoft Windows 7	Propietaria	0 €	Alta	Alto
C++	Libre	0 €	Baja	Medio
Eclipse	Libre	0 €	Media	Alto
Access	Libre	0 €	Baja	Bajo
Valoración total	70 / 100			

Tabla 14: Valoración de la alternativa 6

Valoración de riesgos: al igual que en la *Alternativa 6,* el hecho de utilizar C++ como lenguaje de programación supone un gran riesgo para el éxito del proyecto, debido a la **falta de experiencia** en el desarrollo de aplicaciones con este lenguaje. Además, se plantea el uso de una base de datos *Access*, la cual presenta **menor rendimiento** y fiabilidad que *MySQL* o *SQL Server* una vez el sistema ha sido puesto en producción.

Alternativa 7				
Componente	**Licencia**	**Coste**	**Experiencia**	**Rendimiento**
Linux Red Hat	Libre	0 €	Baja	Medio
C	Libre	0 €	Media	Bajo
Eclipse	Libre	0 €	Media	Alto
MySQL	Libre	0 €	Media	Alto
Valoración total	60 / 100			

Tabla 15: Valoración de la alternativa 7

Valoración de riesgos: los principales riesgos de esta alternativa son la falta de conocimiento del sistema operativo empleado (GNU/Linux Red Hat), y la falta de experiencia y rendimiento esperado del lenguaje de programación propuesto, lo que implica una gran probabilidad de que aparezcan problemas en cualquier fase de desarrollo del producto.

2.1.3 Selección de la solución

Solución	Valoración
Alternativa 1	71.7 / 100
Alternativa 2	62 / 100
Alternativa 3	**100 / 100**
Alternativa 4	89.1 / 100
Alternativa 5	77.5 / 100
Alternativa 6	70 / 100
Alternativa 7	60 / 100

Tabla 16: Comparativa de alternativas de solución

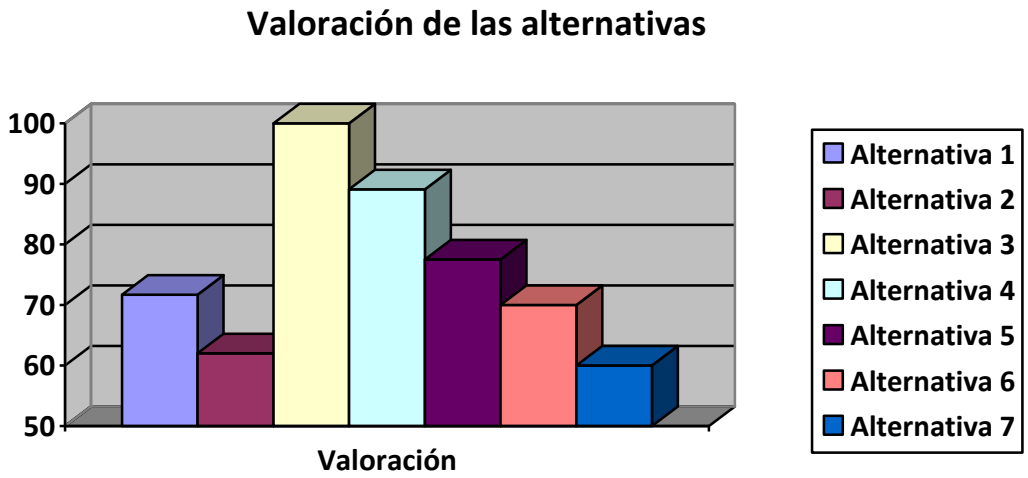

Ilustración 10: Valoración ponderada de las alternativas de solución

Finalmente, tras un detallado análisis de todas las alternativas de diseño, cabe destacar que la **Alternativa 3** es la solución que más se adapta a las prioridades del cliente. Las propiedades de esta alternativa que destacan sobre el resto son:

- Coste monetario nulo.
- Alta experiencia en las herramientas de trabajo a utilizar más importantes.
- Alto potencial de dichas herramientas para obtener un máximo rendimiento con respecto a las necesidades planteadas por el cliente.

A pesar de su elección, se muestran los costes ocultos de esta alternativa en caso de no formar parte de la Universidad Carlos III de Madrid y aprovechar el programa *MSDN Academic Alliance* de Microsoft.

Concepto	Coste	Cantidad	Coste total
Licencias de Microsoft Windows 7 Ultimate [9]	319 €	1	319 €
Licencias de Microsoft Visual Studio 2010 Professional [10]	415 €	1	415 €
			734 €

Tabla 17: Costes ocultos de la alternativa elegida

3 Análisis

Introducción

El objetivo de este capítulo es obtener una especificación a alto nivel del sistema que se va a construir. No define cómo solucionar el problema que se plantea por parte del cliente, sino que se capturan las necesidades que se deben resolver y modelar durante una posterior fase de diseño.

Definición del Sistema

3.1.1 Determinación del Alcance del Sistema

El módulo a desarrollar formará parte de una aplicación de gestión del conocimiento denominada *knowledgeManager*. El objetivo final de esta herramienta es aportar la funcionalidad necesaria para que un usuario final sea capaz de gestionar y controlar un sistema de organización del conocimiento (*KOS, Knowledge Organization System*).

Más concretamente, el módulo estará basado en la gestión de un **vocabulario controlado**, que servirá de base para el resto de la aplicación. Se define un vocabulario controlado como un conjunto de términos que pueden mantener entre sí relaciones de distintos tipos, cuyo objetivo final radica en convertir el lenguaje natural de un documento cualquiera en un lenguaje controlado, con el fin de ser utilizado en posteriores procesos de *indexing* y *retrieval*.

Además, el módulo también está apoyado sobre una parte de gestión de la **light ontology**, que gestiona todo lo relacionado con el significado sintáctico y semántico que se le puede asignar a posteriori a un término o a una relación entre términos.

Este módulo debe servir al **experto del dominio** para gestionar y mantener un *KOS* totalmente controlado. El resultado será el punto de partida para que los analistas y arquitectos software de la compañía sean capaces de definir especificaciones correctas de requisitos de una elevada calidad de un determinado dominio.

3.1.2 Identificación del Entorno Tecnológico

El sistema a desarrollar consiste en un módulo de *knowledgeManager*, una aplicación de mayor tamaño. Por tanto, el entorno tecnológico mínimo en que debe funcionar en una máquina cliente es el mismo que dicha aplicación:

- Microsoft .NET Framework 4.0 instalado.
- Procesador a 1.5GHz.
- 512 MB de memoria RAM.
- 64 GB de disco duro.
- Sistema operativo: Windows XP, Windows Vista, Windows 7 (cualquier revisión de todos ellos).

3.1.3 Especificación de Estándares y Normas

3.1.3.1 Restricciones Generales

- Todas las operaciones de acceso a datos están abstraídas mediante una DAL. Por lo tanto todas las clases de la aplicación que requieran acceso a la base de datos deben tener definido un objeto "engine" de tipo `Cake.Engine.CAKEEngine,` contenido en la librería Cake.Engine.dll.
- Todas las operaciones de indexación y normalización están abstraídas mediante un componente externo. Por lo tanto todas las clases de la aplicación que lo requieran, deben tener definido un objeto "indexer" de tipo `Cake.Indexer.Indexer,` contenido en la librería Cake.Indexer.dll.
- Todos los controles de usuario de la capa de presentación deben heredar de un control de usuario común para toda la aplicación, con el fin de maximizar la reutilización de código, de tipo `KnowledgeManagerBaseUserControl`. Este asunto de análisis puede ser ignorado, ya que se incluye en las librerías de la aplicación.
- Todas las clases de negocio de la BLL deben heredar de una clase común de tipo `RepositoryBase`.
- Los comentarios de descripción de clases y métodos deben estar correctamente formados para generar documentación XML.
    - En el caso de las clases, es suficiente indicar un pequeño resumen de la función de la clase.
    - En el caso de los métodos, es necesario añadir el resumen de responsabilidades del método, fecha y autor. Además se puede añadir información sobre los parámetros de entrada y el valor de retorno, si se considera oportuno.

3.1.3.2 Supuestos y Dependencias

El módulo desarrollado no supone el *knowledgeManager* completo. Por lo tanto se utiliza el mismo estilo gráfico de la capa de presentación que en el resto de módulos de la aplicación, con el fin de mantener la homogeneidad de todos sus componentes de cara a la versión final de la aplicación.

3.1.3.3 Entorno Operacional

Los equipos de desarrollo utilizados para la elaboración del código de la aplicación tendrán una configuración homogénea:

- Sistema operativo Windows 7 Ultimate 64-bit
- Visual Studio 2010 Professional
- .NET Framework 4.0
- Microsoft Office 2010
- Tortoise SVN 1.7.7 64-bit

3.1.4 Identificación de los Usuarios Participantes y Finales

En este apartado se distinguen dos tipos de usuarios del sistema. Por un lado los *stakeholders*, todo aquel interesado en la construcción del sistema de información. Por otro lado, los usuarios finales que van a utilizar la aplicación.

- **Stakeholders**
  - *The Reuse Company* [15]: interesado en el desarrollo del proyecto para incluir el módulo en su software *knowledgeManager*.
  - Universidad Carlos III de Madrid [16]: entidad donde se presenta el módulo como TFG y, además, lugar donde se lleva a cabo el trabajo mediante una beca.
- **Usuarios finales**
  - Experto del dominio: persona encargada de gestionar la ontología de un determinado dominio en una compañía de desarrollo software.
  - Arquitectos y analistas software: utilizarán otras herramientas cuyas entradas serán las salidas de este módulo, y viceversa.
  - Jefe de proyecto: tomará decisiones sobre el uso del módulo dentro de la compañía.
  - Responsable de calidad: tomará decisiones en herramientas externas que pueden afectar al funcionamiento del módulo.

3.1.5 Estudio de la Seguridad Requerida en el Proceso de Análisis del Sistema de Información

Aunque el desarrollo del módulo de *knowledgeManager* se desarrolla como TFG para la Universidad Carlos III de Madrid, es condición indispensable que tanto el código desarrollado, como las librerías de *Cake* utilizadas y la documentación generada son confidenciales y propiedad de la empresa *The Reuse Company*.

Establecimiento de Requisitos

Se plasman todos los requisitos capturados para el proyecto. Se utiliza un formato estándar para mostrar todos y cada uno de los requisitos que se exponen en este apartado.

RSX - YYY: Nombre del requisito			
Autor:	Autor del requisito	**Tipo:**	Tipo del requisito
Fuente:	Fuente de la que procede	**Fecha:**	dd/mm/aaaa
Prioridad:	A/M/B	**Complejidad:**	A/M/B
Necesidad:	A/M/B	**Estado**:	Propuesto
Objetivo:	Una línea con el objetivo principal del requisito		
Precondiciones:	Una línea en caso de que haya alguna precondición		
Descripción:	Descripción clara y concisa del requisito		

Tabla 18: Plantilla de requisitos

La nomenclatura de los requisitos se hará de la siguiente manera: RSX-YYY, donde YYY serán tres dígitos auto numéricos e identificativos de cada requisito, y X se corresponderá con uno o dos caracteres que especificarán el tipo de requisito:

- **F**: Requisito funcional
- **R**: Requisito de rendimiento
- **In**: Requisito de interfaz
- **S**: Requisito de seguridad
- **Ca**: Requisito de calidad

3.1.6 Obtención de Requisitos

Siguiendo la plantilla definida en el apartado anterior, se exponen los requisitos capturados para el proyecto.

RSF - 001: Datos de carga inicial					
Autor:	Borja López Gómez		**Tipo:**	Funcional	
Fuente:	The Reuse Company		**Fecha:**	09/04/2012	
Prioridad:	Alta	**Necesidad:**	Alta	**Complejidad:**	Baja

Estabilidad:		**Verificable:**	Sí	**Estado**:	Propuesto
Objetivo:	Sentar las bases para poder ejecutar la aplicación por vez primera.				
Precondiciones:	Debe existir la base de datos con la que se vaya a hacer la conexión.				
Descripción:	La base de datos a utilizar debe contener en todo momento al menos dos etiquetas sintácticas (NOUN y VERB) y un lenguaje cualquiera.				
Trazabilidad:					

Tabla 19: RSF-001

RSF - 002: Visualización de términos			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	09/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	El usuario visualice en pantalla los términos de la ontología.		
Precondiciones:			
Descripción:	El sistema debe ser capaz de mostrar todos los términos existentes en la ontología. Por cada término se debe mostrar su identificador, nombre, etiqueta sintáctica, semántica, idioma, si/no pertenece al dominio.		
Trazabilidad:	RSF-003, RSF-008, RSF-041, RSF-042, RSF-043, RSF-044, RSF-045, RSF-046, RSF-047, RSF-048		

Tabla 20: RSF-002

RSF - 003: Acciones sobre términos			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	09/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	**Alta**	**Estado**:	Propuesto
Objetivo:	Ofrecer al usuario distintas opciones sobre la vista de términos.		
Precondiciones:			
Descripción:	En la vista de términos, el sistema debe ser capaz de mostrar al usuario las distintas opciones disponibles sobre términos, mediante un menú contextual que se abre con el botón derecho: • Crear un término. • Modificar un término. • Duplicar un término. • Eliminar un término. • Refrescar vista.		
Trazabilidad:	RSF-002, RSF-004, RSF-005, RSF-006, RSF-007, RSF-041, RSF-042, RSF-043, RSF-044, RSF-045		

Tabla 21: RSF-003

RSF - 004: Creación de un término			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	09/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto

Objetivo:	Crear un nuevo término en la ontología.
Precondiciones:	
Descripción:	El usuario debe ser capaz de crear un nuevo término en la ontología, a partir de los datos de carga iniciales. Los datos a rellenar por el usuario son: Nombre, etiqueta sintáctica, semántica, lenguaje, impacto en el dominio, impacto en el lenguaje, si/no pertenece al dominio.
Trazabilidad:	RSF-003, RSF-005, RSF-006, RSF-007

Tabla 22: RSF-004

RSF - 005: Modificación de un término.			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	10/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Modificar la información de un término existente en la ontología.		
Precondiciones:	Debe existir al menos un término en la ontología.		
Descripción:	El usuario debe ser capaz de modificar la información de un término ya existente en la ontología. . Los datos modificables por el usuario son: Nombre, etiqueta sintáctica, semántica, lenguaje, impacto en el dominio, impacto en el lenguaje, si/no pertenece al dominio.		
Trazabilidad:	RSF-003, RSF-004, RSF-006, RSF-007		

Tabla 23: RSF-005

RSF - 006: Duplicado de un término			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	10/04/2012
Prioridad:	Media	**Complejidad:**	Alta
Necesidad:	Baja	**Estado**:	Propuesto
Objetivo:	Duplicar la información de un término existente en la ontología.		
Precondiciones:	Debe existir al menos un término en la ontología.		
Descripción:	El usuario debe ser capaz de duplicar un término existente en la ontología.		
Trazabilidad:	RSF-003, RSF-004, RSF-005, RSF-007		

Tabla 24: RSF-006

RSF - 007: Borrado de un término			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	10/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Borrar un término existente en la ontología.		
Precondiciones:	Debe existir al menos un término en la ontología.		
Descripción:	El usuario debe ser capaz de eliminar un término existente en la ontología. Previamente, se debe mostrar un mensaje de confirmación del usuario: **¿Estás seguro de que quieres borrar el término?** Por último, se debe mostrar un mensaje indicando si el proceso ha concluido satisfactoriamente o si, por el contrario, ha habido algún error.		

Trazabilidad:	RSF-003, RSF-004, RSF-005, RSF-006

Tabla 25: RSF-007

RSF - 008: Búsqueda de un término			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	11/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Agilizar el proceso de búsqueda de un término mediante filtros.		
Precondiciones:			
Descripción:	El usuario debe ser capaz de buscar un término utilizando distintos filtros de búsqueda, mostrándose los resultados en la misma vista de términos. Se debe poder filtrar por Nombre, etiqueta sintáctica, semántica, idioma, si/no está ene l dominio. En el caso de existir más de un criterio, la búsqueda debe ser multicampo.		
Trazabilidad:	RSF-002		

Tabla 26: RSF-008

RSF - 009: Visualización de etiquetas sintácticas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	El usuario visualice en pantalla las etiquetas sintácticas de la ontología.		

Precondiciones:	
Descripción:	El sistema debe ser capaz de mostrar todas las etiquetas sintácticas existentes en la ontología. Por cada etiqueta sintáctica debe mostrarse el Nombre, las etiquetas sintácticas hijas y los términos asociados a dicha etiqueta sintáctica.
Trazabilidad:	RSF-010, RSF-015, RSF-016, RSF-017, RSF-018, RSF-046, RSF-047, RSF-048

Tabla 27: RSF-009

RSF - 010: Acciones sobre etiquetas sintácticas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Ofrecer al usuario distintas opciones sobre la vista de etiquetas sintácticas.		
Precondiciones:			
Descripción:	En la vista de etiquetas sintácticas, el sistema debe ser capaz de mostrar al usuario las distintas opciones disponibles sobre etiquetas sintácticas, mediante un menú contextual que se abre con el botón derecho: • Crear una etiqueta sintáctica. • Modificar una etiqueta sintáctica. • Duplicar una etiqueta sintáctica. • Eliminar una etiqueta sintáctica. • Refrescar vista.		
Trazabilidad:	RSF-009, RSF-011, RSF-012, RSF-013, RSF-014		

Tabla 28: RSF-010

RSF - 011: Creación de una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Crear una nueva etiqueta sintáctica en la ontología.		
Precondiciones:			
Descripción:	El usuario debe ser capaz de crear una nueva etiqueta sintáctica en la ontología, a partir de los datos de carga iniciales. Los campos a rellenar por el usuario son: Nombre, etiqueta sintáctica padre y atributos de relevancia de la etiqueta sintáctica respecto al dominio.		
Trazabilidad:	RSF-010, RSF-012, RSF-013, RSF-014		

Tabla 29: RSF-011

RSF - 012: Modificación de una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Modificar la información de una etiqueta sintáctica existente en la ontología.		
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología.		
Descripción:	El usuario debe ser capaz de modificar la información de una etiqueta sintáctica ya existente en la ontología. Los campos modificables por el usuario son: Nombre, etiqueta		

	sintáctica padre y atributos de relevancia de la etiqueta sintáctica respecto al dominio.
Trazabilidad:	RSF-010, RSF-011, RSF-013, RSF-014

Tabla 30: RSF-012

RSF - 013: Duplicado de una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Media	**Complejidad:**	Alta
Necesidad:	Baja	**Estado**:	Propuesto
Objetivo:	Duplicar la información de una etiqueta sintáctica existente en la ontología.		
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología.		
Descripción:	El usuario debe ser capaz de duplicar una etiqueta sintáctica existente en la ontología.		
Trazabilidad:	RSF-010, RSF-011, RSF-012, RSF-014		

Tabla 31: RSF-013

RSF - 014: Borrado de una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Borrar una etiqueta sintáctica de la ontología.		

Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología.
Descripción:	El usuario debe ser capaz de eliminar una etiqueta sintáctica existente en la ontología. Previamente, se debe mostrar un mensaje de confirmación del usuario: **¿Estás seguro de que quieres borrar la etiqueta sintáctica?** Por último, se debe mostrar un mensaje indicando si el proceso ha concluido satisfactoriamente o si, por el contrario, ha habido algún error.
Trazabilidad:	RSF-010, RSF-011, RSF-012, RSF-013

Tabla 32: RSF-014

RSF - 015: Búsqueda de una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	12/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Agilizar el proceso de búsqueda de una etiqueta sintáctica mediante filtros.		
Precondiciones:			
Descripción:	El usuario debe ser capaz de buscar una etiqueta sintáctica utilizando distintos filtros de búsqueda, mostrándose los resultados en la misma vista de etiquetas sintácticas. Se debe poder filtrar por Nombre, sólo etiquetas sintácticas raíces, sólo etiquetas sintácticas de tipo nombre, sólo etiquetas sintácticas de tipo verbo.		
Trazabilidad:	RSF-009, RSF-016, RSF-017, RSF-018		

Tabla 33: RSF-015

RSF - 016: Visualización de etiquetas sintácticas hijas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	13/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Conocer en todo momento si una etiqueta sintáctica tiene etiquetas sintácticas hijas.		
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología.		
Descripción:	El sistema debe ser capaz de mostrar todas las etiquetas sintácticas hijas de la etiqueta sintáctica seleccionada, si tiene. En cada etiqueta sintáctica hija se debe mostrar su nombre.		
Trazabilidad:	RSF-009, RSF-015, RSF-017, RSF-018, RSF-046, RSF-047, RSF-048		

Tabla 34: RSF-016

RSF - 017: Visualización de términos asociados a una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	13/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Conocer en todo momento si hay términos asociados a una etiqueta sintáctica.		
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología.		
Descripción:	El sistema debe ser capaz de mostrar todos los términos		

	que están asociados a la etiqueta sintáctica seleccionada, si tiene alguno. Por cada término se debe mostrar su Nombre, su semántica y su etiqueta sintáctica.
Trazabilidad:	RSF-009, RSF-015, RSF-016, RSF-018, RSF-019, RSF-020, RSF-021, RSF-046, RSF-047, RSF-048

Tabla 35: RSF-017

RSF - 018: Visualización de términos asociados a una etiqueta sintáctica o a sus hijas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	13/04/2012
Prioridad:	Alta	**Complejidad:**	Media
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Conocer en todo momento si hay términos asociados a una etiqueta sintáctica o a cualquiera de sus etiquetas sintácticas hijas.		
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología.		
Descripción:	Además de mostrar todos los términos que están asociados a la etiqueta sintáctica seleccionada, el usuario debe tener la opción de decidir si quiere mostrar también, sobre la misma vista, los términos asociados a cualquier etiqueta sintáctica hija de la etiqueta sintáctica seleccionada. Por cada término se debe mostrar su Nombre, su semántica y su etiqueta sintáctica.		
Trazabilidad:	RSF-009, RSF-015, RSF-016, RSF-017, RSF-019, RSF-020, RSF-021, RSF-046, RSF-047, RSF-048		

Tabla 36: RSF-018

RSF - 019: Acciones sobre términos de una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	13/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Ofrecer al usuario distintas opciones sobre términos en la vista de etiquetas sintácticas.		
Precondiciones:	Debe existir una etiqueta sintáctica seleccionada		
Descripción:	En la vista de etiquetas sintácticas, el sistema debe ser capaz de mostrar al usuario las distintas opciones disponibles sobre etiquetas sintácticas, mediante un menú contextual que se abre con el botón derecho: • Vincular un término a la etiqueta sintáctica. • Mover términos de otra etiqueta sintáctica.		
Trazabilidad:	RSF-017, RSF-018, RSF-020, RSF-021		

Tabla 37: RSF-019

RSF - 020: Vincular un término a una etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	13/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Facilitar que el usuario pueda vincular un término a una etiqueta sintáctica.		
Precondiciones:	Debe existir una etiqueta sintáctica seleccionada.		
Descripción:	Desde la vista de términos de una etiqueta sintáctica, el		

	usuario debe ser capaz de vincular un término a la etiqueta sintáctica seleccionada.
Trazabilidad:	RSF-017, RSF-018, RSF-019

Tabla 38: RSF-020

RSF - 021: Mover términos de otra etiqueta sintáctica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	13/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Facilitar que el usuario pueda mover términos entre etiquetas sintácticas.		
Precondiciones:	Debe existir al menos una etiqueta sintáctica seleccionada.		
Descripción:	Desde la vista de términos de una etiqueta sintáctica, el usuario debe ser capaz de mover todos los términos de una determinada etiqueta sintáctica a la etiqueta sintáctica seleccionada.		
Trazabilidad:	RSF-017, RSF-018, RSF-019		

Tabla 39 RSF-021

RSF - 022: Visualización de semánticas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	16/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	El usuario visualice en pantalla las semánticas de la		

	ontología.
Precondiciones:	
Descripción:	El sistema debe ser capaz de mostrar todas las semánticas existentes en la ontología. Por cada semántica debe mostrarse el Nombre, las semánticas hijas y los términos asociados a dicha semántica.
Trazabilidad:	RSF-023, RSF-028, RSF-029, RSF-030, RSF-031, RSF-046, RSF-047, RSF-048

Tabla 40 RSF-022

RSF - 023: Acciones sobre semánticas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	16/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Ofrecer al usuario distintas opciones sobre la vista de semánticas.		
Precondiciones:			
Descripción:	En la vista de semánticas, el sistema debe ser capaz de mostrar al usuario las distintas opciones disponibles sobre semánticas, mediante un menú contextual que se abre con el botón derecho: • Crear una semántica. • Modificar una semántica. • Duplicar una semántica. • Eliminar una semántica. • Refrescar la vista.		
Trazabilidad:	RSF-024, RSF-025, RSF-026, RSF-027		

Tabla 41 RSF-023

RSF - 024: Creación de una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	16/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Crear una nueva semántica en la ontología.		
Precondiciones:			
Descripción:	El usuario debe ser capaz de crear una nueva semántica, a partir de los datos de carga iniciales. Los campos a rellenar por el usuario son: Nombre, semántica padre, ponderación y atributos que determinan las propiedades de la relación.		
Trazabilidad:	RSF-023, RSF-025, RSF-026, RSF-027		

Tabla 42 RSF-024

RSF - 025: Modificación de una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	16/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Modificar la información de una semántica existente en la ontología.		
Precondiciones:	Debe existir al menos una semántica en la ontología.		
Descripción:	El usuario debe ser capaz de modificar la información de una semántica ya existente en la ontología. Los campos modificables por el usuario son: Nombre, semántica padre, ponderación y atributos que determinan las propiedades de la relación		

Trazabilidad:	RSF-023, RSF-024, RSF-026, RSF-027

Tabla 43 RSF-025

RSF - 026: Duplicado de una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Media	**Complejidad:**	Alta
Necesidad:	Baja	**Estado**:	Propuesto
Objetivo:	Duplicar la información de una semántica existente en la ontología.		
Precondiciones:	Debe existir al menos una semántica en la ontología.		
Descripción:	El usuario debe ser capaz de duplicar semántica existente en la ontología.		
Trazabilidad:	RSF-023, RSF-024, RSF-025, RSF-027		

Tabla 44 RSF-026

RSF - 027: Borrado de una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Borrar una semántica de la ontología.		
Precondiciones:	Debe existir al menos una semántica en la ontología.		
Descripción:	El usuario debe ser capaz de eliminar una semántica existente en la ontología. Previamente, se debe mostrar un		

	mensaje de confirmación del usuario: **¿Estás seguro de que quieres borrar la semántica?** Por último, se debe mostrar un mensaje indicando si el proceso ha concluido satisfactoriamente o si, por el contrario, ha habido algún error.
Trazabilidad:	RSF-023, RSF-024, RSF-025, RSF-026

Tabla 45 RSF-027

RSF - 028: Búsqueda de una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Agilizar el proceso de búsqueda de una semántica mediante filtros.		
Precondiciones:			
Descripción:	El usuario debe ser capaz de buscar una semántica utilizando distintos filtros de búsqueda, mostrándose los resultados en la misma vista de semánticas. Se debe poder filtrar por Nombre, si/no son mostradas en el tesauro, sólo semánticas raíces.		
Trazabilidad:	RSF-022, RSF-029, RSF-030, RSF-031		

RSF - 029: Visualización de semánticas hijas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Alta	**Complejidad:**	Baja

Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Conocer en todo momento si una semántica tiene semánticas hijas.		
Precondiciones:	Debe existir al menos una semántica en la ontología.		
Descripción:	El sistema debe ser capaz de mostrar todas las semánticas hijas de la semántica seleccionada, si tiene. En cada semántica hija se debe mostrar su nombre.		
Trazabilidad:	RSF-022, RSF-028, RSF-030, RSF-031, RSF-046, RSF-047, RSF-048		

Tabla 46: RSF-029

RSF - 030: Visualización de términos asociados a una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Conocer en todo momento si hay términos asociados a una semántica.		
Precondiciones:	Debe existir al menos una semántica en la ontología.		
Descripción:	El sistema debe ser capaz de mostrar todos los términos que están asociados a la semántica seleccionada, si tiene alguno. Por cada término se debe mostrar su Nombre, su semántica y su etiqueta sintáctica.		
Trazabilidad:	RSF-022, RSF-028, RSF-029, RSF-032, RSF-033, RSF-034, RSF-035, RSF-046, RSF-047, RSF-048		

Tabla 47: RSF-030

RSF - 031: Visualización de términos asociados a una semántica o a sus hijas			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Alta	**Complejidad:**	Media
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Conocer en todo momento si hay términos asociados a una semántica o a cualquiera de sus semánticas hijas.		
Precondiciones:	Debe existir al menos una semántica en la ontología.		
Descripción:	Además de mostrar todos los términos que están asociados a la semántica seleccionada, el usuario debe tener la opción de decidir si quiere mostrar también, sobre la misma vista, los términos asociados a cualquier semántica hija de la semántica seleccionada. Por cada término se debe mostrar su Nombre, su semántica y su etiqueta sintáctica.		
Trazabilidad:	RSF-022, RSF-028, RSF-029, RSF-032, RSF-033, RSF-034, RSF-035, RSF-046, RSF-047, RSF-048		

Tabla 48: RSF-031

RSF - 032: Acciones sobre términos de una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Ofrecer al usuario distintas opciones sobre términos en la vista de semánticas.		
Precondiciones:	Debe existir una semántica seleccionada		
Descripción:	En la vista de semánticas, el sistema debe ser capaz de		

	mostrar al usuario las distintas opciones disponibles sobre semánticas, mediante un menú contextual que se abre con el botón derecho: • Vincular un término a la semántica. • Mover términos de otra semántica. • Eliminar la semántica de un término.
Trazabilidad:	RSF-030, RSF-031, RSF-033, RSF-034, RSF-035

Tabla 49: RSF-032

RSF - 033: Vincular un término a una semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Facilitar que el usuario pueda vincular un término a una semántica.		
Precondiciones:	Debe existir una semántica seleccionada.		
Descripción:	Desde la vista de términos de una semántica, el usuario debe ser capaz de vincular un término a la semántica seleccionada.		
Trazabilidad:	RSF-030, RSF-031, RSF-032		

Tabla 50: RSF-033

RSF - 034: Mover términos de otra semántica			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Media	**Complejidad:**	Baja

Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Facilitar que el usuario pueda mover términos entre semánticas.		
Precondiciones:	Debe existir al menos una semántica seleccionada.		
Descripción:	Desde la vista de términos de una semántica, el usuario debe ser capaz de mover todos los términos de una determinada semántica a la semántica seleccionada.		
Trazabilidad:	RSF-030, RSF-031, RSF-032		

Tabla 51: RSF-034

RSF - 035: Eliminar la semántica de un término			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	17/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Facilitar que el usuario pueda eliminar la semántica de un término.		
Precondiciones:	Debe existir al menos una semántica seleccionada.		
Descripción:	Desde la vista de términos de una etiqueta sintáctica, el usuario debe ser capaz de eliminar la semántica de un término que actualmente está asociado a la semántica seleccionada.		
Trazabilidad:	RSF-030, RSF-031, RSF-032		

Tabla 52: RSF-035

RSF - 036: Visualización de sugerencias de términos			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	18/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	El usuario visualice en pantalla las sugerencias de términos existentes en el sistema.		
Precondiciones:			
Descripción:	El sistema debe ser capaz de mostrar todas las sugerencias existentes en la base de datos. Por cada sugerencia debe mostrarse el nombre del término, el autor de la sugerencia, la fecha de creación (dd/mm/aaaa hh:mm), la acción sugerida, la descripción, el estado, la respuesta, la fecha de respuesta (dd/mm/aaaa hh:mm) y si/no se exige notificación de respuesta.		
Trazabilidad:	RSF-037, RSF-038, RSF-039, RSF-40, RSF-046, RSF-047, RSF-048		

Tabla 53: RSF-036

RSF - 037: Visualización de sugerencias de relaciones entre términos			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	18/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	El usuario visualice en pantalla las sugerencias de relaciones existentes en el sistema.		
Precondiciones:			

Descripción:	El sistema debe ser capaz de mostrar todas las sugerencias de relaciones existentes en la base de datos. Por cada sugerencia debe mostrarse el nombre del término 1, el nombre de la relación, el nombre del término 2, el autor de la sugerencia, la fecha de creación (dd/mm/aaaa hh:mm), la acción sugerida, la descripción, el estado, la respuesta, la fecha de respuesta (dd/mm/aaaa hh:mm) y si/no se exige notificación de respuesta.
Trazabilidad:	RSF-036, RSF-038, RSF-039, RSF-040, RSF-046, RSF-047, RSF-048

Tabla 54: RSF-037

RSF - 038: Acciones sobre sugerencias			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	18/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Ofrecer al usuario distintas opciones sobre la vista de sugerencias.		
Precondiciones:			
Descripción:	En la vista de sugerencias (tanto de términos como de relaciones), el sistema debe ser capaz de mostrar al usuario las distintas opciones disponibles sobre una sugerencia, mediante un menú contextual que se abre con el botón derecho: • Cambiar el estado. • Crear respuesta. • Refrescar la vista.		
Trazabilidad:	RSF-036, RSF-037, RSF-039		

Tabla 55: RSF-038

RSF - 039: Modificación de una sugerencia			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	18/04/2012
Prioridad:	Alta	**Complejidad:**	Alta
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Modificar la información de una sugerencia existente en el sistema.		
Precondiciones:	Debe existir al menos una sugerencia en el sistema.		
Descripción:	El usuario debe ser capaz de modificar la información de una sugerencia ya existente en el sistema. Los campos modificables por el usuario son: Descripción, estado y respuesta.		
Trazabilidad:	RSF-036, RSF-037, RSF-038		

Tabla 56: RSF-039

RSF - 040: Búsqueda de una sugerencia de términos			
Autor:	Borja López Gómez	**Tipo:**	Funcional
Fuente:	The Reuse Company	**Fecha:**	19/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Agilizar el proceso de búsqueda de una sugerencia de términos mediante filtros.		
Precondiciones:			
Descripción:	El usuario debe ser capaz de buscar una sugerencia de términos utilizando distintos filtros de búsqueda, mostrándose los resultados en la misma vista de sugerencias de términos. Se debe poder filtrar por nombre		

	del término, autor de la sugerencia, acción sugerida, estado de la sugerencia, si/no ha sido respondida.
Trazabilidad:	RSF-036, RSF-037

Tabla 57: RSF-040

RSIn - 041: Listado de todos los términos			
Autor:	Borja López Gómez	**Tipo:**	Interfaz
Fuente:	The Reuse Company	**Fecha:**	23/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Facilitar la localización de términos.		
Precondiciones:			
Descripción:	El sistema debe tener una opción en la interfaz para visualizar todos los términos de la ontología.		
Trazabilidad:	RSF-002, RSF-003, RSF-046, RSF-047, RSF-048		

Tabla 58: RSF-041

RSIn - 042: Listado de términos del dominio			
Autor:	Borja López Gómez	**Tipo:**	Interfaz
Fuente:	The Reuse Company	**Fecha:**	23/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Facilitar la localización de términos.		
Precondiciones:			

Descripción:	El sistema debe tener una opción en la interfaz para visualizar sólo los términos cuya etiqueta sintáctica sea NOUN o hija de NOUN y pertenezcan al dominio.
Trazabilidad:	RSF-002, RSF-003, RSF-046, RSF-047, RSF-048

Tabla 59: RSF-042

RSIn - 043: Listado de verbos del dominio			
Autor:	Borja López Gómez	**Tipo:**	Interfaz
Fuente:	The Reuse Company	**Fecha:**	23/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Facilitar la localización de términos.		
Precondiciones:			
Descripción:	El sistema debe tener una opción en la interfaz para visualizar sólo los términos cuya etiqueta sintáctica sea VERB o hija de VERB y pertenezcan al dominio.		
Trazabilidad:	RSF-002, RSF-003, RSF-046, RSF-047, RSF-048		

Tabla 60: RSF-043

RSIn - 044: Listado de términos candidatos al dominio			
Autor:	Borja López Gómez	**Tipo:**	Interfaz
Fuente:	The Reuse Company	**Fecha:**	23/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Facilitar la localización de términos.		

Precondiciones:	
Descripción:	El sistema debe tener una opción en la interfaz para visualizar sólo los términos cuya etiqueta sintáctica sea UNCLASSIFIED_NOUN o hija de UNCLASSIFIED_NOUN.
Trazabilidad:	RSF-002, RSF-003, RSF-046, RSF-047, RSF-048

Tabla 61: RSF-044

RSIn - 045: Listado de verbos candidatos al dominio			
Autor:	Borja López Gómez	**Tipo:**	Interfaz
Fuente:	The Reuse Company	**Fecha:**	23/04/2012
Prioridad:	Alta	**Complejidad:**	Baja
Necesidad:	Alta	**Estado**:	Propuesto
Objetivo:	Facilitar la localización de términos.		
Precondiciones:			
Descripción:	El sistema debe tener una opción en la interfaz para visualizar sólo los términos cuya etiqueta sintáctica sea UNCLASSIFIED_VERB o hija de UNCLASSIFIED_VERB.		
Trazabilidad:	RSF-002, RSF-003, RSF-046, RSF-047, RSF-048		

Tabla 62: RSF-045

RSIn - 046: Color de fondo en los listados			
Autor:	Borja López Gómez	**Tipo:**	Interfaz
Fuente:	The Reuse Company	**Fecha:**	24/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto

Objetivo:	Mejorar la visualización de elementos en los listados.
Precondiciones:	
Descripción:	El sistema debe marcar con color una elemento de un listado una vez se haya completado una operación de inserción (color verde), modificación (color amarillo) o borrado (color rojo) sobre dicho elemento.
Trazabilidad:	RSF-002, RSF-009, RSF-016, RSF-017, RSF-018, RSF-022, RSF-029, RSF-30, RSF-31, RSF-36, RSF-37, RSF-41, RSF-42, RSF-43, RSF-44, RSF-45

Tabla 63: RSF-046

RSCa - 047: Ordenación de los listados			
Autor:	Borja López Gómez	**Tipo:**	Calidad
Fuente:	The Reuse Company	**Fecha:**	25/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Mejorar la navegación y usabilidad de los listados.		
Precondiciones:			
Descripción:	El sistema debe permitir la ordenación de los elementos de los listados haciendo click en la cabecera de la columna que se quiera ordenar.		
Trazabilidad:	RSF-002, RSF-009, RSF-016, RSF-017, RSF-018, RSF-022, RSF-029, RSF-30, RSF-31, RSF-36, RSF-37, RSF-41, RSF-42, RSF-43, RSF-44, RSF-45		

Tabla 64: RSF-047

RSCa - 048: Número de elementos de los listados			
Autor:	Borja López Gómez	**Tipo:**	Calidad
Fuente:	The Reuse Company	**Fecha:**	25/04/2012
Prioridad:	Media	**Complejidad:**	Baja
Necesidad:	Media	**Estado**:	Propuesto
Objetivo:	Mejorar la usabilidad de los listados.		
Precondiciones:			
Descripción:	El sistema debe mostrar el número de elementos que hay en cada listado.		
Trazabilidad:	RSF-002, RSF-009, RSF-016, RSF-017, RSF-018, RSF-022, RSF-029, RSF-30, RSF-31, RSF-36, RSF-37, RSF-41, RSF-42, RSF-43, RSF-44, RSF-45		

Tabla 65: RSF-048

3.1.7 Especificación de Casos de Uso

Para visualizar con más facilidad todos los casos de uso, se han dividido por subsistemas. Así, tenemos casos de uso de terminología, de etiquetas sintácticas, de semánticas y de sugerencias.

3.1.7.1 *Terminología*

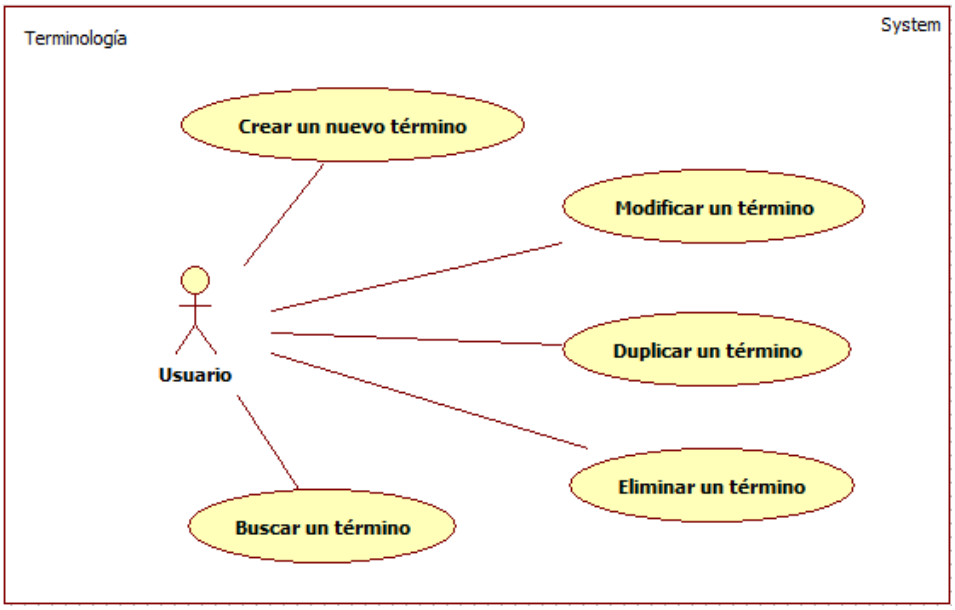

Ilustración 11: Casos de uso de Terminología

Caso de uso: Crear un nuevo término	
Identificador:	CU-001
Descripción:	El usuario tiene la posibilidad de crear un nuevo término que será introducido en la ontología.
Precondiciones:	Debe existir al menos una 'etiqueta sintáctica' en la ontología, ya que es un atributo obligatorio a la hora de crear un nuevo término. Debe existir al menos un 'lenguaje' en la ontología, ya que también es obligatorio a la hora de crear un nuevo término.
Postcondiciones:	Un nuevo término ha sido introducido en la ontología.
Escenario:	1. Entrar en el apartado de Terminología. 2. En el menú contextual, elegir la opción de añadir término. 3. Rellenar en un nuevo formulario los atributos que forman un nuevo término. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	No está permitido crear un término sin 'etiqueta sintáctica'. No está permitido crear un término sin 'lenguaje'.

	No está permitido crear dos términos con el mismo nombre y misma 'etiqueta sintáctica'.

Tabla 66: CU-001

Caso de uso: Modificar un término	
Identificador:	CU-002
Descripción:	El usuario tiene la posibilidad de modificar un término que ha sido creado previamente en la ontología.
Precondiciones:	Debe existir al menos un término en la ontología y debe haber al menos un término seleccionado.
Postcondiciones:	Las modificaciones sobre el término son almacenadas en la ontología.
Escenario:	1. Entrar en el apartado de Terminología. 2. Seleccionar un término y elegir la opción de modificar un término en el menú contextual. 3. Rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en la ontología. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	Se repiten las mismas condiciones de fallo que en CU-001.

Tabla 67: CU-002

Caso de uso: Duplicar un término	
Identificador:	CU-003
Descripción:	El usuario tiene la opción de duplicar un término que ya existe en la ontología. Al producirse la operación se deben tener en cuenta las restricciones de validación que deben seguirse a la hora de crear un nuevo término (CU-001). No olvidemos que una operación de duplicación es una nueva inserción con una serie de datos idénticos a otro término.
Precondiciones:	Debe existir al menos un término en la ontología y debe haber al menos un término seleccionado.
Postcondiciones:	Un nuevo término se crea a partir de la información de otro término ya existente previamente en la ontología.
Escenario:	1. Entrar en el apartado de Terminología.

	2. Seleccionar un término y elegir la opción de duplicar un término en el menú contextual. 3. Confirmar los diálogos y rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en la ontología, ya que se siguen las restricciones marcadas en CU-001. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	Si el usuario cancela el formulario habilitado para rellenar los atributos del nuevo término, éste no se almacena en la ontología. Se repiten las mismas condiciones de fallo que en CU-001.

Tabla 68: CU-003

Caso de uso: Eliminar un término	
Identificador:	CU-004
Descripción:	El usuario tiene la posibilidad de eliminar un término que ya existe en la ontología.
Precondiciones:	Debe existir al menos un término en la ontología y debe haber al menos un término seleccionado.
Postcondiciones:	Un término existente ha sido eliminado de la ontología.
Escenario:	1. Entrar en el apartado de Terminología. 2. Seleccionar un término y elegir la opción de eliminar un término en el menú contextual. 3. Confirmar el borrado.
Condiciones de fallo:	

Tabla 69: CU-004

Caso de uso: Buscar un término	
Identificador:	CU-005
Descripción:	El usuario tiene la posibilidad de buscar un término existente en la ontología mediante filtros de búsqueda.
Precondiciones:	
Postcondiciones:	Ninguno, uno o varios términos se muestran en el listado de términos.

Escenario:	1. Entrar en el apartado de Terminología. 2. Introducir los criterios de búsqueda deseados. 3. Hacer click en el botón de Buscar.
Condiciones de fallo:	

Tabla 70: CU-005

3.1.7.2 Etiquetas sintácticas

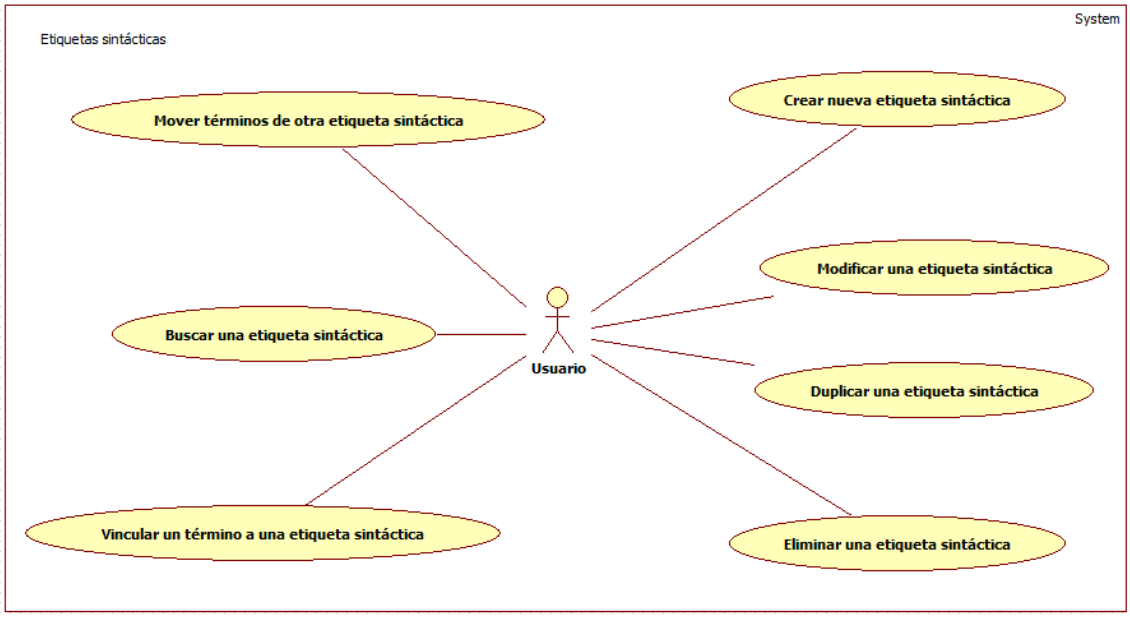

Ilustración 12: Casos de uso de Etiquetas sintácticas

Caso de uso: Crear una nueva Etiqueta Sintáctica	
Identificador:	CU-006
Descripción:	El usuario tiene la posibilidad de crear una nueva etiqueta sintáctica, que será introducida en la ontología.
Precondiciones:	Debe existir al menos una 'etiqueta sintáctica' en la ontología, ya que la etiqueta sintáctica raíz es un atributo obligatorio a la hora de crear una nueva etiqueta sintáctica.
Postcondiciones:	Una nueva etiqueta sintáctica ha sido introducida en la ontología.
Escenario:	1. Entrar en el apartado de Etiquetas Sintácticas.

	2. En el menú contextual, elegir la opción de añadir etiqueta sintáctica. 3. Rellenar en un nuevo formulario los atributos que forman una nueva etiqueta sintáctica. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	No está permitido crear una etiqueta sintáctica sin rellenar el campo 'etiqueta sintáctica raíz'. No está permitido que existan dos etiqueta sintácticas con el mismo 'nombre' y la mismo 'etiqueta sintáctica raíz'. No está permitido crear una etiqueta sintáctica de tipo 'nombre' y 'verbo' a la vez. Debe ser de ningún o sólo uno de los dos tipos.

Tabla 71: CU-006

Caso de uso: Modificar una etiqueta sintáctica	
Identificador:	CU-007
Descripción:	El usuario tiene la posibilidad de modificar una etiqueta sintáctica que ha sido creado previamente en la ontología.
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología y debe haber al menos una etiqueta sintáctica seleccionada.
Postcondiciones:	Las modificaciones sobre la etiqueta sintáctica son almacenadas en la ontología.
Escenario:	1. Entrar en el apartado de Etiquetas Sintácticas. 2. Seleccionar una etiqueta sintáctica y elegir la opción de modificar una etiqueta sintáctica en el menú contextual. 3. Rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en la ontología. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	Se repiten las mismas condiciones de fallo que en CU-006.

Tabla 72: CU-007

Caso de uso: Duplicar una etiqueta sintáctica	
Identificador:	CU-008
Descripción:	El usuario tiene la opción de duplicar una etiqueta sintáctica que ya existe en la ontología.
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología y debe haber al menos una etiqueta sintáctica seleccionado.
Postcondiciones:	Una nueva etiqueta sintáctica se crea a partir de la información de otra etiqueta sintáctica ya existente previamente en la ontología.
Escenario:	1. Entrar en el apartado de Etiquetas Sintácticas. 2. Seleccionar una etiqueta sintáctica y elegir la opción de duplicar una etiqueta sintáctica en el menú contextual. 3. Confirmar los diálogos y rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en la ontología, ya que se siguen las restricciones marcadas en CU-006. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	Si el usuario cancela el formulario habilitado para rellenar los atributos de la nueva etiqueta sintáctica, éste no se almacena en la ontología. Se repiten las mismas condiciones de fallo que en CU-006.

Tabla 73: CU-008

Caso de uso: Eliminar una etiqueta sintáctica	
Identificador:	CU-009
Descripción:	El usuario tiene la posibilidad de eliminar una etiqueta sintáctica que ya existe en la ontología. Para que una etiqueta sintáctica sea borrada correctamente, no debe acarrear ninguna dependencia. Es decir, no debe haber ningún término que tenga asociado dicha etiqueta sintáctica.
Precondiciones:	Debe existir al menos una etiqueta sintáctica en la ontología y debe haber al menos una etiqueta sintáctica seleccionada.
Postcondiciones:	Una etiqueta sintáctica existente ha sido eliminada de la

	ontología.
Escenario:	1. Entrar en el apartado de Etiquetas Sintácticas. 2. Seleccionar una etiqueta sintáctica y elegir la opción de eliminar una etiqueta sintáctica en el menú contextual. 3. Confirmar el borrado.
Condiciones de fallo:	No se puede eliminar una etiqueta sintáctica que tenga algún término asociado. Si ocurre el caso, se debe mostrar al usuario qué término o términos dependen de la etiqueta sintáctica que se quiere eliminar.

Tabla 74: CU-009

Caso de uso: Buscar una etiqueta sintáctica	
Identificador:	CU-010
Descripción:	El usuario tiene la posibilidad de buscar una etiqueta sintáctica existente en la ontología mediante filtros de búsqueda.
Precondiciones:	
Postcondiciones:	Ninguna, una o varias etiquetas sintácticas se muestran en el listado de etiquetas sintácticas.
Escenario:	1. Entrar en el apartado de Etiquetas sintácticas. 2. Introducir los criterios de búsqueda deseados. 3. Hacer click en el botón de Buscar.
Condiciones de fallo:	

Tabla 75: CU-010

Caso de uso: Vincular un término a una etiqueta sintáctica	
Identificador:	CU-011
Descripción:	El usuario tiene la posibilidad de vincular un término a una etiqueta sintáctica, desde la vista de etiquetas sintácticas.
Precondiciones:	Debe existir una etiqueta sintáctica seleccionada.
Postcondiciones:	El término queda vinculado a la etiqueta sintáctica seleccionada.
Escenario:	1. Entrar en el apartado de Etiquetas sintácticas.

	2. Seleccionar una etiqueta sintáctica. 3. Hacer click con el botón derecho en el listado de términos asociados a la etiqueta sintáctica seleccionada previamente. 4. Elegir la opción 'Añadir término' 5. Buscar un término y hacer click en Aceptar.
Condiciones de fallo:	

Tabla 76: CU-011

Caso de uso: Mover términos a otra etiqueta sintáctica	
Identificador:	CU-012
Descripción:	El usuario tiene la posibilidad de mover todos los términos de una etiqueta sintáctica a otra, desde la vista de etiquetas sintácticas.
Precondiciones:	Debe existir una etiqueta sintáctica seleccionada.
Postcondiciones:	Los términos de la etiqueta sintáctica de origen quedan vinculados a la etiqueta sintáctica seleccionada (destino).
Escenario:	1. Entrar en el apartado de Etiquetas sintácticas. 2. Seleccionar una etiqueta sintáctica (destino). 3. Hacer click con el botón derecho en el listado de términos asociados a la etiqueta sintáctica seleccionada previamente. 4. Elegir la opción 'Mover términos' 5. Buscar la etiqueta sintáctica origen y hacer click en Aceptar.
Condiciones de fallo:	

Tabla 77: CU-012

3.1.7.3 Semánticas

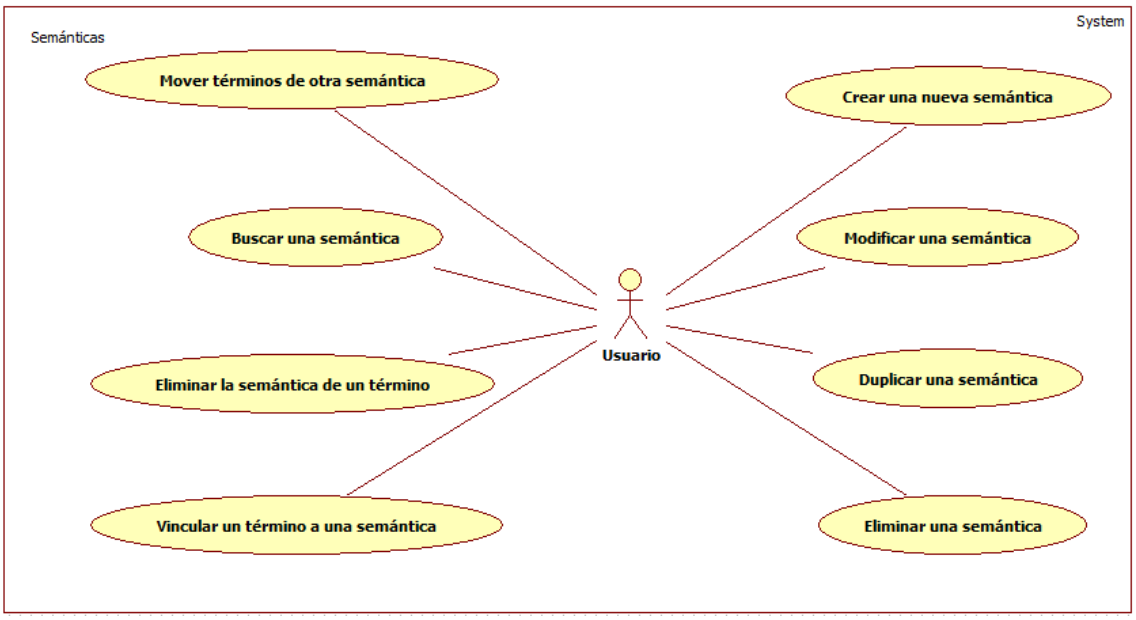

Ilustración 13: Casos de uso de Semánticas

Caso de uso: Crear una nueva semántica	
Identificador:	CU-013
Descripción:	El usuario tiene la posibilidad de crear una nueva semántica que será introducida en la ontología.
Precondiciones:	
Postcondiciones:	Una nueva semántica ha sido introducida en la ontología.
Escenario:	1. Entrar en el apartado de Semánticas. 2. En el menú contextual, elegir la opción de añadir semántica. 3. Rellenar en un nuevo formulario los atributos que forman una nueva semántica. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	No está permitido crear una semántica sin nombre. No pueden existir dos semánticas con el mismo nombre. El atributo 'Ponderación' debe ser en cualquier caso mayor o igual a cero. Si una semántica es visible en el tesauro, debe completarse información sobre sus roles. Si el atributo

	de ‘Simétrica’ está marcado, sólo es necesario rellenar la descripción y abreviación del rol A. Si el atributo de ‘Simétrica’ no está marcado, es necesario rellenar la descripción y abreviación de ambos roles, A y B.

Tabla 78: CU-013

Caso de uso: Modificar una semántica	
Identificador:	CU-014
Descripción:	El usuario tiene la posibilidad de modificar una semántica que ha sido creada previamente en la ontología.
Precondiciones:	Debe existir al menos una semántica en la ontología y debe haber al menos una semántica seleccionada.
Postcondiciones:	Las modificaciones sobre la semántica son almacenadas en la ontología.
Escenario:	1. Entrar en el apartado de Semánticas. 2. Seleccionar una semántica y elegir la opción de modificar una semántica en el menú contextual. 3. Rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en la ontología. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	Se repiten las mismas condiciones de fallo que en CU-013.

Tabla 79: CU-014

Caso de uso: Duplicar una semántica	
Identificador:	CU-015
Descripción:	El usuario tiene la opción de duplicar una semántica que ya existe en la ontología. Al producirse la operación se deben tener en cuenta las restricciones de validación que deben seguirse a la hora de crear una nueva semántica (CU-013). No olvidemos que una operación de duplicación es una nueva inserción con una serie de datos idénticos a otra semántica.
Precondiciones:	Debe existir al menos una semántica en la ontología y debe haber al menos una semántica seleccionada
Postcondiciones:	Una nueva semántica creada a partir de la información

	de otra semántica ya existente ha sido introducida en la ontología.
Escenario:	1. Entrar en el apartado de Semánticas. 2. Seleccionar una semántica y elegir la opción de duplicar una semántica en el menú contextual. 3. Confirmar los diálogos y rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en la ontología, ya que se siguen las restricciones marcadas en CU-013. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	Si el usuario cancela el formulario habilitado para rellenar los atributos de la nueva semántica, ésta no se almacena en la ontología. Se repiten las mismas condiciones de fallo que en CU-013.

Tabla 80: CU-015

Caso de uso: Eliminar una semántica	
Identificador:	CU-016
Descripción:	El usuario tiene la posibilidad de eliminar una semántica que ya existe en la ontología. Para que una semántica sea borrada correctamente, no debe acarrear ninguna dependencia. Es decir, no debe haber ningún término que tenga asociado dicha semántica.
Precondiciones:	Debe existir al menos una semántica en la ontología y debe haber al menos una semántica seleccionada
Postcondiciones:	Una semántica existente ha sido eliminada de la ontología.
Escenario:	1. Entrar en el apartado de Semánticas. 2. Seleccionar una semántica y elegir la opción de eliminar una semántica en el menú contextual. 3. Confirmar el borrado.
Condiciones de fallo:	No se puede eliminar una semántica que tenga algún término asociado. Si ocurre el caso, se debe mostrar al usuario qué término o términos dependen de la semántica que se quiere eliminar.

Tabla 81: CU-016

Caso de uso: Buscar una semántica	
Identificador:	CU-017
Descripción:	El usuario tiene la posibilidad de buscar una semántica existente en la ontología mediante filtros de búsqueda.
Precondiciones:	
Postcondiciones:	Ninguna, una o varias semánticas se muestran en el listado de semánticas.
Escenario:	1. Entrar en el apartado de Semánticas. 2. Introducir los criterios de búsqueda deseados. 3. Hacer click en el botón de Buscar.
Condiciones de fallo:	

Tabla 82: CU-017

Caso de uso: Vincular un término a una semántica	
Identificador:	CU-018
Descripción:	El usuario tiene la posibilidad de vincular un término a una semántica, desde la vista de semánticas.
Precondiciones:	Debe existir una semántica seleccionada.
Postcondiciones:	El término queda vinculado a la semántica seleccionada.
Escenario:	1. Entrar en el apartado de Semánticas. 2. Seleccionar una semántica. 3. Hacer click con el botón derecho en el listado de términos asociados a la semántica seleccionada previamente. 4. Elegir la opción ‘Añadir término’ 5. Buscar un término y hacer click en Aceptar.
Condiciones de fallo:	

Tabla 83: CU-018

Caso de uso: Mover términos a otra semántica	
Identificador:	CU-019
Descripción:	El usuario tiene la posibilidad de mover todos los términos de una semántica a otra, desde la vista de semánticas.
Precondiciones:	Debe existir una semántica seleccionada.
Postcondiciones:	Los términos de la semántica de origen quedan vinculados a la semántica seleccionada (destino).
Escenario:	1. Entrar en el apartado de Semánticas. 2. Seleccionar una semántica (destino). 3. Hacer click con el botón derecho en el listado de términos asociados a la semántica seleccionada previamente. 4. Elegir la opción 'Mover términos' 5. Buscar la semántica origen y hacer click en Aceptar.
Condiciones de fallo:	

Tabla 84: CU-019

Caso de uso: Eliminar la semántica de un término	
Identificador:	CU-020
Descripción:	El usuario tiene la posibilidad de eliminar la semántica de un término desde la vista de semánticas.
Precondiciones:	Debe existir al menos una semántica seleccionada, y un término seleccionado en el listado de términos asociados a una semántica.
Postcondiciones:	El término seleccionado no tiene semántica.
Escenario:	1. Entrar en el apartado de Semánticas. 2. Seleccionar una semántica que tenga términos asociados. 3. Hacer click con el botón derecho sobre un término. 4. Elegir la opción 'Eliminar semántica'.
Condiciones de fallo:	

Tabla 85: CU-020

3.1.7.4 Sugerencias

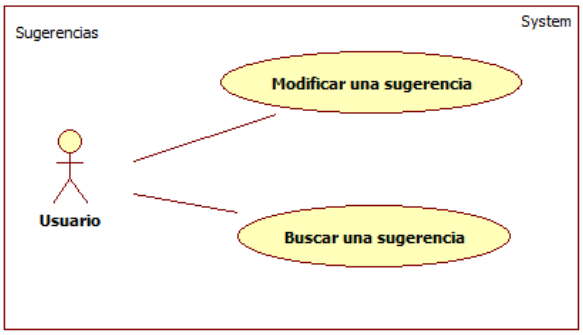

Ilustración 14: Casos de uso de Sugerencias

Caso de uso: Modificar una sugerencia	
Identificador:	CU-021
Descripción:	El usuario tiene la posibilidad de modificar una sugerencia existente.
Precondiciones:	Debe existir al menos una sugerencia seleccionada.
Postcondiciones:	Las modificaciones sobre la sugerencia son almacenadas en base de datos.
Escenario:	1. Entrar en el apartado de Sugerencias. 2. Seleccionar una sugerencia y hacer doble click sobre ella. 3. Rellenar en un nuevo formulario los atributos tal y como se desea que sean almacenados en base de datos. 4. Pulsar sobre el botón de aceptar.
Condiciones de fallo:	No está permitido que el campo 'Estado' quede vacío. No está permitido que el campo 'Respuesta' quede vacío.

Tabla 86: CU-021

Caso de uso: Buscar una sugerencia	
Identificador:	CU-022
Descripción:	El usuario tiene la posibilidad de buscar una sugerencia existente en base de datos mediante filtros de búsqueda.
Precondiciones:	

Postcondiciones:	Ninguna, una o varias sugerencias se muestran en el listado de sugerencias.
Escenario:	1. Entrar en el apartado de Sugerencias. 2. Introducir los criterios de búsqueda deseados. 3. Hacer click en el botón de Buscar.
Condiciones de fallo:	

Tabla 87: CU-022

Identificación de Subsistemas de Análisis

Debido a la complejidad de representar toda la funcionalidad especificada en un solo sistema, se divide el problema en varios subsistemas más especializados. El criterio utilizado para llevar a cabo esta división ha sido la homogeneidad de procesos y afinidad de requisitos.

3.1.8 Determinación de Subsistemas de Análisis

- **Formulario principal**: se encarga de mostrar la interfaz de usuario común a todos los subsistemas.
- **Terminología**: se encarga de representar la funcionalidad de la parte de la aplicación relacionada con términos.
- **Etiquetas sintácticas**: se encarga de representar la funcionalidad de la parte de la aplicación relacionada con etiquetas sintácticas.
- **Semánticas**: se encarga de representar la funcionalidad de la parte de la aplicación relacionada con semánticas.
- **Sugerencias**: representa la funcionalidad de la parte de la aplicación relacionada con sugerencias de términos y relaciones del tesauro.

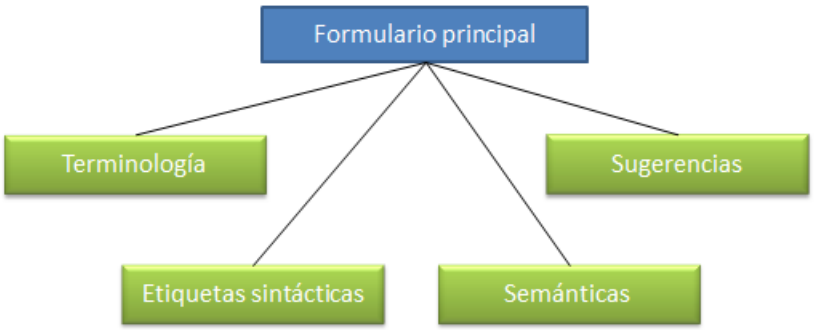

Ilustración 15: Subsistemas de análisis

Análisis de casos de uso

Una vez se han definido los casos de uso que van a formar parte de la aplicación, se identifican las clases u objetos necesarios para llevar a cabo la funcionalidad especificada en cada caso de uso. No se realiza un análisis y diseño detallado de cada clase, ya que se profundizará en detalles durante la fase de diseño.

Para alcanzar este objetivo, se utilizan tres tipos de clases:

- **Clases de entidad**: representan la información que contendrá la clase.
- **Clases de interfaz**: describen la interacción entre el sistema y los actores.
- **Clases de control**: representan la secuencia de transacciones y la lógica de negocio.

3.1.9 Identificación de las clases asociadas a un Caso de Uso

- Clases de entidad:
    - **VocabularyDataSet**: contiene los atributos que forman un término.
    - **RulesFamilies_DataSet**: contiene los atributos que forman una etiqueta sintáctica.
    - **GrammaticalDataSet**: contiene los atributos que forman una gramática.
    - **SuggestionDataSet**: contiene los atributos que forman una sugerencia.
- Clases de interfaz
    - **MainForm**: formulario de apertura, común para toda la aplicación.
    - **AllTerms**: muestra todos los términos.
    - **Term**: muestra la información referida a un término.
    - **CandidateTerms**: muestra los términos candidatos a formar parte del dominio.
    - **TermTags**: muestra todas las etiquetas sintácticas.
    - **TermTag**: muestra la información referida a una etiqueta sintáctica.
    - **SelectTermTag**: selector empleado para realizar búsqueda de etiquetas sintácticas fuera del formulario de etiquetas sintácticas.
    - **Semantics**: muestra todas las semánticas.
    - **Semantic**: muestra la información referida a una semántica.

- **SelectSemantic**: selector empleado para realizar búsqueda de semánticas fuera del formulario de semánticas.
- **Suggestions**: muestra todas las sugerencias de términos.
- **SuggestedRelationsihp**: muestra todas las sugerencias de relaciones entre términos.
- **Suggestion**: muestra la información referida a una sugerencia.

- Clases de control
  - **VocabularyRepository**: contiene la lógica de negocio del subsistema de terminología.
  - **RulesFamiliesRepository**: contiene la lógica de negocio del subsistema de etiquetas sintácticas.
  - **GrammaticalsRepository**: contiene la lógica de negocio del subsistema de semánticas.
  - **SuggestionsRepository**: contiene la lógica de negocio del subsistema de sugerencias.

3.1.10 Descripción de la Interacción de Objetos

Para describir los casos de interacción entre objetos se utilizan diagramas de secuencia. Ya que los subsistemas realizan operaciones similares, se realizan dichos diagramas para las principales acciones del subsistema de semánticas.

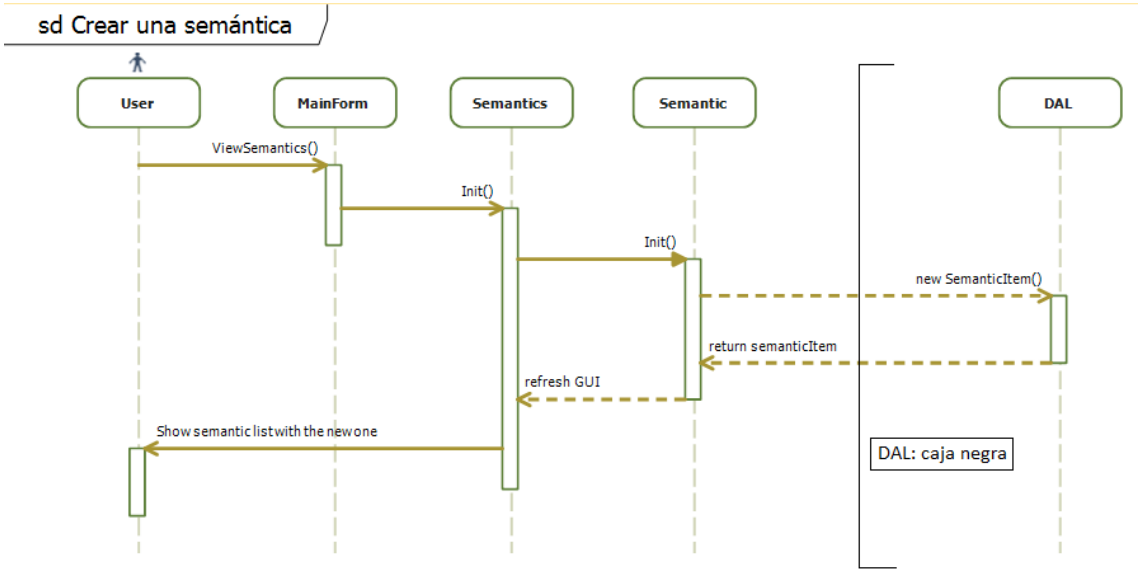

Ilustración 16: Diagrama de secuencia para crear una nueva semántica

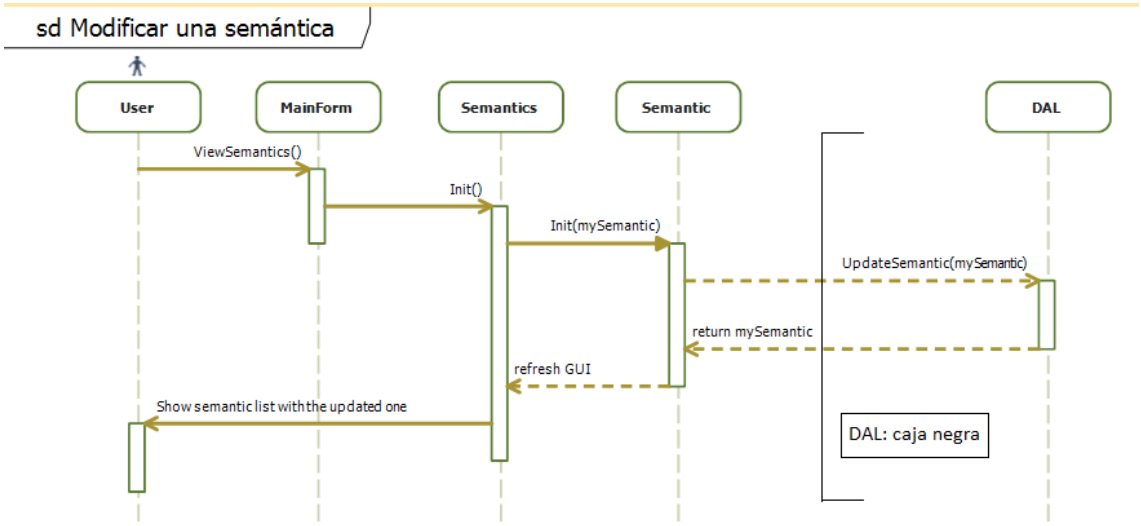

Ilustración 17: Diagrama de secuencia para modificar una semántica

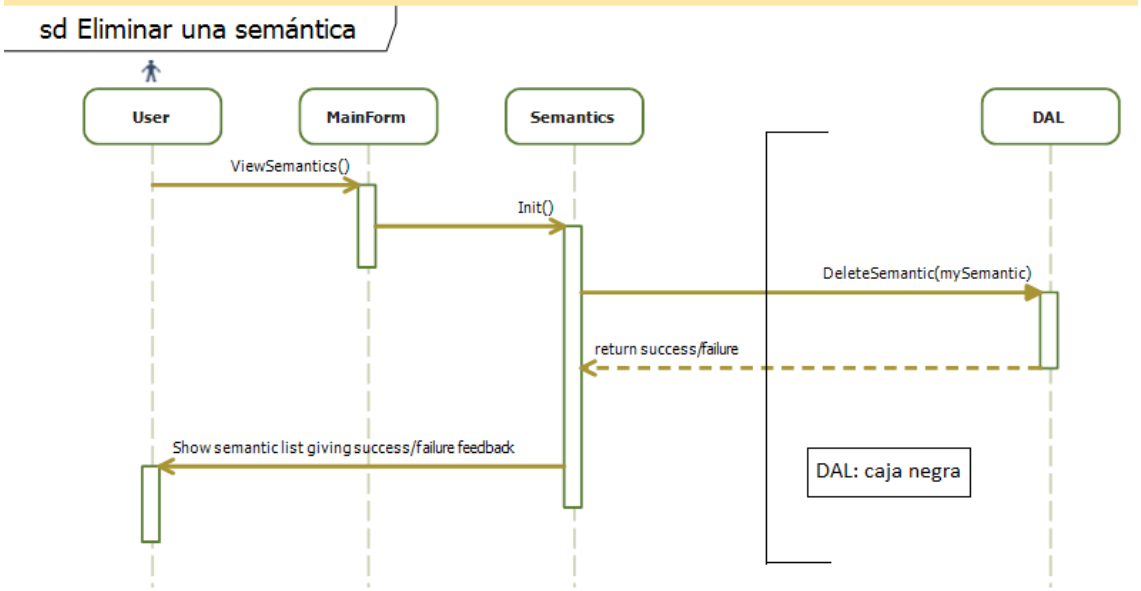

Ilustración 18: Diagrama de secuencia para eliminar una semántica

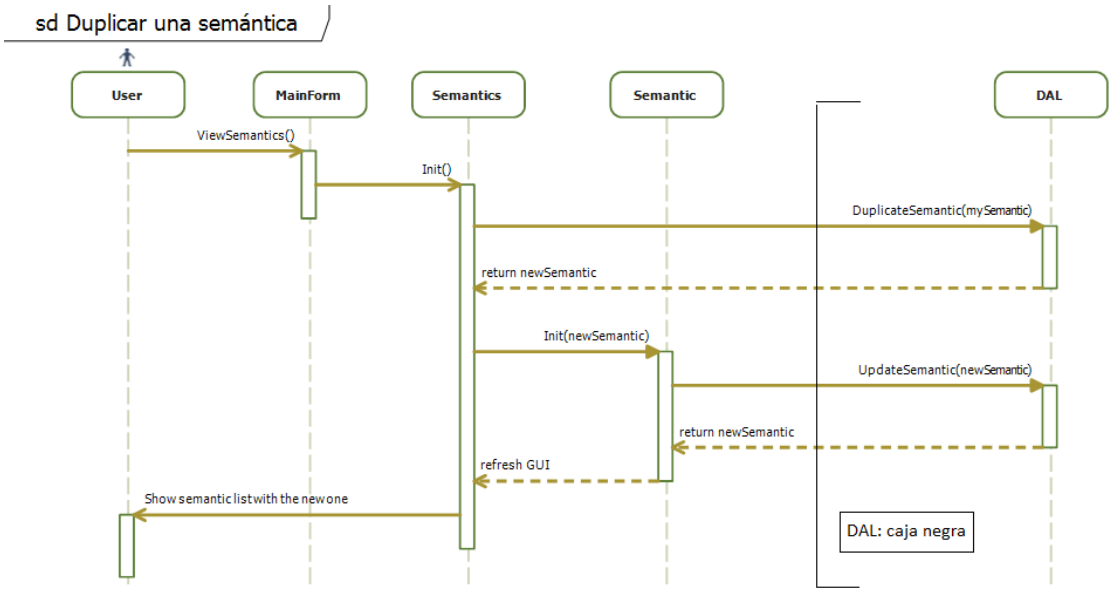

Ilustración 19: Diagrama de secuencia para duplicar una semántica

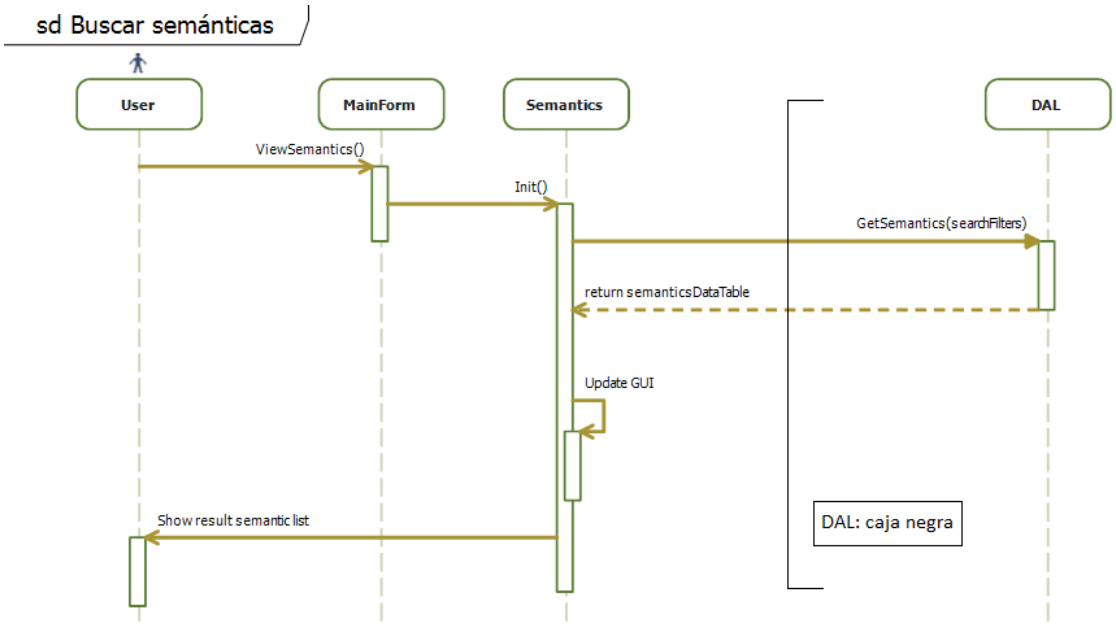

Ilustración 20: Diagrama de secuencia para buscar semánticas

Análisis de Clases

Una vez definidas en primera instancia las clases que van a formar parte de la aplicación, lo siguiente es definir las propiedades que va a tener cada una de ellas. Al igual que durante la definición de clases, no se profundiza en los

detalles de las propiedades de cada clase, ya que esta tarea se hará durante la fase de diseño de la aplicación.

3.1.11 Identificación de Responsabilidades y Atributos

En esta sección se definen los atributos y funciones que debe tener cada clase, de forma que se satisfagan los requisitos especificados. Dichas funciones se detallan más durante la fase de diseño.

Con el objetivo de simplificar el análisis, se divide el apartado según los subsistemas identificados previamente.

3.1.11.1 Terminología

VocabularyDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• Tabla de datos que contiene los atributos referidos a un término
Operaciones:	

Tabla 88: Análisis de VocabularyDataSet

VocabularyRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de terminología.
Atributos:	
Operaciones:	• Función que devuelve una tabla de datos con todos los términos de la ontología. • Función que devuelve una tabla de datos con todos los términos que coincidan con los criterios introducidos por el usuario en el filtro de búsqueda. • Función que actualiza los datos de un término de la ontología. • Función que inserta un nuevo término en la ontología. • Función que duplica un término. • Función que elimina un término.

Tabla 89: Análisis de VocabularyRepository

AllTerms	
Responsabilidades:	Interfaz de listado de todos los términos.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de terminología. • Tabla de datos de términos.
Operaciones:	• Petición a la BLL para obtener todos los términos de la ontología. • Captura el evento para crear un nuevo término. • Captura el evento para editar los datos del término seleccionado. • Captura el evento para duplicar el término seleccionado. • Captura el evento para eliminar los términos seleccionados.

Tabla 90: Análisis de AllTerms

CandidateTerms	
Responsabilidades:	Interfaz de listado de términos candidatos.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de terminología. • Tabla de datos de términos.
Operaciones:	• Petición a la BLL para obtener todos los términos candidatos de la ontología.

Tabla 91: Análisis de CandidateTerms

Term	
Responsabilidades:	Interfaz para mostrar los datos de un término.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de términos.
Operaciones:	• Aceptar los cambios realizados en el formulario y realizar la correspondiente petición a la BLL para insertar o editar un término. • Cancelar los cambios realizados en el formulario. • Validar que los campos introducidos son válidos.

Tabla 92: Análisis de Term

3.1.11.2 Etiquetas sintácticas

Rules_FamiliesDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• Tabla de datos que contiene los atributos referidos a una etiqueta sintáctica.
Operaciones:	

Tabla 93: Análisis de Rules_FamiliesDataSet

RulesFamiliesRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de etiquetas sintácticas.
Atributos:	
Operaciones:	• Función que devuelve una tabla de datos con todas las etiquetas sintácticas de la ontología. • Función que devuelve una tabla de datos con todas las etiquetas sintácticas que tienen como padre otra determinada etiqueta sintáctica. • Función que devuelve una tabla de datos con todos los términos asociados a una determinada etiqueta sintáctica. • Función que devuelve una tabla de datos con todas las etiquetas sintácticas que coincidan con los criterios introducidos por el usuario en el filtro de búsqueda. • Función que actualiza los datos de una etiqueta sintáctica de la ontología. • Función que inserta una nueva etiqueta sintáctica en la ontología. • Función que duplica una etiqueta sintáctica. • Función que elimina una etiqueta sintáctica.

Tabla 94: Análisis de RulesFamiliesRepository

TermTags	
Responsabilidades:	Interfaz de listado de etiquetas sintácticas.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de etiquetas sintácticas. • Objeto de la BLL que permite acceder a la lógica de negocio de terminología. • Tabla de datos de las etiquetas sintácticas. • Tabla de datos de las etiquetas sintácticas hijas. • Tabla de datos de los términos asociados a la etiqueta sintáctica seleccionada.
Operaciones:	• Petición a la BLL para obtener todas las etiquetas sintácticas en la ontología. • Petición a la BLL para obtener todas las etiquetas sintácticas hijas asociadas a la etiqueta sintáctica seleccionada. • Petición a la BLL para obtener todos los términos asociados a la etiqueta sintáctica seleccionada. • Captura el evento para crear una nueva etiqueta sintáctica. • Captura el evento para editar los datos de la etiqueta sintáctica seleccionada. • Captura el evento para duplicar la etiqueta sintáctica seleccionada. • Captura el evento para eliminar las etiquetas sintácticas seleccionadas.

Tabla 95: Análisis de TermTags

TermTag	
Responsabilidades:	Interfaz para mostrar los datos de una etiqueta sintáctica.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de etiquetas sintácticas.
Operaciones:	• Aceptar los cambios realizados en el formulario y realizar la correspondiente petición a la BLL para insertar o editar una etiqueta sintáctica. • Cancelar los cambios realizados en el formulario.

	• Validar que los campos introducidos son válidos.

Tabla 96: Análisis de TermTag

SelectTermTag	
Responsabilidades:	Interfaz selector de una etiqueta sintáctica.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de etiquetas sintácticas. • Tabla de datos de etiquetas sintácticas.
Operaciones:	• Realizar la petición a la BLL para obtener las etiquetas sintácticas que coincidan con el criterio de búsqueda introducido por el usuario. • Aceptar y almacenar la etiqueta sintáctica elegida por el usuario. • Cancelar selección.

Tabla 97: Análisis de SelectTermTag

3.1.11.3 Semánticas

GrammaticalDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• Tabla de datos que contiene los atributos referidos a una semántica.
Operaciones:	

Tabla 98: Análisis de GrammaticalDataSet

GrammaticalsRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de semánticas.
Atributos:	
Operaciones:	• Función que devuelve una tabla de datos con todas las semánticas de la ontología. • Función que devuelve una tabla de datos con todas las semánticas que tienen como padre otra determinada semántica.

	• Función que devuelve una tabla de datos con todos los términos asociados a una determinada semántica. • Función que devuelve una tabla de datos con todas las semánticas que coincidan con los criterios introducidos por el usuario en el filtro de búsqueda. • Función que actualiza los datos de una semántica de la ontología. • Función que inserta una nueva semántica en la ontología. • Función que duplica una semántica. • Función que elimina una semántica.

Tabla 99: Análisis de GrammaticalRepository

Semantics	
Responsabilidades:	Interfaz de listado de semánticas.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de semánticas. • Objeto de la BLL que permite acceder a la lógica de negocio de terminología. • Tabla de datos de las semánticas. • Tabla de datos de las semánticas hijas. • Tabla de datos de los términos asociados a la semántica seleccionada.
Operaciones:	• Petición a la BLL para obtener todas las semánticas de la ontología. • Petición a la BLL para obtener todas las semánticas hijas asociadas a la semántica seleccionada. • Petición a la BLL para obtener todos los términos asociados a la semántica seleccionada. • Captura el evento para crear una semántica. • Captura el evento para editar los datos de la semántica seleccionada. • Captura el evento para duplicar la semántica seleccionada.

	• Captura el evento para eliminar las semánticas seleccionadas.

Tabla 100: Análisis de Semantics

Semantic	
Responsabilidades:	Interfaz para mostrar los datos de una semántica.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de semánticas.
Operaciones:	• Aceptar los cambios realizados en el formulario y realizar la correspondiente petición a la BLL para insertar o editar una semántica. • Cancelar los cambios realizados en el formulario. • Validar que los campos introducidos son válidos.

Tabla 101: Análisis de Semantic

SelectSemantic	
Responsabilidades:	Interfaz selector de una semántica.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de semánticas. • Tabla de datos de semánticas.
Operaciones:	• Realizar la petición a la BLL para obtener las semánticas que coincidan con el criterio de búsqueda introducido por el usuario. • Aceptar y almacenar la semántica elegida por el usuario. • Cancelar selección.

Tabla 102: Análisis de SelectSemantic

3.1.11.4 Sugerencias

SuggestionDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• Tabla de datos que contiene los atributos referidos a una sugerencia.
Operaciones:	

Tabla 103: Análisis de SuggestionDataSet

SuggestionsRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de sugerencias.
Atributos:	
Operaciones:	• Función que devuelve una tabla de datos con todas las sugerencias de la ontología. • Función que devuelve una tabla de datos con todas las sugerencias de la ontología creadas por un determinado usuario. • Función que devuelve una tabla de datos con todas las sugerencias que coincidan con los criterios introducidos por el usuario en el filtro de búsqueda. • Función que modifica el estado de una sugerencia existente. • Función que modifica la respuesta en una sugerencia existente.

Tabla 104: Análisis de SuggestionsRepository

Suggestions	
Responsabilidades:	Interfaz de listado de sugerencias de términos.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de sugerencias. • Tabla de datos de las sugerencias.
Operaciones:	• Petición a la BLL para obtener todas las sugerencias de la ontología.

	• Captura el evento para editar una sugerencia. • Captura el evento para mostrar sugerencias que coincidan con los criterios del filtro de búsqueda.

Tabla 105: Análisis de Suggestions

SuggestedRelationship	
Responsabilidades:	Interfaz de listado de sugerencias de relaciones entre términos. Utiliza la misma interfaz que “Suggestions”, luego será una clase heredada.
Atributos:	
Operaciones:	• Inicializador que habilita el atributo diferenciador que indica que se deben mostrar las sugerencias de relaciones.

Tabla 106: Análisis de SuggestedRelationship

Suggestion	
Responsabilidades:	Interfaz para mostrar los datos de una sugerencia.
Atributos:	• Objeto de la BLL que permite acceder a la lógica de negocio de sugerencias.
Operaciones:	• Aceptar los cambios realizados en el formulario y realizar la correspondiente petición a la BLL para editar la respuesta de la sugerencia. • Cancelar los cambios realizados en el formulario. • Validar que la respuesta de la sugerencia introducida no es una cadena vacía.

Tabla 107: Análisis de Suggestion

3.1.12 Identificación de Asociaciones y Agregaciones

En esta sección se definen las asociaciones y agregaciones más representativas para cada clase.

Como **asociación** se entiende una relación entre instancias de clases. Por lo tanto los objetos de la clase de un extremo de la asociación deben conocer los objetos del otro extremo de la relación.

Por otro lado, una **agregación** es un tipo de asociación en la que un objeto forma parte del otro, es decir, el todo se relaciona con sus partes. También se denomina como relación “parte de”.

Terminología ←→ Etiquetas sintácticas	
Tipo:	Asociación
Clases (cardinalidad):	• `Término (*)` • `Etiqueta sintáctica (1)`
Descripción:	Un **término** está asociado a una sola **etiqueta sintáctica**, y cada **etiqueta sintáctica** puede estar asociada a 0 o más **términos**.

Tabla 108: Asociación Terminología-Etiquetas Sintácticas

Terminología ←→ Semánticas	
Tipo:	Asociación
Clases (cardinalidad):	• `Término (*)` • `Semántica (0..1)`
Descripción:	Un **término** está asociado a ninguna o una sola **semántica**, y cada **semántica** puede estar asociada a 0 o más **términos**.

Tabla 109: Asociación Terminología-Semánticas

Terminología ←→ Sugerencias	
Tipo:	Agregación
Clases (cardinalidad):	• `Término (1..2)` • `Sugerencia (0..*)`
Descripción:	Un **término** aparece en ninguna o muchas **sugerencias**, y cada **sugerencia** está formada por 1 o 2 **términos**.

Tabla 110: Agregación Terminología-Sugerencias

Semánticas ←→ Sugerencias	
Tipo:	Agregación
Clases (cardinalidad):	• `Semántica (0..1)` • `Sugerencia (0..*)`
Descripción:	Una **semántica** aparece en ninguna o muchas **sugerencias**, y cada **sugerencia** está formada por 0 o 1 **semánticas**.

Tabla 111: Agregación Semánticas-Sugerencias

Elaboración del Modelo de Datos

La definición del modelo de datos puede entenderse como una actividad más de diseño que de análisis. Pero con el fin de facilitar la comprensión del problema en esta fase, se definen las **entidades** que van a formar parte del sistema de información.

Se parte de un modelo de datos ya existente, elaborado previamente para construir un metamodelo basado en relaciones, compuesto por toda clase de elementos (textuales, modelos de diseño, código, bases de datos, etc) destinado a generalizar su gestión en aplicaciones de ordenador [19], [20].

No se utiliza el modelo de datos completo, ya que es enorme y hay gran cantidad de tablas que no se utilizan durante este módulo. De esta forma, se han definido las siguientes entidades:

- **VOCABULARY**: Representa la información de un **término** de la ontología. Es la entidad fundamental del sistema. Contiene una lista muy amplia de atributos debido a que posteriormente el módulo desarrollado será reutilizado para abarcar una mayor funcionalidad.
    - El atributo *Type* hace referencia a una etiqueta sintáctica de la ontología.
    - El atributo *Grammatical* hace referencia a una semántica de la ontología.
- **RULES_FAMILIES**: Representa la información de una **etiqueta sintáctica** de la ontología.
- **GRAMMATICAL**: Representa la información de una **semántica** de la ontología.
- **SUGGESTION**: Representa la información de una **sugerencia** de la ontología.

- El atributo *RelationshipType* hace referencia a una semántica de la ontología.
- Los atributos *CodTerm1* y *CodTerm2* hacen referencia a un término de la ontología, respectivamente.

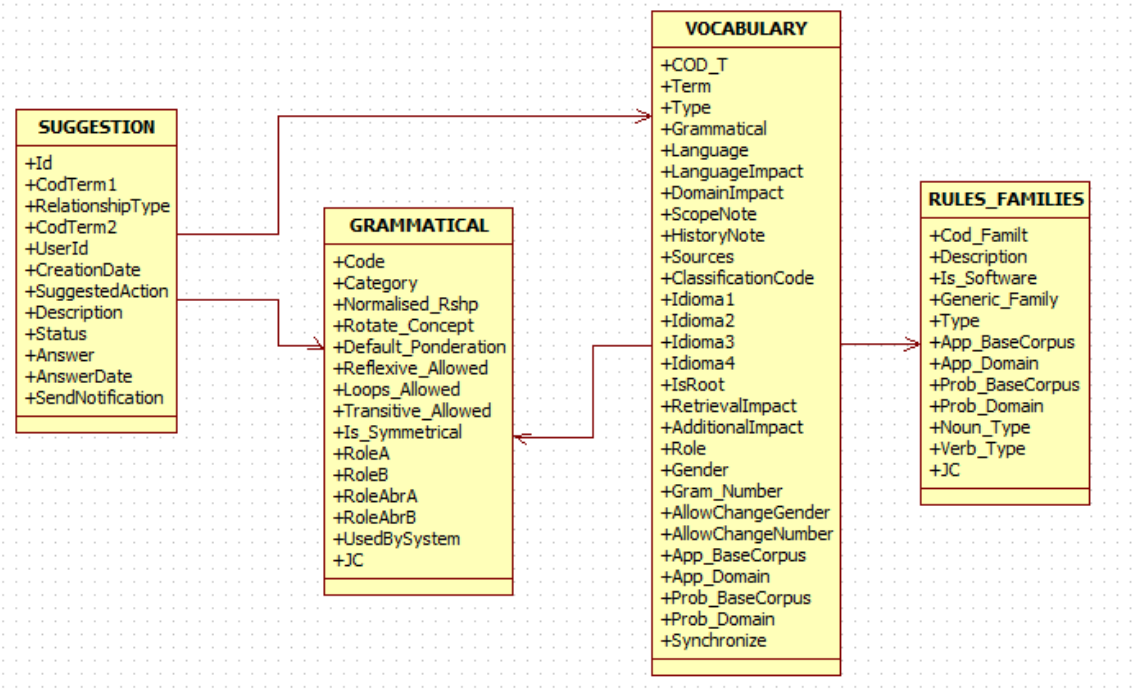

Ilustración 21: Modelo de datos

3.1.13 Especificación de Necesidades de Migración de Datos y Carga Inicial

A la hora de realizar altas, bajas, modificaciones o consultas de datos desde la interfaz de la aplicación, no se altera la estructura de ninguna tabla de la base de datos.

En cuanto a la carga inicial de datos, hay que tener en cuenta una serie de requisitos mínimos obligatorios para que la aplicación pueda iniciarse satisfactoriamente. Aunque ya ha sido definido en la fase de especificación de requisitos, se recuerda en este capítulo.

- La base de datos debe contener la etiqueta sintáctica NOUN.
- La base de datos debe contener la etiqueta sintáctica VERB.

No supone ningún problema para los miembros del equipo de desarrollo ni a la hora de realizar el despliegue de la aplicación, ya que junto con el código

fuente entregable se incluye un fichero de restauración de un ejemplo de base de datos válida.

Definición de Interfaces de Usuario

En este apartado se describen las principales características de la interfaz de usuario. Se sigue la misma interfaz que en el resto del *knowledgeManager*, por lo que no se especifican en profundidad los detalles de la interfaz.

3.1.14 Especificación de Principios Generales de la Interfaz

Una de las principales características de la capa de presentación de *knowledgeManager* es la reutilización de los *Windows Forms* y los *User Control*. Existe uno genérico de cada tipo, con las características comunes, y de ellos heredan el resto de clases de interfaz.

3.1.15 Identificación de Perfiles y Diálogos

En este apartado se adjunta gráficamente cómo debe ser cada diálogo que se muestra al usuario, según del tipo que sea.

Cuando el sistema necesita confirmación por parte del usuario, se lanza un diálogo de solicitud de confirmación.

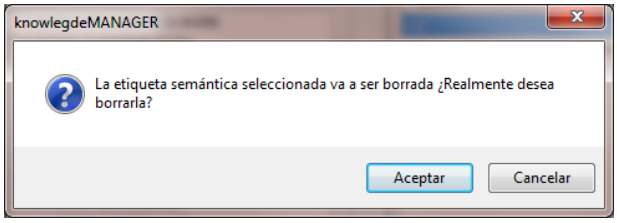

Ilustración 22: Solicitud de confirmación

Cuando el sistema ha llevado a cabo una operación con éxito, se lanza un diálogo de confirmación.

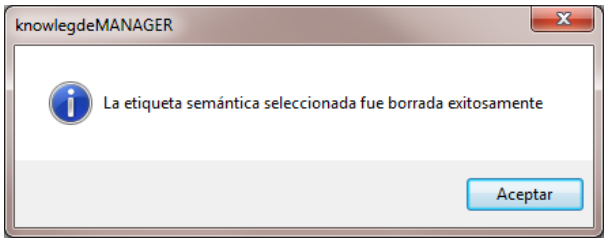

Ilustración 23: Mensaje de confirmación

Cuando el sistema ha intentado llevar a cabo una operación, pero se produjo un error durante el proceso, se lanza un mensaje de error descriptivo.

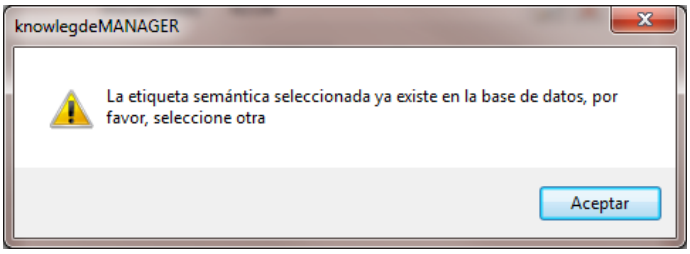

Ilustración 24: Mensaje de error

Cuando se solicita una operación de borrado de un elemento, y dicho elemento no se puede borrar debido a restricciones de integridad a nivel de base de datos, se lanza un mensaje de error descriptivo, personalizado con cada elemento.

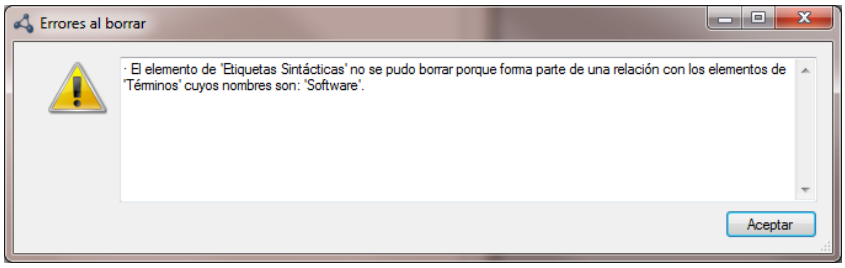

Ilustración 25: Mensaje de error de borrado

3.1.16 Especificación de Formatos Individuales de la Interfaz de Pantalla

En este apartado se define un icono por cada subsistema del sistema de información. Las sugerencias se dividen en dos, al tener distinto icono cada tipo.

Terminología	Etiquetas sintácticas	Semánticas	Sugerencias de términos	Sugerencias de relaciones
				

Tabla 112: Formatos individuales de la interfaz

3.1.17 Especificación del Comportamiento Dinámico de la Interfaz

Existen algunas situaciones en que estando en un determinado subsistema es necesario acceder a un objeto de otro subsistema distinto. Por ejemplo, si contemplamos la opción de crear un nuevo término, es necesario elegir una etiqueta sintáctica para ese término, lo que mezcla dos subsistemas distintos.

Para listar los elementos disponibles del segundo subsistema, al principio se comienza utilizando un control tipo *ComboBox*. Pero cuando hay demasiados elementos en este tipo de controles, no es fácil de recorrer.

Es por ello que se han creado **formularios selectoras** que emulan la funcionalidad de un *ComboBox* (elegir una opción) pero incluyen un filtro de búsqueda, para que encontrar lo que se busca sea más fácil.

Existe un selector de términos, de etiquetas sintácticas y de semánticas.

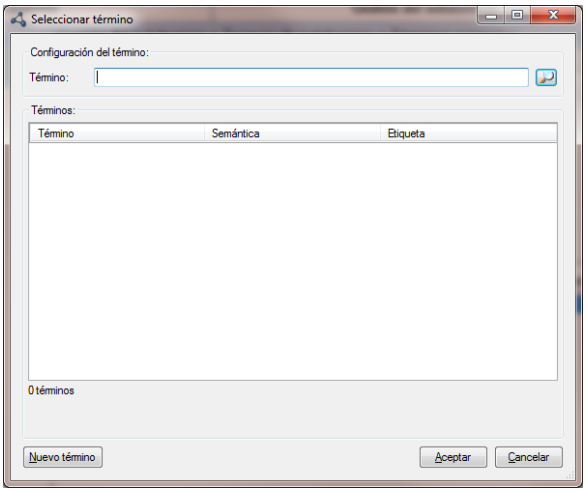

Ilustración 26: Selector de término

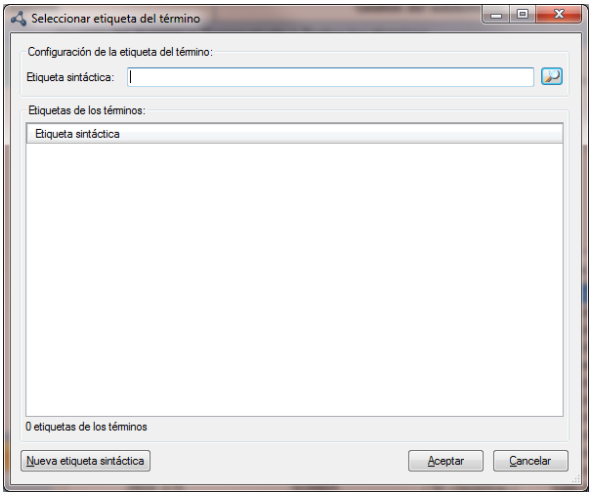

Ilustración 27: Selector de etiqueta sintáctica

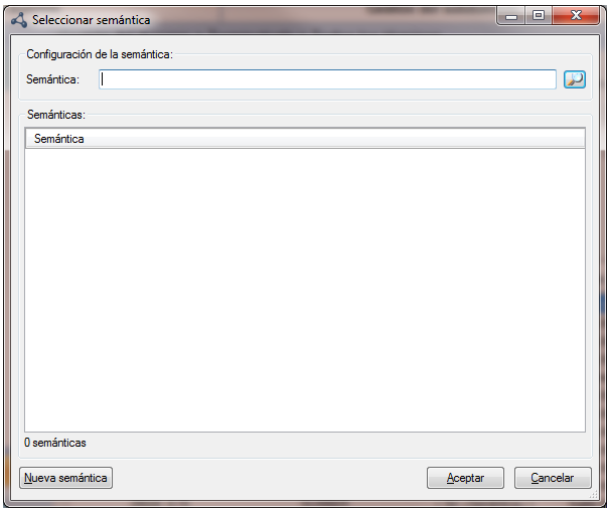

Ilustración 28: Selector de semántica

3.1.18 Especificación de Formatos de Impresión

No aplica.

Análisis de Consistencia y Especificación de Requisitos

Una parte de todo documento de análisis consiste en realizar un propio análisis sobre él. Los requisitos de un sistema deben ser **trazables**, es decir, se pueden identificar todos los elementos del producto relacionados con ese requisito. Así se mantiene la consistencia entre los distintos elementos de un proyecto.

Una matriz de trazabilidad es una buena opción para controlar las dependencias de cada requisito. Pero por temas de espacio y dimensiones de la tabla, se considera que la mejor opción es añadir un campo más, ***Trazabilidad***, en cada tabla de requisitos, de forma que se indican los identificadores de los requisitos con que traza cada uno de los requisitos.

Especificación del plan de pruebas

3.1.19 Definición del Alcance de las Pruebas

El sistema de información cuenta con un plan de pruebas que garantiza que nuestro sistema cumple las necesidades requeridas por el usuario, además de la calidad necesaria.

En este nivel de análisis de la aplicación se desarrollan las pruebas de aceptación, que certifican a alto nivel que la aplicación valida los requisitos especificados por el cliente.

Sin embargo, estas pruebas de aceptación no son lo suficiente detalladas como para decir que el sistema es fiable 100%. Es por ello que durante la fase de

diseño se detallan las pruebas unitarias realizadas, con el fin de obtener un producto funcional y fiable.

3.1.20 Definición de Requisitos del entorno de Pruebas

El entorno de pruebas mínimo es el equivalente al equipo de desarrollo, ya que es el entorno mínimo donde se ejecutará la aplicación una vez acabe la fase de desarrollo. Dichos requisitos son:

- Requisitos software:
  - Microsoft Framework 4.0
  - Visual Studio 2010 Professional
  - SQL Server 2008
- Requisitos hardware:
  - Procesador Intel Pentium Dual Core 2.0GHz
  - 2048 MB de memoria RAM
  - 256 GB totales de disco duro

3.1.21 Definición de las Pruebas de Aceptación del Sistema

Para facilitar el entendimiento de este apartado, al igual que en muchos otros, se dividen las pruebas por subsistema de diseño.

3.1.21.1 Terminología

PA-001: Crear un término	
Descripción:	Se comprueba que se puede crear un nuevo término en la ontología. a) Acceder al listado de todos los términos. b) Hacer click con el botón derecho y elegir la opción 'Añadir término'. c) Introducir los siguientes datos en el nuevo formulario: • Nombre: Termino1 • Etiqueta sintáctica: NOUN • Semántica: dejar el campo vacío • Lenguaje: Español • Impacto en el dominio: 0

	• Impacto en el lenguaje: 0 • Pertenece al dominio: desmarcado d) Hacer click en el botón Aceptar.
Resultado:	Tras hacer click en el botón Aceptar, en el listado de todos los términos aparece el nuevo término 'Termino1' que se acaba de crear.
Requisitos relacionados:	RSF-002, RSF-003, RSF-004

Tabla 113: PA-001

PA-002: Ver los detalles de un término	
Descripción:	Se comprueba que se pueden ver los detalles de un término ya existente en la ontología a) Acceder al listado de todos los términos. b) Hacer doble click sobre el término 'Termino1' creado previamente.
Resultado:	Tras hacer doble click sobre dicho término, se abre un nuevo formulario donde se muestran los datos del término.
Requisitos relacionados:	RSF-002, RSF-003, RSF-005

Tabla 114: PA-002

PA-003: Modificar un término	
Descripción:	Se comprueba que se puede modificar un término ya creado previamente en la ontología. a) Acceder al listado de todos los términos. b) Seleccionar con el botón izquierdo del ratón el término insertado 'Termino1' previamente. c) Hacer click con el botón derecho y elegir la opción 'Modificar término'. d) Modificar el siguiente dato en el nuevo formulario: • **Nombre**: Termino1_Modificado e) Hacer click en el botón Aceptar.

	Tras hacer click en el botón Aceptar, en el listado de todos los términos sigue habiendo un solo término, pero esta vez tiene el nombre 'Termino1_Modificado'.
Requisitos relacionados:	RSF-005

Tabla 115: PA-003

PA-004: Comprobar campos obligatorios de un término	
Descripción:	Se comprueba que los campos obligatorios a la hora de rellenar la información de un término son correctos. a) Acceder al listado de todos los términos. b) Hacer doble click con el botón izquierdo del ratón sobre el término modificado 'Termino1_Modificado' previamente. c) Modificar el siguiente dato en el nuevo formulario: • **Nombre**: *dejar el campo vacío*. d) Hacer click en el botón Aceptar: debe aparecer un mensaje de error sobre la interfaz avisando de que el 'Nombre' es un campo obligatorio. e) Modificar los siguientes datos: • **Nombre**: Termino1_Modificado • **Etiqueta sintáctica**: *dejar el campo vacío*. f) Hacer click en el botón Aceptar: debe aparecer un mensaje de error sobre la interfaz avisando de que la 'Etiqueta sintáctica' es un campo obligatorio. g) Modificar los siguientes datos: • **Etiqueta sintáctica**: NOUN h) Hacer click en el botón Aceptar.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso. Una vez se hace el último click sobre el botón Aceptar, se vuelve al listado de todos los términos con la misma información que había antes de comenzar esta prueba, ya que el estado final es igual que el estado inicial.

Requisitos relacionados:	RSF-002, RSF-003, RSF-004, RSF-005

Tabla 116: PA-004

PA-005: Duplicar un término	
Descripción:	Se comprueba que se puede duplicar la información de un término ya creado previamente en la ontología. a) Acceder al listado de todos los términos. b) Seleccionar con el botón izquierdo del ratón el término insertado 'Termino1_Modificado' previamente. c) Hacer click con el botón derecho y elegir la opción 'Duplicar término'. d) Confirmar el duplicado del término. e) Una vez se ha abierto el diálogo de los detalles del nuevo término, introducir los siguientes datos: • **Nombre**: Termino1_Duplicado f) Hacer click en el botón Aceptar.
Resultado:	Tras hacer click en el botón Aceptar, en el listado de todos los términos ahora hay dos términos, uno con el nombre 'Termino1_Modificado' y otro con el nombre 'Termino1_Duplicado'.
Requisitos relacionados:	RSF-006

Tabla 117: PA-005

PA-006: Buscar un término mediante filtros de búsqueda	
Descripción:	Se comprueba que el filtro de búsqueda funciona correctamente. a) Acceder al listado de todos los términos. b) Escribir los siguientes datos en el filtro de búsqueda: • **Nombre**: TerminoQueNoExiste c) Hacer click sobre el botón 'Buscar': no se debe mostrar ningún resultado en el listado de todos los términos, ya que no hay ningún término que coincida con los

	criterios de búsqueda introducidos. d) Borrar el criterio de búsqueda introducido previamente en el campo Nombre. e) Seleccionar los siguientes datos en el filtro de búsqueda: • **Etiqueta sintáctica**: NOUN f) Hacer click sobre el botón 'Buscar': se deben mostrar los dos términos, 'Termino1_Modificado' y 'Termino1_Duplicado', ya que ambos están vinculados a dicha etiqueta sintáctica. g) Borrar el criterio de búsqueda introducido previamente en el campo Etiqueta sintáctica. h) Escribir los siguientes datos en el filtro de búsqueda: • **Nombre**: Termino1_Duplicado i) Hacer click con el botón 'Buscar': se debe mostrar un solo término, cuyo nombre es 'Termino1_Duplicado'.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso.
Requisitos relacionados:	RSF-002, RSF-008

Tabla 118: PA-006

PA-007: Eliminar un término	
Descripción:	Se comprueba que se puede eliminar un término ya creado previamente en la ontología. a) Acceder al listado de todos los términos. b) Seleccionar con el botón izquierdo del ratón el término insertado 'Termino1_Modificado' previamente. c) Hacer click con el botón derecho y elegir la opción 'Eliminar término'. d) Confirmar el borrado del término. e) Seleccionar con el botón izquierdo del ratón el término insertado 'Termino1_Duplicado' previamente. f) Hacer click con el botón derecho y elegir la opción 'Eliminar término'.

	g) Confirmar el borrado del término.
Resultado:	Tras confirmar el último borrado y volver al listado de todos los términos, no debe haber ningún término.
Requisitos relacionados:	RSF-002, RSF-007

Tabla 119: PA-007

3.1.21.2 Etiquetas sintácticas

PA-008: Crear una etiqueta sintáctica	
Descripción:	Se comprueba que se puede crear una nueva etiqueta sintáctica en la ontología. a) Acceder al listado de etiquetas sintácticas. b) Hacer click con el botón derecho y elegir la opción 'Añadir etiqueta sintáctica. c) Introducir los siguientes datos en el nuevo formulario: • Nombre: Etiqueta1 • Etiqueta sintáctica padre: NOUN • Estadísticas del dominio: 0,0 • App. Dominio: 0 • Estadísticas del corpus: 0,0 • App. Corpus: 0 • Tipo 'Nombre': desmarcado • Tipo 'Verbo': desmarcado d) Hacer click en el botón Aceptar.
Resultado:	Tras hacer click en el botón Aceptar, en el listado de etiquetas sintácticas aparece la nueva etiqueta sintáctica 'Etiqueta1' que se acaba de crear.
Requisitos relacionados:	RSF-010, RSF-011

Tabla 120: PA-008

PA-009: Ver los detalles de una etiqueta sintáctica	
Descripción:	Se comprueba que se pueden ver los detalles de una etiqueta sintáctica ya existente en la ontología a) Acceder al listado de etiquetas sintácticas. b) Hacer doble click sobre la etiqueta sintáctica 'Etiqueta1' creada previamente.
Resultado:	Tras hacer doble click sobre dicha etiqueta sintáctica, se abre un nuevo formulario donde se muestran sus datos.
Requisitos relacionados:	RSF-009, RSF-010

Tabla 121: PA-009

PA-010: Modificar una etiqueta sintáctica	
Descripción:	Se comprueba que se puede modificar una etiqueta sintáctica ya creada previamente en la ontología. a) Acceder al listado de etiquetas sintácticas. b) Seleccionar con el botón izquierdo del ratón la etiqueta sintáctica insertada 'Etiqueta1' previamente. c) Hacer click con el botón derecho y elegir la opción 'Modificar etiqueta sintáctica'. d) Modificar el siguiente dato en el nuevo formulario: • **Nombre**: Etiqueta1_Modificada e) Hacer click en el botón Aceptar.
	Tras hacer click en el botón Aceptar, en el listado de etiquetas sintácticas sigue habiendo una sola etiqueta, pero esta vez tiene el nombre 'Etiqueta1_Modificada'.
Requisitos relacionados:	RSF-009, RSF-010, RSF-12

Tabla 122: PA-010

PA-011: Comprobar campos obligatorios de una etiqueta sintáctica	
Descripción:	Se comprueba que los campos obligatorios a la hora de rellenar la información de una etiqueta sintáctica son

	correctos. a) Acceder al listado de etiquetas sintácticas. b) Hacer doble click con el botón izquierdo del ratón sobre la etiqueta sintáctica 'Etiqueta1_Modificada' previamente. c) Modificar el siguiente dato en el nuevo formulario: • **Nombre**: *dejar el campo vacío*. d) Hacer click en el botón Aceptar: debe aparecer un mensaje de error sobre la interfaz avisando de que el 'Nombre' es un campo obligatorio. e) Modificar los siguientes datos: • **Nombre**: Etiqueta1_Modificada • **Etiqueta sintáctica padre**: *dejar el campo vacío*. f) Hacer click en el botón Aceptar: debe aparecer un mensaje de error sobre la interfaz avisando de que la 'Etiqueta sintáctica padre' es un campo obligatorio. g) Modificar los siguientes datos: • **Etiqueta sintáctica padre**: NOUN h) Hacer click en el botón Aceptar.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso. Una vez se hace el último click sobre el botón Aceptar, se vuelve al listado de etiquetas sintácticas con la misma información que había antes de comenzar esta prueba, ya que el estado final es igual que el estado inicial.
Requisitos relacionados:	RSF-010, RSF-011, RSF-012

Tabla 123: PA-011

PA-012: Duplicar una etiqueta sintáctica	
Descripción:	Se comprueba que se puede duplicar la información de una etiqueta sintáctica ya creada previamente en la ontología. a) Acceder al listado de etiquetas sintácticas. b) Seleccionar con el botón izquierdo del ratón la etiqueta sintáctica insertada 'Etiqueta1_Modificada' previamente. c) Hacer click con el botón derecho y elegir la opción 'Duplicar etiqueta sintáctica'. d) Confirmar el duplicado de la etiqueta sintáctica. e) Una vez se ha abierto el diálogo de los detalles de la nueva etiqueta sintáctica, introducir los siguientes datos: • **Nombre**: Etiqueta1_Duplicada f) Hacer click en el botón Aceptar.
Resultado:	Tras hacer click en el botón Aceptar, en el listado de etiquetas sintácticas ahora hay dos etiquetas sintácticas, uno con el nombre 'Etiqueta1_Modificada' y otro con el nombre 'Etiqueta1_Duplicada'.
Requisitos relacionados:	RSF-013

Tabla 124: PA-012

PA-013: Buscar una etiqueta sintáctica mediante filtros de búsqueda	
Descripción:	Se comprueba que el filtro de búsqueda funciona correctamente. a) Acceder al listado de etiquetas sintácticas. b) Escribir los siguientes datos en el filtro de búsqueda: • **Etiqueta sintáctica**: EtiquetaQueNoExiste c) Hacer click sobre el botón 'Buscar': no se debe mostrar ningún resultado en el listado de etiquetas sintácticas, ya que no hay ninguna etiqueta sintáctica que coincida con los criterios de búsqueda introducidos. d) Borrar el criterio de búsqueda introducido previamente en el campo Etiqueta sintáctica.

	e) Escribir los siguientes datos en el filtro de búsqueda: • **Etiqueta sintáctica**: Etiqueta1_Duplicada f) Hacer click con el botón 'Buscar': se debe mostrar una sola etiqueta sintáctica, cuyo nombre es 'Etiqueta1_Duplicada'.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso.
Requisitos relacionados:	RSF-009, RSF-015

Tabla 125: PA-013

PA-014: Eliminar una etiqueta sintáctica	
Descripción:	Se comprueba que se puede eliminar una etiqueta sintáctica ya creada previamente en la ontología. a) Acceder al listado de etiquetas sintácticas. b) Seleccionar con el botón izquierdo del ratón la etiqueta sintáctica insertada previamente 'Etiqueta1_Modificada'. c) Hacer click con el botón derecho y elegir la opción 'Eliminar etiqueta sintáctica'. d) Confirmar el borrado de la etiqueta sintáctica. e) Seleccionar con el botón izquierdo del ratón la etiqueta sintáctica insertada previamente 'Etiqueta1_Duplicada'. f) Hacer click con el botón derecho y elegir la opción 'Eliminar etiqueta sintáctica'. g) Confirmar el borrado del término.
Resultado:	Tras confirmar el último borrado y volver al listado de etiquetas sintácticas, no debe haber ningún elemento.
Requisitos relacionados:	RSF-014

Tabla 126: PA-014

3.1.21.3 Semánticas

PA-015: Crear una semántica	
Descripción:	Se comprueba que se puede crear una nueva semántica en la ontología. e) Acceder al listado de semánticas. f) Hacer click con el botón derecho y elegir la opción 'Añadir semántica'. g) Introducir los siguientes datos en el nuevo formulario: • Nombre: Semántica1 • Ponderación: 0 • Etiqueta sintáctica normalizada: vacío • Rotate concept: desmarcado • Permitir reflexiva: desmarcado • Permitir bucles: desmarcado • Permitir transitiva: desmarcado • Simétrica: desmarcado • Mostrar en el tesauro: desmarcado h) Hacer click en el botón Aceptar.
Resultado:	Tras hacer click en el botón Aceptar, en el listado de semánticas aparece la nueva semántica 'Semántica1' que se acaba de crear.
Requisitos relacionados:	RSF-023, RSF-024

Tabla 127: PA-015

PA-016: Ver los detalles de una semántica	
Descripción:	Se comprueba que se pueden ver los detalles de una semántica ya existente en la ontología a) Acceder al listado de semánticas. b) Hacer doble click sobre la semántica 'Semántica1' creada previamente.
Resultado:	Tras hacer doble click sobre dicha semántica, se abre un

	nuevo formulario donde se muestran sus datos.
Requisitos relacionados:	RSF-022, RSF-023

Tabla 128: PA-016

PA-017: Modificar una semántica	
Descripción:	Se comprueba que se puede modificar una semántica ya creada previamente en la ontología. a) Acceder al listado de semánticas. b) Seleccionar con el botón izquierdo del ratón la semántica insertada 'Semántica1' previamente. c) Hacer click con el botón derecho y elegir la opción 'Modificar semántica'. d) Modificar el siguiente dato en el nuevo formulario: • **Nombre**: Semántica1_Modificada e) Hacer click en el botón Aceptar.
	Tras hacer click en el botón Aceptar, en el listado de semánticas sigue habiendo una sola semántica, pero esta vez tiene el nombre 'Semántica1_Modificada'.
Requisitos relacionados:	RSF-022, RSF-023, RSF-025

Tabla 129: PA-017

PA-018: Comprobar campos obligatorios de una semántica	
Descripción:	Se comprueba que los campos obligatorios a la hora de rellenar la información de una semántica son correctos. a) Acceder al listado de semánticas. b) Hacer doble click con el botón izquierdo del ratón sobre la semántica 'Semántica1_Modificada' previamente. c) Modificar el siguiente dato en el nuevo formulario: • **Nombre**: *dejar el campo vacío*. d) Hacer click en el botón Aceptar: debe aparecer un mensaje de error sobre la interfaz avisando de que el 'Nombre' es un campo obligatorio.

	e) Modificar los siguientes datos: • **Nombre**: Semántica1_Modificada f) Hacer click en el botón Aceptar.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso. Una vez se hace el último click sobre el botón Aceptar, se vuelve al listado de semánticas con la misma información que había antes de comenzar esta prueba, ya que el estado final es igual que el estado inicial.
Requisitos relacionados:	RSF-024, RSF-025

Tabla 130: PA-018

PA-019: Duplicar una semántica	
Descripción:	Se comprueba que se puede duplicar la información de una semántica ya creada previamente en la ontología. a) Acceder al listado de semánticas. b) Seleccionar con el botón izquierdo del ratón la semántica insertada 'Semántica1_Modificada' previamente. c) Hacer click con el botón derecho y elegir la opción 'Duplicar semántica'. d) Confirmar el duplicado de la semántica. e) Una vez se ha abierto el diálogo de los detalles de la nueva semántica, introducir los siguientes datos: • **Nombre**: Semántica1_Duplicada f) Hacer click en el botón Aceptar.
Resultado:	Tras hacer click en el botón Aceptar, en el listado de semánticas ahora hay dos etiquetas sintácticas, uno con el nombre 'Semántica1_Modificada' y otro con el nombre 'Semántica1_Duplicada'.
Requisitos relacionados:	RSF-026

Tabla 131: PA-019

PA-020: Buscar una semántica mediante filtros de búsqueda	
Descripción:	Se comprueba que el filtro de búsqueda funciona correctamente. g) Acceder al listado de semánticas. h) Escribir los siguientes datos en el filtro de búsqueda: • **Semántica**: SemanticaQueNoExiste i) Hacer click sobre el botón 'Buscar': no se debe mostrar ningún resultado en el listado de semánticas, ya que no hay ninguna semántica que coincida con los criterios de búsqueda introducidos. j) Borrar el criterio de búsqueda introducido previamente en el campo Semántica. k) Escribir los siguientes datos en el filtro de búsqueda: • **Semántica**: Semántica1_Duplicada l) Hacer click con el botón 'Buscar': se debe mostrar una sola semántica, cuyo nombre es 'Semántica1_Duplicada'.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso.
Requisitos relacionados:	RSF-022, RSF-028

Tabla 132: PA-020

PA-021: Eliminar una semántica	
Descripción:	Se comprueba que se puede eliminar una semántica ya creada previamente en la ontología. a) Acceder al listado de semánticas. b) Seleccionar con el botón izquierdo del ratón la semántica insertada previamente 'Semántica1_Modificada'. c) Hacer click con el botón derecho y elegir la opción 'Eliminar semántica'. d) Confirmar el borrado de la semántica. e) Seleccionar con el botón izquierdo del ratón la

	semántica insertada previamente 'Semántica1_Duplicada'. f) Hacer click con el botón derecho y elegir la opción 'Eliminar semántica'. g) Confirmar el borrado del término.
Resultado:	Tras confirmar el último borrado y volver al listado de semánticas, no debe haber ningún elemento.
Requisitos relacionados:	RSF-027

Tabla 133: PA-021

3.1.21.4 Sugerencias

PA-022: Modificar una sugerencia	
Descripción:	Se comprueba que se puede modificar una sugerencia ya existente en base de datos. Ya que el sistema no permite crear sugerencias, para llevar a cabo esta prueba es condición indispensable que exista al menos una sugerencia ya creada. a) Acceder al listado de sugerencias de términos. b) Hacer doble click con el botón izquierdo del ratón sobre una sugerencia. c) Modificar el siguiente dato en el nuevo formulario: • **Respuesta**: Respuesta para PA-022 • **Status**: Aprobada d) Hacer click en el botón Aceptar.
	Tras hacer click en el botón Aceptar, en el listado de sugerencias sigue existiendo dicha sugerencia, pero su información ha sido cambiada correctamente.
Requisitos relacionados:	RSF-036, RSF-037, RSF-038, RSF-039

Tabla 134: PA-022

PA-023: Comprobar campos obligatorios de una sugerencia

Descripción:	Se comprueba que los campos obligatorios a la hora de rellenar la información de una sugerencia son correctos. Ya que el sistema no permite crear sugerencias, para llevar a cabo esta prueba es condición indispensable que exista al menos una sugerencia ya creada. a) Acceder al listado de sugerencias de términos. b) Hacer doble click con el botón izquierdo del ratón sobre una sugerencia. c) Modificar el siguiente dato en el nuevo formulario: • **Respuesta**: *dejar el campo vacío*. d) Hacer click en el botón Aceptar: debe aparecer un mensaje de error sobre la interfaz avisando de que la 'Respuesta' es un campo obligatorio. e) Modificar los siguientes datos: • **Respuesta**: Respuesta para PA-023 f) Hacer click en el botón Aceptar.
Resultado:	En la 'Descripción' de esta misma tabla se especifica el correcto *feedback* que debe dar la aplicación durante el proceso. Una vez se hace el último click sobre el botón Aceptar, se vuelve al listado de sugerencias de términos.
Requisitos relacionados:	RSF-038, RSF-039

Tabla 135: PA-023

4 Diseño

Introducción

El objetivo de este capítulo es elaborar el diseño de la solución con un nivel de profundidad mayor del obtenido en el capítulo de análisis, donde es capturaron a más alto nivel las necesidades que debe cumplir el sistema de información.

El objetivo del sistema a desarrollar es satisfacer los requisitos planteados durante la fase de análisis. En resumen, el objetivo es crear de forma satisfactoria un módulo que gestione correctamente los subsistemas estudiados, y que sea tanto verificable como validado por parte del cliente.

Definición de la arquitectura del sistema

A continuación se definen aspectos de la arquitectura del sistema. Se lleva a cabo un estudio a nivel global, es decir, la interacción del módulo desarrollado con el resto del sistema. Además se definen las excepciones que se pueden dar en el sistema, y se establece la normativa de código que se aplica durante el desarrollo.

4.1.1 Definición de niveles de arquitectura

Cada subsistema de diseño aplica una **arquitectura basada en tres capas**, de modo que se continúa dividiendo la complejidad de cada subsistema.

- **Capa de presentación**: también denominada *PL (Presentation Layer)*. Separa la responsabilidad de mostrar la interfaz de usuario y la interacción con el mismo. Haciendo uso de esta capa, se utiliza el patrón de diseño *facade*.
- **Capa de negocio**: también denominada *BLL (Business Logic Layer)*. Tiene la responsabilidad de almacenar todas las funciones que determinan la lógica de negocio de la aplicación.
- **Capa de datos**: también denominada *DAL (Data Access Logic)*. Su responsabilidad es realizar la interacción con la base de datos, realizando una gestión óptima de las operaciones sobre base de datos.

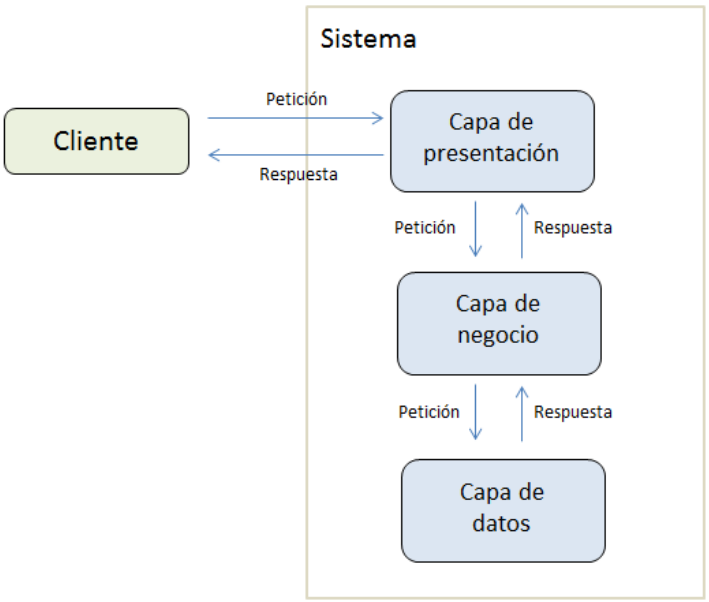

Ilustración 29: Diseño del modelo arquitectónico en tres capas

4.1.2 Especificación de excepciones

Se ha diseñado el módulo de *knowledgeManager* para evitar el lanzamiento de excepciones.

Por un lado, las excepciones que se capturan en la BLL son registradas en un fichero de *log* y además devuelven valores nulos a la PL para saber que algo ha ido mal durante el proceso.

Por el contrario, las excepciones que se capturan en la PL suelen ser debidas a fallos de interacción entre el sistema y el usuario, por lo que al ser capturadas muestran un mensaje al usuario tipo *MessageBox*, además de ser registradas en el *log*.

4.1.3 Especificación de estándares y normas de diseño y construcción

Para la codificación del sistema es necesario seguir una serie de normas para mantener homogeneidad de código en todo el proyecto:

- **Atributos**: se escriben en minúscula. Si el nombre del atributo abarca más de una palabra, se escribe la inicial de cada palabra en mayúscula, exceptuando la primera que es íntegra en minúscula.
- **Constantes**: se escriben en mayúscula. Si el nombre de la constante abarca más de una palabra, se separan mediante guión bajo.
- **Funciones**: se escriben en minúscula, excepto la primera letra de cada palabra, que se escribe en mayúscula.

- **Clases**: se escriben en minúscula, excepto la primera letra de cada palabra, que se escribe en mayúscula.

No es totalmente estricto, pero tanto los comentarios como el nombre de los atributos, constantes, funciones y clases deben ser en inglés. Cualquier norma que no esté especificada en este estándar, es decisión del desarrollador.

Diseño de la arquitectura de diseño

Uno de los principales objetivos de un buen diseño debe ser la separación de responsabilidades. Cualquier sistema de información debe permitir ser **escalable**, con el fin de facilitar en la medida de lo posible los cambios de requisitos que se puedan dar durante (o tras) el desarrollo de la aplicación. Por una parte **se reduce la complejidad del sistema**, y por otra **se facilita su mantenimiento**.

4.1.4 Diseño de subsistemas de diseño

Con el fin de comenzar a dividir las responsabilidades del sistema, durante la fase de análisis se dividió el sistema completo en varios subsistemas, dependiendo de la funcionalidad que se desarrolle en cada uno de ellos. Se mantienen los subsistemas diseñados previamente y se especifican formalmente:

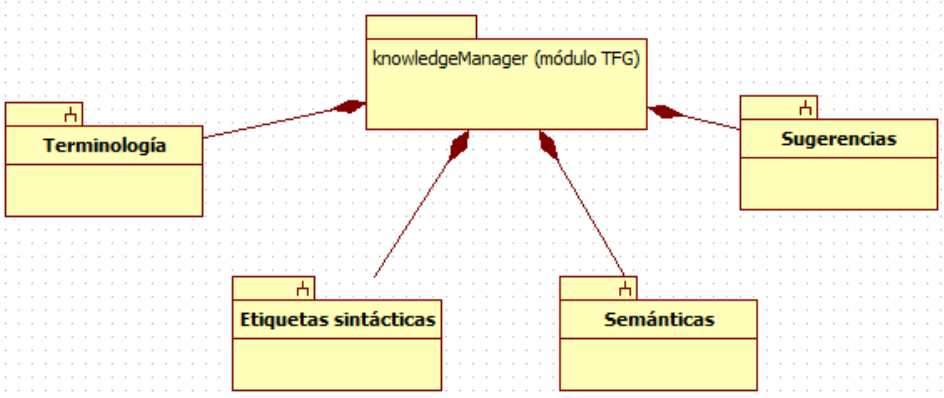

Ilustración 30: Subsistemas de diseño de la aplicación

Por otra parte, el módulo desarrollado durante este TFG se acopla dentro del proyecto completo del *knowledgeManager*. Por ello, es necesario que este módulo interactúe con otros componentes, lo que supone el uso de librerías externas ya desarrolladas.

A continuación se muestra el diagrama de interacciones en las que interviene este módulo con el resto del sistema:

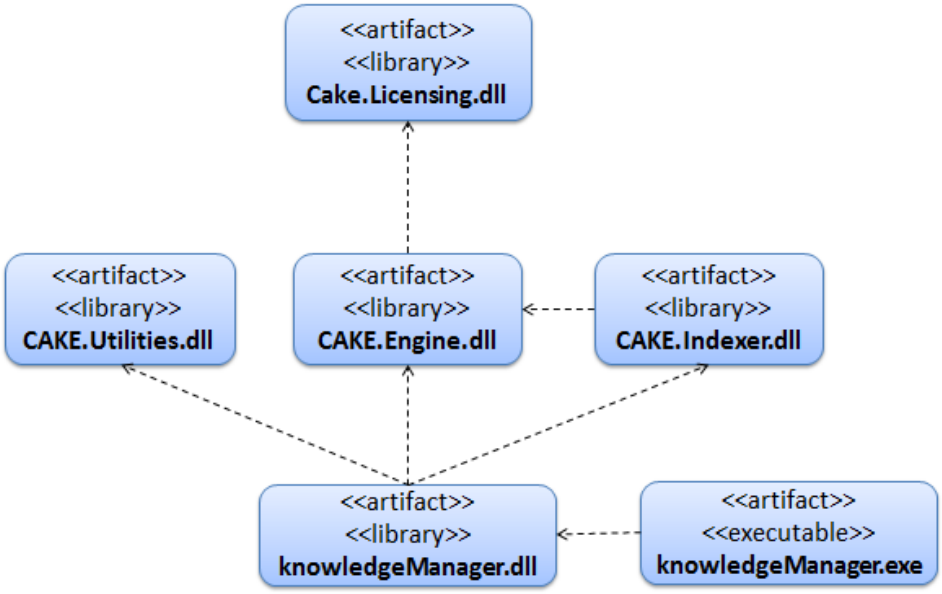

Ilustración 31: Diseño de componentes del sistema

Completando al diagrama anterior, existen cuatro piedras angulares en las que el módulo a desarrollar se basa:

- **Cake.Engine**: lleva a cabo toda la comunicación con la base de datos. Es la capa DAL de toda la aplicación, y se inicializa una sola vez al iniciar la aplicación.
- **Cake.Indexer**: se encarga de todas las operaciones de *indexing* y *retrieval*, e interactúa con la base de datos a través del *Cake.Engine*.
- **Cake.Licensing**: lleva a cabo todo lo relacionado con el licenciamiento de la aplicación.
- **Cake.Utilities**: gestiona el formulario de conexión a la base de datos y la ventana de *splash* de espera.

Todos ellos se tratan de *sistemas caja negra*. Por ello no se profundiza en el análisis y diseño de cada uno de ellos.

Revisión de la interfaz de usuario

Se mantiene la interfaz de usuario definida en el análisis. Dichas interfaces se desarrollan mediante controles de usuario que son integrados en un formulario principal, común a toda la aplicación. Así, dependiendo la opción del menú que se utilice, se mantiene el mismo formulario exterior pero cambia el control de usuario que se utilice en cada momento.

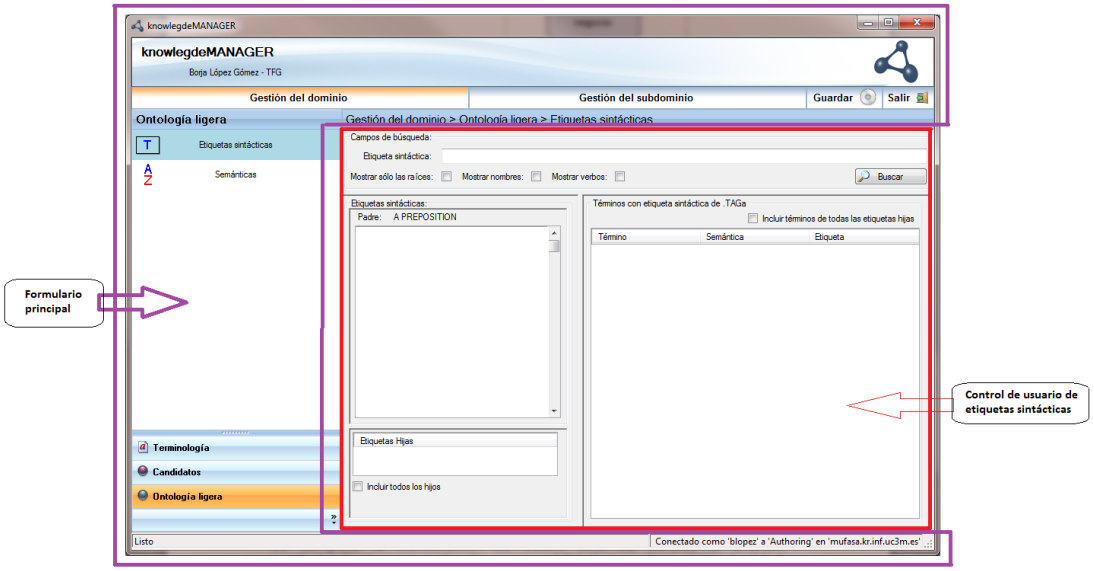

Ilustración 32: Diseño de la interfaz de usuario

Diseño de clases

A continuación se profundiza en el análisis de clases llevado a cabo durante el capítulo de análisis. Se parte de la identificación de clases que se hizo durante dicha fase, para completar todas las clases que se consideren necesarias para el desarrollo del sistema. Además se estudian en detalle cuáles son los atributos y funciones necesarias en cada clase.

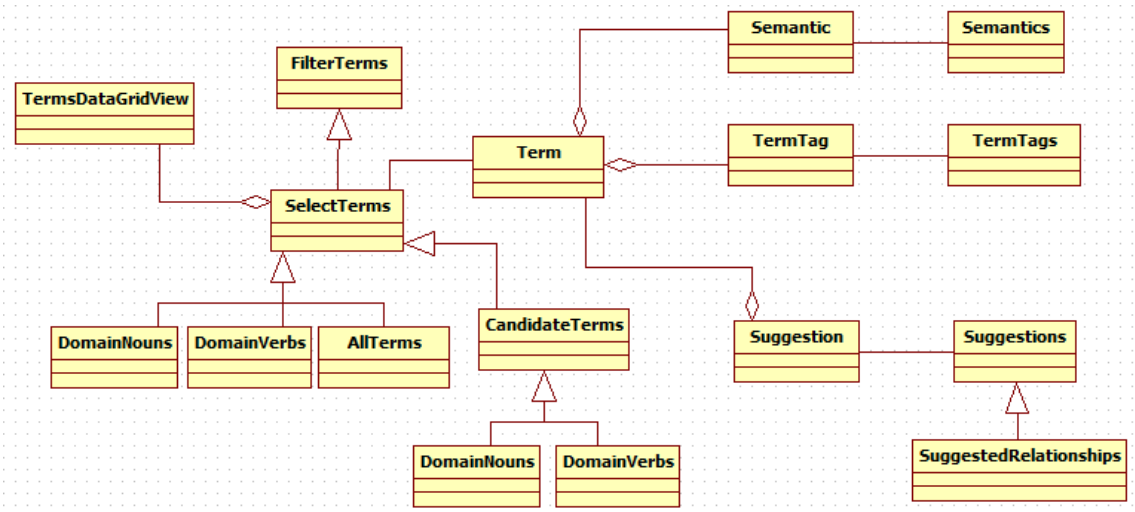

Ilustración 33: Diagrama de clases

4.1.5 Identificación de clases

Se completa la identificación de clases obtenida durante el análisis, manteniendo la separación de responsabilidades en tres capas (entidad, interfaz, control).

- Clases de entidad:
  - **VocabularyDataSet**: contiene los atributos que forman un término.
  - **RulesFamilies_DataSet**: contiene los atributos que forman una etiqueta sintáctica.
  - **GrammaticalDataSet**: contiene los atributos que forman una gramática.
  - **SuggestionDataSet**: contiene los atributos que forman una sugerencia.
- Clases de interfaz
  - **AllTerms**: muestra todos los términos.
  - **DomainNouns**: muestra los términos cuya etiqueta sintáctica sea nombre.
  - **DomainVerbs**: muestra los términos cuya etiqueta sintáctica sea verbo.
  - **FilterTerms**: contiene las opciones del filtro de búsqueda de términos.
  - **SelectTerms**: control de usuario que se emplea para incluir a FilterTerms y TermDataGridView, ya que se usa junto en varios sitios.
  - **TermDataGridView**: grid de datos empleado para mostrar un listado de términos.
  - **SelectTerm**: selector empleado para realizar búsqueda de términos fuera de cualquier formulario de terminología.
  - **Term**: muestra la información referida a un término.
  - **CandidateTerms**: muestra los términos candidatos a formar parte del dominio.
  - **DomainNouns**: muestra los términos candidatos a formar parte del dominio cuya etiqueta sintáctica sea nombre.
  - **DomainVerbs**: muestra los términos candidatos a formar parte del dominio cuya etiqueta sintáctica sea verbo.
  - **TermTags**: muestra todas las etiquetas sintácticas.
  - **TermTag**: muestra la información referida a una etiqueta sintáctica.

  - **SelectTermTag**: selector empleado para realizar búsqueda de etiquetas sintácticas fuera del formulario de etiquetas sintácticas.
  - **Semantics**: muestra todas las semánticas.
  - **Semantic**: muestra la información referida a una semántica.
  - **SelectSemantic**: selector empleado para realizar búsqueda de semánticas fuera del formulario de semánticas.
  - **Suggestions**: muestra todas las sugerencias de términos.
  - **SuggestedRelationship**: muestra todas las sugerencias de relaciones entre términos.
  - **Suggestion**: muestra la información referida a una sugerencia.
  - **DeleteErrors**: muestra mensajes de error al usuario cuando no se puede llevar a cabo una operación de borrado en cualquier subsistema.
- Clases de control
  - **VocabularyRepository**: contiene la lógica de negocio del subsistema de terminología.
  - **RulesFamiliesRepository**: contiene la lógica de negocio del subsistema de etiquetas sintácticas.
  - **GrammaticalsRepository**: contiene la lógica de negocio del subsistema de semánticas.
  - **SuggestionsRepository**: contiene la lógica de negocio del subsistema de sugerencias.

4.1.6 Especificación de clases

Una vez se han definido todas las clases que van a formar parte de la aplicación, se realiza una especificación detallada de las propiedades que tiene cada una de ellas. Se agrupan por subsistema.

4.1.6.1 Terminología

VocabularyDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• `private VocabularyDataTable vocabularyDataTable:` Tabla de datos que contiene los atributos referidos a un término. • `private TermForComboBoxDataTable vocabularyDataTableForComboBox:` Tabla de

	datos que contiene los atributos necesarios para mostrar un término en un control *ComboBox* desplegable.
Operaciones:	

Tabla 136: Diseño de VocabularyDataSet

VocabularyRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de terminología.
Atributos:	• `private CE_Director directorRepository:` objeto que gestiona la lógica de negocio para las funciones que requieran información del tesauro.
Operaciones:	• `public VocabularyDataSet.VocabularyDataTable GetTermsForDataGridView():` devuelve una tabla de datos con todos los términos de la ontología. • `public VocabularyDataSet.VocabularyDataTable GetTermsForDataGridView(Term_Tag selectedTermTag):` devuelve una tabla de datos con todos los términos de la ontología cuya etiqueta sintáctica sea la recibida por parámetro. • `public VocabularyDataSet.VocabularyDataTable GetTermsForDataGridView(SemanticItem selectedSemanticItem):` devuelve una tabla de datos con todos los términos de la ontología semántica sea la recibida por parámetro. • `public VocabularyDataSet.VocabularyDataTable GetTermsForDataGridView(String termName, Term_Tag termTag, Language language, SemanticItem semantic, bool belongsToDomain):` devuelve una tabla de datos con todos los términos de la ontología cuyos campos coincidan con los pasados por parámetro. • `public VocabularyDataSet.VocabularyDataTable`

	MakeAVocabularyDataTableFromListOfTerms (bool belongsToDomain, VocabularyDataSet.VocabularyDataTable dataTable, List<Term> termsList): devuelve una tabla de datos con todos los términos que contiene la lista pasada por parámetro. • public Term GetTermByName(String termName): devuelve el primer término de la ontología que contenga la cadena pasada por parámetro. • public Term GetTermByNameExact(String termName): devuelve el primer término de la ontología cuyo nombre coincida exactamente con la cadena pasada por parámetro. • public bool ValidateTerm(Term editingTerm, Term_Tag termTag, SemanticItem semantic, String termName, bool belongsToDomain, Language language, int domainImpact, int languageImpact): comprueba si el término y sus atributos (recibido todo como parámetro) es válido para ser almacenado en la ontología. Devuelve *false* en caso de no ser válido. • public bool UpdateTerm(Term editingTerm, Term_Tag termTag, SemanticItem semantic, String termName, bool belongsToDomain, Language language, int domainImpact, int languageImpact, Term scopeNoteTerm): tras invocar a la función ValidateTerm, envía a la capa de datos la petición para modificar la información de un término en la ontología. • public bool InsertTerm(Term_Tag termTag, SemanticItem semantic, String termName, bool belongsToDomain, Language language, int domainImpact, int languageImpact, string scopeNoteTerm, ref Term newTerm): tras invocar a la función ValidateTerm, envía a la capa de datos la petición para insertar la información de un nuevo término en la ontología. • public Term InsertBlankTerm(Term_Tag termTag, Language language): inserta un término por defecto con la etiqueta sintáctica y el

	lenguaje pasados por parámetro. • `public bool DuplicateTerm(Term originalTerm, ref Term newTerm)`: envía a la capa de datos la petición para crear un nuevo término a partir de la información de uno ya existente. • `public bool ValidateDeleting(Term deletingTerm, ref List<Special_Sentence> specialSentences, ref List<Contraction> contractions, ref List<S_Exception> substituteException, ref List<Artifact> artifactsWithKes, ref List<ClarifierRule> clarifierRule, ref List<Pattern_SLOT> boilerplates, ref List<Artifact> artifactsWithMetas, ref List<Qualification> qualification, ref List<Locution> locution, ref List<Suggestion> suggestions)`: comprueba si existe algún element que impida el borrado de un término, debido a las restricciones de integridad existentes a nivel de base de datos. Devuelve *false* en caso de no ser posible el borrado del término. • `public bool DeleteTerm(Term deletingTerm)`: envía a la capa de datos la petición de borrado del término.

Tabla 137 Diseño de VocabularyRepository

4.1.6.2 Etiquetas sintácticas

Rules_FamiliesDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• `private TermTagsDataTable termTagsDataTable`: Tabla de datos que contiene los atributos referidos a una etiqueta sintáctica. • `private TermTagForComboBoxDataTable termTagsDataTableForComboBox`: Tabla de datos que contiene los atributos necesarios para mostrar una etiqueta sintáctica en un control ComboBox desplegable.

Operaciones:	

Tabla 138 Diseño de Rules_FamiliesDataSet

RulesFamiliesRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de etiquetas sintácticas.
Atributos:	• `private static List<RulesFamilies> compulsoryTermTags:` lista de las etiquetas sintácticas de carácter obligatorio.
Operaciones:	• `public Rules_FamiliesDataSet.TermTagForComboBoxDataTable GetTermTagsForComboBox():` devuelve una tabla de datos con todas las etiquetas sintácticas de la ontología, adaptada para controles *ComboBox* desplegables. • `public Rules_FamiliesDataSet.TermTagsDataTable GetTermTagsForDataGridView():` devuelve una tabla de datos con todas las etiquetas sintácticas de la ontología. • `public Rules_FamiliesDataSet.TermTagsDataTable GetTermTagsOnlyRootForDataGridView():` devuelve una tabla de datos con las etiquetas sintácticas de la ontología que no tengan ningún padre (roots). • `public Rules_FamiliesDataSet.TermTagsDataTable GetTermTagsForDataGridView(Term_Tag selectedTermTag):` devuelve una tabla de datos con todas las etiquetas sintácticas de la ontología que coincidan con la etiqueta sintáctica pasada por parámetro. • `public Rules_FamiliesDataSet.TermTagsDataTable GetTermTagsForDataGridView(String termTagName):` devuelve una tabla de datos con todas las etiquetas sintácticas de la ontología cuyos nombre contenga la cadena pasada por parámetro. • `public`

	`Rules_FamiliesDataSet.TermTagsDataTable GetTermTagsForDataGridView(String termTagName, String onlyRoots, bool nounType, bool verbType)`: devuelve una tabla de datos con todas las etiquetas sintácticas de la ontología cuyos atributos coincidan con los indicados por parámetro. • `public bool ValidateTermTag(Term_Tag editingTermTag, string name, Term_Tag parent, double domainStats, int appDomain, double corpusStats, int appCorpus)`: comprueba si la etiqueta sintáctica y sus atributos (recibido todo como parámetro) es válida para ser almacenada en la ontología. Devuelve *false* en caso de no ser válida. • `public bool UpdateTermTag(Term_Tag editingTermTag, string name, Term_Tag parent, double domainStats, int appDomain, double corpusStats, int appCorpus, bool nounType, bool verbType)`: tras invocar a la función `ValidateTermTag`, envía a la capa de datos la petición para modificar la información de una etiqueta sintáctica en la ontología. • `public bool InsertTermTag(string name, Term_Tag parent, double domainStats, int appDomain, double corpusStats, int appCorpus, bool nounType, bool verbType, ref Term_Tag newTermTag)`: tras invocar a la función `ValidateTermTag`, envía a la capa de datos la petición para insertar la información de una nueva etiqueta sintáctica en la ontología. • `public bool DuplicateTerm(Term_Tag originalTermTag, ref Term_Tag newTermTag)`: envía a la capa de datos la petición para crear una nueva etiqueta sintáctica a partir de la información de una ya existente. • `public bool ValidateDeleting(Term_Tag editingTermTag, ref List<Term> terms, ref List<Role> roles, ref List<Rule> rules, ref List<BigramStatistics> bigrams, ref List<Qualification> qualifications, ref List<ClarifierRule> oldInferenceRules, ref List<ClarifierRule> taggingRules, ref List<Pattern> inferenceRules, ref`

	`List<Locution> locutions, ref List<Term_Tag> termTags):` comprueba si existe algún elemento que impida el borrado de una etiqueta sintáctica, debido a las restricciones de integridad existentes a nivel de base de datos. Devuelve *false* en caso de no ser posible el borrado de la etiqueta sintáctica. • `public bool DeleteTermTag(Term_Tag deletingTermTag):` envía a la capa de datos la petición de borrado de la etiqueta sintáctica.

Tabla 139 Diseño de RulesFamiliesRepository

TermTags	
Responsabilidades:	Interfaz de listado de etiquetas sintácticas.
Atributos:	• `private RulesFamiliesRepository termTagRepository:` objeto de la BLL que permite acceder a la lógica de negocio de etiquetas sintácticas. • `private VocabularyRepository vocabularyRepository:` objeto de la BLL que permite acceder a la lógica de negocio de terminología. • `private DataView termTagsDataView:` DataView que contiene la tabla de datos de las etiquetas sintácticas. • `private DataView subTermTagsDataView:` DataView que contiene la tabla de datos de las etiquetas sintácticas hijas. • `private DataView termsDataView:` DataView que contiene la tabla de datos de los términos asociados a la etiqueta sintáctica seleccionada. • `private Term_Tag selectedTermTag:` referencia a la etiqueta sintáctica seleccionada en cada momento.
Operaciones:	• `private void LoadTermTagsDataGridView():` realiza la petición a la BLL para obtener todas las etiquetas sintácticas en la ontología. • `private void ChangeSelectedTermTag():` actualiza etiquetas sintácticas hijas y términos cada vez que se cambia la etiqueta sintáctica

	padre seleccionada. • `private bool Search():` aplica los filtros de búsqueda. Devuelve *false* en caso de error. • `private void AddTermTag():` muestra el formulario para crear una nueva etiqueta sintáctica. • `private void EditTermTag(Term_Tag originalTermTag):` muestra el formulario para editar los datos de la etiqueta sintáctica seleccionada. • `private void DuplicateTermTag(Term_Tag originalTermTag):` realiza la petición a la BLL para crear una nueva etiqueta sintáctica con la información de la etiqueta sintáctica seleccionada. • `private bool DeleteTermTags(List<int> selectedIndexes):` realiza la petición a la BLL para eliminar las etiquetas sintácticas seleccionadas. • `private void LoadSubTermTagsDataGridView(Term_Tag selectedTermTag):` realiza la petición a la BLL para cargar las etiquetas sintácticas hijas en función de la etiqueta sintáctica seleccionada. • `private void LoadTermsDataGridView(Term_Tag selectedTermTag):` realiza la petición a la BLL para cargar los términos que estén asociados a la etiqueta sintáctica seleccionada. • `private void AddTerm():` abre el formulario para añadir un término a la etiqueta sintáctica seleccionada. • `private void EditTerms():` abre el formulario para mover términos a la etiqueta sintáctica seleccionada desde otra etiqueta sintáctica.

Tabla 140 Diseño de TermTags

TermTag	
Responsabilidades:	Interfaz para mostrar los datos de una etiqueta sintáctica.
Atributos:	• `private RulesFamiliesRepository termTagRepository:` objeto de la BLL que permite acceder a la lógica de negocio de etiquetas sintácticas. • `private Term_Tag editingTermTag:` referencia a la etiqueta sintáctica que se está editando, si existe.
Operaciones:	• `private void Accept():` invoca a las funciones de validación y realiza la correspondiente petición a la BLL para insertar o editar una etiqueta sintáctica. • `private void Cancel():` cancela los cambios llevados a cabo en el formulario. • `private bool ValidateNameTextBox():` valida que el nombre introducido no esté vacío. Devuelve *false* si no es válido. • `private bool ValidateTermTag():` valida que se introduzca una etiqueta sintáctica padre. Devuelve *false* si no es válido. • `private bool ValidateTermTagType():` valida que no se haya seleccionado tanto *nombre* como *verbo* para el tipo de etiqueta sintáctica. Devuelve *false* si no es válido.

Tabla 141 Diseño de TermTag

SelectTermTag	
Responsabilidades:	Interfaz selector de una etiqueta sintáctica.
Atributos:	• `private RulesFamiliesRepository termTagRepository:` objeto de la BLL que permite acceder a la lógica de negocio de etiquetas sintácticas. • `private DataView termTagsDataView:` DataView que contiene la tabla de datos de etiquetas sintácticas. • `private Term_Tag editingTermTag:` referencia a la etiqueta sintáctica que se está

	editando, si existe.
Operaciones:	• `private bool Search(String termTagName):` realiza la petición a la BLL para obtener las etiquetas sintácticas que coincidan con el criterio de búsqueda introducido por el usuario. • `private void Accept():` almacena la etiqueta sintáctica elegida por el usuario y devuelve el control al formulario de origen. • `private void Cancel():` devuelve el control al formulario de origen.

Tabla 142: Diseño de SelectTermTag

4.1.6.3 Semánticas

GrammaticalDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• `private GrammaticalDataTable semanticsDataTable:` Tabla de datos que contiene los atributos referidos a una semántica. • `private GrammaticalForComboBoxDataTable semanticsDataTableForComboBox:` Tabla de datos que contiene los atributos necesarios para mostrar una semántica en un control *ComboBox* desplegable.
Operaciones:	

Tabla 143: Diseño de GrammaticalDataSet

GrammaticalsRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de semánticas.
Atributos:	
Operaciones:	• `public GrammaticalDataSet.GrammaticalDataTable GetSemantics():` devuelve una tabla de datos con todas las semánticas de la ontología. • `public GrammaticalDataSet.GrammaticalForComboBoxD`

	ataTable GetSemanticsForComboBox(): devuelve una tabla de datos con menos campos con todas las semánticas de la ontología. • public GrammaticalDataSet.GrammaticalForComboBoxDataTable GetSemantics(string semantic, bool usedBySystem, bool onlyRoots): devuelve una tabla de datos con las semánticas de la ontología que coincidan con los criterios de búsqueda recibidos por parámetro. • public GrammaticalDataSet.GrammaticalForComboBoxDataTable GetSemanticsWithoutFirstRowForComboBox(): devuelve una tabla de datos con todas las semánticas de la ontología. • public GrammaticalDataSet.GrammaticalForComboBoxDataTable GetSemanticsWithoutFirstRowForComboBox(SemanticItem semantic, bool includeAllChildren): devuelve una tabla de datos con las semánticas de la ontología cuya semántica coincida con la recibida por parámetro. • public GrammaticalDataSet.GrammaticalForComboBoxDataTable GetSemanticsWithoutFirstRowForComboBox(string semanticName): devuelve una tabla de datos con las semánticas de la ontología cuyo nombre contenga la cadena recibida por parámetro. • public bool ValidateSemantic(SemanticItem editingSemantic, string name, SemanticItem nomalizedSemanticItem, byte defaultPonderation, string roleA, string roleB, string roleAbreviationA, string roleAbreviationB): comprueba si la semántica y sus atributos (recibido todo como parámetro) es válida para ser almacenada en la ontología. Devuelve *false* en caso de no ser válida. • public bool UpdateSemantic(SemanticItem editingSemantic, string name, SemanticItem nomalizedSemanticItem, bool rotateConcept, byte defaultPonderation, bool reflexiveAllowed, bool loopsAllowed, bool transitiveAllowed, bool isSymmetrical, bool usedBySystem, string roleA, string roleB, string roleAbreviationA, string

	`roleAbreviationB)`: tras invocar a la función `ValidateSemantic`, envía a la capa de datos la petición para modificar la información de una semántica en la ontología. • `public InsertSemantic(string name, SemanticItem nomalizedSemanticItem, bool rotateConcept, byte defaultPonderation, bool reflexiveAllowed, bool loopsAllowed, bool transitiveAllowed, bool isSymmetrical, bool usedBySystem, string roleA, string roleB, string roleAbreviationA, string roleAbreviationB, ref SemanticItem newSemantic)`: tras invocar a la función `ValidateSemantic`, envía a la capa de datos la petición para insertar la información de una nueva semántica en la ontología. • `public bool DuplicateSemantic(SemanticItem semantic, ref SemanticItem newSemantic)`: envía a la capa de datos la petición para crear una nueva semántica a partir de la información de una ya existente. • `public bool ValidateDeleting(SemanticItem editingSemantic, ref List<Term> terms, ref List<Role> roles, ref List<ClarifierRule_BIT> clarifierRulesBit, ref List<Pattern_SLOT> patternSlots, ref List<SemanticItem> otherSubtypes)`: comprueba si existe algún elemento que impida el borrado de una semántica, debido a las restricciones de integridad existentes a nivel de base de datos. Devuelve *false* en caso de no ser posible el borrado de la semántica. • `public bool DeleteSemantic(SemanticItem deletingSemantic)`: envía a la capa de datos la petición de borrado de la semántica.

Tabla 144: Diseño de GrammaticalsRepository

Semantics	
Responsabilidades:	Interfaz de listado de semánticas.
Atributos:	• `private GrammaticalsRepository semanticsRepository:` objeto de la BLL que permite acceder a la lógica de negocio de semánticas. • `private VocabularyRepository vocabularyRepository:` objeto de la BLL que permite acceder a la lógica de negocio de terminología. • `private DataView semanticsDataView:` DataView que contiene la tabla de datos de las semánticas. • `private DataView subSemanticsDataView:` DataView que contiene la tabla de datos de las semánticas hijas. • `private DataView termsDataView:` DataView que contiene la tabla de datos de los términos asociados a la semántica seleccionada. • `private SemanticItem selectedSemantic:` referencia a la semántica seleccionada en cada momento.
Operaciones:	• `private void LoadSemanticsDataGridView():` realiza la petición a la BLL para obtener todas las semánticas de la ontología. • `private void ChangeSelectedSemantic():` actualiza semánticas hijas y términos cada vez que se cambia la semántica padre seleccionada. • `private bool Search():` aplica los filtros de búsqueda. Devuelve *false* en caso de error. • `private void AddSemantic():` muestra el formulario para crear una nueva semántica. • `private void EditSemantic (SemanticItem originalSemantic):` muestra el formulario para editar los datos de la semántica seleccionada. • `private void DuplicateSemantic(SemanticItem originalSemantic):` realiza la petición a la

	BLL para crear una nueva semántica con la información de la semántica seleccionada. • `private bool DeleteSemantics(List<int> selectedIndexes):` realiza la petición a la BLL para eliminar las semánticas seleccionadas. • `private void LoadSubSemanticsDataGridView(Semantic selectedSemantic):` realiza la petición a la BLL para cargar las semánticas hijas en función de la semántica seleccionada. • `private void LoadTermsDataGridView(Semantic selectedSemantic):` realiza la petición a la BLL para cargar los términos que estén asociados a la semántica seleccionada. • `private void AddTerm():` abre el formulario para añadir un término a la semántica seleccionada. • `private void MoveTerms():` abre el formulario para mover términos a la semántica seleccionada desde otra semántica. • `private void RemoveSelectedTermsFromSemantic():` elimina la semántica de los términos seleccionados.

Tabla 145: Diseño de Semantics

Semantic	
Responsabilidades:	Interfaz para mostrar los datos de una semántica.
Atributos:	• `private GrammaticalsRepository semanticsRepository:` objeto de la BLL que permite acceder a la lógica de negocio de semánticas. • `private SemanticItem editingSemantic:` referencia a la semántica que se está editando, si existe.
Operaciones:	• `private void Accept():` invoca a las funciones de validación y realiza la correspondiente petición a la BLL para insertar o editar una semántica.

	• `private void Cancel():` cancela los cambios llevados a cabo en el formulario. • `private bool ValidateName ():` valida que el nombre introducido no esté vacío. Devuelve *false* si no es válido. • `private bool ValidateRoles():` valida que se introduzca tanto el nombre y abreviatura del rol A como los del rol B. Devuelve *false* si no es válido. • `private bool ValidateTermTagType():` valida que no se haya seleccionado tanto *nombre* como *verbo* para el tipo de etiqueta sintáctica. Devuelve *false* si no es válido.

Tabla 146: Diseño de Semantic

SelectSemantic	
Responsabilidades:	Interfaz selector de una semántica.
Atributos:	• `private GrammaticalsRepository semanticRepository:` objeto de la BLL que permite acceder a la lógica de negocio de semánticas. • `private DataView semanticsDataView:` DataView que contiene la tabla de datos de semánticas. • `private SemanticItem editingSemantic:` referencia a la semántica que se está editando, si existe.
Operaciones:	• `private bool Search(String semanticName):` realiza la petición a la BLL para obtener las semánticas que coincidan con el criterio de búsqueda introducido por el usuario. • `private void Accept():` almacena la semántica elegida por el usuario y devuelve el control al formulario de origen. • `private void Cancel():` devuelve el control al formulario de origen. • `private void AddSemantic():` abre el formulario que permite crear una nueva semántica.

Tabla 147: Diseño de SelectSemantic

4.1.6.4 Sugerencias

SuggestionDataSet	
Responsabilidades:	Lleva a cabo la interacción con la capa de datos para obtener la información necesaria de la base de datos.
Atributos:	• `private SuggestionDataSet.SuggestionDataTable suggestionsDataTable:` Tabla de datos que contiene los atributos referidos a una sugerencia.
Operaciones:	

Tabla 148: Diseño de SuggestionDataSet

SuggestionsRepository	
Responsabilidades:	Clase de la BLL que contiene la lógica de negocio del módulo de sugerencias.
Atributos:	
Operaciones:	• `public SuggestionDataSet.SuggestionDataTable GetSuggestionsForDataGridView(CakeIntegerDictionary<Term> thesaurusTerms, bool suggestedRelationship):` devuelve una tabla de datos con todas las sugerencias que incluyan información sobre algún término del tesauro recibido en el atributo thesaurusTerms. En función del valor *booleano* recibido en *suggestedRelationship* se obtendrá uno de los dos tipos de sugerencias. ○ *True*: se recuperan sugerencias de relaciones → Debe existir *Termino1*, *Termino2* y *Relationship*. ○ *False*: se recuperan sugerencias de términos → Debe existir sólo *Termino1 (Termino2 y Relationship* están con valor *null* en base de datos*)*. • `public SuggestionDataSet.SuggestionDataTable GetSuggestionsForDataGridViewByUser(CakeIntegerDictionary<Term> thesaurusTerms, User user):` devuelve una table de datos con todas las

	sugerencias cuyo usuario creador coincide con el recibido por parámetro. • `public SuggestionDataSet.SuggestionDataTable GetSuggestionsForDataGridView(string term1, SemanticItem relationshipType, string term2, string author, int suggestedAction, int status, bool? answered, bool suggestedRelationship, CakeIntegerDictionary<Term> thesaurusTerms):` devuelve una tabla de datos con todas las sugerencias existentes que estén asociadas a cualquier término del tesauro (contenidos en el parámetro *thesaurusTerms)* y que coincidan con el resto de criterios de búsqueda recibidos por parámetro. • `public SuggestionDataSet.SuggestionDataTable GetSuggestionsByTerm(Term selectedTerm):` devuelve una tabla de datos con las sugerencias cuyo *Termino1* o *Termino2* sea el recibido por parámetro. • `public bool AddAnswer(Suggestion editingSuggestion, string Answer, int selectedStatus):` modifica la sugerencia recibida como parámetro para añadir la respuesta, también recibida por parámetro. Este método no modifica el *Status* de la sugerencia.

Tabla 149: Diseño de SuggestionRepository

Suggestions	
Responsabilidades:	Interfaz de listado de sugerencias de términos.
Atributos:	• `private SuggestionsRepository suggestionsRepository:` objeto de la BLL que permite acceder a la lógica de negocio de sugerencias. • `private DataView suggestionsDataView:` DataView que contiene la tabla de datos de las sugerencias. • `private Suggestion selectedSuggestion:` referencia a la sugerencia seleccionada en cada momento.
Operaciones:	• `private void LoadSuggestionsDataGridView():` realiza la

	petición a la BLL para obtener todas las sugerencias a mostrar. • `private bool Search():` aplica los filtros de búsqueda. Devuelve *false* en caso de error. • `private void ShowDetails(bool answer):` muestra el formulario para visualizar los datos de una sugerencia. • `private void ChangeStatus(int newStatus):` modifica el estado de la sugerencia actual, y establece como nuevo estado el *newStatus* recibido como parámetro.

Tabla 150: Diseño de Suggestions

SuggestedRelationship	
Responsabilidades:	Interfaz de listado de sugerencias de relaciones entre términos. Usa el mismo GUI que la clase *Suggestions*, por lo que hereda de ella y añade un atributo discriminante para saber que sólo se desean recuperar las sugerencias de relaciones entre términos. • Una **sugerencia de relación entre términos** es aquella que está formada por dos términos y una semántica que los relaciona. A nivel de datos, se deberán recuperar aquellas sugerencias cuyo *CodTerm1, CodTerm2* y *RelationshipType* no tenga valor *null*.
Atributos:	
Operaciones:	• `public SuggestedRelationship():` habilita en la interfaz la información y los paneles gráficos del término 2 y la relación.

Tabla 151: Diseño de SuggestedRelationship

Suggestion	
Responsabilidades:	Interfaz para mostrar los datos de una sugerencia.
Atributos:	• `private SuggestionsRepository suggestionssRepository:` objeto de la BLL que permite acceder a la lógica de negocio de sugerencias. • `private Suggestion editingSuggestion:` referencia a la sugerencia cuyos datos se están

	visualizando en el formulario.
Operaciones:	• `private void Accept():` invoca a las funciones de validación y realiza la correspondiente petición a la BLL para añadir una respuesta a la sugerencia. • `private void Cancel():` cancela los cambios llevados a cabo en el formulario. • `private bool ValidateAnswer ():` valida que la respuesta introducida no esté vacía. Devuelve *false* si no es válida.

Tabla 152: Diseño de Suggestion

DeleteErrors	
Responsabilidades:	Interfaz que muestra errores en el caso de los borrados, debido a la integridad referencial existente en el modelo de datos.
Atributos:	
Operaciones:	• `private bool Init(String errorMessage):` Recibe el mensaje de error y lo muestra en un control *textbox* de la interfaz.

Tabla 153: Diseño de DeleteErrors

Diseño físico de datos

En este capítulo se concreta la estructura física de datos que se utilizará en el sistema, una vez se han diseñado y especificado las clases que van a formar parte de él. Aunque es bastante similar al modelo de datos definido durante la fase de análisis, se ha considerado incluir el nuevo modelo ya que durante esta fase se conocen con mayor exactitud los requisitos específicos del sistema, las particularidades del entorno tecnológico, y los distintos aspectos arquitectónicos.

4.1.7 Diseño del modelo físico de datos

El término *PK* se corresponde con la clave primaria de la tabla, mientras que *FK* simboliza las claves ajenas, de forma que de cada fila con dicho identificador sale una flecha que indica a qué tipo de entidad referencia.

Además se incluye el tipo de dato de cada atributo, por si fuera necesario en algún momento durante el desarrollo del sistema.

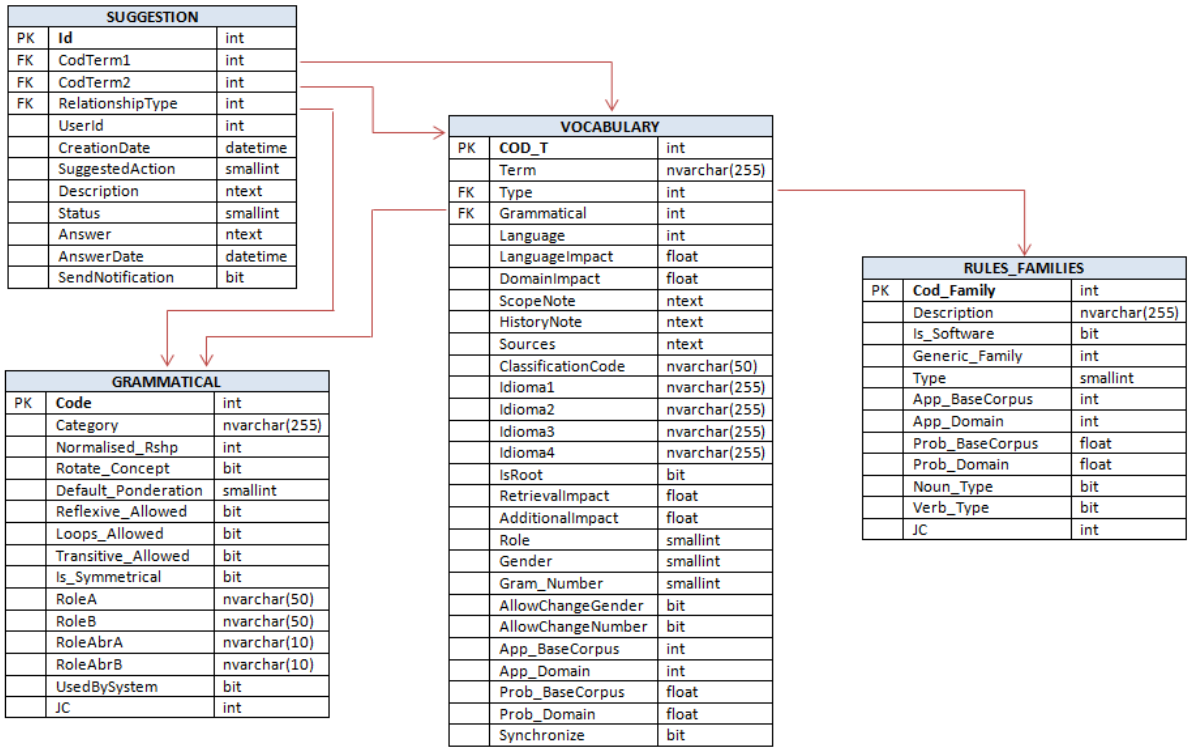

Ilustración 34: Diseño físico del modelo de datos de la aplicación

Recordemos que *vocabulary* hace referencia a un término, *rules_families* a una etiqueta sintáctica, *grammatical* a una semántica y *suggestion* a una sugerencia.

Diseño de la migración y carga inicial de datos

Recordemos que el sistema a desarrollar es un módulo de la aplicación completa *knowledgeManager*. Como tal, no necesita la misma cantidad de datos iniciales para ser ejecutado.

De cualquier modo, para poder ejecutar el módulo es necesario que la base de datos de conexión disponga de algunos datos básicos y esenciales. Dichos datos se encuentran especificados en el apartado de requisitos del sistema.

4.1.8 Especificación del entorno de migración

Para disponer de los datos necesarios, el sistema debe tener instalado un gestor de bases de datos SQL Server 2008. En este caso, el mínimo y recomendado es SQL Server Management Studio 10.0.

4.1.9 Diseño de procedimientos de migración y carga inicial

Para poder ejecutar correctamente la aplicación, es necesario disponer previamente de una base de datos correctamente preparada. Con el fin de no tener que modificar las tablas de la base de datos de forma manual cada vez que se quiera hacer un despliegue de la aplicación, se adjunta un fichero de **copia de seguridad de base de datos (.bak)**, que contiene la información mínima para que la aplicación pueda ser ejecutada.

Teniendo por un lado, el gestor SQL Server Management Studio y por otro, el fichero de base de datos, es necesario importarlo para crear nuestra base de datos. El proceso es el siguiente:

1. Conectar con tu servidor de bases de datos.

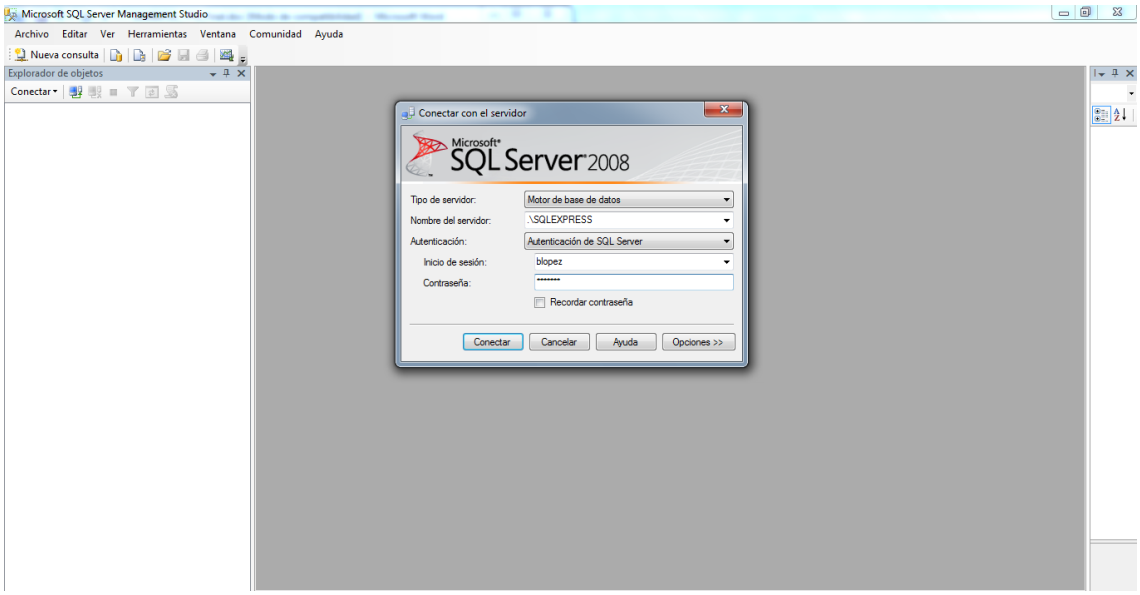

Ilustración 35: Conexión con servidor de SQL Server

2. Elegir la opción ‘Restaurar base de datos’

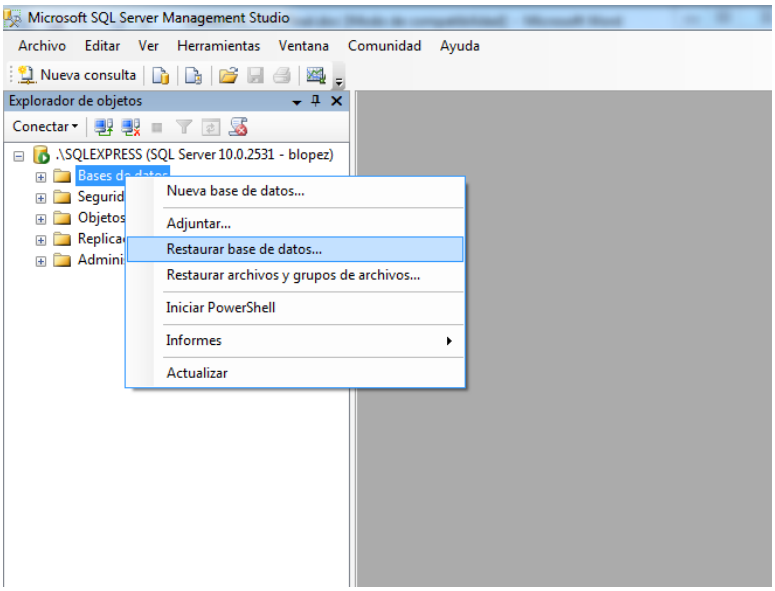

Ilustración 36: Primer paso para restaurar base de datos SQL Server

a. Elegir el nuevo nombre de la base de datos y la ruta del fichero *.bak* del cual se va a restaurar.

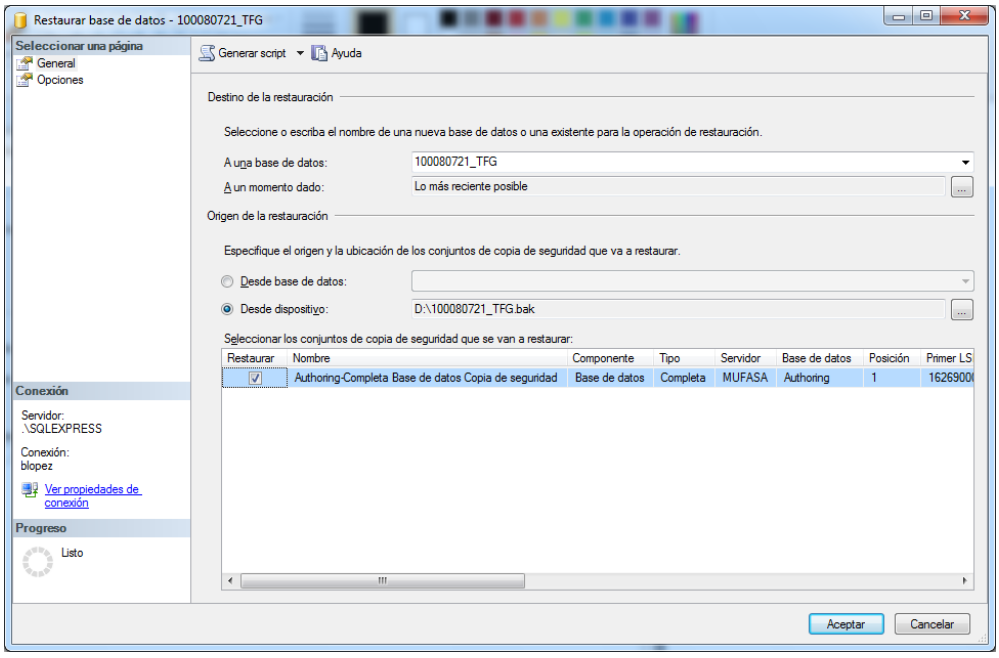

Ilustración 37: Segundo paso para restaurar base de datos SQL Server

b. Comprobar que la base de datos ha sido creada correctamente.

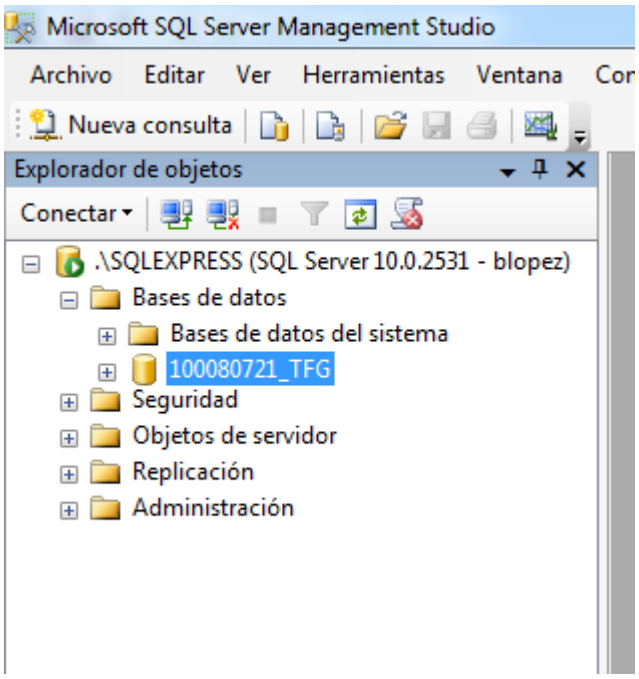

Ilustración 38: La base de datos ha sido restaurada con éxito

Especificación técnica del plan de pruebas

La aplicación a desarrollar está basada en una arquitectura de tres capas que separa por un lado la capa de presentación, y por otro lado la capa de negocio de la aplicación.

Además de las pruebas de aceptación definidas en el capítulo de Análisis, se llevan a cabo pruebas unitarias de todas las funciones públicas de la capa de negocio con el fin de obtener un producto lo más fiable posible.

4.1.10 Especificación del entorno de pruebas

Tanto el entorno software como hardware utilizado para llevar a cabo las pruebas unitarias es el mismo que el que se define en el capítulo de Análisis. Se añade la necesidad de utilizar **nunit,** un software que permite llevar a cabo pruebas unitarias desarrolladas en lenguaje C#.

Se utiliza la versión 2.6.1 de nunit, que a fecha actual es la versión estable más actualizada para el *framework* .NET 4.0 utilizado.

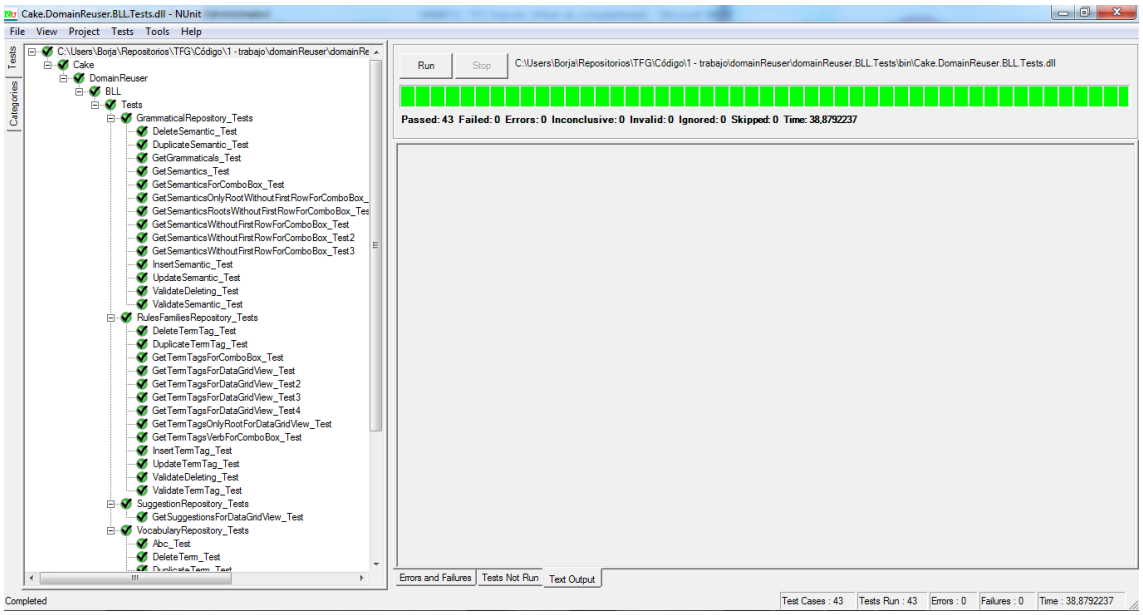

Ilustración 39: Interfaz de nUnit 2.6.1 tras la ejecución de las pruebas

4.1.11 Especificación técnica de niveles de prueba

Con el fin de facilitar la comprensión de las pruebas, se dividen según el criterio de subsistemas de diseño. Así, habrá una clase de pruebas para el subsistema de Terminología, otra para el de Etiquetas sintácticas, otra para el de Semánticas y una última clase para el subsistema de Sugerencias.

Cada método de test es **independiente** del resto. Es decir, el estado final de la base de datos del primer método de test no debe ser el estado inicial del segundo test, sino que deben partir todos de la misma información.

Para facilitar esta tarea, se utiliza una base de datos de desarrollo de Microsoft *Access*. Existe una base de datos *Access* original, que contiene el estado inicial de cada test, y una copia de la original que es sobre la que se trabaja.

Existirá una función etiquetada como [*SetUp]* que se ejecuta automáticamente antes de cada test, cuya función será reemplazar la copia de trabajo con una réplica exacta de la base de datos original descrita en el párrafo anterior. Así aseguramos el mismo estado inicial para todos los tests de cada *suite de pruebas*.

4.1.11.1 Terminología

PU-001: Obtención de términos	
Descripción:	Comprueba la obtención de todos los términos de la ontología, en el caso de que haya más de uno.
Función:	`GetTermsForDataGridView()`
Pasos:	• Invocar llamada a `GetTermsForDataGridView()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 154: PU-001

PU-002: Obtención de términos según etiqueta sintáctica	
Descripción:	Comprueba la obtención de todos los términos de la ontología asociados a una determinada etiqueta sintáctica recibida como parámetro.
Función:	`GetTermsForDataGridView(Term_Tag termTag)`
Pasos:	• Term_Tag myTermTag = engine.NounType; • Invocar llamada a `GetTermsForDataGridView(myTermTag)`
Especificaciones de entrada:	• Se recupera la etiqueta sintáctica NOUN en la variable *myTermTag*.
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 155: PU-002

PU-003: Obtención de términos según etiqueta sintáctica	
Descripción:	Comprueba la obtención de todos los términos de la ontología asociados a una determinada etiqueta sintáctica recibida como parámetro. Además se recuperan los términos de todas las etiquetas sintácticas cuya etiqueta sintáctica raíz sea la especificada como parámetro.
Función:	`GetTermsForDataGridView(Term_Tag termTag, bool includeAllChildren)`
Pasos:	• Term_Tag myTermTag = engine.NounType; • Invocar llamada a `GetTermsForDataGridView(myTermTag, true)`
Especificaciones de entrada:	• Se recupera la etiqueta sintáctica NOUN en la variable *myTermTag*.
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 156: PU-003

PU-004: Obtención de términos según semántica	
Descripción:	Comprueba la obtención de todos los términos de la ontología asociados a una determinada semántica recibida como parámetro.
Función:	`GetTermsForDataGridView(SemanticItem semantic)`
Pasos:	• SemanticItem mySemantic = engine.Semantics_CapabilityType; • Invocar llamada a `GetTermsForDataGridView(mySemantic)`
Especificaciones de entrada:	• Se recupera la semántica CAPABILITY en la variable *mySemantic*.
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.

Produce excepción:	No

Tabla 157: PU-004

PU-005: Obtención de términos según filtro de búsqueda	
Descripción:	Comprueba la obtención de todos los términos de la ontología cuyos atributos coincidan con los especificados por el usuario como criterios de búsqueda. Dichos criterios son recibidos como parámetro.
Función:	`GetTermsForDataGridView(string name, Term_Tag termTag, Language language, SemanticItem semantic, bool belongsTodomain)`
Pasos:	• Term newTerm = new Term(engine, name, termTag, language, semantic); • Invocar llamada a `GetTermsForDataGridView(name, termTag, language, semantic, false)`
Especificaciones de entrada:	• Se inserta un nuevo término: ○ **Name**: newTestTerm ○ **TermTag**: engine.NounType ○ **Semantic**: engine.Semantics_CapabilityType ○ **Language** = engine.EnglishLanguage
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos. En concreto, el número de elementos es 1.
Produce excepción:	No

Tabla 158: PU-005

PU-006: Obtención de términos VERB	
Descripción:	Comprueba la obtención de todos los términos de la ontología cuya etiqueta sintáctica es VERB.
Función:	`GetTermVerb()`
Pasos:	• Invocar llamada a `GetTermVerb()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene cero elementos, ya que no existe ningún término con dicha etiqueta sintáctica.
Produce excepción:	No

Tabla 159: PU-006

PU-007: Obtención de términos por nombre	
Descripción:	Comprueba la obtención de términos según nombre. Esta función no realiza un *matching* completo, sino que comprueba si la cadena recibida como parámetro está contenida en el nombre de algún término.
Función:	`GetTermByName(string name)`
Pasos:	• Term newTerm = new Term(engine, name, termTag, language) • Invocar llamada a `GetTermByName("zZZZz")` y almacenar resultado en un objeto tipo Term.

Especificaciones de entrada:	• Crear un nuevo término
	  ○ **Name**: zzzZZZzzzZZZzzz
	  ○ **TermTag**: engine.NounType
	  ○ **Language**: engine.WorkingLanguage

Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera un término, que coincide con el insertado previamente.
Produce excepción:	No

Tabla 160: PU-007

PU-008: Obtención de términos por nombre	
Descripción:	Comprueba la obtención de términos según nombre. Esta función realiza un *matching* del nombre completo.
Función:	`GetTermByNameExact(string name)`
Pasos:	• Term newTerm = new Term(engine, name, termTag, language) • Invocar llamada a `GetTermByNameExact(name)` y almacenar resultado en un objeto tipo Term.

Especificaciones de entrada:	• Crear un nuevo término
	  ○ **Name**: zzzZZZzzzZZZzzz
	  ○ **TermTag**: engine.NounType
	  ○ **Language**: engine.WorkingLanguage

Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera un término, que coincide con el insertado previamente.
Produce excepción:	No

Tabla 161: PU-008

PU-009: Validar atributos de un término	
Descripción:	Valida los atributos que forman un término, con el fin de realizar posteriormente una operación de inserción o edición.
Función:	`ValidateTerm(Term editingTerm, Term_Tag newTermTag, SemanticItem newSemantic, string name, bool belongsToDomain, Language language, int domainImpact, int languageImpact)`
Pasos:	• Term newTerm = new Term(engine, name, termTag, language) • Crear el atributo `bool validated` • Primera validación: `validated = myVocabularyRepository.ValidateTerm(newTerm, engine.VerbToBEType, null, “newName”, false,`

	`engine.WorkingLanguage, 0, 0)` ○ **Devuelve *`true`*.** • Segunda validación: `validated = myVocabularyRepository.ValidateTerm(newTerm, engine.Unclassified_NounType, engine.Semantics_Composition, "newName", true, engine.WorkingLanguage, 0, 0)` ○ **Devuelve *`true`*.** • Tercera validación: `validated = myVocabularyRepository.ValidateTerm(newTerm, engine.Unclassified_NounType, engine.Semantics_Composition, string.Empty, true, engine.WorkingLanguage, 0, 0)` ○ **Devuelve *`false`*.**
Especificaciones de entrada:	• Crear un nuevo término ○ **Name**: zzzZZZzzzZZZzzz ○ **TermTag**: engine.NounType ○ **Language**: engine.WorkingLanguage
Especificaciones de salida:	Se han especificado en los *Pasos* de ejecución.
Produce excepción:	No

Tabla 162: PU-009

PU-010: Creación de un término	
Descripción:	Comprueba la correcta creación de un término en la ontología.
Función:	`InsertTerm(Term_Tag termTag, Semantic semantic, string name, bool belongsToDomain, int domainImpact, int languageImpact, string scopeNote, ref Term newTerm)`
Pasos:	• Term myNewTerm = null • Invocar llamada a `InsertTerm(termTag, semantic, name, domain, Language, DomainImpact, LanguageImpact, scopeNoteTerm, ref myNewTerm)`. • Comprobar que `myNewTerm` es distinto de *null* y tiene

	los valores con los que se llamó a la función del caso de prueba. • Comprobar con la función de *retrieval* `GetTermByNameExact(string name)` que el término se ha insertado correctamente. El término devuelto debe tener los mismos atributos que se incluyeron como parámetro en la llamada a la función de inserción. ○ `Term check = myVocabularyRepository.getTermByNameExact("myNewTermToTestInsertTermFunction")`
Especificaciones de entrada:	• Crear un nuevo término ○ **Name**: myNewTermToTestInsertTermFunction ○ **Domain**: false ○ **TermTag**: engine.NounType ○ **Language**: engine.WorkingLanguage ○ **SemanticItem**: null ○ **DomainImpact**: 0 ○ **LanguageImpact**: 0 ○ **ScopeNoteTerm**: string.Empty
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera un término, cuyos atributos son los que se pasaron como parámetro en la inserción. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 163: PU-010

PU-011: Duplicado de un término	
Descripción:	Comprueba el correcto duplicado de la información de un término de la ontología.
Función:	`DuplicateTerm(Term originalTerm, ref Term duplicatedTerm)`
Pasos:	• Crear un término. `Term originalTerm = new Term(termTag, semantic, name, domain,`

	`language, DomainImpact, LanguageImpact, scopeNoteTerm)` • Invocar llamada a `DuplicateTerm(originalTerm, ref duplicatedTerm)`. ○ Devuelve **true**. • Comprobar que `duplicatedTerm` es distinto de *null* y tiene los valores con los que se creó el término original, a excepción del nombre que debe ser **`myNewTermToTestDuplicateTermfunction0`**.

Especificaciones de entrada:	• Crear un nuevo término
	  ○ **Name**: myNewTermToTestDuplicateTermFunction
	  ○ **Domain**: false
	  ○ **TermTag**: engine.NounType
	  ○ **Language**: engine.WorkingLanguage
	  ○ **SemanticItem**: null
	  ○ **DomainImpact**: 0
	  ○ **LanguageImpact**: 0
	  ○ **ScopeNoteTerm**: string.Empty

Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera un término, cuyos atributos son los mismos con los que se creó el término original, a excepción del nombre, al que se le añade un 0 al final. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 164: PU-011

PU-012: Edición de un término	
Descripción:	Comprueba la correcta edición de la información de un término de la ontología.
Función:	`UpdateTerm(Term editingTerm, Term_Tag newTermTag, SemanticItem newSemantic, string name, bool belongsToDomain, Language language, int domainImpact, int LanguageImpact, Term`

	`scopeNoteTerm)`
Pasos:	• Crear un término. `Term myTerm = new Term(termTag, semantic, name, domain, language, DomainImpact, LanguageImpact, scopeNoteTerm)` • Invocar llamada a `UpdateTerm(myTerm, engine.VerbTOBEType, null, "newName", false, language, domainImpact, languageImpact, null)`. ○ Devuelve **true**. • Comprobar mediante la function de *retrieval* `GetTermByNameExact("newName")` que el término editado previamente tiene los nuevos atributos *termTag* y *semantic* actualizados correctamente.

Especificaciones de entrada:	• Crear un nuevo término
	  ○ **Name**: myNewTermToTestUpdateTermFunction
	  ○ **Domain**: false
	  ○ **TermTag**: engine.NounType
	  ○ **Language**: engine.WorkingLanguage
	  ○ **SemanticItem**: null
	  ○ **DomainImpact**: 0
	  ○ **LanguageImpact**: 0
	  ○ **ScopeNoteTerm**: string.Empty

Especificaciones de salida:	Tras la llamada a la función de este caso de prueba, es necesario recuperar el término modificado mediante alguna función de *retrieval*. El término devuelto tiene los nuevos valores. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 165: PU-012

PU-013: Borrado de un término	
Descripción:	Comprueba el correcto borrado de un término de la ontología.
Función:	`DeleteTerm(Term deletingTerm)`
Pasos:	• Crear un término. `Term myTerm = new Term(termTag, semantic, name, domain, language, DomainImpact, LanguageImpact, scopeNoteTerm)` • Comprobar mediante la function de *retrieval* `GetTermByNameExact(name)` que el término eliminado sí existe. • Invocar llamada a `DeleteTerm(myTerm)`. ○ Devuelve **true**. • Comprobar mediante la function de *retrieval* `GetTermByNameExact(name)` que el término eliminado ya no existe.

Especificaciones de entrada:	• Crear un nuevo término
	  ○ **Name**: myNewTermToTestDeleteTermFunction
	  ○ **Domain**: false
	  ○ **TermTag**: engine.NounType
	  ○ **Language**: engine.WorkingLanguage
	  ○ **SemanticItem**: null
	  ○ **DomainImpact**: 0
	  ○ **LanguageImpact**: 0
	  ○ **ScopeNoteTerm**: string.Empty

Especificaciones de salida:	Tras eliminar un término e intentar recuperarlo después, dicho término ya no existe.
Produce excepción:	No

Tabla 166: PU-013

4.1.11.2 Etiquetas sintácticas

PU-014: Obtención de etiquetas sintácticas	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas de la ontología. Devuelve una tabla de datos reducida.
Función:	`GetTermTagsForComboBox()`
Pasos:	• Invocar llamada a `GetTermTagsForComboBox ()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 167: PU-014

PU-015: Obtención de etiquetas sintácticas verbales	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas verbales de la ontología.
Función:	`GetTermTagsVerbForComboBox()`
Pasos:	• Invocar llamada a `GetTermTagsVerbForComboBox()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 168: PU-015

PU-016: Obtención de etiquetas sintácticas	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas de la ontología.

Función:	`GetTermTagsForDataGridView()`
Pasos:	• Invocar llamada a `GetTermTagsForDataGridView()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 169: PU-016

PU-017: Obtención de etiquetas sintácticas raíces	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas raíces de la ontología, es decir, las que no tienen etiqueta sintáctica padre.
Función:	`GetTermTagsOnlyRootForDataGridView()`
Pasos:	• Invocar llamada a `GetTermTagsOnlyRootForDataGridView ()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 170: PU-017

PU-018: Obtención de etiquetas sintácticas según etiqueta raíz	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas de la ontología cuya etiqueta sintáctica padre es la especificada como parámetro.
Función:	`GetTermTagsForDataGridView(Term_Tag termTag)`
Pasos:	• `Term_Tag termTag = engine.NounType`

	• Invocar llamada a `GetTermTagsForDataGridView(termTag)`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 171: PU-018

PU-019: Obtención de etiquetas sintácticas hijas según descripción	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas hijas de la ontología cuya descripción de la etiqueta sintáctica raíz sea igual a la cadena pasada como parámetro.
Función:	`GetTermTagsForDataGridView(string parentDescription)`
Pasos:	• `string parentDescription = engine.NounType.Description` • Invocar llamada a `GetTermTagsForDataGridView(parentDescription)`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 172: PU-019

PU-020: Obtención de etiquetas sintácticas según etiqueta raíz	
Descripción:	Comprueba la obtención de todas las etiquetas sintácticas de la ontología cuyo padre sea el especificado como parámetro. Además, cabe la opción de incluir recursivamente las hijas de las obtenidas en un primer nivel, mediante el segundo parámetro de la función.
Función:	`GetTermTagsForDataGridView(Term_Tag parent, bool includeAllRecursiveChildren)`
Pasos:	• `Term_Tag parent = engine.NounType` • Invocar llamada a `GetTermTagsForDataGridView(parent, false)` • Comprobar que la tabla de datos que devuelve la llamada tiene más de 0 elementos. • Invocar llamada a `GetTermTagsForDataGridView(parent, true)` • Comprobar que la tabla de datos que devuelve la llamada tiene más de 0 elementos.
Especificaciones de entrada:	
Especificaciones de salida:	Se especifican los detalles en los *Pasos*.
Produce excepción:	No

Tabla 173: PU-020

PU-021: Validar atributos de una etiqueta sintáctica	
Descripción:	Valida los atributos que forman una etiqueta sintáctica, con el fin de realizar posteriormente una operación de inserción o edición.
Función:	`ValidateTermTag(Term_Tag editingTermTag, string name, Term_Tag parentTermTag, string name, double domainStats, int appDomain, double corpusStats, int appCorpus)`
Pasos:	• `Term_Tag` myTermTag =

	engine.NotGroupingNounType
	• Crear el atributo `bool success`
	• Primera validación: `success = myTermTagRepository.ValidateTermTag(myTermTag, name + name, engine.NounType, 0, 0, 0, 0)`
	  ○ **Devuelve *true*.**
	• Segunda validación: `success = myTermTagRepository. ValidateTermTag (myTermTag, string.Empty, engine.NounType, 0, 0, 0, 0)`
	  ○ **Devuelve *false*.**

Especificaciones de entrada:	
Especificaciones de salida:	Se han especificado en los *Pasos* de ejecución.
Produce excepción:	No

Tabla 174: PU-021

PU-022: Validar borrado de una etiqueta sintáctica	
Descripción:	Valida si una etiqueta sintáctica puede ser borrada o no. En caso negativo, se debe a que tiene alguna dependencia con algún elemento a través de clave ajena de base de datos.
Función:	`ValidateDeleting(Term_Tag deletingTermTag, List<Term> terms, List<Role> roles, List<Rule> rules, List<BigramStatistic> bigrams, List<Qualification> qualifications, List<ClarifierRule> oldInferenceRules, List<ClarifierRule> taggingRules, List<Pattern> inferenceRules, List<Locution> locutions, List<Term_Tag> termTags)`
Pasos:	• `Term_Tag` myTermTag = engine.NounType • Crear el atributo `bool success` • Primera validación de borrado: `success = myTermTagRepository.ValidateDeleting(myTermTag, ref terms, ref roles, ref rules, ref`

	`bigrams, ref qualifications, ref oldInferenceRules, ref taggingRules, ref inferenceRules, ref locutions, ref termTags)` ○ **Devuelve *`false`*.** ○ El atributo *terms* debe tener más de 0 elementos. • `myTermTag = new Term_Tag(engine, newName, engine.NounType, false)` • Segunda validación de borrado: `success = myTermTagRepository.ValidateDeleting(myTermTag, ref terms, ref roles, ref rules, ref bigrams, ref qualifications, ref oldInferenceRules, ref taggingRules, ref inferenceRules, ref locutions, ref termTags)` ○ **Devuelve *`true`*.**
Especificaciones de entrada:	• Crear una nueva etiqueta sintáctica tras la primera validación. ○ **Name**: myNewTermTag ○ **TermTag**: engine.NounType ○ **IsSoftware**: false
Especificaciones de salida:	Se han especificado en los *Pasos* de ejecución.
Produce excepción:	No

Tabla 175: PU-022

PU-023: Creación de una etiqueta sintáctica	
Descripción:	Comprueba la correcta creación de una etiqueta sintáctica en la ontología.
Función:	`InsertTermTag(string nombre, Term_Tag parentTermTag, double domainStats, int appDomain, double corpusStats, int appCorpus, bool nounType, bool verbType, ref Term_Tag newTermTag)`
Pasos:	• `Term_Tag` myNewTermTag = null • Invocar llamada a `InsertTermTag(name,`

	`engine.NounType, 0, 0, 0, 0, false, false, ref myNewTermTag`). • Comprobar que `myNewTermTag` es distinto de *null* y tiene los valores con los que se llamó a la función del caso de prueba.
Especificaciones de entrada:	• Crear una nueva etiqueta sintáctica ○ **Name**: myNewTermTagToTestInserted ○ **ParentTermTag**: engine.NounType ○ **DomainStats**: 0 ○ **AppDomain**: 0 ○ **CorpusStats**: 0 ○ **AppCorpus**: 0 ○ **LanguageImpact**: 0 ○ **NounType**: false ○ **VerbType**: false
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera una etiqueta sintáctica, cuyos atributos son los que se pasaron como parámetro en la inserción. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 176: PU-023

PU-024: Duplicado de una etiqueta sintáctica	
Descripción:	Comprueba el correcto duplicado de la información de una etiqueta sintáctica de la ontología.
Función:	`DuplicateTermTag(Term_Tag originalTermTag, ref Term_Tag duplicatedTermTag)`
Pasos:	• Crear una etiqueta sintáctica. `Term_Tag originalTermTag = new Term_Tag(engine, name, false, engine.NounType)` • Invocar llamada a `DuplicateTermTag(originalTermTag, ref duplicatedTermTag)`.

	○ Devuelve **true**. • Comprobar que `duplicatedTermTag` es distinto de *null* y tiene los valores con los que se creó la etiqueta sintáctica original.
Especificaciones de entrada:	• Crear una nueva etiqueta sintáctica mediante su constructor: ○ **Name**: myNewTermTagToTestInserted ○ **IsSoftware**: false ○ **ParentTermTag**: engine.NounType
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera una etiqueta sintáctica, cuyos atributos son los mismos con los que se creó la etiqueta sintáctica original. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 177: PU-024

PU-025: Edición de una etiqueta sintáctica	
Descripción:	Comprueba la correcta edición de la información de una etiqueta sintáctica de la ontología.
Función:	`UpdateTermTag(string nombre, Term_Tag parentTermTag, double domainStats, int appDomain, double corpusStats, int appCorpus, bool nounType, bool verbType)`
Pasos:	• Crear una etiqueta sintáctica. `Term_Tag newTermTag = null` • `myTermTagsRepository.InsertTermTag(name, engine.NounType, 0, 0, 0, 0, false, false, ref newTermTag)` • Invocar llamada a `UpdateTermTag(newTermTag, "updatedName", engine.NounType, 0, 0, 0, 0, false, false)`. ○ Devuelve **true**. • Comprobar mediante la function de *retrieval* `GetTermTagsForDataGridView ("updatedName")` que la etiqueta sintáctica

	editada previamente tiene los nuevos atributos.
Especificaciones de entrada:	• Crear una nueva etiqueta sintáctica ○ **Name**: myNewTermTagToTestUpdated ○ **ParentTermTag**: engine.NounType ○ **DomainStats**: 0 ○ **AppDomain**: 0 ○ **CorpusStats**: 0 ○ **AppCorpus**: 0 ○ **LanguageImpact**: 0 ○ **NounType**: false ○ **VerbType**: false
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba, es necesario recuperar la etiqueta sintáctica modificada mediante alguna función de *retrieval*. La etiqueta sintáctica devuelta tiene los nuevos valores. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 178: PU-025

PU-026: Borrado de una etiqueta sintáctica	
Descripción:	Comprueba el correcto borrado de una etiqueta sintáctica de la ontología.
Función:	`DeleteTermTag(Term_Tag deletingTermTag)`
Pasos:	• Invocar a `myTermTagsRepository.DeleteTermTag(engine.NounType)` ○ Devuelve **false.** • Comprobar mediante la function de *retrieval* `GetTermTagsForDataGridView(engine.NounType.Description)` que la etiqueta sintáctica que no se pudo borrar anteriormente sigue existiendo. • Crear una etiqueta sintáctica. `Term_Tag myTermTag = new TermTag(engine, name,`

	`engine.NounType, false)` • Invocar a `myTermTagsRepository.DeleteTermTag(myTermTag)` ○ Devuelve **true.** • Comprobar mediante la function de *retrieval* `GetTermTagsForDataGridView (name)` que la etiqueta sintáctica eliminada ya no existe.

Especificaciones de entrada:	• Crear una nueva etiqueta sintáctica mediante su constructor:
	  ○ **Name**: myNewTermTagToTestDeleted
	  ○ **IsSoftware**: false
	  ○ **ParentTermTag**: engine.NounType

Especificaciones de salida:	Tras eliminar una etiqueta sintáctica e intentar recuperarla después, dicha etiqueta sintáctica ya no existe.
Produce excepción:	No

Tabla 179: PU-026

4.1.11.3 Semánticas

PU-027: Obtención de semánticas	
Descripción:	Comprueba la obtención de todas las semánticas de la ontología.
Función:	`GetSemanticsForComboBox()`
Pasos:	• Invocar llamada a `GetSemanticsForComboBox()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 180: PU-027

PU-028: Obtención de semánticas	
Descripción:	Comprueba la obtención de las etiquetas sintácticas de la ontología cuyo nombre coincida con la cadena pasada como parámetro.
Función:	`GetSemantics(string semantic, bool? UsedBySystem, bool onlyRoots)`
Pasos:	• Invocar llamada a `GetSemantics(engine.GeneralizationSpecialization.Name, null, false)`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 181: PU-028

PU-029: Obtención de semánticas	
Descripción:	Comprueba la obtención de todas las semánticas de la ontología.
Función:	`GetSemanticsWithoutFirstRowForComboBox()`
Pasos:	• Invocar llamada a `GetSemanticsWithoutFirstRowForComboBox()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 182: PU-029

PU-030: Obtención de semánticas raíces	
Descripción:	Comprueba la obtención de todas las semánticas raíces de la ontología, es decir, las que no tienen semántica padre.
Función:	`GetSemanticsRootWithoutFirstRowForComboBox()`
Pasos:	• Invocar llamada a `GetSemanticsRootWithoutFirstRowForComboBox()`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 183: PU-030

PU-031: Obtención de semánticas según semántica raíz	
Descripción:	Comprueba la obtención de todas las semánticas de la ontología cuya semántica padre es la especificada como parámetro.
Función:	`GetSemanticsWithoutFirstRowForComboBox(SemanticItem parentSemantic, bool includeAllChildren)`
Pasos:	• `SemanticItem parentSemantic = engine.Semantics_ActionType` • Invocar llamada a `GetSemanticsWithoutFirstRowForComboBox(parentSemantic, false)`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 184: PU-031

PU-032: Obtención de semánticas hijas según descripción	
Descripción:	Comprueba la obtención de todas las semánticas hijas de la ontología cuya descripción de la semántica raíz sea igual a la cadena pasada como parámetro.
Función:	`GetSemanticsWithoutFirstRowForComboBox(string parentDescription)`
Pasos:	• `string parentDescription = engine.Semantics_ActionType.Name` • Invocar llamada a `GetSemanticsWithoutFirstRowForComboBox(parentDescription)`
Especificaciones de entrada:	
Especificaciones de salida:	La tabla de datos obtenida tras la llamada a la función tiene más de cero elementos.
Produce excepción:	No

Tabla 185: PU-032

PU-033: Validar atributos de una semántica	
Descripción:	Valida los atributos que forman una semántica, con el fin de realizar posteriormente una operación de inserción o edición.
Función:	`ValidateSemantic(SemanticItem editingSemantic, string name, SemanticItem parentSemantic, byte defaultPonderation, string roleA, string roleB, string roleAbreviationA, string roleAbreviationB)`
Pasos:	• SemanticItem mySemantic = engine.Semantics_ActionType • Crear el atributo `bool success` • Primera validación: `success = myGrammaticalRepository.ValidateSemantic(mySemantic, "newName", null, 0, string.Empty, string.Empty, string.Empty, string.Empty)`

	○ **Devuelve *true*.** • Segunda validación: `success = myGrammaticalRepository.ValidateSemantic(mySemantic, null, null, 0, string.Empty, string.Empty, string.Empty, string.Empty)` ○ **Devuelve *false*.**
Especificaciones de entrada:	
Especificaciones de salida:	Se han especificado en los *Pasos* de ejecución.
Produce excepción:	No

Tabla 186: PU-033

PU-034: Validar borrado de una etiqueta sintáctica	
Descripción:	Valida si una semántica puede ser borrada o no. En caso negativo, se debe a que tiene alguna dependencia con algún elemento a través de clave ajena de base de datos.
Función:	`ValidateDeleting(Term_Tag deletingTermTag, List<Term> terms, List<Role> roles, List<ClarifierRuleBit> clarifierRulesBit, List<Pattern_SLOT> patternSlots, List<Semantic> otherSubtypes)`
Pasos:	• SemanticItem mySemantic = engine.Semantics_ActionType • Crear el atributo `bool success` • Primera validación de borrado: `success = myGrammaticalRepository.ValidateDeleting(mySemantic, ref terms, ref roles, ref clarifierRulesBit, ref patternSlots, ref otherSubtypes)` ○ **Devuelve *false*.** ○ El atributo *terms* debe tener más de 0 elementos. • `mySemantic = new SemanticItem(engine, name, null, false)`

	• Segunda validación de borrado: `success = myGrammaticalRepository.ValidateDeleting(mySemantic, ref terms, ref roles, ref clarifierRulesBit, ref patternSlots, ref otherSubtypes)` ○ **Devuelve *true*.**

Especificaciones de entrada:	• Crear una nueva gramática tras la primera validación.
	  ○ **Name**: newSemantic
	  ○ **ParentSemantic**: null
	  ○ **RotateConcept**: false

Especificaciones de salida:	Se han especificado en los *Pasos* de ejecución.
Produce excepción:	No

Tabla 187: PU-034

PU-035: Creación de una etiqueta sintáctica	
Descripción:	Comprueba la correcta creación de una semántica en la ontología.
Función:	`InsertSemantic(string nombre, SemanticItem parentSemantic, bool rotateConcept, byte defaultPonderation, bool reflexiveAllowed, bool loopsAllowed, bool transitiveAllowed, bool isSymmetrical, bool usedBySystem , string roleA , string roleB , string roleAbreviationA, string roleAbreviationB, ref SemanticItem newSemantic)`
Pasos:	• SemanticItem myNewSemantic = null • Invocar llamada a `InsertSemantic(name, null, false, 0, false, false, false, false, false, string.Empty, string.Empty, string.Empty, string.Empty)`. • Comprobar que `myNewSemantic` es distinto de *null* y tiene los valores con los que se llamó a la función del caso de prueba.

Especificaciones de entrada:	• Crear una nueva semántica ○ **Name**: myNewSemanticToTestInserted ○ **ParentSemantic**: null ○ **RotateConcept**: false ○ **DefaultPonderation**: 0 ○ **ReflexiveAllowed**: false ○ **LoopsAllowed**: false ○ **TransitiveAllowed**: false ○ **IsSymmetrical**: false ○ **UsedBySystem**: false ○ **RoleA**: string.Empty ○ **RoleB**: string.Empty ○ **RoleAbreviationA**: string.Empty ○ **RoleAbreviationB**: string.Empty
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera una semántica, cuyos atributos son los que se pasaron como parámetro en la inserción. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 188: PU-035

PU-036: Duplicado de una semántica	
Descripción:	Comprueba el correcto duplicado de la información de una semántica de la ontología.
Función:	`DuplicateSemantic(SemanticItem originalSemantic, ref SemanticItem duplicatedSemantic)`
Pasos:	• Crear una semántica. `SemanticItem myGrammaticalRepository.InsertSemantic(name, null, false, 0, false, false, false, false, false, string.Empty, string.Empty, string.Empty, string.Empty)`

	• Invocar llamada a `DuplicateSemantic(originalSemantic, ref duplicatedSemantic)`. ○ Devuelve **true**. • Comprobar que `duplicatedSemantic` es distinto de *null* y tiene los valores con los que se creó la semántica original.

Especificaciones de entrada:	• Crear una nueva semántica:
	  ○ **Name**: myNewSemantic
	  ○ **ParentSemantic**: null
	  ○ **RotateConcept**: false
	  ○ **DefaultPonderation**: 0
	  ○ **ReflexiveAllowed**: false
	  ○ **LoopsAllowed**: false
	  ○ **TransitiveAllowed**: false
	  ○ **IsSymmetrical**: false
	  ○ **UsedBySystem**: false
	  ○ **RoleA**: string.Empty
	  ○ **RoleB**: string.Empty
	  ○ **RoleAbreviationA**: string.Empty
	  ○ **RoleAbreviationB**: string.Empty

Especificaciones de salida:	Tras la llamada a la función de este caso de prueba se recupera una semántica, cuyos atributos son los mismos con los que se creó la semántica original. Se especifican más detalles en los *Pasos*.
Produce excepción:	No

Tabla 189: PU-036

PU-037: Edición de una semántica	
Descripción:	Comprueba la correcta edición de la información de una semántica de la ontología.
Función:	`UpdateSemantic(string nombre, SemanticItem parentSemantic, bool rotateConcept, byte defaultPonderation, bool reflexiveAllowed, bool loopsAllowed, bool transitiveAllowed, bool isSymmetrical, bool usedBySystem , string roleA , string roleB , string roleAbreviationA, string roleAbreviationB)`
Pasos:	• SemanticItem myNewSemantic = null • Invocar llamada a `InsertSemantic(name, null, false, 0, false, false, false, false, false, string.Empty, string.Empty, string.Empty, string.Empty)`. • Invocar llamada a `UpdateSemantic("myNewNameToUpdate", null, false, 0, false, false, false, false, false, string.Empty, string.Empty, string.Empty, string.Empty)`. ○ Devuelve **true**.

Especificaciones de entrada:	• Crear una nueva semántica
	  ○ **Name**: newSemantic
	  ○ **ParentSemantic**: null
	  ○ **RotateConcept**: false
	  ○ **DefaultPonderation**: 0
	  ○ **ReflexiveAllowed**: false
	  ○ **LoopsAllowed**: false
	  ○ **TransitiveAllowed**: false
	  ○ **IsSymmetrical**: false
	  ○ **UsedBySystem**: false
	  ○ **RoleA**: string.Empty
	  ○ **RoleB**: string.Empty
	  ○ **RoleAbreviationA**: string.Empty

	○ **RoleAbreviationB**: string.Empty
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba, el último resultado devuelto es **true**.
Produce excepción:	No

Tabla 190: PU-037

PU-038: Borrado de una semántica	
Descripción:	Comprueba el correcto borrado de una etiqueta sintáctica de la ontología.
Función:	`DeleteSemantic(Semantic deletingSemantic)`
Pasos:	• Invocar a `myGrammaticalRepository.DeleteSemantic(engine.Semantics_ActionType)` ○ Devuelve **false.** • Comprobar mediante la function de *retrieval* `GetSemanticsWithoutFirstRowForComboBox(engine.Semantics_ActionType.Name)` que la semántica que no se pudo borrar anteriormente sigue existiendo. • Crear una semántica. `SemanticItem mySemantic = new SemanticItem(engine, name, null, false)` • Invocar a `myGrammaticalRepository.DeleteSemantic(mySemantic)` ○ Devuelve **true.** • Comprobar mediante la function de *retrieval* `GetSemanticsWithoutFirstRowForComboBox (name)` que la semántica eliminada ya no existe.
Especificaciones de entrada:	• Crear una nueva semántica mediante su constructor: ○ **Name**: myNewTermTagToTestDeleted ○ **ParentSemantic**: null ○ **RotateConcept**: false

Especificaciones de salida:	Tras eliminar una semántica e intentar recuperarla después, dicha semántica ya no existe.
Produce excepción:	No

Tabla 191: PU-038

PU-039: Edición de una sugerencia	
Descripción:	Comprueba la correcta edición de la información de una sugerencia del sistema. No se pueden crear sugerencias desde *knowledgeManager*, por lo que esta prueba se llevará a cabo sólo si ya hay alguna sugerencia creada.
Función:	`UpdateSuggestion(Suggestion editingSuggestion, string descripcion, string estado, string respuesta)`
Pasos:	• Seleccionar la primera sugerencia: `Suggestion` mySuggestion = engine.Suggestions.FirstOrDefault().Value; • Si mySuggestion es *null*, no se sigue la prueba, ya que significa que no hay sugerencias creadas. • Invocar llamada a `UpdateSuggestion(mySuggestion, "newDescription", newStatus", "new answer")`. ○ Devuelve **true**. • Volver a obtener mySuggestion = engine.Suggestions.FirstOrDefault().Value; • Comprobar que los atributos de *mySuggestion* coinciden con los que se invocaron a la función *UpdateSuggestion*.
Especificaciones de entrada:	
Especificaciones de salida:	Tras la llamada a la función de este caso de prueba, el último resultado devuelto es **true**.
Produce	No

excepción:	

Tabla 192: PU-039

4.1.12 Revisión de la planificación de pruebas

Según la planificación inicial, las pruebas no deben suponer el mayor esfuerzo en cuanto a tiempo dedicado, ya que se avanza en paralelo con la codificación del sistema.

Mientras que no haya cambios bruscos en la especificación del sistema y/o en la planificación, se mantiene la idea de llevar a cabo las pruebas unitarias a la vez que la codificación, en lugar de emplear una gran cantidad de horas de *testing* una vez que el sistema haya sido totalmente construido.

5 Planificación

Se realiza una planificación de tareas **en función de la experiencia** adquirida durante el desarrollo del Grado en Ingeniería Informática de la UC3M y las experiencias profesionales fuera de ella.

No se utilizan herramientas complejas para llevar a cabo una estimación del esfuerzo necesario para llevar a cabo la aplicación, ya que se trata de un sistema no lo suficientemente complejo como para aplicar este tipo de técnicas.

Esta planificación se realiza en función de la normativa de la universidad con respecto a los créditos ECTS. Recordamos que el presente trabajo de fin de grado equivale a 14 créditos ECTS, que traducidos en esfuerzo suponen alrededor de **300 horas de trabajo** del alumno.

Para simplificar el reparto de horas, se divide el trabajo a realizar en distintas fases:

Tarea	Horas estimadas	Fecha de inicio	Fecha de fin
Investigación, definición del estado del arte	40	01/03/2012	30/03/2012
Planificación	2	26/03/2012	30/03/2012
Presupuesto	3	26/03/2012	30/03/2012
Análisis	30	02/04/2012	04/05/2012
Diseño	50	09/04/2012	29/06/2012
Implementación	95	16/04/2012	10/08/2012
Pruebas	30	01/05/2012	10/08/2012
Seguimiento	10	01/06/2012	17/08/2012
Debrief	10	06/08/2012	17/08/2012

Tabla 193: Estimación de horas inicial

La fecha de inicio del proyecto, como vemos en la tabla, es el día 1 de marzo de 2012. Con el fin de conocer una aproximación final del proyecto, se realiza una estimación del tiempo que va a suponer cada tarea.

Por necesidades de la propia universidad y ajenas al proyecto, se decide que se va a seguir una jornada laboral variable. No se trabaja todos los meses por igual. El comienzo del proyecto se hará de forma relajada pero constante. La parte media supondrá una mayor carga de trabajo, mientras que la parte final

será de nuevo más relajada, si se cumplen los plazos establecidos para cada fase.

Mes	Días laborables	Jornada laboral	Total horas
Marzo	22	2 horas / día	44 horas
Abril	22	2 horas / día	44 horas
Mayo	22	3 horas / día	66 horas
Junio	22	3 horas / día	66 horas
Julio	15	2 horas / día	30 horas
Agosto	10	2 horas / día	20 horas
			270 horas

Tabla 194: Reparto de las horas en meses

A continuación se incluye un desglose de tareas a realizar cada mes, dividido en semanas.

Marzo					
Tarea	Semana 1 (2 días)	Semana 2 (5 días)	Semana 3 (5 días)	Semana 4 (5 días)	Semana 5 (5 días)
Estado del arte	4	10	10	10	6
Planificación	0	0	0	0	2
Presupuesto	0	0	0	0	3

Tabla 195: Planificación de marzo

Estimación de marzo: Se dedican las cuatro primeras semanas a investigar sobre el estado del arte y a evaluar qué opciones de mercado aportan una posible solución al problema planteado por el cliente. Una vez comprendida la primera fase, se realiza una planificación y se elabora un presupuesto para fin

de mes. Se dedica una hora más la última semana al estado del arte para completar las 40 horas estimadas en un principio.

- Se finaliza el Estado del arte → 40 horas totales
- Se finaliza al planificación → 2 horas totales
- Se finaliza el presupuesto → 3 horas totales

Abril					
Tarea	Semana 1 (5 días)	Semana 2 (5 días)	Semana 3 (5 días)	Semana 4 (5 días)	Semana 5 (1 día)
Análisis	10	7	5	4	0
Diseño	0	3	3	4	2
Implementación	0	0	2	2	0

Tabla 196: Planificación de abril

Estimación de abril: Una vez comprendido el problema, se comienza a analizar en detalle el problema planteado por el cliente. Comienzan las fases de análisis y diseño del sistema en paralelo, con una gran carga de trabajo del mes, y ligeramente posterior se comienza con la planificación, debido al uso de metodologías ágiles de desarrollo.

Mayo					
Tarea	Semana 1 (4 días)	Semana 2 (5 días)	Semana 3 (5 días)	Semana 4 (5 días)	Semana 5 (4 día)
Análisis	4	0	0	0	0
Diseño	5	5	5	5	5
Implementación	2	9	9	9	6
Pruebas	1	1	1	1	1
Seguimiento	0	0	0	0	0

Tabla 197: Planificación de mayo

Estimación de mayo: Durante la primera semana de mayo se finaliza el análisis, de forma que durante el resto del mes se pone más hincapié en el diseño y codificación del sistema. A su vez, se comienzan las pruebas para avanzar en paralelo junto con el desarrollo.

- Se finaliza el análisis: 30 horas totales

Junio					
Tarea	Semana 1 (1 días)	Semana 2 (5 días)	Semana 3 (5 días)	Semana 4 (5 días)	Semana 5 (5 día)
Diseño	0	4	4	3	2
Implementación	0	10	10	11	12
Pruebas	0	1	1	1	1
Seguimiento	3	0	0	0	0

Tabla 198: Planificación de junio

Estimación de junio: el diseño entra en su recta final, ya que se finaliza a final de mes, y se pone el foco de atención en la implementación del sistema. A la vez se sigue dedicando tiempo a realizar continuas pruebas.

- Se finaliza el diseño: 50 horas totales

Julio					
Tarea	Semana 1 (5 días)	Semana 2 (5 días)	Semana 3 (5 días)	Semana 4 (5 días)	Semana 5 (2 día)
Implementación	5	3	0	------	------
Pruebas	2	7	7	------	------
Seguimiento	3	0	3	------	------

Tabla 199: Planificación de julio

Estimación de julio: se trabaja durante las tres primeras semanas. La planificación durante este mes puede ser cambiante, ya que se llevan a cabo dos seguimientos con el fin de determinar cuánta carga de trabajo restante queda y tratar de que no se descontrole. Se sigue avanzando en la implementación y pruebas del sistema.

Agosto					
Tarea	Semana 1 (3 días)	Semana 2 (5 días)	Semana 3 (5 días)	Semana 4 (5 días)	Semana 5 (5 día)
Implementación	------	5	0	0	0
Pruebas	------	5	0	0	0
Seguimiento	------	0	1	0	0
Debrief	------	3	7	0	0

Tabla 200: Planificación de agosto

Estimación de agosto: Hay que dedicar algo más de esfuerzo durante la segunda semana para acabar de probar todo el sistema y corregir errores, con el fin de dedicar la tercera semana íntegra a hacer un seguimiento final y sacar conclusiones del proyecto. Según la estimación realizada, el proyecto finaliza al final de la tercera semana de agosto, es decir, el **17 de agosto de 2012**.

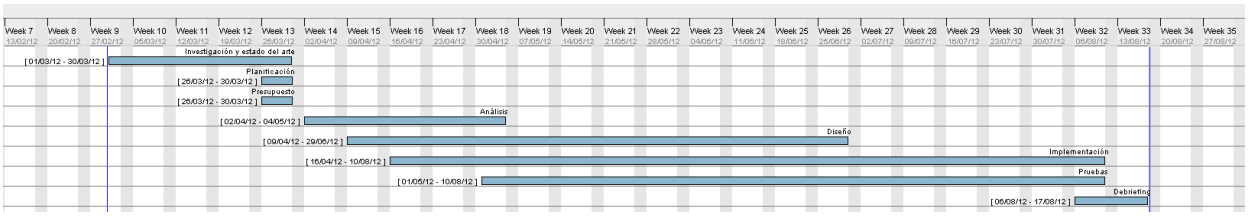

Ilustración 40: Diagrama de Gantt según planificación

Fecha de finalización estimada del sistema de información: 17 de agosto de 2012.

Avance del proyecto estimado para fecha de fin: 100% completado.

6 Presupuesto

A partir de las fases definidas en la *Planificación*, se definen **distintos roles** para llevar a cabo cada una de ellas. Para calcular el coste por hora de cada rol, se parte del coste bruto anual. Se consideran 14 pagas por año, y cada mes tiene 22 días laborables, de forma que cada día laborable son 8 horas de trabajo.

Rol	Sueldo bruto anual	Sueldo bruto por hora (coste)
Jefe de proyecto	115000 € / año	46.67 € / hora
Analista	75000 € / año	30.44 € / hora
Programador	28000 € / año	11.37 € / hora

Tabla 201: Salario bruto por rol

Una vez conocido el coste por hora de cada rol, se calcula el número de horas que trabaja cada uno de ellos para conocer el **coste total en cuanto a personal de desarrollo**.

	Jefe de proyecto	Analista	Programador
Estado del arte	10 horas	30 horas	0 horas
Planificación	2 horas	0 horas	0 horas
Presupuesto	3 horas	0 horas	0 horas
Análisis	5 horas	25 horas	0 horas
Diseño	10 horas	40 horas	0 horas
Implementación	0 horas	0 horas	95 horas
Pruebas	0 horas	0 horas	30 horas
Seguimiento	8 horas	2 horas	0 horas
Debrief	7 horas	3 horas	0 horas
Horas por rol	45 horas	100 horas	125 horas
Coste por rol	2100.15 €	3044 €	1421.25 €
		Total	**6565.4 €**

Tabla 202: Horas planificadas para cada rol

Además de los costes del personal del equipo de desarrollo, se incluyen los costes de material y transporte que supone el proyecto. En cuanto al portátil, para calcular el valor se considera que es útil durante dos años, es decir 730 días.

Material	Cantidad	Cálculo realizado	Coste
Ordenador portátil	1 (113 días de uso)	(CostePortatil/DiasUtiles) * DiasDeUso = 950/730*113	147.1 €
Microsoft Windows 7	1	MSDN Academic Alliance	0 €
Visual Studio 2010	1	MSDN Academic Alliance	0 €
Microsoft Office 2010	1	MSDN Academic Alliance	0 €
CD-ROM	5	5*0,2	1 €
Conexión a internet	6 meses	CosteMes*NumeroMeses = 39,9*6	239.4 €
Transporte	6 meses	CosteMes*NumeroMeses = 37,9*6	227.4 €
Total			**614,9 €**

Tabla 203: Coste de material, viaje y fungible

Por último se hace un resumen con el coste total del proyecto. A partir del coste de personal más material, transporte y fungible, se añade un porcentaje para costes indirectos del proyecto que puedan no haberse tenido en cuenta (15%), y un margen de riesgo que en este caso será del 8%.

Resumen del presupuesto		
Concepto	Cantidad	Total
Costes directos del proyecto		6565.4 €

Costes indirectos del proyecto (15%)	984.81 €	7550.21 €
Margen de riesgo (8%)	604.02 €	8154.23 €
Beneficio (15%)	1223.14 €	9377.35 €
IVA (18%)	1687.93 €	7493.03 €
Precio final		**11065.28 €**

Tabla 204: Resumen del presupuesto y precio final

7 Debriefing

Resultados obtenidos

En este apartado se comparan los resultados iniciales planificados con los resultados reales obtenidos. No se utilizan algoritmos complejos, sino que se utilizan los datos obtenidos del seguimiento realizado durante todo el ciclo de vida del proyecto.

Cuando se acabó de estudiar el estado del arte durante el mes de marzo, se realizó una planificación y un presupuesto en función de la experiencia con otras estimaciones anteriores. Ya se había trabajado previamente con la tecnología actual, por lo que los factores de riesgo tal vez no serían tan abultados como en proyectos anteriores.

En un principio se planificó que la fecha de fin del proyecto sería el 17 de agosto de 2012. Se fijó esa fecha tope para tener margen de maniobra en caso de que las cosas no fueran como se habían previsto, ya que la fecha de entrega del TFG para corrección del tribunal es el día 5 de septiembre de 2012.

Por un lado se ha producido un ligero **retraso en cuanto a la fecha de entrega**, ya que la fecha real de finalización del proyecto ha sido el día 31 de agosto. Por otro lado, **se han conseguido todos los objetivos fijados** durante el análisis y diseño del sistema, y se ha construido al 100% el módulo del *knowledgeManager* que se había pensado desde un principio.

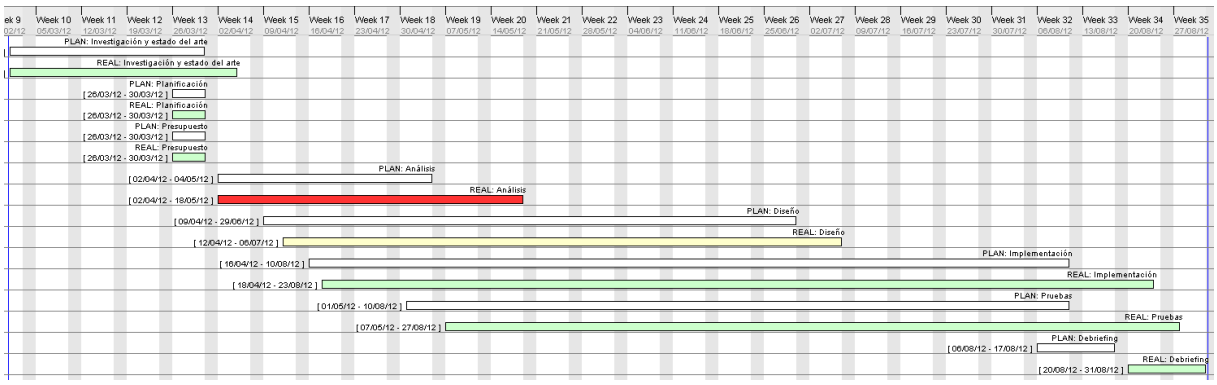

Ilustración 41: Comparación Planificado-Real

En la gráfica se aprecia la comparativa entre el esfuerzo planificado y el esfuerzo real. Las tareas en blanco representan las fechas de inicio y fin planificadas, y la inmediatamente inferior son las fechas de inicio y fin reales de dicha tarea (**verde** significa que no produce retraso en las tareas posteriores; **amarillo** significa que produce un ligero retraso en las tareas posteriores; **rojo** significa que retrasa considerablemente el resto del proyecto).

La conclusión ha sido dedicar horas extra durante el mes de agosto, ya que se había planificado dedicar durante dicho mes 10 días a 2 horas por día para un total de 20 horas, una planificación demasiado optimista ya que se trata de la

recta final del proyecto. En concreto comenzó a demorar el documento de análisis, lo que ha supuesto un efecto dominó sobre el diseño y la codificación y *testing* del sistema. En total se han utilizado unas 20 horas más de lo esperado, 290 horas reales en total frente a las 270 estimadas inicialmente.

Evaluación y conclusiones

Uno de los puntos más importantes tanto a nivel individual como a nivel grupal que he aprendido durante los cuatro años en la universidad es la importancia de analizar un producto una vez ha sido acabado.

En este caso he sido yo solo quien ha desarrollado el proyecto completo, por lo que no tengo ningún grupo al que valorar. Aun así, mantengo en la memoria de mi TFG este apartado de *reflexión* y *lecciones aprendidas* aunque no sea obligatorio en los criterios de evaluación.

En mi opinión, la realización del TFG me ha sido de gran utilidad para reforzar todos los procesos de ingeniería por los que pasa un proyecto software a lo largo del ciclo de vida. Hemos tenido varias asignaturas quizá demasiado teóricas, en las que se nos daban ciertas pautas sobre cómo hacer las cosas, pero realmente no teníamos la opción de poner en práctica toda esa parte teórica.

Seguramente el hecho de coordinar un grupo de personas para realizar un proyecto común es una de las tareas más complicadas en el mundo de la ingeniería. Durante el último año de carrera (4º curso de grado en Ingeniería Informática), gran parte de las prácticas eran simulaciones de desarrollo de un proyecto real entre un número “grande” de gente. Sin embargo, es de agradecer el desarrollo individual del TFG, ya que es infinitamente más sencillo de planificar, organizar, replanificar y ver cómo de bien o de mal estoy avanzando en él.

Destacar también el uso de metodologías ágiles. Aunque no se aplique estrictamente al cien por cien una determinada metodología de desarrollo ágil, no se ha llevado a cabo el proyecto según una metodología de diseño tradicional, donde quizá la documentación tiene demasiado peso con respecto al esfuerzo total del proyecto bajo mi punto de vista, por lo que he visto durante la carrera en asignaturas de desarrollo tradicional.

El principal problema encontrado durante el desarrollo del sistema de información ha sido, a mi juicio, no dimensionar del todo bien el trabajo a desarrollar. A priori no parece un sistema demasiado grande, pero a la mínima que comienza a retrasarse, el retraso crece exponencialmente. Finalmente la funcionalidad ha sido acabada en tiempo.

Por último, con vistas a líneas futuras y desarrollo de mi carrera profesional en la empresa, darme cuenta de mis errores y aprender de ellos para evitar que me vuelvan a ocurrir en el futuro.

8 Anexo A: Manual de usuario

Introducción

El módulo desarrollado para *knowledgeManager* permite gestionar el dominio de una ontología a través de:

a) Un vocabulario controlado mediante una lista de **términos**.

b) Una ontología ligera mediante **etiquetas sintácticas** y **semánticas**.

A continuación se detallan las distintas acciones que la aplicación nos permite realizar. No se muestran detalles para la instalación y conexión con la base de datos, ya que es necesario adquirir una licencia por parte de *The Reuse Company*.

1. Formulario inicial.
2. Creación de un elemento del dominio.
3. Modificación de un elemento del dominio.
4. Borrado de un elemento del dominio.
5. Duplicado de un elemento del dominio.
6. Gestión del subdominio: sugerencias

Un “elemento del dominio” engloba a términos, etiquetas sintácticas y semánticas. Las acciones a realizar de todos ellos son similares, por lo que se tomará uno de ellos como referencia.

Formulario inicial

Una vez se introducen correctamente las credenciales para acceder a la aplicación, se accede a un primer formulario en el que encontramos un menú horizontal superior con dos partes diferenciadas de la aplicación:

a) **Gestión del dominio**: gestión del vocabulario controlado y la ontología ligera.

b) **Gestión del subdominio**: gestión de las sugerencias de términos y relaciones entre términos.

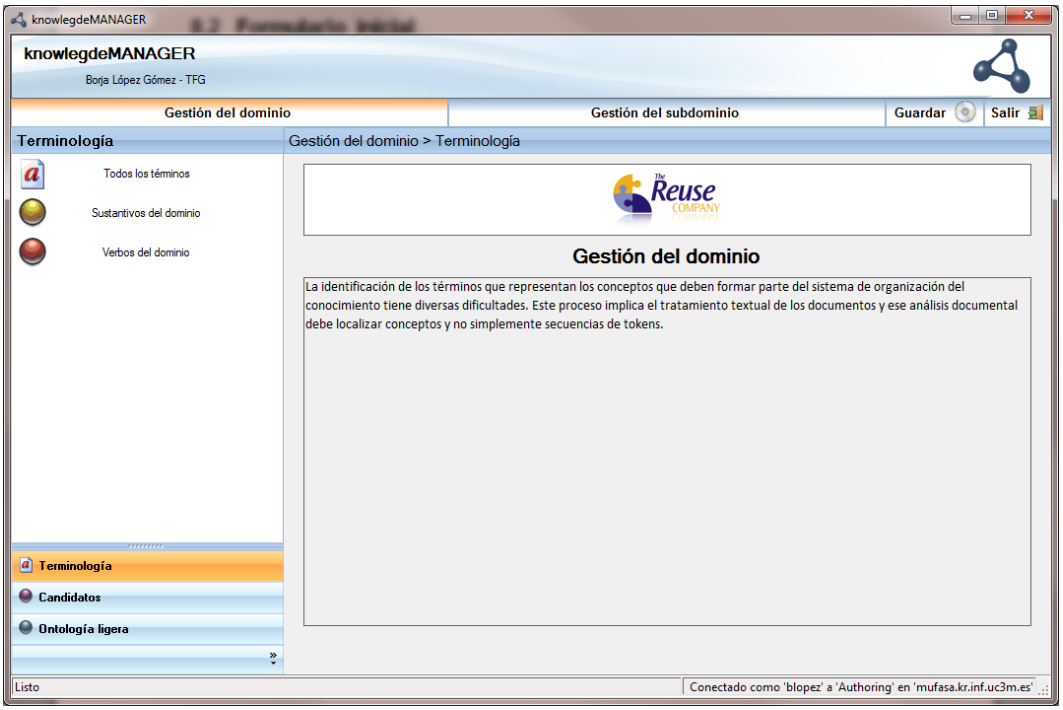

Ilustración 42: Formulario inicial de la aplicación

Creación de un elemento del dominio

Como comentamos en el apartado anterior, un "elemento del dominio" se refiere tanto a términos, como a etiquetas sintácticas y semánticas. Al tener funcionalidades muy similares entre ellos, se coge uno para hacer de modelo, por ejemplo etiquetas sintácticas.

Lo primero es entrar a la sección de etiquetas sintácticas: Gestión del dominio → Ontología ligera → Etiquetas sintácticas

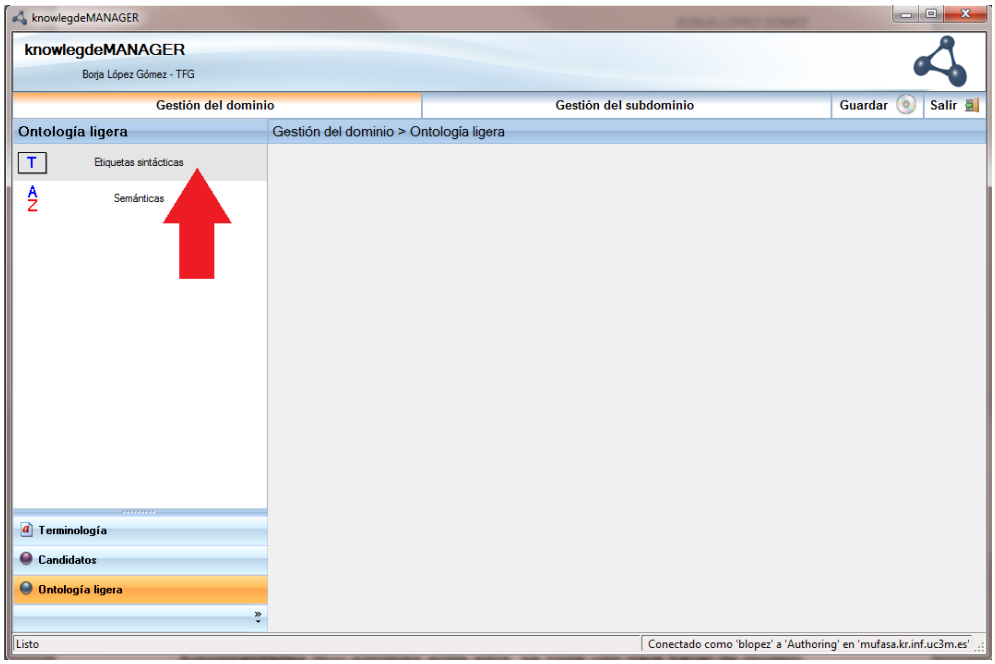

Ilustración 43: Acceder a la gestión de etiquetas sintácticas

A continuación hacemos click con el botón derecho sobre el listado de semánticas (superior izquierda), y elegimos la opción 'Añadir etiqueta sintáctica'.

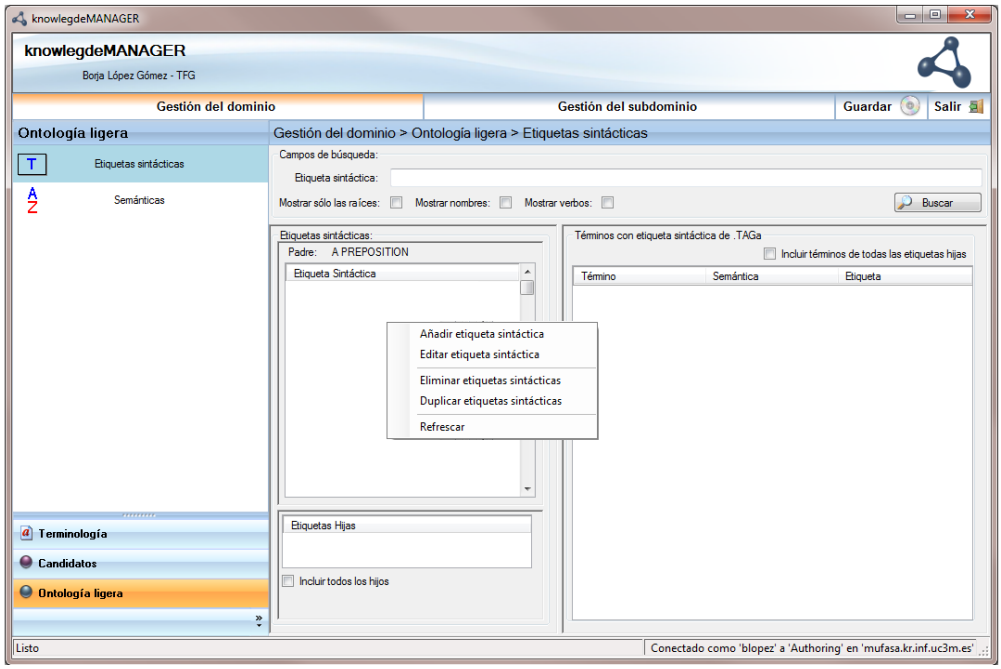

Ilustración 44: Opción añadir un nuevo elemento

Se abre un nuevo formulario donde debemos rellenar los atributos que va a tener nuestro nuevo elemento, y hacer click sobre el botón Aceptar. Cabe

destacar que, en caso de haber insertado incorrectamente algún atributo, como por ejemplo dejar vacío un atributo obligatorio, se muestra un icono de error de tal forma que poniendo el ratón encima aparece un mensaje descriptivo sobre qué está pasando.

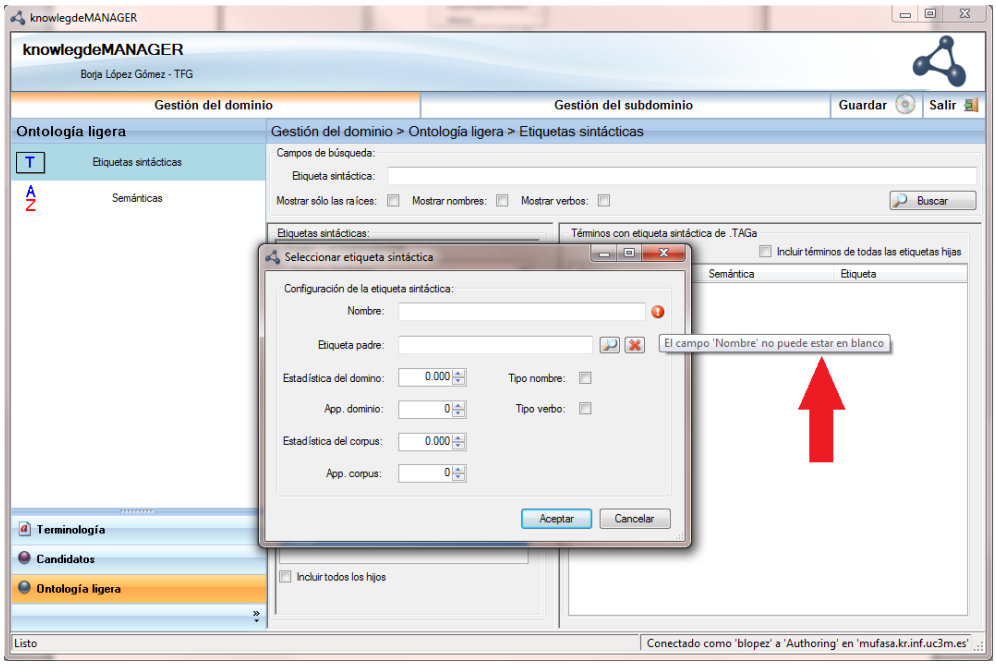

Ilustración 45: Validación de atributos en la interfaz

Modificación de un elemento del dominio

Al igual que en el apartado anterior, el primer paso es ir hasta la ventana de gestión de etiquetas sintácticas. Se asume que el usuario ya está familiarizado con la navegación de la aplicación, por lo que no se adjunta ilustración de nuevo.

Seleccionamos el elemento que deseamos editar, y con el botón derecho seleccionamos la opción ‘Editar etiqueta sintáctica’.

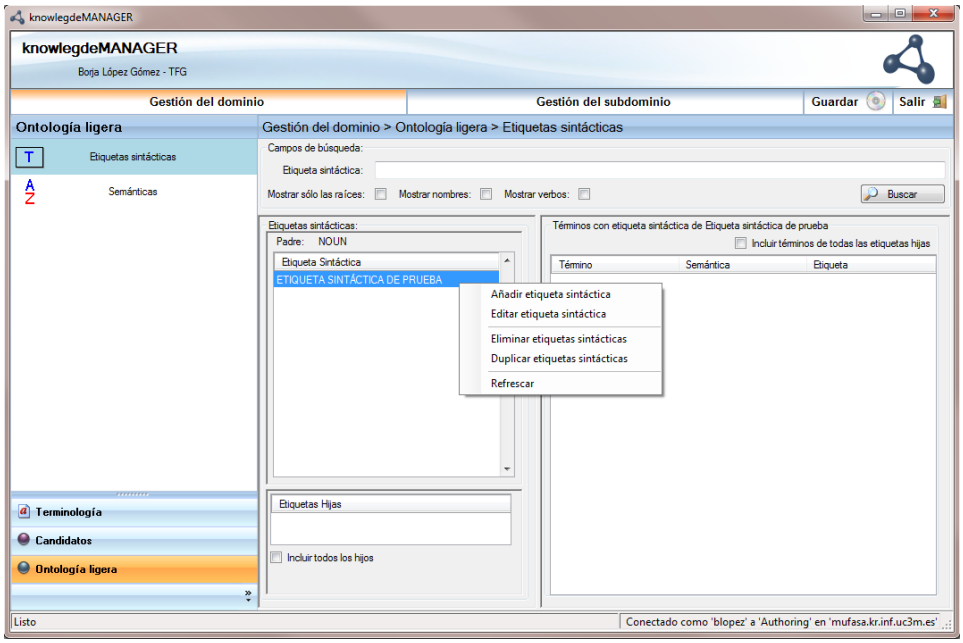

Ilustración 46: Opción de editar un elemento existente

De nuevo se nos abre un formulario con todos los datos del elemento. Ahora es cuando se debe modificar los datos necesarios y hacer click en Aceptar para validar los cambios.

Borrado de un elemento del dominio

Una vez más, partimos de la ventana con el listado de elementos. Para eliminar un elemento, se selecciona el que se quiera borrar, se hace click con el botón derecho y se elige la opción de 'Eliminar etiqueta sintáctica'. El sistema pide confirmación por parte del usuario, y se elimina el elemento seleccionado.

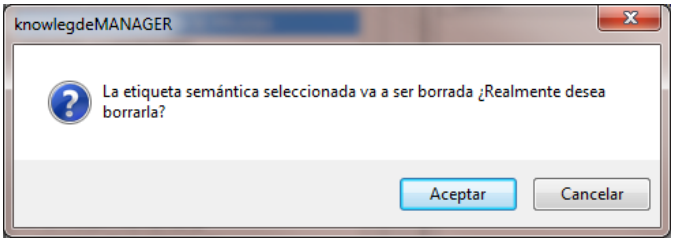

Ilustración 47: Confirmación de borrado de una etiqueta sintáctica

Duplicado de un elemento del dominio

Una vez más, partimos de la ventana con el listado de elementos. Para duplicar un elemento, se selecciona el que se quiera duplicar, se hace click con el botón derecho y se elige la opción de 'Duplicar etiqueta sintáctica'.

En ocasiones no pueden coexistir dos elementos del mismo tipo con la misma información, por lo que el sistema te pide que tras el duplicado cambies algún dato.

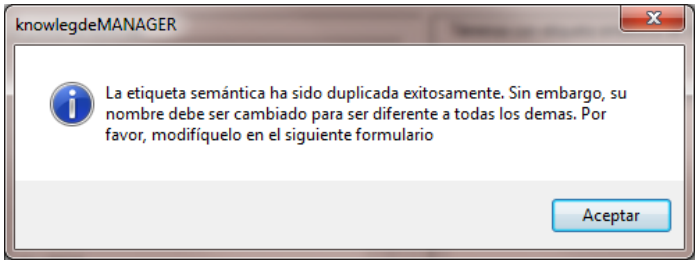

Ilustración 48: Solicitud de cambio duplicar una etiqueta sintáctica

Gestión del subdominio: sugerencias

El apartado de Sugerencias es un tanto especial, ya que no es posible la creación de sugerencias desde *knowledgeManager*. En concreto, una *sugerencia*, como su propio nombre indica, es una idea que otra persona ha tenido para crear un término o una relación entre términos desde otra aplicación, como por ejemplo desde *Requirements Authoring Tool* [11].

Por lo tanto, la única operación que se puede hacer sobre una sugerencia es modificar sus atributos:

- **Estado**: Sugerida/Rechazada/Aprobada/Ejecutada
- **Respuesta**: justificación del cambio de estado que leerá el creador.

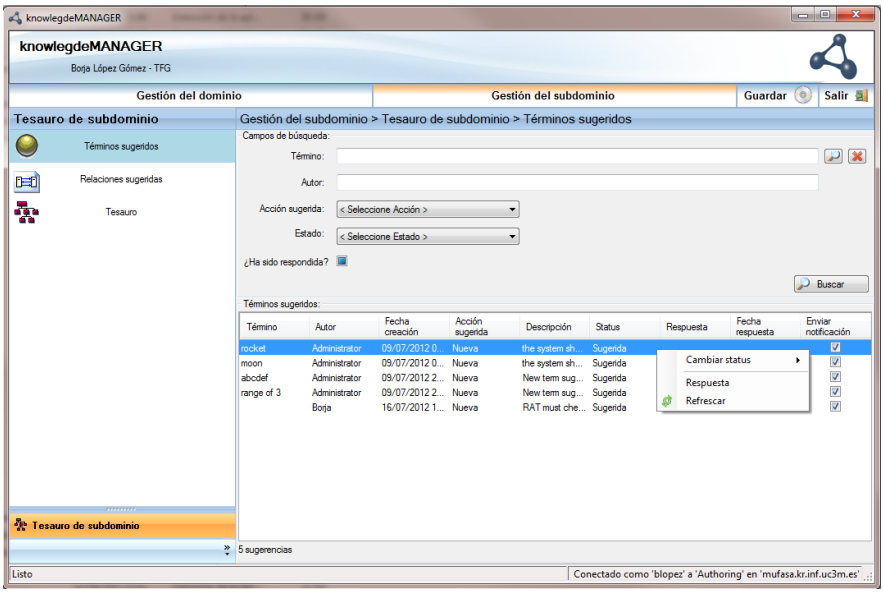

Ilustración 49: Opciones sobre una sugerencia

9 Anexo B: Glosario

Definiciones

- **Broader term**: relación del tesauro que indica jerarquía padre-hijo.
- **Business Logic Layer**: capa de negocio de una aplicación software que contiene la lógica de negocio de dicha aplicación.
- **Click**: acción de pulsar el botón izquierdo del ratón.
- **Cloud computing**: almacenamiento de información en computadores ajenos donde la transmisión de datos se hace a través de internet.
- **ComboBox**: control de .Net que emula un menú desplegable.
- **Comma-Separated values**: ficheros de información masiva donde los distintos campos están separados entre sí por punto y coma.
- **Computer Aided Knowledge Environment**: entorno de herramientas destinadas a facilitar procesos de reutilización, *indexing* y *retrieval* de información.
- **Data Access Layer**: capa de datos de una aplicación software que se encarga de la comunicación entre la aplicación y la base de datos.
- **Etiqueta sintáctica**: se encargan de identificar cada término como perteneciente a una clase o categoría gramatical.
- **Etiqueta sintáctica hija**: etiqueta sintáctica que tiene jerárquicamente otra etiqueta sintáctica por encima.
- **Etiqueta sintáctica padre**: etiqueta sintáctica que tiene jerárquicamente otra etiqueta sintáctica por debajo.
- **European Credit Transfer System**: sistema utilizado por las universidades europeas para convalidar asignaturas dentro del plan Bolonia.
- **eXtensible Markup Language**: lenguaje de etiquetado utilizado para intercambiar información.
- **Facade**: Patrón de diseño que se utiliza para indicar que se trata de una interfaz.
- **Feedback**: realimentación sobre un determinado proceso.
- **Framework**: conjunto de herramientas empleadas para llevar a cabo una acción.
- **Gazetteers**: diccionario o directorio geográfico que contiene información acerca de lugares y nombres de lugares.
- **Graphical User Interface**: interfaz de usuario utilizada para que un ser humano pueda interaccionar con un computador.
- **Hardware**: partes tangibles de un sistema informático.

- **Indexación**: proceso por el cual se obtiene y almacena información.
- **Indexing**: nombre que se le da al proceso de *indexación*.
- **Information technology**: relacionado con las tecnologías de la información.
- **Knowledge organization system**: sistema de organización del conocimiento, capaz de almacenar jerárquicamente una determinada información.
- **KnowledgeManager**: software gestor de un sistema de organización del conocimiento, creado y desarrollado por *The Reuse Company*.
- **Macintosh**: nombre que se le da a los ordenadores desarrollados por la compañía *Apple*.
- **Meta-ontología**: conceptos que se limitan a un determinado campo.
- **MySql**: sistema gestor de bases de datos de Sun.
- **Narrower term**: relación del tesauro que indica jerarquía padre-hijo.
- **Normalización**: conjunto de procesos que se aplican sobre una entrada para obtener una determinada salida.
- **Null**: dato vacío.
- **NUnit**: librería utilizada para realizar pruebas unitarias de código.
- **Open-source**: código libre que puede transmitirse sin necesidad de utilizar licencias que cuestan dinero.
- **Personal-computer**: ordenador usado generalmente para uso cotidiano.
- **Presentation layer**: capa de presentación de una aplicación software que contiene la interfaz de usuario.
- **Proceso integral**: procesos que se llevan a cabo durante todo el ciclo de vida de un proyecto software.
- **Related term**: relación del tesauro.
- **Resource Description Framework**: lenguaje o descripción cocneptual empleado para representar información en la web.
- **Retrieval**: procesos de recuperación de la información.
- **Root**: raíz, que no tiene jerárquicamente nada por encima.
- **Semántica**: significado, sentido o interpretación que se le da a un término.
- **Semántica hija**: semántica que jerárquicamente tiene otra semántica por encima.
- **Semántica padre**: semántica que jerárquicamente tiene otra semántica por debajo.

- **SetUp**: atributo de NUnit que indica que un método se ejecuta justo antes de la ejecución de cada método de prueba.
- **Sistema caja blanca**: sistema que se conoce su funcionamiento interno.
- **Sistema caja negra**: sistema del que no se conoce su funcionamiento interno, tan sólo sus entradas y las salidas que proporciona.
- **Sistema empotrado**: sistemas dedicados generalmente a cubrir funciones en un sistema de tiempo real.
- **Software**: componentes lógicos de un sistema informático que hacen posible su funcionamiento.
- **Sql Server**: sistema gestor de base de datos de Microsoft.
- **Stakeholder**: cualquier persona interesada en el desarrollo de una acción.
- **Suite de pruebas**: conjunto de métodos de prueba.
- **Tablet**: computadora portátil ligeramente mayor que un teléfono móvil.
- **Testing**: proceso de prueba.
- **Universo del discurso**: conjunto de cosas de las que se habla en un determinado contexto.
- **Ventana de splash**: ventana intermedia de espera que es mostrada cuando se ejecuta un proceso que tarde cierto tiempo.
- **Web Ontology Language**: lenguaje de marcado utilizado para publicar y transmitir información utilizando ontologías a través de la web.
- **Wordnet**: base de datos léxica en inglés que agrupa un conjunto de sinónimos, proporcionando definiciones cortas y generales.

Acrónimos

- **BLL**: Business Logic Layer
- **BT**: Broader Term
- **CAKE**: Computer Aided Knowledge Environment
- **CSV**: Comma-Separated Values
- **DAL**: Data Access Layer
- **ECTS**: European Credit Transfer System
- **GUI**: Graphical User Interface
- **IT**: Information Technology
- **KOS**: Knowledge Organization System
- **NT**: Narrower Term

- **OWL**: Web Ontology Language
- **RDF**: Resource Description Framework
- **RT**: Related Term
- **TFG**: Trabajo de Fin de Grado
- **XML**: eXtensible Markup Language

10 Anexo C: Bibliografía

- **[1]:** [WEB1] Informe del caos, *the chaos report*. http://www.pmhut.com/the-chaos-report-2009-on-it-project-failure**.**
    - Última visita: marzo 2012
- **[2]**: [WEB2] Visión general de KOS. http://www.clir.org/pubs/reports/pub91/1knowledge.html/
    - Última visita: marzo 2012
- **[3]**: [WEB3] Ontology: overview. http://www.iva.dk/bh/lifeboat_ko/CONCEPTS/ontology.htm
    - Última visita: marzo 2012
- **[4]**: [WEB4] Definición de ontología. http://elies.rediris.es/elies18/531.html
    - Última visita: agosto 2012
- **[5]**: [WEB5] Ontologías en profundidad. http://www.hipertexto.info/documentos/ontologias.htm
    - Última visita: abril 2012
- **[6]**: [WEB6] Ciclo de vida del software. http://www.comusoft.com/wp-content/uploads/2011/02/ciclos-de-vida-del-software.jpg
    - Última visita: mayo 2012
- **[7]**: [WEB7] Esfuerzos por fase durante el ciclo de vida. http://mdjesus.files.wordpress.com/2010/05/pu.gif
    - Última visita: mayo 2012
- **[8]**: [WEB8] Lenguajes de programación más usados, evolución histórica. http://www.genbetadev.com/lenguajes-de-programacion/ranking-2011-de-lenguajes-mas-usados-java-se-mantiene-lider-y-objective-c-dobla-sus-resultados
    - [Última visita: marzo 2012
- **[9]**: [WEB9] Información de la licencia Windows 7 Ultimate. http://emea.microsoftstore.com/es/es-ES/Microsoft/Windows-7-Ultimate
    - [Última visita: marzo 2012

- **[10]**: [WEB10] Información de la licencia VS 2010 Professional. http://www.microsoft.com/visualstudio/en-us/products/2010-editions/professional/overview
    - Última visita: marzo 2012
- **[11]**: [WEB11] Información sobre el software *Requirements Authoring Tool.* http://www.reusecompany.com/index.php?option=com_content&view=category&layout=blog&id=216&Itemid=184%2C&lang=es
    - Última visita: julio 2012
- **[12]**: [WEB12] Información sobre el software *Requirements Quality Analizer.* http://www.reusecompany.com/index.php?option=com_content&view=category&layout=blog&id=53&Itemid=80&lang=es
    - Última visita: junio 2012
- **[13]**: [WEB13] Gestor de ontologías *SWOOP.* http://code.google.com/p/swoop/
    - Última visita: agosto 2012
- **[14]**: [WEB14] Gestor de ontologías DOME. http://dome.sourceforge.net/
    - Última visita: agosto 2012
- **[15]**: [WEB15] Página principal de *The Reuse Company*. http://www.reusecompany.com
    - Última visita: agosto 2012
- **[16]**: [WEB16] Paper sobre ontologías. http://www.elprofesionaldelainformacion.com/contenidos/2007/noviembre/03.pdf
    - Última visita: marzo 2012
- **[17]**: [WEB17] Gestor de ontologías PROTEGE. http://protege.semanticweb.org/
    - Última visita: agosto 2012
- **[18]**: [WEB18] Proyecto de tesis de Vicente Palacios Madrid – UC3M. http://163.117.147.101:9000/semse/index2.php?option=com_docman&task=doc_view&gid=13&Itemid=39

  - [Ultima visita: marzo 2012]

- **[19]**: Llorens J., Morato J., Genova G. RSHP: An information representation model based on relationships. In: Ernesto Damiani, Lakhmi C. Jain, Mauro Madravio (Eds.), Soft Computing in Software Engineering (Studies in Fuzziness and Soft Computing Series, Vol. 159), Springer, pp 221-253. 2004.
- **[20]**: Llorens, J., Fuentes J.M., Diaz I. RSHP: A Scheme to Classify Information in a Domain Analysis Environment. IEEE-AAAI- ICEIS 2001. International Conference on Enterprise Information Systems. Actas del congreso, pp. 686-690. Setúbal, Portugal. 2001
- **[21]**: Sanchez-Cuadrado, Sonia. Definición de una metodología para la construcción automatizada de sistemas de organización del conocimiento. Directores: José Antonio Moreiro González y Jorge Morato Lara. Universidad Carlos III de Madrid, Departamento de Biblioteconomía y Documentación. 2007.
- **[22]**: A. Fraga. A Methodology for reusing any kind of knowledge at low cost: Universal Knowledge Reuse. Advisors: Juan Llorens and Gonzalo Génova. Carlos III of Madrid University. Computer Science Engineering. 2010.
- **[23]**: Roger S. Pressman. Ingeniería del Software: un enfoque práctico, 5ª ed. 2002. McGraw-Hill.
- **[24]**: Ian Sommerville. Ingeniería del Software, 7ª ed. 2005. Pearson Addison.
- **[25]**: Llorens, J.; Fuentes, J.: Computer Aided Knowledge Environment CAKE. The Reuse Company, Winter 2004.
- **[26]**: Guarino, N.: Formal ontology in information systems, 2004.
- **[27]**: Definición de ontología de Gruber. http://tomgruber.org/writing/ontology-definition-2007.htm
  - Última visita: agosto 2012
- **[28]**: Lambe, P.: Organizing knowledge: taxonomies, knowledge and organisational effectiveness. Chandos, 2007