INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

UNIVERSIDAD CARLOS III DE MADRID
- Escuela Politécnica Superior -

PROYECTO FIN DE CARRERA

DESARROLLO DE ONTOLOGÍAS SOBRE BBDD RELACIONALES: ONTOLOGÍA QUE SOPORTA LA BÚSQUEDA AVANZADA DE INFORMACIÓN

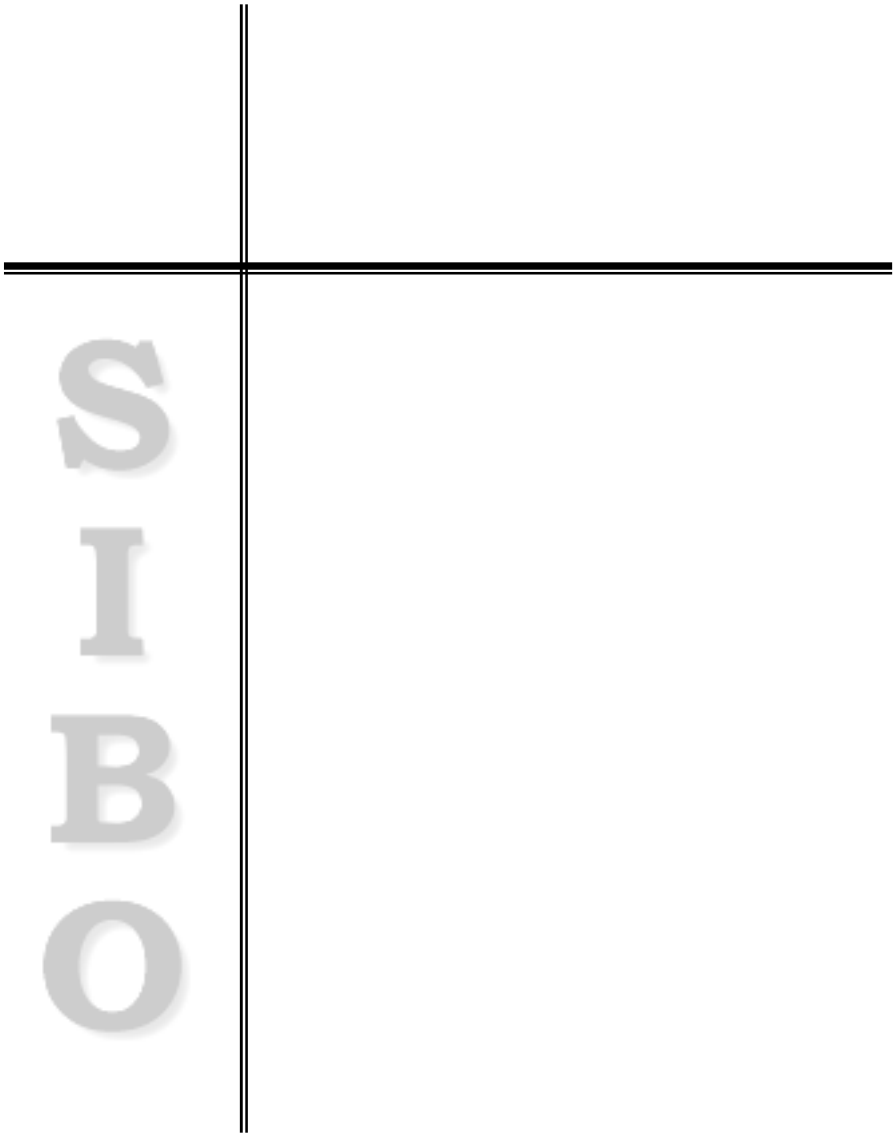
PROYECTO SIBO
Sistema de Información Basado en Ontologías

Autor: Ignacio Herrero Jiménez

Tutor: Javier Calle

Junio 2004

AGRADECIMIENTOS



A mis padres por su paciencia y comprensión en este largo camino

A mi hermano por acompañarme desde el primer día de mi vida

A mis colegas por esos veranos y los locos fines de semana necesarios para desconectar del estrés de la carrera

A mis amigos de la uni por hacerme más agradable las clases: Paqui, Mike, Carlos, Silvestre, Almudena, Miguel, Marcos, Javi, Victor...seguro que me olvido de muchos, pero es sin mala intención

A mi tutor Javi, por responderme a cada duda y aguantar mis avalanchas de e-mails y preguntas. A César por mantener el sistema y tenerme siempre listo el servidor de Oracle. A Samuel por esos conocimientos de Java (qué hubiera hecho sin ti?!)

A Elena por saber aportar en cada momento lo necesario

A todos los que de alguna u otra forma han hecho posible que haya terminado esta carrera, gracias

Nacho

ÍNDICES

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Iniciación al proyecto

SIBO

Antes de empezar este proyecto, se invita al lector a hacer la siguiente reflexión: en qué grado nuestra vida y la vida de las empresas, dependen de la informática y de los sistemas de información.

El empleo de ordenadores a gran escala comenzó en las empresas como una ayuda a la ejecución de ciertas operaciones. Hoy en día, millones de empresas se hundirían si no pudiesen utilizar sus sistemas informáticos para su gestión, producción, etc. El trabajo de los empleados, que antes se enfocaba a producir manualmente, ahora se centra en crear sistemas, operarlos y mantenerlos.

Pero más allá de los cambios en la informática industrial, es la revolución de la informática doméstica la que ha dado un giro en nuestro modo de vivir. Desde mediados de los 90, con la aparición de Windows95, el mundo de la informática doméstica tuvo su primera gran expansión. Por esos años también, lo que ahora todo el mundo utiliza sin apenas conocimientos de informática daba sus primeros pasos: la “world wide web”, coloquialmente conocida por internet. En la actualidad, es difícil encontrar un hogar en el que, al menos, no haya un ordenador.

Debido a la gran y rápida expansión de las nuevas tecnologías, como teléfonos móviles, agendas electrónicas, ordenadores portátiles, etc., se ha hecho imprescindible hacer estas tecnologías accesibles a usuarios sin un bagaje tecnológico. El número de personas que accede a internet crece cada día y también lo hace la información que se genera y se maneja a diario. Si esto es notorio en el ámbito doméstico, su importancia se multiplica en el entorno empresarial. La información que tiene que almacenar y gestionar una empresa cada vez es mayor y se hace inviable que seres humanos puedan mantener y actualizar los millones de bytes almacenados.

Una vez llegados a este punto, y con las expectativas que existen en torno a la informática, el siguiente paso es crear interfaces “inteligentes”, que permitan interactuar con los sistemas actuales, y que son cada día mas difíciles de diseñar y mantener. Una de las muchas tecnologías que permiten aproximar este tipo de sistemas es la referente a las ontologías.

Este proyecto se centra en el desarrollo de un nuevo sistema: SIBO (Sistema de Información Basado en Ontologías). Consta de dos partes bien diferenciadas. En primer lugar está la ontología, que es la razón de ser de este trabajo y el núcleo del proyecto. Por otro lado,

con el fin de hacer más agradable el acceso a la ontología se ha desarrollado una interfaz gráfica. Estos dos puntos se explicarán detalladamente más adelante.

El presente documento comienza con un somero estudio de la tecnología protagonista en este proyecto: las ontologías. A modo de introducción a este campo, se exponen sus características, componentes, uso y metodologías de desarrollo. El segundo capítulo está dedicado a relatar cuál ha sido el desarrollo de este proyecto, desde el punto de partida hasta su implementación, pasando por sus fases de análisis, diseño, implementación y pruebas. El tercer capítulo recoge las conclusiones de este trabajo y algunas de las líneas futuras más llamativas que presenta. Por otro lado, el cuarto capítulo está destinado a contener la bibliografía usada y recomendada a los lectores. El quinto y último capítulo, recoge los anexos que contienen documentación de referencia que puede resultar de utilidad, como parte de la documentación formal (documento de especificación de requisitos, manual de usuario), documentación de la carga de pruebas y algunos de los *scripts* de generación más relevantes.

CAPÍTULO 1

Conocimientos previos sobre las ontologías

1. Introducción. Necesidad de su uso
2. Definiciones de ontología
3. Clasificación de ontologías
4. Aplicaciones
5. Desarrollo de ontologías
6. Algunos sistemas actuales

1.1 INTRODUCCIÓN: NECESIDAD DE SU USO

En la sociedad actual, la información disponible es cada vez mayor. Esto irá en aumento y será imposible manejarla, mantenerla, entenderla, etc. Actualmente ya se han empezado a aplicar técnicas de inteligencia artificial en documentación para superar los límites de los métodos clásicos. El campo en el que más se han utilizado es en el de la recuperación de información, debido al aumento de ésta, que hace inviable métodos tradicionales.

Algunos de los problemas que pueden surgir cuando se maneja gran cantidad de información son los siguientes:

- Los actuales buscadores basados en palabras clave suelen devolver información irrelevante que usa una cierta palabra con un significado diferente del que se pretende en la búsqueda (polisemia), y pierden información cuando no reconocen palabras diferentes pero con el mismo significado que la buscada (sinonimia).
- Actualmente se requiere lectura humana para extraer información relevante de un origen, debido a que los agentes automáticos no tienen la capacidad necesaria para reconocer dicha información en representación textual.
- Mantener orígenes textuales débilmente estructurados representa una tarea dificultosa y consumidora de tiempo cuando tales orígenes son de un tamaño considerable. Mantener esas colecciones consistentes y al día requiere de información interpretable por computador y de semántica que ayude a detectar anomalías automáticamente.
- La utilidad de sitios web adaptativos que permitan su reconfiguración dinámica de acuerdo al perfil del usuario u otros aspectos relevantes requiere una representación computable de la semántica involucrada.

La documentación de los datos para dotarlos de una semántica resulta fundamental a la hora de la búsqueda y recuperación de los mismos. Los datos por sí solos, sin tener una semántica asociada, no son de utilidad, ya que resultan ambiguos. Hay, por lo tanto, que introducir esa semántica añadiendo datos que describan a su vez los propios datos, o lo que es lo mismo, metadatos. Sin embargo, para dar una descripción formal de contenidos mediante el modelado de conceptos y relaciones, hace falta algo más potente que los metadatos. De este modo se llega a las ontologías. La principal diferencia entre ambos es que los metadatos estructuran contenido y las ontologías estructuran la semántica de un recurso.

Por otro lado, la utilización de los metadatos en las ontologías es necesaria para poder definir qué partes de la ontología pueden ser compartidas o reutilizadas. Sin embargo, las ontologías van más allá pues son capaces de almacenar otro tipo de conocimiento: la semántica, a través de los conceptos y relaciones entre ellos. Las ontologías permiten realizar una representación declarativa del conocimiento de un dominio, que puede ser comunicado entre personas y máquinas.

La información que aparece en internet va a poder interpretarse por los ordenadores sin necesidad de intervención humana. Por ejemplo, dos programas tienen que saber que el campo "estado" de una base de datos y el campo "estado civil" de otra base de datos contienen la misma información. Para que esto ocurra, es necesario que la información de las páginas web, o de cualquier base de datos, se codifique mediante ontologías.

En resumen, se necesita unificar los contenidos semánticos por medio de ontologías que formalicen este conocimiento de forma consensuada (ontología negociada entre las distintas partes involucradas) y reutilizable (para poder compartir conocimiento). Es necesario un lenguaje común con suficiente capacidad expresiva y de razonamiento para representar la semántica de las ontologías.

Por último, hay que apuntar que este método puede ser aplicado tanto a pequeños dominios, el caso de este proyecto, como a grandes dominios, que será presumiblemente la web semántica, donde el dominio en el que se aplicará la ontología será toda internet.

1.2 DIVERSAS DEFINICIONES DE ONTOLOGÍA

En el sentido filosófico, ontología hace referencia a la esencia misma del ser, a su existencia (*onto*:ser). Para los sistemas de inteligencia artificial, lo que "existe" es lo que puede representarse. No obstante, la definición más extendida de ontología es la formulada por Gruber: "*Especificación explícita y formal sobre una conceptualización compartida*" [Gruber, 1993; extendida Studer, 1998].

Una ontología también puede ser entendida como una descripción formal de los conceptos y las relaciones entre conceptos. Como una especificación explícita y formal de una conceptualización, es decir, el modelo abstracto de un universo de discurso y que es compartido

por una comunidad de agentes (humanos o no humanos). Se dice que es **formal** porque debe ser entendible por todos y debe estar estandarizada; **explícita**, porque los conceptos, propiedades, relaciones, funciones, restricciones y axiomas son definidos explícitamente, y **compartida**, porque una ontología recoge conocimiento consensuado, no privado de un individuo sino aceptado por un grupo.

Así pues, se puede decir que una **ontología** proporciona estructura y contenidos de forma explícita y que codifica las reglas implícitas de una parte de la realidad. Proporciona una estructura y contenidos que son independientes del fin y del dominio de la aplicación en que se usarán o reutilizarán sus definiciones. Por consiguiente, las definiciones procedentes de las ontologías pueden ser utilizadas en diferentes dominios y en diferentes sistemas, cada uno con sus fines.

A continuación, con la intención de seguir formalizando el concepto de ontología, se recogen en la siguiente tabla algunas de las muchas definiciones que existen hoy en día sobre el concepto de ontología.

AUTOR/AÑO	DEFINICIÓN
Bruker, 1999	"*Es una representación explícita de una conceptualización cognitiva, es decir, la descripción de los componentes de conocimiento relevantes en el ámbito de la modelización*".
Neches, 1991	"*Define los términos básicos y relaciones para comprender el vocabulario de un área, así como, las reglas de combinación de los términos y relaciones para definir extensiones al vocabulario*".
B.Swartout, R.Patil y otros, 1997	"*Es una jerarquía de un conjunto estructurado de términos para describir un dominio que puede ser usado como principio, como un esqueleto firme para una base de conocimiento*".
McGraw	"*El conocimiento es un conjunto de descripciones, relaciones y procedimientos tales como: (i) descripciones simbólicas de conceptos, (ii) descripciones simbólicas de relaciones y (iii) procedimientos para manipular ambos tipos de descripciones*

Una ontología es una especificación de una conceptualización de un dominio particular, que debe ser representada de una manera formal, legible y utilizable por los ordenadores. Una conceptualización es una visión abstracta y simplificada del mundo real. Las ontologías son la base de un conocimiento que va a describir una serie de medios para especificar conocimiento.

Una ontología define una taxonomía de conceptos (una clasificación de conceptos y relaciones entre ellos y un conjunto de reglas de inferencia). En general, los modelos ontológicos definen la relación taxonomía como reflexiva, transitiva y asimétrica. En tanto que la transitiva permite inferir en la estructura, las restantes se prestan para comprobar consistencia.

Desde otro punto de vista, podemos ver una ontología como el resultado de seleccionar un dominio y aplicar sobre él un método formal para representar los conceptos y las relaciones existentes entre ellos en el dominio que estamos modelando. El propósito general de este formalismo es servir como medio para la distribución de conocimiento y reutilización. Esta especificación puede tomar la forma de un conjunto de definiciones de un vocabulario formal usado por un conjunto de agentes.

Este tipo de definición establece *qué* es una ontología; no *para qué* sirve. En otras palabras, una definición de ontología no espera aportar un medio para ejecutar tareas, como resolver un problema, y debería ser tan independiente como fuera posible.

Como ya se ha dicho anteriormente, las ontologías permiten representar el conocimiento. El conocimiento es información que ha sido categorizada, interpretada, aplicada y revisada, por tanto, es una simple codificación de los hechos que incluye la habilidad de utilizarlos en la interacción con el mundo para construir un modelo mental que represente en forma precisa el objeto y las acciones que con él mismo y sobre él se pueden efectuar.

Una ontología define el vocabulario de un dominio acotado mediante un conjunto de términos básicos y las relaciones entre dichos términos. Una ontología estaría englobada dentro de lo que se considera un lenguaje controlado, pero amplía su potencialidad puesto que tiene la capacidad de generar nuevos términos por medio de inferencias y otras formas de razonamiento gracias a las relaciones entre conceptos y al conjunto de axiomas.

La creación de una ontología reporta de inmediato la ventaja de que se hace explícita la categorización de los elementos y relaciones que intervienen en el modelo de conocimiento, de forma que, por un lado el modelo de conocimiento puede ser editado y gestionado, y por otro, es posible transmitirlo de manera que un sistema “entienda” la conceptualización que se ha utilizado en otro. Este hecho es fundamental a la hora de convertir el proceso de creación en una labor de ingeniería más que en una labor artesanal, además de proporcionar un conocimiento del

dominio reusable y mantenible. En cuanto a este último concepto de reusabilidad, una ontología es la definición de un vocabulario común con el que expresar el conocimiento, y su utilidad está en que permite a los sistemas hacer referencias a componentes de conocimiento de otros, siempre que ambos compartan la misma conceptualización. De esta manera, una ontología compartida sólo necesita describir un vocabulario común para hablar sobre un dominio [Mizoguchi, 1997].

La gran diferencia con el resto de bases de conocimiento es el propósito de la codificación de su conocimiento: las ontologías son diseñadas con suficiente abstracción y generalidad como para compartir y reutilizar el conocimiento [Guerrero y Lozano, 1999].

Una ontología contiene definiciones que nos proveen del vocabulario para referirse a un dominio [1]. Las definiciones dependen del lenguaje que usamos para describirlas. Algunas características típicas de las ontologías son:

- La existencia de ontologías múltiples: el objetivo de una ontología es hacer explícito algún punto de vista, para ello, a veces, se necesita combinar dos o más ontologías.
- La identificación de niveles de abstracción de las ontologías. Estos niveles de generalización dan una topología de ontologías. La idea es caracterizar una red de ontologías usando multiplicidad y abstracción.
- La multiplicidad de la representación: un concepto puede ser identificado de muchas formas, por lo que pueden existir múltiples representaciones de un mismo concepto.
- El mapeo de ontologías para establecer relaciones entre los elementos de una o más ontologías, así como, conexiones, especializaciones, generalizaciones, etcétera.

1.3 CLASIFICACIÓN DE ONTOLOGÍAS

Las ontologías se pueden clasificar atendiendo a varios criterios. A continuación se exponen los diferentes tipos de ontologías según los criterios más extendidos en la actualidad [2].

1.3.1 Según su **uso / usabilidad**: unas ontologías se utilizan para representar el conocimiento mientras que la meta de otras es describir tareas y resolver problemas.

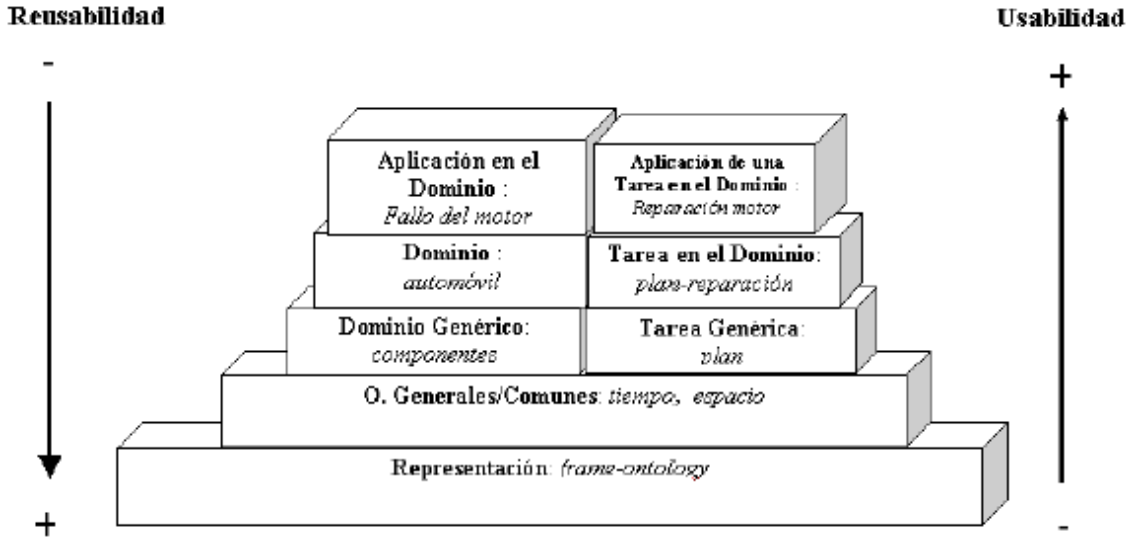


Figura 1.1

Se puede decir que cuanto más usable es una ontología en un contexto determinado, menos reusable será para otro contexto.

Las meta-ontologías, ontologías de dominio y ontologías de aplicación capturan el conocimiento estático del dominio independientemente del método de resolución de problemas que se pueda utilizar. Por otro lado las ontologías de PSMs (Problem Solving Methods), ontologías de tareas y ontologías de dominio-tarea están destinadas a conceptualizar y exponer conocimiento de resolución de problemas. A continuación se explican brevemente los diferentes tipos de ontologías según su uso:

- **Aplicación**: modelado de un dominio particular. No reutilizable. Contiene todas las definiciones que son necesarias para modelar el conocimiento requerido para una aplicación particular en un dominio dado. Típicamente son una mezcla de conceptos provenientes de ontologías de dominio y de ontologías genéricas. Las ontologías de aplicación no se construyen con el propósito de lograr usabilidad en diferentes dominios.
- **Dominio-Tarea**: reutilizable en un dominio dado enfocado a resolver problemas

- **Tareas**: vocabulario para resolver problemas genéricos. Proveen un sistemático vocabulario de los términos usados para resolver problemas asociados con tareas particulares, ya sean dependientes o no del dominio.
- **Dominio**: expresan conceptualizaciones que son específicas de un dominio particular, colocando restricciones en la estructura y contenido de ese dominio de conocimiento mediante axiomas que se cumplen siempre entre los elementos de dicho dominio. Su principal cometido es permitir el reuso de la ontología para diferentes aplicaciones que involucren al mismo dominio (reutilizable en un dominio dado). Los conceptos de las ontologías de dominio son a menudo definidos como especializaciones de conceptos existentes en ontologías genéricas.
- **Meta-Ontologías**: reutilizable a través de dominios. Son similares a las de dominio, pero los conceptos que definen se consideran genéricos a través de diferentes áreas de conocimiento y por ello reusables en diferentes dominios. Típicamente, las ontologías genéricas definen conceptos como estado, evento, proceso, acción, etc.
- **General / Común**: Incluyen vocabulario relacionado con cosas, eventos, tiempo, espacio, causalidad, comportamiento, función, etc.
- **Ontología de representación**: conceptualización de formalismos de representación de conocimiento. Formalización de primitivas de representación. Las ontologías genéricas y de dominio son descritas usando ontologías de representación.

1.3.2 Según la **cantidad y tipo de conceptualización**: esta categorización clasifica según la funcionalidad específica de cada ontología:

- **terminológicas**: especifican los términos que son usados para representar conocimiento en el universo del discurso. Se utilizan habitualmente para unificar vocabulario en un dominio determinado.
- **de información**: especifican la estructura de almacenamiento de las bases de datos. Ofrecen un marco para el almacenamiento estandarizado de información.
- **de modelado del conocimiento**: especifican conceptualizaciones del conocimiento. Contienen una rica estructura interna y suelen estar ajustadas al uso particular del conocimiento que describen.

1.3.3 Según el **nivel formal del lenguaje** en el cual está expresada. Las ontologías tratan de definir, organizar y estructurar el conocimiento de un dominio. Esta tarea se puede realizar a distintos niveles de formalidad (abstracción):

- **Altamente informal**: escrito en lenguaje natural.
- **Semi-Informal**: lenguaje natural restringido.
- **Semi-Formal**: lenguaje artificial definido formalmente.
- **Rigurosamente formal**: definido en un lenguaje con semántica formal.

1.3.4 La naturaleza del **tipo de relación** en una ontología puede ser también un criterio de clasificación. Las relaciones en una ontología son vínculos entre conceptos que son usados para construir una jerarquía de conceptos que clasifica desde alto nivel (conceptos generales) a bajo nivel (conceptos particulares). Se dividen en:

- **Disjuntas**: estas relaciones están limitadas a "es un subtipo de" (es subclase). En este tipo de ontologías, la herencia múltiple es usualmente permitida.
- **n-dimensionales**: la división en subconceptos puede también permitir crecer en más de una dimensión, en este caso las categorías resultantes no serán disjuntas, y estaremos hablando de una ontología con una gran dimensión.

1.3.5 El nivel de densidad puede variar entre ontologías, y los conceptos pueden estar, más o menos, vinculados a la tarea que están destinados a realizar. Por esta razón, algunas ontologías serán más dependientes del dominio que otras. Según este criterio, las ontologías pueden ser:

- **Ontologías taxonómicas**: aquellas construidas usando solamente una gran taxonomía.
- **Ontologías axiomáticas**: aquellas que recogen conjuntos de pequeñas taxonomías.

1.4 APLICACIONES

Uno de los fines más interesantes de una ontología es poder compartir conocimiento sin necesidad de intervención humana. La idea es que los datos puedan ser utilizados y "comprendidos" por los ordenadores sin la necesidad de supervisión de una persona, es decir, que las máquinas se entiendan entre sí.

Las ontologías proporcionan una comprensión compartida y consensuada del conocimiento de un dominio que puede ser comunicada entre personas y sistemas heterogéneos. Para poder reutilizar conocimientos de otros sistemas es necesario conocer y estar conforme con la terminología y con su significado. Desde este punto de vista, se llaman acuerdos ontológicos a los acuerdos terminológicos necesarios para reutilizar conocimiento. Se trata de convertir la información en conocimiento. Las ontologías representan el conocimiento de un dominio definiendo formalmente los conceptos y relaciones de dicho dominio, con la capacidad de realizar deducciones con ese conocimiento, es decir, con axiomas que podrán aplicarse en los diferentes dominios que trate el conocimiento almacenado para realizar inferencias: aprender a partir del conocimiento actual, de su base de conocimiento.

Por lo tanto, las ontologías definen conceptos y relaciones de algún dominio, de forma compartida y consensuada. A continuación se presentan algunos de los usos más frecuentes de las ontologías, aunque las amplias expectativas de la investigación en este campo hacen prever un aumento considerable de éstos en un futuro cercano.

Los sistemas expertos, por ejemplo, se han aplicado al entorno profesional puesto que éstas son técnicas que tratan de competir con los expertos humanos. Su aplicación se centra en actividades muy formalizadas en las que se suele necesitar bastante experiencia, la cual, sin embargo, resulta difícil adquirir principalmente porque no son actividades muy comunes (catalogación de mapas, música, etc).

Este amplio espectro de posibles tareas es la razón de la dificultad de encontrar un acuerdo general en sus contenidos y estructuras, además de la escasez de una amplia distribución de estándares y lenguajes para ontologías. En la literatura especializada se destacan los siguientes usos para una ontología:

Aplicaciones en ingeniería del conocimiento

- **Ingeniería del conocimiento**

  El aporte de las ontologías en el proceso de ingeniería del conocimiento se encuentra en las siguientes etapas:

- modelado conceptual: donde se crea un glosario de la terminología del dominio de la aplicación (los conceptos), se definen relaciones entre dichos términos y restricciones en su uso. Este modelo conceptual explícito es la ontología.
- construcción de la base de conocimiento: usando la ontología definida en la etapa anterior como un conjunto de esquemas o contenedores de conocimiento, se puebla la base de conocimiento con instancias del dominio bajo la forma de reglas, hechos, eventos y restricciones.

Las ontologías, conjuntamente con los métodos de solución de problemas (PSMs), son prometedores candidatos para posibilitar el reuso de componentes en ingeniería del conocimiento. En tanto que las ontologías definen el conocimiento declarativo del dominio a un nivel genérico, los PSMs especifican conocimiento de razonamiento sobre dicho dominio.

- **Procesamiento del lenguaje natural**

Una ontología puede mantener la definición de elementos gramaticales del lenguaje y sus relaciones, permitiendo, por ejemplo, el análisis sintáctico de un texto. Agentes software podrán entender la estructura de un idioma, serán capaces de entender a una persona a través de su voz, con todas las ventajas que esto aporta. Se podrán implementar mejores correctores ortográficos, traductores software de mayor calidad ya que la traducción se procesará a nivel conceptual y no terminológico, etc.

- **Distribución del conocimiento desde el punto de vista de la reutilización**

Uno de los fines principales de la utilización de una ontología en un sistema es poder reutilizar conocimiento para sistemas futuros. Se podrán integrar ontologías para la constitución de una nueva, más grande, que mejore la conceptualización que aportaban todas ellas por separado. Sistemas distintos podrán entender la información almacenada en una ontología sobre un dominio y trabajar con dicha información aunque no haya sido generada por y para ellos.

- **Distribución de conocimiento como un camino para resolver la integración de sistemas basados en conocimiento**

Integración inteligente de información. Como ya se ha señalado en puntos anteriores, la distribución de un conocimiento de forma estandarizada se traducirá en importantes

mejoras en el desarrollo de agentes inteligentes, los cuales tendrán a su disposición un mayor número de bases de conocimiento disponibles.

- **Implementación de agentes software inteligentes**

  Para ayudar a los estudiantes a encontrar y usar globalmente recursos de aprendizaje distribuidos en distintas máquinas y representados de forma entendible por agentes. Una posible meta de esta tecnología es poder disponer de un agente software para cada dominio, para cada tarea que tenga que realizar un humano facilitándole la obtención de resultados.

- **Creación y construcción de cursos de manera colaborativa (desarrollo e integración de sistemas de aprendizaje)**

  Para esto es necesario presentar el conocimiento de una materia de forma que se pueda referir a ello en términos pedagógicos, es decir, más cercano al nivel de uso de un profesor explicando. Esto puede llevarse a cabo mediante la creación de un dominio en el que se incluyen los elementos conceptuales y didácticos que conforman una materia de estudio desde el punto de vista del profesor, proporcionando relaciones entre los elementos que describan aspectos educativos, como por ejemplo saber que un problema *ejercita* un determinado concepto o que éste es *prerrequisito* para la comprensión de otros o bien saber que un problema es *fácil* o *difícil*, etc. Este mayor nivel de abstracción hace posible el diseño utilizando términos del tipo *exponer ejemplos de dificultad media que ilustren este concepto* o también *mostrar los prerrequisitos de este concepto junto con un ejemplo fácil de cada uno de ellos.*

  Un caso particular es la reutilización de material de aprendizaje por la *fertilización cruzada*, creando importantes efectos de sinergia [Nilsson, 2001] (reutilización y compartición de conocimiento).

Aplicaciones en sistemas de información

- **Interoperabilidad entre sistemas heterogéneos**

  Las ontologías se presentan como una importante solución para lograr una integración inteligente. En particular en el área de bases de datos, una ontología puede ser un elemento clave asociado a un mediador que integra datos provenientes de “wrappers” que recogen información de fuentes heterogéneas. Con una ontología terminológica, se

pueden organizar los términos que son usados en interacciones entre sistemas heterogéneos, de manera que reconozca cuándo una aplicación está usando un término que es más general o más específico que otro que está en uso por otra aplicación. Si la ontología es formal, se puede contar con una definición más completa sobre cómo se relaciona un término de un origen con el de otro, y eventualmente usar axiomas definidos que los vinculen por igualdad o que expresen un término exactamente en función del otro, lo que permite establecer correspondencias seguras y automáticas entre ellos.

- **Sistemas de información cooperativos**

  El objetivo es que múltiples sistemas de información sean capaces de trabajar de forma cooperativa combinando sus datos y funcionalidades. Para poder entenderse se ayudan de las ontologías.

- **Medio de distribución de conocimiento dentro aplicaciones software y entre aplicaciones software**

  Comunicación entre aplicaciones sin intervención humana, utilizando estándares y protocolos para el entendimiento recíproco.

- **Recuperación de información, mejoras en la formulación de consultas**

  Añadir semántica a las consultas y no hacerlas únicamente por palabras clave proporciona una calidad superior en los resultados de una búsqueda. Las consultas no serán tratadas desde el punto de vista terminológico sino conceptual. De este modo, se reducirá el número de resultados "basura" y no se omitirán aquellos resultados, que aún siendo conceptualmente sinónimos al de la consulta, actualmente los buscadores no son capaces de encontrar por ser distintos terminológicamente.

- **Normalización de sistemas documentales**

  El aumento de la masa de documentos no se ha traducido en una mayor información para el usuario, sino que generalmente lo que provoca es la recuperación de "basura" cuando se quiere acceder a cierta información. Gracias al uso de *thesaurus* se ha favorecido el crecimiento de las posibilidades documentales, tanto en su dimensión cuantitativa (más rapidez con menos impedimentos), como cualitativa (manuscritos,

voces, imágenes, etc). La documentación generada por los nuevos sistemas contará con las siguientes características: a) Identificación del contenido documental (información) mediante el uso de códigos de identificación y descriptores; b) Etiquetado de la documentación legible en las fuentes y en los instrumentos de representación (catálogos, listados, bases de datos, etc.); c) Establecimiento de una estructura base que permita la relación de los términos empleados en la identificación documental (thesaurus) y que debe alcanzar hasta las unidades mínimas de descripción; d) Creación de instrumentos auxiliares para la recuperación de la información: índices, tablas, etc.

Aplicaciones para la web semántica

- **Indización de documentos** (web semántica)

  El procedimiento para indexar un sitio web con apoyo de una ontología terminológica, en grandes pinceladas, es el siguiente:

  En una primera etapa se extraen los términos más relevantes de cada página con ayuda de las marcas de HTML, luego se asocian a dichos términos conceptos candidatos. A continuación, se evalúa la capacidad de representación de la página de cada uno de los conceptos candidatos, lo cual, determina su nivel de representatividad; por último, se construye el índice. A cada concepto de la ontología se le asocian las páginas del sitio web donde está contenido. De esta forma las consultas no serán procesadas a un nivel terminológico, sino conceptual, con los beneficios que eso supone en el grado de acierto de las respuestas generadas.

- **Agrupamiento o "clustering"**

  Las técnicas de clustering permiten el crecimiento de un sistema mediante la adición de procesadores o CPU a la unidad primitiva. La ontologías aportan las herramientas para que los distintos equipos puedan entenderse entre sí y funcionar como si fueran uno solo. [*véase glosario - anexo E*]

- **Servicios web**

  Web semántica: las ontologías representarán los datos en la red de tal forma que puedan ser utilizados y comprendidos por las máquinas sin necesidad de la intervención humana.

- **Comercio electrónico**

  Aplicaciones que faciliten el comercio electrónico. Un ejemplo de un sistema de este tipo es MKBEEM (Mercado electrónico europeo multilingüe basado en la gestión de conocimiento), un portal multilingüe de comercio electrónico que combina procesamiento basado en ontologías y procesamiento de lenguaje humano.

1.5 DESARROLLO DE ONTOLOGÍAS

Cada vez que un ingeniero de conocimiento comienza la construcción de un nuevo sistema, empieza la laboriosa tarea de adquirir los conocimientos de un dominio y su estructuración y organización en un conjunto de representaciones intermedias que darán lugar al modelo conceptual del sistema.

Con el fin de paliar el cuello de botella que supone la adquisición de conocimiento, y de disminuir los tiempos y recursos, humanos y materiales, empleados en la conceptualización del nuevo sistema, el ingeniero de conocimiento puede plantearse la reutilización de conocimientos procedentes de otros sistemas. La inspección y el estudio detallado de otras bases de conocimiento ya construidas permiten al ingeniero de conocimiento determinar si dichos conocimientos son útiles para su nuevo sistema. Sin embargo, se encuentra con numerosos problemas, de entre los que caben destacar:

- La heterogeneidad de los paradigmas de representación de los conocimientos.
- Cada sistema tiene expresado los conocimientos en un lenguaje o en una herramienta.
- Se presentan problemas semánticos. El sistema del cual se reutilizarán los conocimientos puede asignar un significado a un concepto que no coincide con el significado que dicho concepto tendrá en el nuevo sistema.
- Problemas de sinónimos.
- Cada sistema tiene suposiciones implícitas que hacen difícil su reutilización.

Para resolver alguno de estos problemas con los que se encuentran el ingeniero de conocimiento a la hora de reutilizar conocimiento, se están desarrollando ontologías en diferentes dominios. Existen también problemas relacionados directamente con la construcción de sistemas basados en conocimiento:

- La adquisición de los conocimientos, y cómo trasladar los conocimientos humanos, tal y como existen corrientemente en los textos y en las mentes de los expertos de un dominio, a una representación abstracta efectiva, denominada conceptualización.
- La representación de los conocimientos en términos de estructuras de información que una máquina pueda procesar.
- La generación de inferencias, lo que equivale a decir cómo hacer uso de esas estructuras abstractas de información para generar información útil en el contexto de un caso de uso específico.

Para dar solución a estos problemas simultáneamente, se deben satisfacer los dos requisitos siguientes:

- **Modular**. Durante el desarrollo de la base de conocimientos, es improbable que los expertos presenten todos los hechos y las relaciones necesarias para plasmar íntegramente la experiencia en el dominio. Los expertos humanos tienden a olvidar y a simplificar los detalles concernientes a sus conocimientos, de manera que los sistemas deben aumentar sus conocimientos paulatinamente. Debido a que los conocimientos impartidos al sistema son considerablemente empíricos, y dado que los conocimientos en los dominios se desarrollan rápidamente, es necesario que se puedan hacer cambios fácilmente y de una forma gradual y modular.
- **De programación**. El uso fundamental de un sistema computacional no es crear secuencias de instrucciones para realizar tareas, sino expresar y manipular las descripciones de procesos computacionales y de los objetos sobre los cuales se llevan a cabo. En este sentido, conviene que los sistemas sean declarativos y no imperativos.

Por otro lado, los expertos y usuarios del dominio deberían ser capaces de usar una base de conocimientos directamente, tanto durante la fase de adquisición de conocimientos como durante su explotación. Para ello, las ontologías deben proporcionar definiciones de conceptos en lenguaje natural y en lenguaje formal, las cuales pueden ser reutilizadas tanto por expertos como por ingenieros de conocimiento. De este modo, el sistema debe soportar lenguajes de alto nivel cuyos elementos primitivos son los atributos y las asociaciones de un problema particular en el dominio, en lugar de conceptos de programación.

Este enfoque contrasta con el enfoque tradicional en el cual el ingeniero de conocimiento hace de intermediario entre la base de conocimientos y el experto, pues, hasta

ahora, trasladar los conocimientos del cerebro de los expertos a un sistema computacional es casi siempre un trabajo lento y de colaboración entre ambos. De donde se desprende la necesidad de métodos más automáticos para transferir y transformar los conocimientos dentro de su representación computacional.

Adquisición de conocimientos

La adquisición de conocimientos es el proceso de recolección de información, a partir de cualquier fuente, necesaria para construir un sistema basado en conocimiento. La adquisición de conocimientos en la actualidad es una labor de artesanía, hecha a medida para cada caso, de modo que pueda ajustarse a las características de cada situación particular. Así pues, durante la adquisición de conocimiento, el ingeniero sólo está, prácticamente, equipado con una grabadora, un lápiz y un papel.

Desde el punto de vista de un sistema basado en el conocimiento, existen distintas fuentes de adquisición del mismo: libros y manuales, documentación formal, documentación informal, presentaciones, publicaciones especializadas, investigación, visitas, expertos, etc. Para cada fuente se dispondrán de una serie de técnicas de educción que permitirán extraer el conocimiento útil.

Diseño de una ontología

En la construcción de cualquier sistema, se deben tomar una serie de decisiones de diseño, inducidas por los resultados del análisis de las características que se desean en el sistema. A continuación se presentan algunas recomendaciones existentes acerca de las características que debería tener una ontología. Es muy útil aplicarlo en la etapa de validación y verificación de un proyecto software destinado a la construcción de una ontología.

- Sobre los términos
  - **Claridad y objetividad**: una ontología debe proveer al usuario con el significado del término de forma objetiva. Todas las definiciones deben además estar documentadas en lenguaje natural.
  - **Completitud/integridad**: las expresiones han de estar expresadas en términos necesarios y suficientes.
  - **Máxima extensibilidad monótona**: una ontología debe soportar la definición de nuevos términos basándose en el vocabulario existente, de manera, que no requiera

la revisión de las definiciones existentes. Esto se logra manteniendo cierto equilibro entre ser lo suficientemente específicos en la definición, como para permitir el uso para el que la ontología se construye, pero no demasiado como para que la ontología sea de poca utilidad para otros usos futuros. Debe anticipar usos futuros y permitir extensiones.

- Sobre la organización
  - **Principio de distinción ontológica**: las clases en una ontología deben ser disjuntas, con el objetivo de obtener una mejor base estructural que posibilite y favorezca su futura reutilización. Un elemento no puede ser clasificado en múltiples categorías. Por ejemplo, una *asociación* puede ser vista como un *grupo* de personas, pero dos asociaciones diferentes pueden involucrar al mismo grupo de personas.
  - **Diversificación de jerarquías**: permiten mecanismos de herencia múltiple. Esto se logra representando suficiente conocimiento en la ontología y usando tantos criterios de clasificación como sea posible para que agregar un término sea sencillo en la medida en que puede ser fácilmente especificado desde los conceptos preexistentes y los criterios de clasificación.
  - **Modularidad**, que disminuya el acoplamiento entre módulos. Se persigue el ideal de alta cohesión y bajo acoplamiento.

- Sobre la conceptualización
  - **Coherencia** para asegurar que las inferencias realizadas de ésta, sean consistentes con las definiciones que contiene la propia ontología. Por lo tanto, sus axiomas deberán ser lógicamente consistentes.
  - **Minimización de la distancia semántica** entre conceptos emparentados. Conceptos similares estarán agrupados y representados utilizando las mismas primitivas.
  - **Sesgo de codificación mínimo** (*Minimal encoding bias*): especifica el nivel de conocimiento; debe minimizarse sin depender de una codificación particular a nivel de símbolo.
  - **Mínimo compromiso ontológico**: una ontología debe hacer la menor cantidad posible de afirmaciones acerca del mundo que está siendo modelado, permitiendo de esta forma que las partes que están comprometidas con la ontología (los

diferentes agentes que la usarán) tengan libertad de especializar e instanciar la ontología cuando sea necesario.

Construcción de una ontología

Actualmente, no existe un acuerdo entre la comunidad involucrada que permita la formulación de una metodología estándar para la construcción de ontologías. Cada grupo de desarrollo usa su propio conjunto de principios, criterios de diseño y etapas en el proceso de construcción de una ontología. En palabras de Jones (1998), "en la actualidad la construcción de ontologías es mucho más un arte que una ciencia".

Es necesario un conocimiento profundo para poder desarrollar una metodología viable para la construcción y mantenimiento de una ontología. El estándar IEEE 1074-1995 describe un camino para desarrollar un proceso de construcción de software, sin especificar los ciclos de vida concretos o el orden temporal de los diferentes pasos que lo componen. Aunque es un estándar muy abierto, la mayoría de sus puntos pueden ser considerados en un desarrollo de software basado en ontologías. Aparte del IEEE existen centenares de sugerencias de desarrollo de ontologías realizadas por varios autores. La mayoría de ellas tiene un fuerte acoplamiento con el campo de ingeniería del conocimiento y de este modo hereda muchos de sus rasgos. En la mayor parte de los casos, se recomienda un proceso de formalización para pasar del diseño conceptual a una implementación concreta de un conocimiento ontológico.

Fases en el desarrollo de una ontología

Prácticamente todas estas metodologías centran el ciclo de vida en un prototipo evolutivo. Las fases más habituales en una metodología de desarrollo de sistemas basados en conocimiento son las siguientes:

1. **Especificación**: especificar el ámbito de aplicación (contexto) y el punto de vista del modelado. El contexto de aplicación describe el dominio de aplicación, los objetos de interés del dominio, y las tareas que van a realizarse por la ontología, es decir, para qué va a construirse. Es la tarea de identificar el propósito, alcance y granularidad de la ontología, y tiene como salida un documento en lenguaje natural.
2. **Adquisición de conocimiento**: se ejecuta habitualmente en paralelo a la especificación, y las técnicas más frecuentemente utilizadas para extraer información son *(i)* entrevistas con expertos y *(ii)* análisis del *corpus*, aunque existen otras muchas.

3. **Conceptualización**: identificación de conceptos, verbos, instancias o propiedades usando una técnica de representación informal.
4. **Integración**: especificar cómo se van a integrar las ontologías existentes; especificar cuáles se van a integrar; y por último, si se va a usar una ontología existente, reutilizando así conocimiento. Por lo tanto, sólo es ejecutado cuando se incluyen definiciones de otras ontologías.
5. **Implementación**: la ontología es formalmente representada en un lenguaje, como por ejemplo *Ontolingua*.
6. **Evaluación**: se emplean técnicas de validación y verificación para buscar redundancias, inconsistencias y para comprobar si es completa. Debe considerarse la posible reutilización de la ontología para usos futuros.
7. **Documentación y Reutilización**: comparación de todos los documentos obtenidos de las otras actividades. Se hará de forma paralela a los puntos anteriores. Debe cumplirse la metodología usada para la construcción, las diferencias semánticas con las ontologías seleccionadas, justificación de las decisiones tomadas, evaluación, conocimiento adicional para usarla, etc. También ha de ser indexada y colocada con las ontologías existentes para su posible reutilización.

Otros esfuerzos dirigidos a la creación de una metodología válida para el desarrollo de ontologías han adoptado un punto de vista mas general, renunciando a establecer un conjunto de pasos predefinidos. En este camino, la metodología es vista como un conjunto de pasos útiles a seguir cuando creamos un sistema basado en ontologías. Ésta es la aproximación de la metodología IDEF5 la cual sugiere los siguientes pasos guía:

- **Organización y propósito**: establecimiento de un criterio de cumplimiento, objeto o propósito y punto de vista de la ontología.
- **Colección de datos**: uso de técnicas típicas de adquisición de conocimiento para recolectar datos nuevos.
- **Análisis de datos**: uso de los nuevos datos para listar los objetos de interés en el dominio y sus límites.
- **Desarrollo inicial de la ontología**: desarrollo de una ontología preliminar con protoconceptos, descripciones iniciales de tipos, relaciones y propiedades.

- **Refinamiento de la ontología y validación**: refinamiento iterativo y prueba de los protoconceptos. Seguimiento de un proceso deductivo de instanciación de ontologías y comparación del resultado con la estructura de la ontología.

Hay muchas otras metodologías como: Metodología de Gruninger y Fox, Metodología de Unschold y King, Methontology, etc.

Lenguajes de especificación e implementación

En un lenguaje, en general, se busca expresividad y uso. Las ontologías son teorías formales acerca de un dominio de discurso y por eso requieren de un lenguaje lógico formal para ser expresadas. En inteligencia artificial se han desarrollado muchos lenguajes para este fin. Algunos basados en lógica de predicados de primer orden, como *KIF* y *Cycl* que proporcionan poderosas primitivas de modelado y la posibilidad de construir fórmulas que les permite convertirse en términos de otras fórmulas.

Otros lenguajes están basados en marcos, con más poder expresivo pero menos capacidad de inferencia, como *Ontolingua* y *Frame Logic*. Generalmente proveen un conjunto bastante rico de primitivas, pero imponen restricciones sintácticas muy fuertes a la hora de combinar esas primitivas y en el modo en que se utilizan para definir una clase. Finalmente, existen otros orientados a ser robustos en el razonamiento como *Loom* y *Classic*. Éstos proveen DL (Description Logics) que aportan un conjunto más restringido de primitivas (se restringe su cantidad para lograr claridad semántica, capacidad de decisión y la posibilidad de proveer más procedimientos de razonamiento) pero permite que sus primitivas sean combinadas en expresiones booleanas arbitrariamente y usadas para definir diferentes tipos de clases.

Herramientas de anotación

Las herramientas de anotación permiten estructurar la información que ahora está publicada mediante su clasificación sobre la base de conceptos semánticos, y añadir información adicional, estructurada o no, a los contenidos actuales de la web y a los de nueva creación. Se consideran herramientas de anotación:

- **Editores de ontologías**, facilitan la tarea de su definición, unión de diferentes ontologías y su desarrollo distribuido.

- **Servidores de anotaciones**, cuya finalidad es mantener los repositorios de conocimiento generado a través de las anotaciones.
- **Herramientas de anotación externa**, permiten asociar metainformación a páginas web que ya están presentes en la web. La metainformación no se almacena dentro de la misma página, sino de forma externa en un repositorio destinado específicamente a mantener las anotaciones. Estos repositorios suelen ser bases de datos RDF (que actualmente se implementan sobre sistemas gestores de base de datos relacionales). Esta metainformación también puede estar basada en una ontología. Dentro de esta categoría encontraremos tanto editores como servidores de anotaciones. Las herramientas de anotación externa harán posible la anotación de las páginas web visitadas por los miembros de una comunidad de usuarios. La metainformación relacionada con la página web (introducida a través de las anotaciones) se almacenará en un repositorio común, al cual podrán acceder todos los miembros de la comunidad.
- **Herramientas del propietario**, partiendo de una ontología definida previamente, permiten incluir la información estructurada, la metainformación, en la misma página web. Esta información se incluye normalmente en alguno de los nuevos lenguajes de marcado promovidos por el W3C (World Wide Web Consortium), como son XML, o RDF, este último es un lenguaje definido específicamente para representar metainformación (RDF, se suele implementar a través de XML). Para la definición de las ontologías se suele usar DAML, aunque actualmente el W3C (www.w3.org), está trabajando en la definición de un lenguaje estándar para definir ontologías, OWL (Ontology Working Group). Las herramientas de autor facilitarán la estructuración de la información publicada en la web, ya sea en el ámbito de intranet, extranet o internet, haciendo posible la inclusión de metainformación en los documentos publicados.

Herramientas para construir y mantener ontologías

A continuación se enumeran algunos de los editores de ontologías de uso más extendido, así como herramientas existentes para construir y mantener ontologías:

- ***Ontolingua*** es una herramienta basada en la web. Provee de un entorno de colaboración distribuido para navegar, crear, editar, modificar y utilizar ontologías [3].

- ***WebOnto***, también una herramienta basada en la web, pero completamente gráfica [4].

- ***ProtégéWin***, una herramienta también gráfica, basada en Windows [5].

- ***OntoSaurus***, basada en la web, similar a Ontolingua, pero usa representación en Loom [6].

- ***ODE***, una herramienta basada en Windows, con aspectos básicos y textuales [7].

- ***KADS22***, también gráfica y textual, para construir ontologías y estrategias de razonamiento [8].

- ***Apollo*** es una aplicación amigable de modelado de conocimiento [9]. El modelado está basado en torno a los principios básicos tales como clases, instancias, funciones, relaciones, etc. La interfaz de usuario tiene una arquitectura abierta y está escrita en lenguaje de programación Java.

- ***LinkFactory*** es una herramienta utilizada para construir completos sistemas de terminología corporativa capaz de extraer valor significativo de gran cantidad de datos no estructurados almacenados en bases de datos de contenido corporativo [10].

- ***OILEd*** es un editor de ontologías que permite al usuario construir ontologías utilizando DAML+OIL [11].

- ***OntoEdit*** *Free and Professional versions* permite crear y gestionar ontologías. Confía en los estándares del W3C y ofrece muchas interfaces exportables a la mayor parte de lenguajes de representación de ontologías. Esta herramienta permite crear, navegar y modificar ontologías [12].

- ***OpenKnoME*** es la piedra angular de la aplicación utilizada por los motores de conocimiento *topThing*. Es un sistema de gestión del conocimiento y un motor de ontologías. Desde el 2001 el código fuente está abierto para la comunidad académica y la clínica sin ánimo de lucro [13].

- ***Protégé-2000*** es un editor de ontologías y editor de bases de conocimiento. Es también de código abierto; herramienta Java que proporciona una arquitectura extensible para la creación de aplicaciones de bases de conocimiento customizadas [14].

- ***SymOntoX*** es un software que almacena y gestiona un dominio de ontología [15].

Otros mecanismos de descripción, representación e implementación de ontologías

A continuación se exponen algunos de los lenguajes utilizados para definir de manera estándar ontologías en diversos entornos (especialmente en la web) [16]:

- El **XML** (Extensible Markup Language) es un lenguaje basado en demarcación que proporciona un formato para describir datos de manera estructurada e independiente de aplicaciones o proveedores. Es un subconjunto de SGML (Standard Generalized Markup Language), de manera que todo documento XML está escrito en SGML también. Se le considera como extensible porque en XML se pueden definir etiquetas que demarcan por su nombre la semántica de los datos que encapsulan; de esta manera, conociendo las etiquetas usadas, cualquier aplicación podrá entender el contenido de un documento XML.

  Mediante una **DTD** (Document Type Definition) que es una descripción de documentos, es posible validar la estructura de un documento XML. Contiene la lista de todos los elementos, atributos, notaciones y entidades que se pueden usar en el tipo de documento al que se refiere la DTD.

- Un **XML Schema** es un medio de definir restricciones de la sintaxis y la estructura de documentos XML y tiene el mismo propósito que una DTD, pero significativas ventajas:

  - Las definiciones realizadas en un XML Schema son en sí mismas documentos XML, por lo que no es necesario un segundo lenguaje como se debe usar en las DTD. Además, todo lo desarrollado para documentos XML puede usarse para documentos de tipo XML Schema.
  - Proveen un conjunto de tipos de datos mucho mas rico que el que puede ser definido actualmente en una DTD.

- Permiten definir anidamientos en la estructura de una manera mucho más rica que las DTD.
- Usan el mecanismo de espacios de nombres de XML para combinar documentos XML provenientes de orígenes heterogéneos.

Los lenguajes ontológicos se destinan a especificar teorías de dominio, y los XML Schema son una forma de proporcionar restricciones de integridad para orígenes de información.

- **RDF**: XML aporta información semántica como un efecto colateral de describir la estructura del documento, ya que define una estructura en árbol para un documento de manera que las hojas del mismo contienen la información. Se puede observar entonces que la estructura y la semántica de un documento XML están entrelazadas. El RDF (Resource Description Framework), proporciona un medio de agregar semántica a un documento sin referirse a su estructura. RDF es una aplicación XML recomendada como estándar por la W3C. El modelo de datos de RDF provee tres tipos de objetos: recursos, propiedades y sentencias. La diferencia fundamental entre XML y RDF es que el primero es un lenguaje para modelar datos, mientras que RDF es un lenguaje para modelar metadatos. Todo lo expresable en RDF lo es en sintaxis lineal de XML, sin embargo, RDF aporta un modo estándar de representar metadatos en XML. Usando directamente XML para representar metadatos, podrían obtenerse varias representaciones diferentes.

- **RDF Schema** (RDFS) fue definido sobre el lenguaje RDF para ofrecer un vocabulario particular para modelar clases y jerarquías de propiedades y otras primitivas básicas que pudiesen ser referenciadas desde modelos RDF. El rol de RDFS es definir una ontología simple, que documentos RDF particulares puedan chequear, para decidir su consistencia.

Otra posibilidad de diseño es confeccionar un modelo de conocimiento, haciendo una analogía con un modelo de datos del tipo entidad-relación, con sus jerarquías, clases, subclases e instancias, pero este modelo presenta algunas limitaciones:

- El modelo de datos toma un solo punto de vista del mundo. Describe los objetos o instancias de interés, pero bajo una sola posible interpretación. Si uno quiere reutilizar algún término, se hace evidente que el término puede tener diferentes interpretaciones

dependiendo del dominio. La reutilización de conocimiento complejo es imposible sin tener en cuenta los diferentes puntos de vista.

- Por otro lado, existen desarrollos en modelo de datos orientados a objetos, que sin embargo, siguen siendo pobres en su representación de relaciones entre objetos.

Ante estas limitaciones, una posible solución es hacer accesible la semántica de la información almacenada: qué contiene, qué propiedades tiene y cómo puede usarse. Si algún agente entiende la ontología, puede usar la información.

Así como existe una frontera difusa entre conocimiento e información, existe una frontera difusa entre ontologías y modelos de datos. Finalmente, una ontología se puede ver como un modelo de datos de conocimiento. Una ontología especifica una conceptualización, una forma de ver al mundo. Por lo que cada ontología incorpora un punto de vista.

Klein encuentra que la relación existente entre una ontología y un XML Schema es equivalente a la existente entre el modelo entidad relación extendido y el esquema relacional de una base de datos [23]. El modelo relacional proporciona una descripción de las bases de datos orientada a la implementación, en tanto el modelo entidad relación provee un marco para modelar orígenes de información requeridos para una aplicación.

Puede resumirse entonces que expresar una ontología en XML Schema es posible, pero su definición debería ser previamente realizada en un lenguaje ontológico y luego trasladada a XML Schema.

1.6 ALGUNOS SISTEMAS ACTUALES

Después de haber enumerado y analizado algunos de los campos en los que se pueden aplicar ontologías, a continuación, se muestran ejemplos específicos de sistemas reales que utilizan una ontología para su funcionamiento. Cabe resaltar que el uso de ontologías, en el ámbito de internet y de sistemas de recuperación de información, está aumentando continuamente, siendo muy rápida, por tanto, la evolución de su estado del arte. Sirvan estos pocos ejemplos para componer una fotografía del panorama actual.

- **MKBEEM** (*Multilingual Knowledge Based European e-Market* - Mercado electrónico europeo multilingüe basado en la gestión de conocimiento)

Se trata de un portal multilingüe de comercio electrónico que combina procesamiento basado en ontologías y procesamiento de lenguaje natural con el fin de proporcionar servicios multilingües de mediación para el comercio electrónico [17]. La combinación del procesamiento del lenguaje natural con ontologías desempeña un papel decisivo en:

- La creación de una correspondencia entre una consulta en lenguaje humano y su representación en términos ontológicos.
- La producción de servicios combinados a partir de productos o servicios procedentes de los catálogos de diferentes proveedores de contenido.
- La creación de una correspondencia entre consultas ontológicas, que proporcionan una visión abstracta de la información existente, y sus representaciones ontológicas de los catálogos. Los catálogos recogen los servicios ofrecidos, que deben tener implementada una consulta ontológica asociada.
- La descripción del dominio de conocimiento. Las ontologías proporcionan una representación consensual de dominios permitiendo los intercambios independientemente del lenguaje final, el servicio o el proveedor de contenidos.

Las ontologías se usan para la clasificación e indexación de catálogos, para el filtrado de las consultas del usuario, para facilitar los diálogos multilingües hombre-máquina entre el usuario y el agente software, y para inferir información relevante que satisfagan las peticiones del usuario. Concretamente MKBEEM introduce las ventajas de una concepción multilingüe en todos los estados del ciclo de la información (generación y mantenimiento de contenidos multilingües, traducción automática, e interpretación y mejora de la interactividad natural y de la usabilidad del servicio con introducción de lenguaje natural no sujeto a restricciones).

Éste es un ejemplo típico del uso de ontologías en el ámbito del comercio electrónico basado en lenguaje natural. Para una mayor claridad, a continuación se muestra en dos figuras (figura 1.2, figura 1.3), la arquitectura del sistema y el interfaz de usuario.

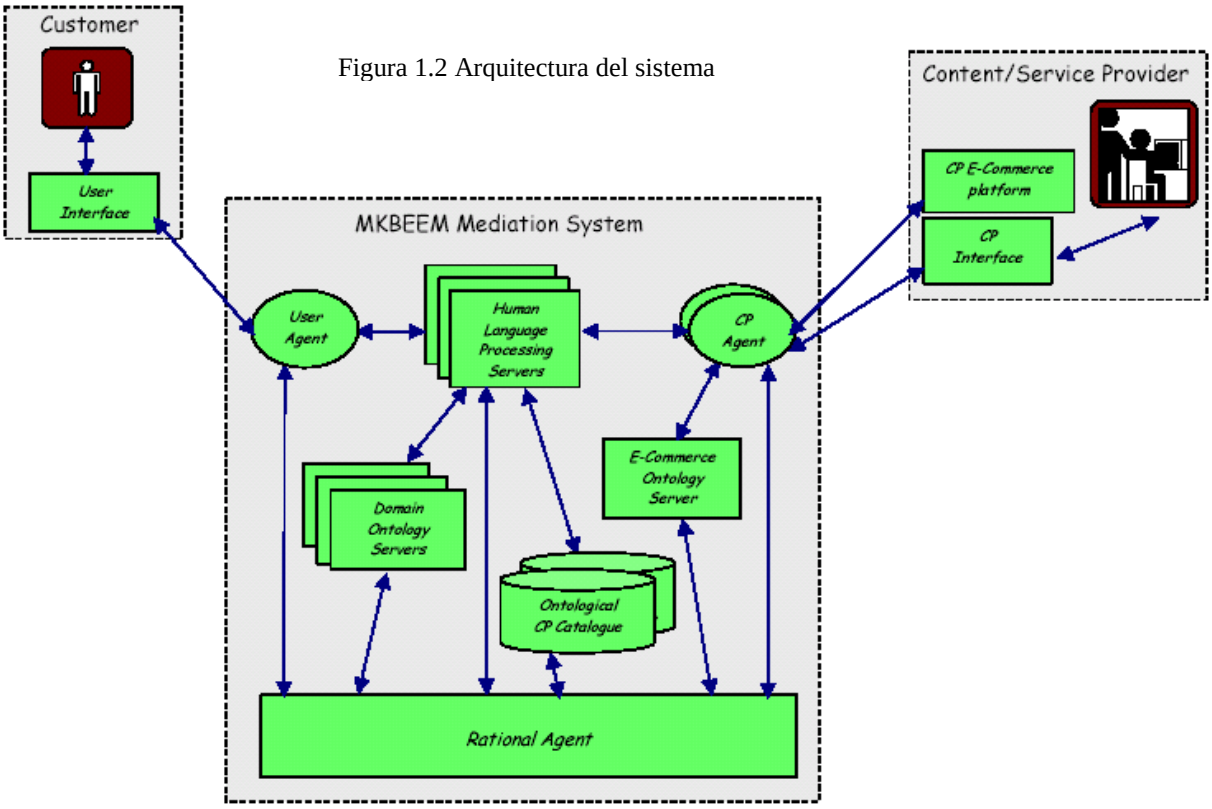


Figura 1.2 Arquitectura del sistema

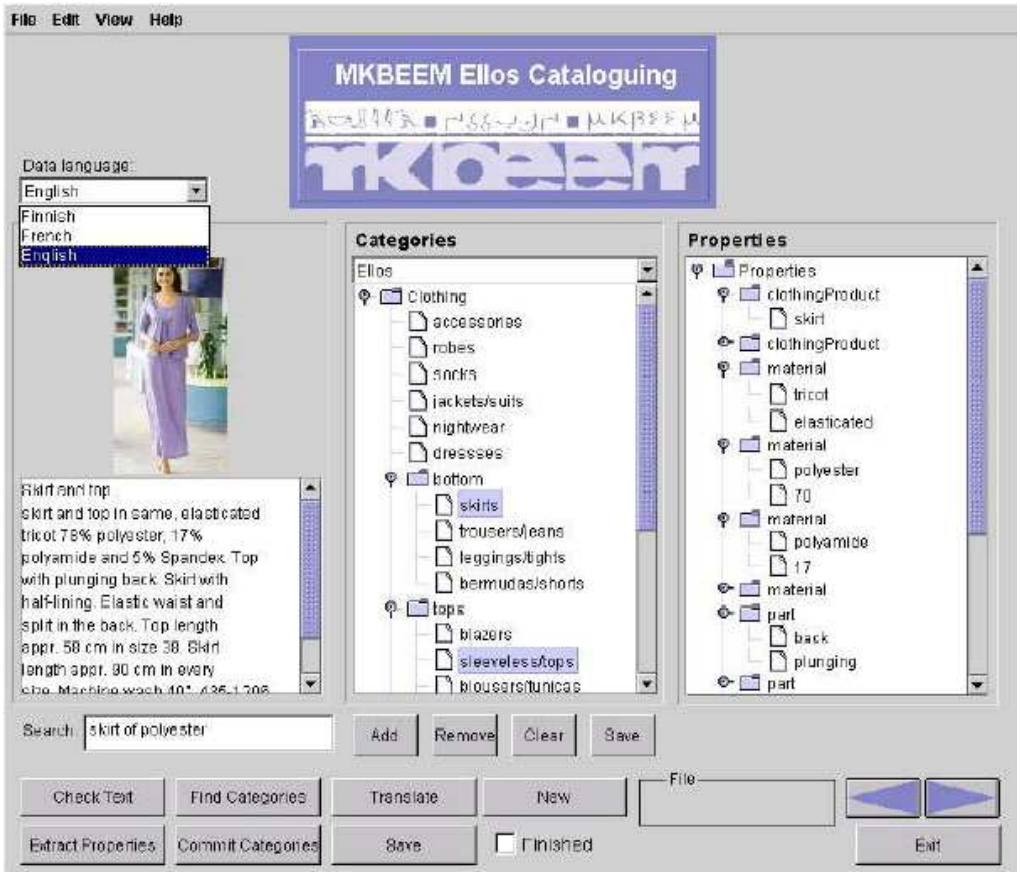


Figura 1.3 Interfaz de usuario MKBEEM

- **MESIA** (Modelo Computacional para Extracción Selectiva de Información de textos cortos)

Se trata de un prototipo cuyo objetivo es la recuperación de documentos de la web de la Comunidad Autónoma de Madrid utilizando recursos lingüísticos, para mejorar los resultados que ofrecen los buscadores tradicionales [18]. Para conseguirlo, se añaden a la consulta características semánticas y estructurales obtenidas a partir del análisis del contenido de las páginas por el sistema que integra recursos disponibles para el tratamiento automático del castellano (Aries y EWN-español).

Básicamente, el sistema transforma una consulta en lenguaje natural en una consulta formal, extendiendo los términos significativos de la consulta formal utilizando conocimiento lingüístico (mediante variaciones morfológicas y términos sinónimos o semánticamente relacionados). A continuación, con la información obtenida tras efectuar la consulta, un módulo "wrapper" se encarga de extraer la información textual que será tratada posteriormente por el módulo analizador de cadenas significativas. Este análisis produce una estructura de rasgos semánticos que describen superficialmente el texto de una página. Las estructuras de rasgos obtenidas para las páginas resultado de una búsqueda se envían tanto a un gestor de conocimiento extraído (se encarga de gestionar la ontología) para almacenarlas para futuras consultas, como a un presentador de resultados para organizarlas y visualizarlas al usuario.

La consulta introducida por el usuario se almacena junto con la consulta generada por MESIA, así como la información y las estadísticas sobre las consultas efectuadas. Además, este sistema también puede recibir directamente una consulta formal para buscar la respuesta en la base de datos de Documentos Clasificados junto con la ontología del dominio sin necesidad de hacer la búsqueda en web.

- **SCASEM** (Sistema de catalogación semántica)

Es un proyecto del Grupo Planeta para adaptar al nuevo mercado la gestión de sus fondos (enciclopédicos, iconográficos, cartográficos y audiovisuales), que constituyen el Banco de Contenidos Planeta (BCP), junto con la Base de Datos de Conocimiento (BDCon) y un conjunto de herramientas lingüísticas [19]. El BCP está organizado y codificado mediante el uso de sistemas de gestión de contenidos basados en SGML y XML, sistemas de catalogación automática y de búsqueda por lenguaje natural.

Los objetivos que persigue, principalmente, el BCP son:

- La clasificación de forma unificada de todo el material documental.
- La reutilización de material tanto documental como de conocimientos.
- Evitar que el material se pierda o se vuelva inutilizable.
- Evitar la necesidad de reclasificar material a medida que vaya creciendo el Banco de Contenidos.

La base de conocimiento (BDCon) contiene una ontología multidimensional y constituye la herramienta central del sistema de clasificación del banco (el núcleo del sistema). Esta ontología está organizada alrededor de una columna vertebral compuesta por un extenso lemario conectado con un diccionario de raíces, y una serie de estructuras de conocimiento. Cada elemento del banco de conocimiento (BCP) está identificado de forma unívoca utilizando distintos parámetros:

- El tipo de entidad del que se trata, organizado en un árbol tipológico (ATIP).
- El área temática a la que pertenece la entidad, organizado en un árbol temático (ATEM).
- El tipo de entidad desde el punto de vista del continente, no del contenido (tipo de palabra o tipo de documento) tomando en cuenta el soporte en que se encuentra. Organizado en un árbol de tipos de soporte (ASOP).
- El lugar al que se puede asociar la entidad, organizado en una estructura geográfica (EGEO).
- El período temporal o fecha al que se puede asociar la entidad, organizado en una estructura cronológica (ECRON).

La función del BDCon es clasificar los contenidos del BCP para que los usuarios encuentren con exactitud y precisión el material buscado. También sirve de apoyo semántico para las herramientas de tratamiento de texto. Además, para tener un lugar donde relacionar las entidades entre sí, se ha creado una estructura llamada RelCon, de Relaciones de Conocimiento donde se relacionan, en principio, objetos concretos como personas, obras, lugares, conceptos, etc.

Por último, el tercer elemento del BCP lo constituyen un conjunto de herramientas lingüísticas dedicadas al tratamiento automático o semiautomático de los contenidos textuales del Banco de Contenidos. Herramientas tales como recuperación "inteligente" de información, catalogación automática de documentos, creación automática de hipertexto, comprobación y corrección automática de textos, control automático de contenidos desde el punto de vista de las actualizaciones, codificación automática de textos, creación automática de resúmenes, extracción de información y conocimiento de los contenidos del BCP, creación de un sistema de búsqueda en lenguaje natural para el usuario final, etc.

- **PEGASUS** (Presentation modeling Environment for Generic Adaptive hypermedia SUpport Systems)

Es un sistema de presentación dinámica en entornos web para representaciones personalizadas del conocimiento [20]. Se trata de un sistema genérico de presentación para sistemas hipermedia de enseñanza adaptativa altamente independiente de la representación del conocimiento del dominio y del mantenimiento del estado de la aplicación. La generalidad se consigue proporcionando un marco de aplicación para la definición de ontologías que mejor se ajustan a un dominio o un autor concreto. La presentación de las páginas se describe en términos de clases y relaciones de la ontología. Se utiliza un modelo explícito de la presentación, separado de los contenidos del curso, para permitir a los diseñadores un extenso control sobre la generación de todos los aspectos de la presentación, con un coste de desarrollo moderado.

En PEGASUS, las ontologías se utilizan para proporcionar la máxima flexibilidad en la representación del conocimiento pedagógico. Son, además, un elemento esencial para conseguir la separación entre presentación y contenidos. La ontología del dominio en PEGASUS consiste en un conjunto de las clases que mejor se adecuan a un campo de aplicación determinado, o que reflejen la visión particular de un autor sobre el dominio. Las ontologías incluyen elementos para representar información sobre la materia (un teorema tiene un enunciado y una demostración), información pedagógica (las lecciones tienen niveles de dificultad), e información sobre el estado del usuario y del entorno en tiempo ejecución (un concepto es conocido o no por el estudiante). Todo este conocimiento se recoge mediante la definición de atributos para las clases, y relaciones entre clases.

- **Generación Automática de una Base de Datos desde Documentos de la Web**

El propósito de este sistema es la extracción de información de documentos HTML y la construcción de una base de datos a partir de dicha información [21]. Para alcanzar este fin se utiliza una ontología de dominio, capaz de interpretar el contenido de un fichero HTML para almacenarlo en la base de datos.

El sistema reconoce la información, contenido relevante de una página web, la clasifica, infiere los conceptos de los datos que son encontrados y reconocidos dentro de las restricciones definidas para la extracción de la información, y por último, consolida la información que se encuentra en forma de tablas y sus títulos encontrada en las páginas web analizadas (la almacena en la base de datos). Para profundizar más y poder entender mejor su funcionamiento, a continuación se ofrece la arquitectura del sistema (figura 1.4).

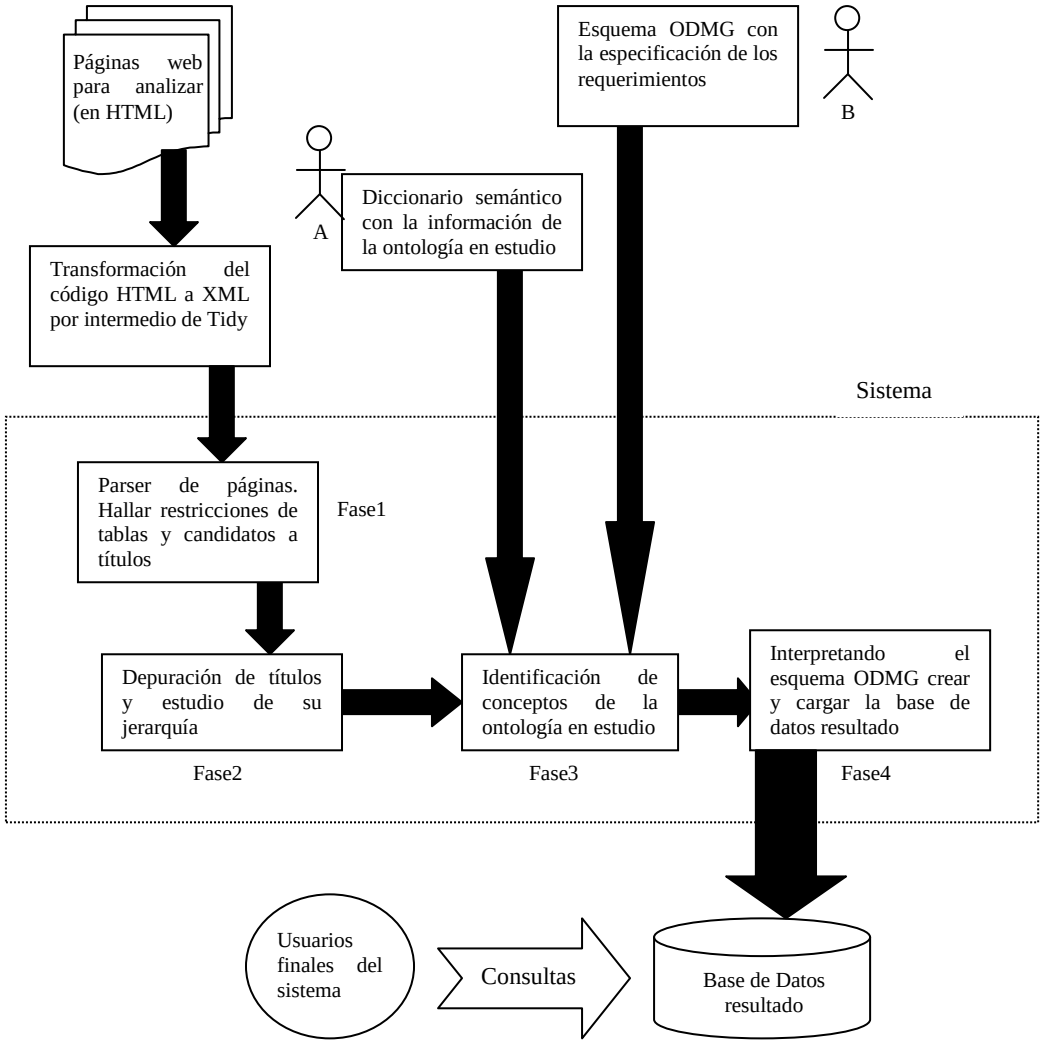


Figura 1.4 Arquitectura del sistema

- **KeyConcept**: Motor de búsqueda conceptual

Es un motor de búsqueda que recupera documentos usando una combinación de conceptos y palabras clave [22]. Los documentos se clasifican automáticamente asociados. Los conceptos relacionados con la consulta son introducidos manualmente por el usuario, o determinados automáticamente mediante una pequeña descripción textual de la consulta.

Para su funcionamiento, KeyConcept utiliza una ontología para estructurar la información. Durante el proceso de indización, se incluye en el índice información acerca de los conceptos con los que se relaciona cada documento. El proceso de recuperación utiliza dicho índice y soporta consultas que empleen sólo palabras claves, sólo conceptos, o una combinación de los dos. El usuario puede seleccionar la importancia relativa de cada criterio (cruce por palabras o cruce por conceptos) a través del factor α incluido en la fórmula:

$$\text{Puntaje documento} = (\alpha \times \text{puntaje conceptual}) + ((1-\alpha) \times \text{puntaje palabras clave})$$

Del mismo modo que en el caso anterior, a continuación, con el fin de profundizar más y poder entender mejor su funcionamiento, se muestra la arquitectura del sistema (figura 1.5).

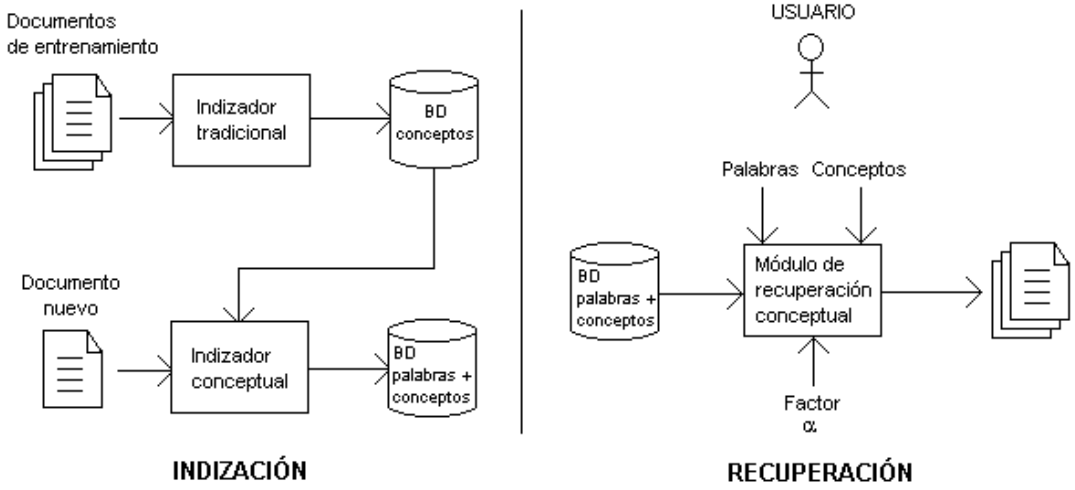


Figura 1.5 Arquitectura del sistema KeyConcept

CAPÍTULO 2

Desarrollo del proyecto

1. Fase de Especificación de requisitos
2. Fase de Análisis
3. Fase de Diseño
4. Validación (ejemplos)
5. Fase de Implementación
    - 5.1 Descripción general de la ontología
    - 5.2 Definición y explicación de los disparadores implementados
    - 5.3 Definición y explicación de los procedimientos implementados
    - 5.4 Otros elementos implementados
6. Fase de Pruebas

2.1 FASE DE ESPECIFICACIÓN DE REQUISITOS

Como al inicio de cualquier proyecto, la primera fase es la de marcar los objetivos y especificar los requisitos. Es decir, describir las necesidades que debe cubrir el sistema y los objetivos a alcanzar. Tras numerosas entrevistas con el cliente, representado en la figura del tutor de este proyecto, así como con la ayuda de representaciones formales de casos específicos siguiendo los modelos obtenidos se fue generando progresivamente, en varias versiones, el documento de especificación de requisitos que se encuentra en el anexo A. Estas representaciones formales se exponen en el apartado 2.4 de este capítulo.

La finalidad de esta primera fase es esbozar una visión general sobre el trabajo que se va a realizar, los conocimientos necesarios a adquirir, una planificación temporal informal, etc. Una vez definidos y comprendidos cada uno de los requisitos se pasa a la fase de análisis.

El objetivo del proyecto es crear un sistema que no sólo almacene datos e información, sino que almacene conocimiento que aporte un valor añadido a la información almacenada. Del mismo modo, se pretende que la información almacenada pueda ser entendida, gestionada y actualizada por agentes software. Dichos agentes utilizarán el conocimiento disponible en la ontología para poder realizar mejor la función que tengan asignada. Como se puede deducir de estas palabras, el objetivo es el diseño e implementación de una ontología. Es importante que quede claro en este punto, que no es una de las metas del proyecto el contenido concreto de la ontología. No se busca insertar un conocimiento adecuado para un dominio concreto sino, únicamente, diseñar e implementar el *contenedor* y posibilitar mecanismos para su cumplimentación, uso y mantenimiento.

Los requisitos se abordan desde diferentes perspectivas. Por un lado, hay que definir detalladamente los usuarios del sistema, las funciones del sistema, los interfaces software, etc. Se requieren cuatro perfiles de usuario: un editor de conocimiento, un usuario estándar y dos agentes software.

El editor de conocimiento es el encargado de introducir los datos en la ontología, de insertar conocimiento y eliminarlo cuando sea necesario. Es una tarea fundamental puesto que el buen funcionamiento de los diferentes procedimientos implementados depende, en gran medida, de la calidad de la información almacenada. Para incrementar la usabilidad del sistema y la facilidad de acceso, se pide también, implementar al editor de conocimiento un interfaz

amigable desde el que pueda desarrollar su trabajo sin tener que acceder directamente a través de la consola del SQL Plus. Sólo utilizará esta consola para resolver problemas de administración que no puedan ser tratados desde el interfaz.

Los agentes software no serán desarrollados como parte de este proyecto, pero se han de diseñar e implementar todas las funciones necesarias que posteriormente serán utilizadas por dichos agentes. Se necesita que se realicen procedimientos que aporten conocimiento y faciliten la tarea de los agentes software. Como ya se verá más adelante en este capítulo, se requieren funciones que trabajen con los términos y conceptos de la ontología. En un principio, existirán dos tipos de agentes software cuyo fin será el de recuperar información de internet. Por un lado, un agente software (asíncrono) que basará sus búsquedas en Google ayudándose para ello de la ontología (términos, conceptos). Por otro lado, un agente (síncrono) que dependerá al 100% de la ontología puesto que todas sus búsquedas las realizará con las fuentes (direcciones web) almacenadas en la ontología.

Por último, se pide la implementación de procedimientos que serán ejecutados, en un futuro, por un agente humano. Las necesidades de un usuario estándar se reducen a consultas del sistema de información (valores de atributos, fuentes), así como, consultas ontológicas que le puedan ser de utilidad (conceptos, términos, etc). No es objetivo de este proyecto desarrollar un interfaz gráfico para este perfil de usuario.

El documento de especificación de requisitos (anexo A) incluye un análisis de viabilidad de las especificaciones dadas. Tras haber estudiado cada requisito y haber analizado dicho estudio de viabilidad se puede concluir que el proyecto es viable y que se pueden alcanzar todos y cada uno de los requisitos demandados por el cliente puesto que se cuenta con los conocimientos y con la tecnología necesaria para ello (máquinas, software, etc).

Una vez descubiertos y recopilados los requisitos y sus complicaciones a nivel tecnológico, se pasa a la siguiente fase: análisis, preparándose paralelamente el equipo de desarrollo en las tecnologías involucradas (PL/SQL y Jdeveloper).

Se invita al lector a consultar el anexo A para conocer los pormenores del documento de especificación de requisitos, si así lo estima oportuno, antes de pasar a leer el siguiente apartado, donde se trata el análisis llevado a cabo en este proyecto.

2.2 FASE DE ANÁLISIS

Una vez terminada la primera fase y con el documento de especificación de requisitos redactado y aprobado comienza la tarea del análisis. En este apartado se expone de forma detallada los principales problemas y decisiones tomadas durante la fase de análisis. Dichas decisiones son fundamentales para poder obtener un diagrama entidad relación válido en la fase siguiente: diseño.

Se trata de construir una ontología, y como ya se ha visto en el capítulo 1, hay muchas tecnologías disponibles para ello. El cliente ha propuesto la implementación de la ontología en SQL, lo que orienta, en gran medida, este análisis. Como consecuencia del lenguaje elegido para la implementación de la ontología, se va a necesitar disponer al terminar la fase de diseño de un diagrama entidad relación consistente y fiel a los requisitos definidos en el documento E.R.S. Por este motivo, en esta fase de análisis, se van a ir identificando posibles entidades y posibilidades de modelar cada uno de los requisitos junto con una justificación para cada decisión tomada.

Necesidades de información

Se pide que el sistema sea capaz de almacenar conceptos. Los conceptos son, junto con las relaciones, los elementos básicos de una ontología. Parece necesaria y evidente la existencia de una entidad destinada a contener conceptos. Así mismo, se pide que cada concepto tenga, al menos, una representación simbólica. Es decir, que cada concepto sea referenciado por uno o varios términos. De inmediato queda descartada la posibilidad de añadir a la entidad que contiene conceptos un atributo con este fin, porque sólo podría contener una de todas las representaciones simbólicas con las que cuenta un concepto. Se podría haber modelado como un atributo multivaluado, pero esta opción tiene la limitación de que no se puede añadir una calificación que exprese el grado de calidad con la que un término referencia a un concepto. Por estos motivos, la mejor solución para almacenar términos es crear una nueva entidad, y en la interrelación existente entre un concepto y un término añadir un atributo para recoger la calidad de dicha representación simbólica. Además, esta solución aporta indirectamente otro beneficio: en los casos en que un término referencia a varios conceptos, se evita que dicho término se repita “n” veces para cada uno de los conceptos que referencia.

Por otra parte están las relaciones, básicas para el funcionamiento de cualquier ontología. Hay una gran variedad de relaciones que se pueden definir para una ontología. Por la

naturaleza de la funcionalidad de una relación es obvio que no se puede incluir en ninguna de las entidades existentes hasta el momento. Por lo tanto, es necesario crear una nueva entidad donde anotar las relaciones de la ontología. Se descartó la posibilidad de tener una jerarquía, donde la raíz fuera una tabla genérica que almacenara las relaciones y en los diferentes subtipos de la jerarquía cada una de ellas, porque no aportaba ningún conocimiento a esta ontología. En el caso de que se hubiera deseado almacenar propiedades diferentes para cada relación o que dichas relaciones se hubieran comportado, o relacionado con otras entidades, de manera distinta unas de otras, hubiera sido la decisión acertada. Avanzando en el análisis del problema, se llega a la conclusión de tener una única entidad donde se almacenen las relaciones disponibles para la ontología.

Otro de los elementos básicos de una ontología son los axiomas, es decir, restricciones impuestas a la ontología en el dominio de discurso. Dado que la ontología se va a implementar en un lenguaje de desarrollo de bases de datos y dado que cualquier base de datos necesita establecer controles para mantener su consistencia, en esta ontología también son imprescindibles dichos controles. Como cualquier otra base de datos, dichos controles son implementados a través de disparadores. En este caso, dichos disparadores realizan el rol de axiomas de la ontología. Posteriormente, en el apartado 2.5 implementación, se explican y justifican todos los disparadores implementados en la ontología.

También se especifica que los conceptos puedan tener atributos para describir sus propiedades y asignarles un valor en función de un momento espacio-temporal. Estos atributos no pueden ser modelados como campos de la entidad que contiene los conceptos puesto que lo que se pretende es construir una ontología lo más general posible, y definir determinados atributos para un concepto limita, en gran medida, este deseo. Cada concepto puede tener un número ilimitado de atributos, y diferentes de un concepto a otro. Tampoco puede ser un campo multivaluado de dicha entidad, puesto que para un atributo se necesita almacenar datos, tales como, el dominio que restringe sus valores o la caducidad (por defecto) de los mismos. Las especificaciones dadas obligan a crear una nueva entidad para introducir los atributos de cada concepto.

Es importante no confundir la definición de un atributo con el valor que posteriormente puede tomar ese atributo. Como ya se ha señalado, un atributo puede ser valorado en diferentes espacios o lugares. Para una ontología más sencilla podría ser una solución válida introducir en la entidad que contiene atributos, un campo valor. Pero en este caso se pretende dar un paso más

allá. Los valores pueden ser diferentes en función de una variable espacio-temporal. Por este motivo, y por intentar aplicar criterios de abstracción, se decide que la entidad de los atributos sólo contenga características propias, y aspectos como su valor, en un espacio determinado y en un momento concreto, se deciden modelar de forma independiente. Para representar el espacio en el cual se instancia un valor hay también diferentes alternativas. Si se añade el campo espacio a la entidad atributo implicaría tener una tupla por cada pareja espacio-atributo. Para evitar todos los valores repetidos que se almacenarían, tanto de espacios como de atributos, la mejor solución, como se explica anteriormente, es sacar esa información a una nueva entidad. Por último, hay que destacar que un espacio puede ser modelado, en esta ontología, como un concepto más.

Al hablar de los atributos y de sus valores, nace la necesidad de controlar dichos valores. Como ocurre en el diseño de cualquier base de datos, el modo de controlar los valores es restringirlo por un dominio. Para representar un dominio se necesita crear una nueva entidad, puesto que los dominios, no sólo almacenan un nombre descriptivo, sino que también almacenan un conjunto de características útiles para describir cada uno de ellos. En adicción a esto, hay una gran diversidad de dominios y no se puede limitar la idea de dominio a definir una cota superior y una cota inferior. Partiendo de esta base, se decide crear una jerarquía dominio donde se hace explícita la existencia de dominios numéricos, caracterizados por una cota inferior y una cota superior, y dominios alfanuméricos que están caracterizados por tener un conjunto de valores válidos, es decir, análogo a un tipo de datos enumerado. Ante la nueva dificultad aparecida, se asume que todos los valores que forman un dominio alfanumérico son conceptos. Esta nueva visión de concepto lleva a la conclusión de que existen dos tipos de conceptos: aquellos que contienen a otros conceptos y aquellos que en sí mismos ya son “un todo”, o dicho de otra manera, son una instancia concreta de un concepto. Por lo tanto, aparece una nueva jerarquía para la entidad concepto, donde se puede ver que hay conceptos genéricos (nodo no hoja) y conceptos instancias (nodo hoja). De este modo, se dice que un dominio alfanumérico es un concepto genérico, donde sus instancias componen el conjunto de valores válidos del dominio. Por otro lado, la entidad espacio es un concepto instancia puesto que contiene instancias concretas, de un concepto superior, donde se valora un atributo (“Madrid”, “Océano Atlántico”, “Júpiter”, “Everest”, etc). Ese concepto superior podría ser referenciado, para los ejemplos dados, con términos tales como “ciudades”, “océanos”, “planetas”, “montañas”, etcétera.

Para finalizar, sólo queda por satisfacer una de las necesidades que contiene el documento de especificación de requisitos. Existe la figura de un agente software, que a través de fuentes (direcciones web) contenidas en la ontología será capaz de acceder a internet para buscar valores actualizados de un atributo. Hay dos alternativas válidas para almacenar las fuentes de cada atributo. Una de ellas, sería incluir en la entidad que contiene los atributos un campo multivaluado y opcional. La segunda opción, es la de crear una nueva entidad. Se ha optado por la segunda opción por varios motivos. En primer lugar, es posible que muchos de los atributos definidos compartan las mismas fuentes y con esta solución no haría falta repetir "n" veces la fuente para cada atributo. En segundo lugar, las fuentes tienen propiedades. Actualmente sólo se ha definido un atributo, pero dada la velocidad con que evoluciona este mundo es posible que en poco tiempo se desearan incluir nuevas propiedades para una dirección web, como por ejemplo la dirección IP del servidor donde se encuentra alojada o la fecha en que se anotó en la ontología. El último de los motivos estudiados es la necesidad de definir un atributo para describir la bondad de una fuente. Este atributo se incluirá en la interrelación existente entre las entidades que contienen atributos y fuentes.

Análisis de la funcionalidad de cada perfil de usuario

Como se describe en el documento de especificación de requisitos se tiene varios perfiles de usuario para este sistema.

- Editor de conocimiento, que realiza inserciones y borrados en la ontología.
- Agentes software, que realizan consultas en la ontología y modifican alguno de los valores almacenados.
- Usuario estándar, que realiza consultas, tanto del sistema de información como de la ontología.

Se pide, por tanto, que se implementen todas las funciones necesarias para consultar la información relevante anotada en la ontología. Estas funciones serán ejecutadas por el perfil del agente software y por el perfil del usuario estándar. Del mismo modo, será necesario implementar procedimientos que permitan a los agentes software actualizar la información anotada.

Para el perfil del editor de conocimiento, se requiere que se implementen todas las funciones necesarias para efectuar inserciones y borrados de conocimiento ontológico. Se especifica en el E.R.S. que el editor de conocimiento ejecutará todas las funciones a través de

una interfaz gráfica sencilla e intuitiva. También se especifica que dicho interfaz deberá ser implementado con un programa que aporte un entorno de ventanas conocido para cualquier usuario familiarizado con Windows. En esta etapa de análisis se estudian los posibles caminos para desarrollar dicho interfaz. Para no complicar posteriormente su diseño, y dado que la funcionalidad del editor está claramente separada en dos partes (inserciones y borrados), el interfaz tendrá dos "índices", uno para añadir conocimiento a la ontología y otro para eliminarlo. Desde estos "índices" se podrán ejecutar todas las operaciones disponibles. Por razones de seguridad se permite, únicamente, al editor de conocimiento acceder directamente a la consola *SQL plus*, si para tareas de administración de la ontología necesita ejecutar determinadas operaciones no disponibles en el interfaz.

En el apartado 2.5 implementación, se describen detalladamente cada una de las funciones implementadas para cada perfil. Es importante señalar que los procedimientos de cada tipo de usuario serán agrupados en un paquete y que cada usuario podrá ejecutar, exclusivamente, las operaciones que ofrece su paquete. De este modo, existirá un paquete para los agentes software, otro para el usuario estándar y otro para el editor de conocimiento.

Al concluir la fase de análisis, se llega a la conclusión de que la mayor parte de las operaciones de consulta de los agentes software son compartidas por el usuario estándar. La principal diferencia entre ambos perfiles es que el agente humano accederá al sistema a través de una interfaz gráfica. Además, los agentes software tienen la propiedad de poder actualizar el sistema de información.

2.3 FASE DE DISEÑO

Se ha visto en las fases anteriores que el objetivo es crear una ontología siguiendo la metodología de creación de bases de datos, por lo tanto, la finalidad de esta fase es construir un diagrama entidad relación definitivo.

Tras la fase de análisis ya se tienen claras las necesidades definidas por el cliente en el documento de especificación de requisitos y qué es lo que se necesita almacenar en función de esos requisitos. En dicha fase se ha tratado detalladamente qué tiene que tener la ontología y se han estudiado las mejores opciones para cada caso, justificando detalladamente cada una de las decisiones. En este punto del proyecto se está en disposición de diseñar el diagrama entidad relación (figura 2.1) apoyándose en todas las conclusiones obtenidas en la fase anterior. Este

diagrama entidad relación representa la estructura (entidades, relaciones, etc) de la nueva ontología diseñada e implementada. Proporciona el esqueleto con el que construir una base de conocimiento añadiendo semántica a la información que contiene. Todo el contenido del diagrama E/R deriva del trabajo realizado en el punto 2.2 de este capítulo, por lo tanto, en este apartado no será necesario explicar muchas decisiones ya tratadas en la fase de análisis. Dicho diagrama entidad relación está compuesto por doce entidades que se explican brevemente a continuación: entidad concepto, entidad genérico, entidad instancia, entidad término, entidad relación, entidad atributo, entidad dominio, entidad numérico, entidad alfanumérico, entidad fuente, entidad valor, entidad espacio.

La entidad concepto almacena las ideas que se intentan formalizar en un determinado universo de discurso. Está formada por una jerarquía donde se representa que un concepto puede ser genérico (nodo no hoja) o instancia (nodo hoja). Se han estudiado distintas maneras de almacenar un concepto pero la más conveniente para este caso es asignarle un código numérico secuencial.

Todo concepto necesita, al menos, una representación simbólica; un término de algún lenguaje con el que ser referenciado. Esta información se almacena en la entidad término. Con el atributo ‘grado’, incluido en la interrelación ‘referencia’, se expresa la “pureza” con la que ese término representa un concepto (expresado en tanto por ciento) y con el atributo ‘lenguaje’, se expresa a qué lenguaje pertenecen los símbolos que componen dicho término (castellano, árabe, ascii, etc).

Los conceptos se interrelacionan entre sí a través de las relaciones contenidas en la entidad relación, siendo fácilmente ampliable para proyectos futuros. La interrelación existente entre la entidad concepto y la entidad relación es ternaria. Hay tres roles bien diferenciados:

rol-1: entidad concepto es primer operando.

rol-2: entidad concepto es segundo operando.

rol-3: entidad relación es operador.

Se hace explícito de este modo que el orden de los conceptos, en general, no es conmutativo. Interrelación “relaciona” (rol1, rol2, rol3).

Por otro lado, los conceptos pueden tener atributos que describan sus propiedades. La entidad atributo recoge esta información, así como la validez que tendrá por defecto cualquier valor para ese atributo, expresado en una unidad de medida temporal (para este proyecto se ha elegido la unidad días). Este atributo 'caducidad' será utilizado, por ejemplo, para calcular una fecha de fin de validez de un valor cuando sólo se introduce una fecha de inicio. Todo atributo está acotado por un dominio y sus valores deben cumplir con las restricciones impuestas por dicho dominio.

Las instancias tendrán un valor restringido por un dominio para cada atributo, pudiendo especificar, opcionalmente, la fecha de inicio y fin de validez de un determinado valor para un atributo, así como la asignación de un espacio determinado donde ese valor tiene vigencia (ciudad, océano, planeta, etc). Un dominio puede ser numérico (definido por una cota superior y una cota inferior) o alfanumérico. En esta ontología se ha considerado el dominio alfanumérico como un concepto más, cuyas instancias forman el conjunto de los posibles valores que puede tomar un atributo.

Por último, se ha de explicar que todo atributo puede tener una fuente de la que obtener datos para que los agentes web puedan actualizar el sistema de información. Las fuentes tienen un peso, y en este caso se ha elegido un rango para peso [0..10] que especifica la bondad de dicha fuente para actualizar un atributo determinado.

Es importante destacar que el diagrama entidad relación distingue, con la tonalidad de las cajas, entre entidades propias de la ontología y entidades propias del sistema de información (cajas grises).

Diagrama Entidad Relación

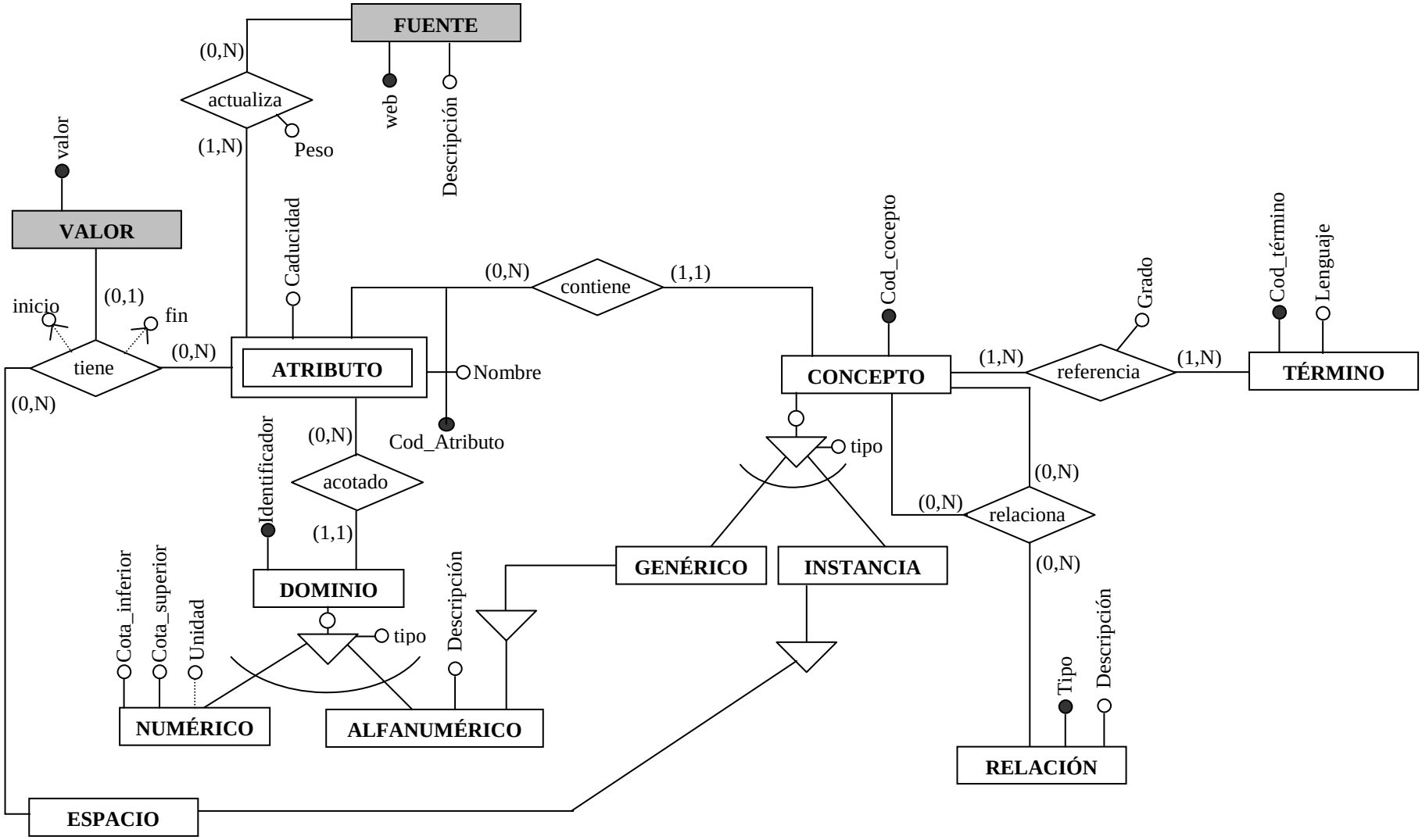


Figura 2.1

2.4 VALIDACIÓN (ejemplos)

Se han realizado un conjunto de ejemplos para mostrar cómo la ontología estructura el conocimiento desde el punto de vista del diagrama entidad relación. Cada uno de los ejemplos mostrados a continuación son representaciones parciales de determinados dominios de un discurso, creados no con la intención de representar fielmente un dominio o escenario, sino con intención didáctica de explicar el diseño de la ontología creada.

- **Ejemplo 1** (figura 2.2)

El objetivo de este ejemplo es el de entender los elementos básicos del diagrama entidad relación (conceptos y relaciones) para poder abordar otros más complejos. Se muestra que cada concepto, referenciado por uno o varios términos, se define con un código único y que dichos conceptos se relacionan, en este caso, mediante una relación 'compuesto de' y otra 'es un'.

- **Ejemplo 2** (figura 2.3)

Este ejemplo se puede dividir en dos partes (dos ámbitos). Por un lado está el ámbito de clima y por otro lado el ámbito de enfermedad.

El ámbito de clima representa que el clima está formado (relación 'compuesto de') por humedad (concepto 2031), viento (concepto 2032), estado del cielo (concepto 2033) y temperatura (concepto 2034).

El ámbito de enfermedad representa que una enfermedad está formada (relación 'compuesto de') por medicación (concepto 2001) y síntomas (concepto 2002). Síntomas como mucosidad (2022), tos (2021) y temperatura (2023), relacionados a través de una relación 'es un'.

La pretensión de este ejemplo es mostrar que un término puede identificar o definir conceptos que pertenecen a ámbitos distintos. De este modo, el término temperatura referencia al concepto 2023 (ámbito enfermedad) y al concepto 2034 (ámbito de clima).

- **Ejemplo 3** (figura 2.4)

La ambición de este ejemplo es poder expresar, de modo gráfico, cómo se instancian los conceptos, cómo los atributos adquieren valores concretos, donde dichos valores están acotados por un determinado dominio. Se tiene una instancia del concepto coche. Dicho concepto (3002) tiene dos atributos, color y año de compra. El atributo año de compra adquiere un valor y está acotado por un dominio numérico restringido al intervalo de años 1900-3000. Por otro lado, el atributo color está acotado por un dominio alfanumérico donde el conjunto de los posibles valores que puede tomar ese atributo lo forman las diferentes instancias de un determinado concepto.

- **Ejemplo 4** (figura 2.5)

Este ejemplo representa el comportamiento de una relación 'función', donde a partir de una instancia (4002) "precio sin iva" es posible crear o valuar, a través de una función IVA, la instancia (4003) correspondiente de "precio con iva".

- **Ejemplo 5** (figura 2.6)

Con este ejemplo se pretende aclarar la entidad espacio que puede resultar un poco gris a priori. El concepto (3000) "parte meteorológico" permite "n" instanciaciones simultáneas, en función del dúo valor-espacio para el atributo temperatura de dicho concepto. Todo espacio es una instancia de un concepto, en este caso del concepto (4000) ciudad.

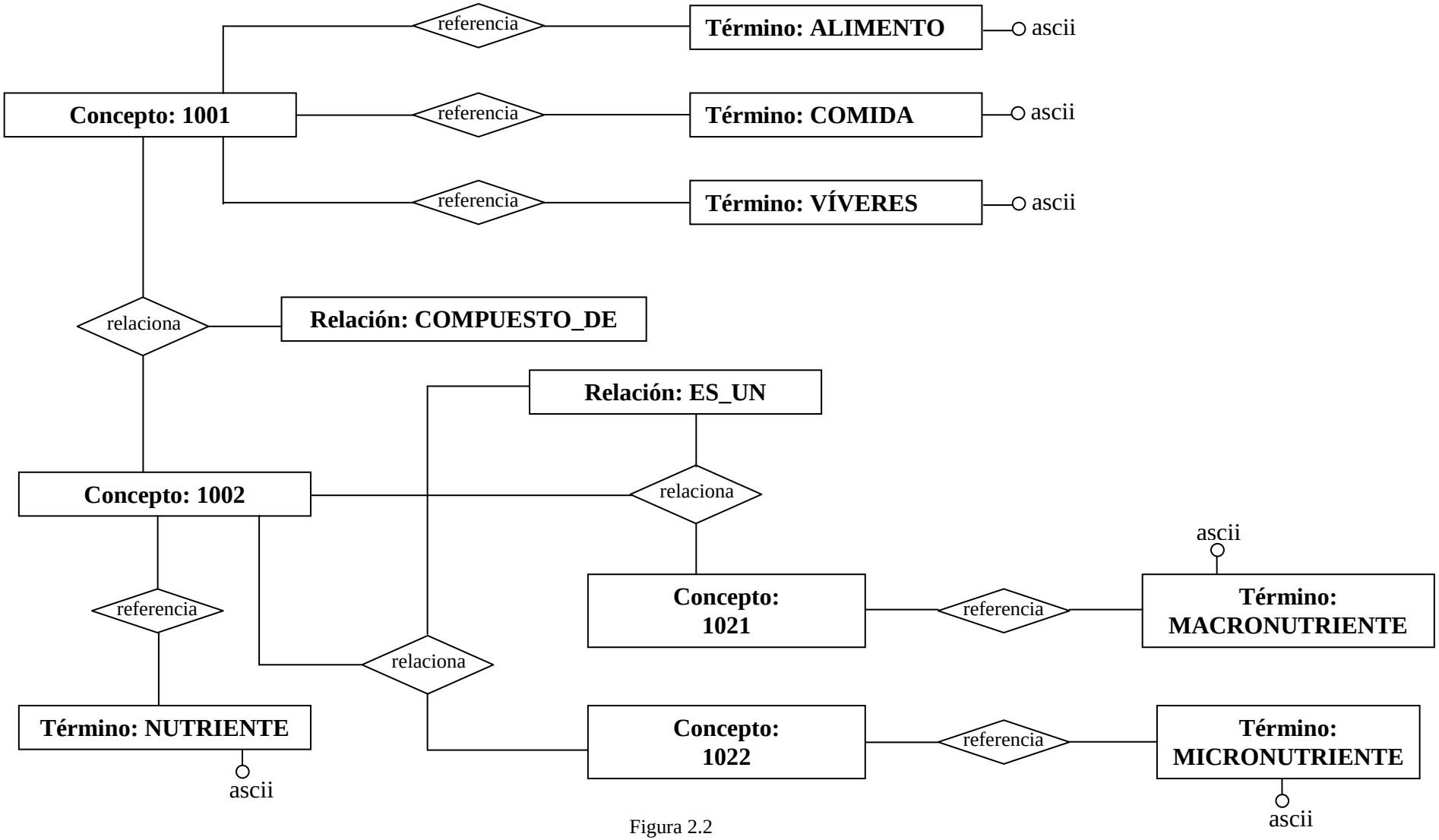

referencia
Término: ALIMENTO
ascii
Concepto: 1001
referencia
Término: COMIDA
ascii
referencia
Término: VÍVERES
ascii
relaciona
Relación: COMPUESTO_DE
Relación: ES_UN
Concepto: 1002
relaciona
ascii
referencia
Concepto:
1021
referencia
Término:
MACRONUTRIENTE
relaciona
Término: NUTRIENTE
ascii
Concepto:
1022
referencia
Término:
MICRONUTRIENTE
ascii

Figura 2.2

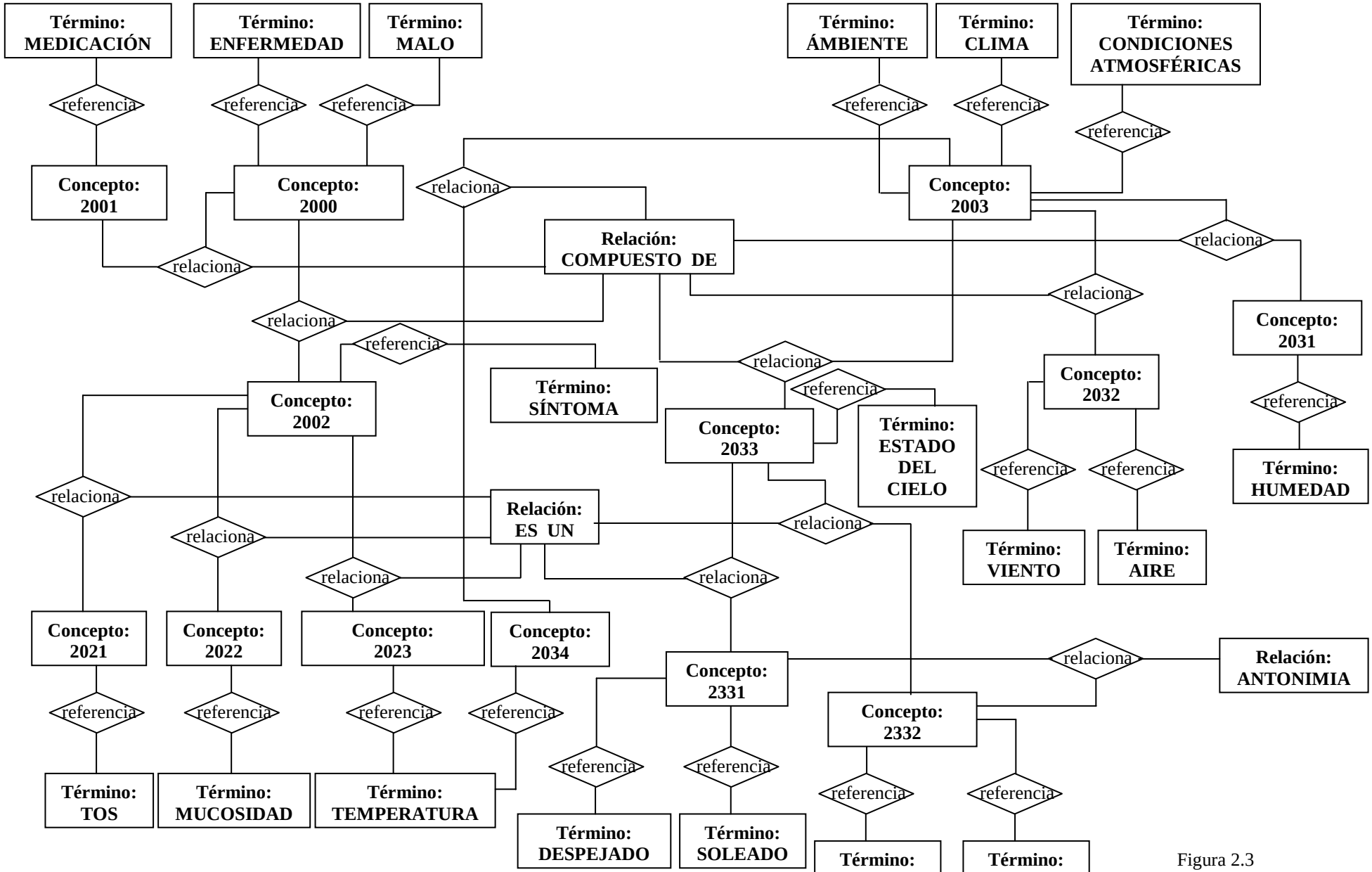

Término: MEDICACIÓN
Término: ENFERMEDAD
Término: MALO
Término: ÁMBIENTE
Término: CLIMA
Término: CONDICIONES ATMOSFÉRICAS
referencia
Concepto: 2001
Concepto: 2000
relaciona
Concepto: 2003
Relación: COMPUESTO DE
Concepto: 2031
Concepto: 2032
Término: SÍNTOMA
Concepto: 2002
Concepto: 2033
Término: ESTADO DEL CIELO
Término: HUMEDAD
Relación: ES UN
Término: VIENTO
Término: AIRE
Concepto: 2021
Concepto: 2022
Concepto: 2023
Concepto: 2034
Concepto: 2331
Concepto: 2332
Relación: ANTONIMIA
Término: TOS
Término: MUCOSIDAD
Término: TEMPERATURA
Término: DESPEJADO
Término: SOLEADO
Término: NUBLADO
Término: NIEBLA

Figura 2.3

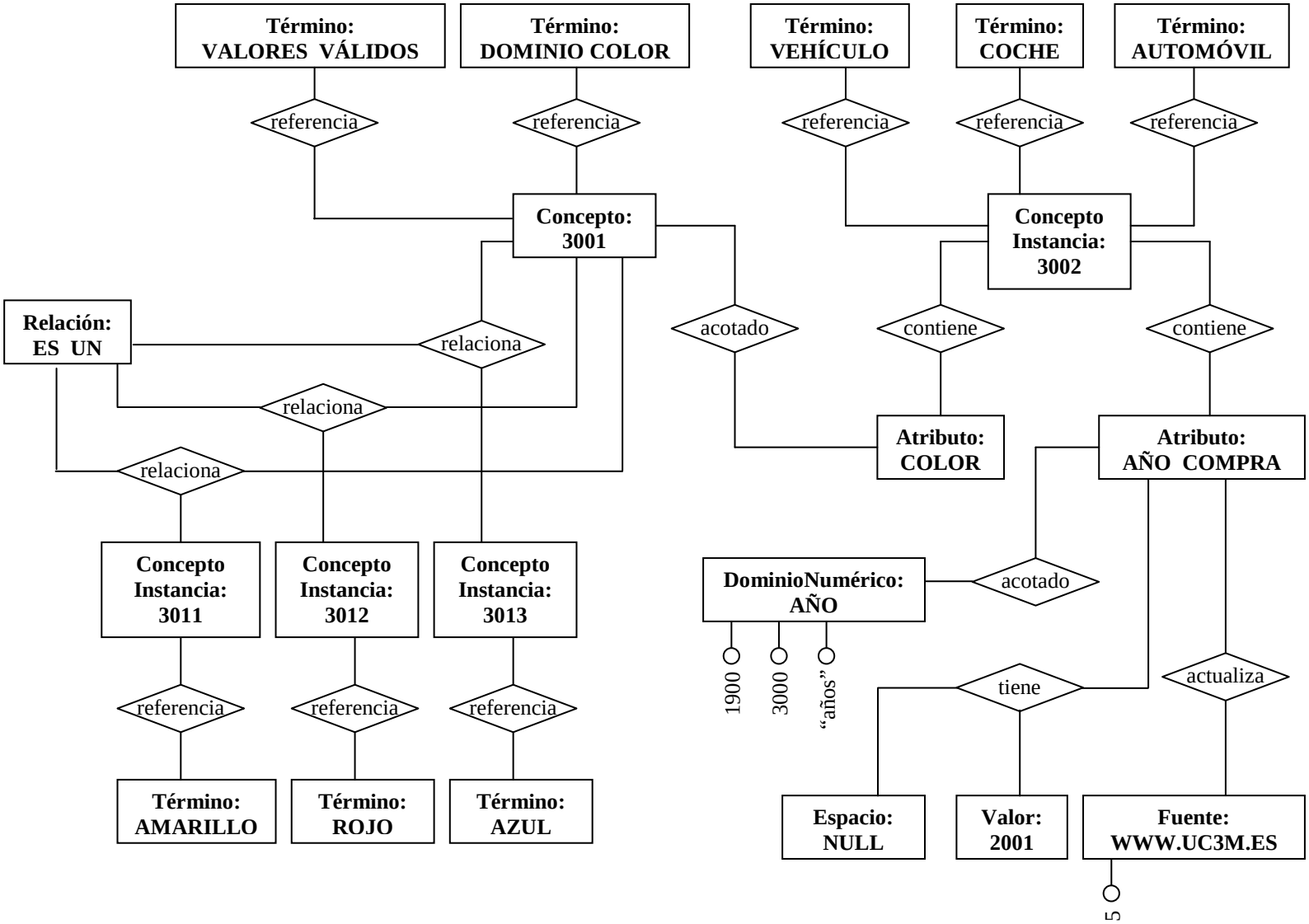

Término: VALORES VÁLIDOS
Término: DOMINIO COLOR
Término: VEHÍCULO
Término: COCHE
Término: AUTOMÓVIL
referencia
referencia
referencia
referencia
referencia
Concepto: 3001
Concepto Instancia: 3002
Relación: ES UN
relaciona
relaciona
relaciona
acotado
contiene
contiene
Atributo: COLOR
Atributo: AÑO COMPRA
Concepto Instancia: 3011
Concepto Instancia: 3012
Concepto Instancia: 3013
DominioNumérico: AÑO
acotado
1900
3000
"años"
tiene
actualiza
referencia
referencia
referencia
Término: AMARILLO
Término: ROJO
Término: AZUL
Espacio: NULL
Valor: 2001
Fuente: WWW.UC3M.ES
5

Figura 2.4

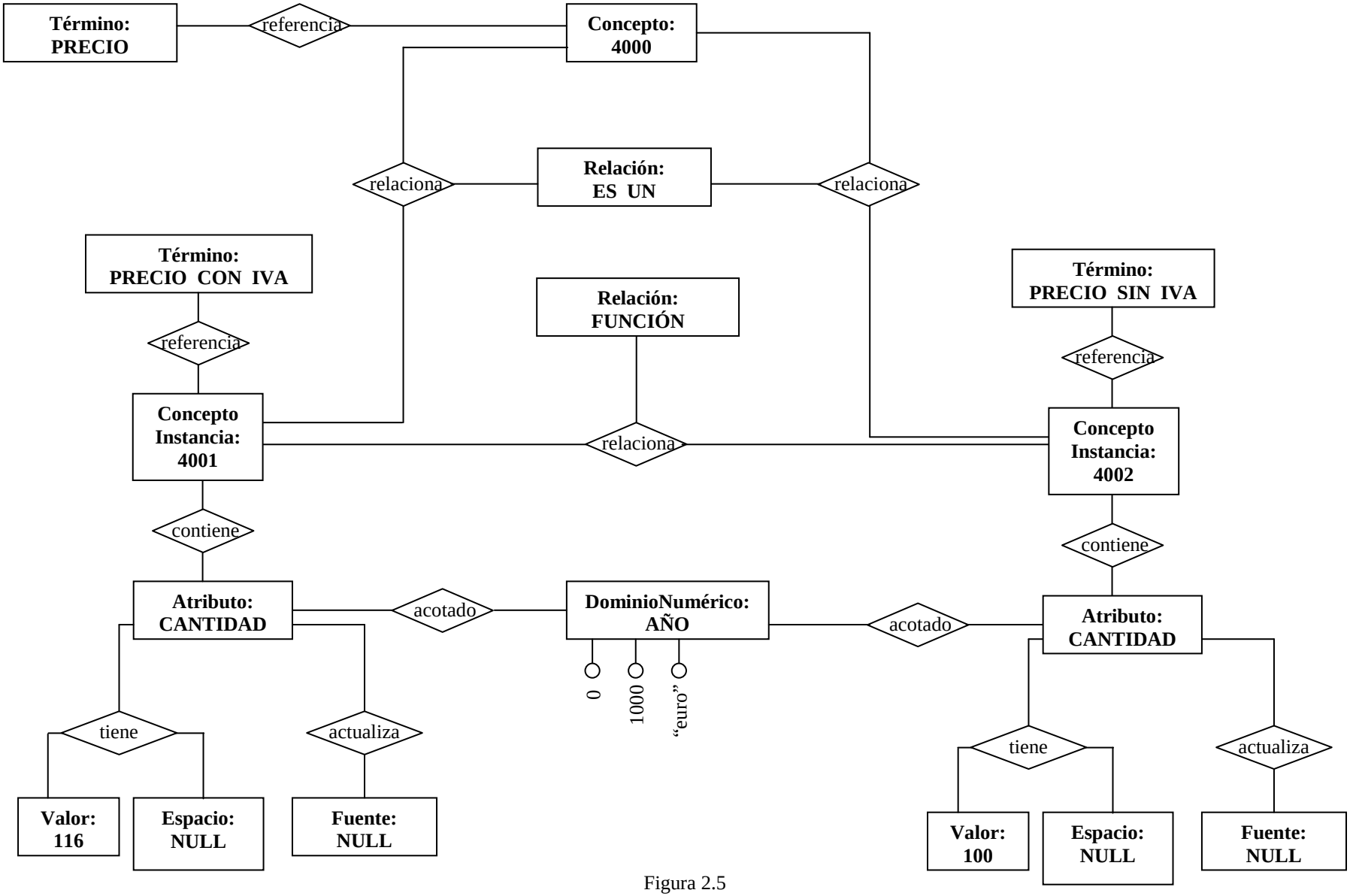

Término:
PRECIO
referencia
Concepto:
4000
relaciona
Relación:
ES UN
relaciona
Término:
PRECIO CON IVA
Relación:
FUNCIÓN
Término:
PRECIO SIN IVA
referencia
referencia
Concepto
Instancia:
4001
relaciona
Concepto
Instancia:
4002
contiene
contiene
Atributo:
CANTIDAD
acotado
DominioNumérico:
AÑO
acotado
Atributo:
CANTIDAD
0
1000
"euro"
tiene
actualiza
tiene
actualiza
Valor:
116
Espacio:
NULL
Fuente:
NULL
Valor:
100
Espacio:
NULL
Fuente:
NULL

Figura 2.5

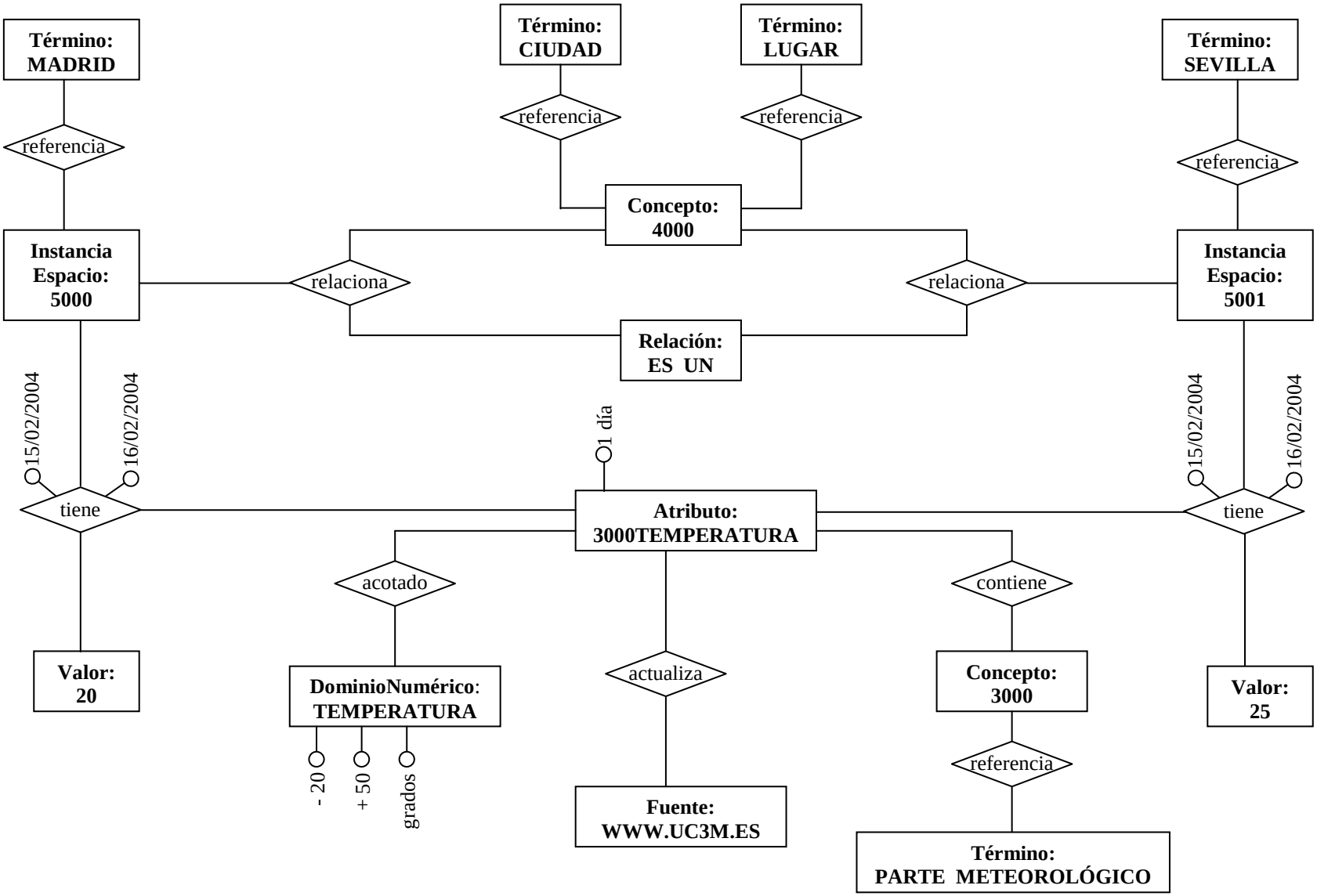

Término: MADRID
Término: CIUDAD
Término: LUGAR
Término: SEVILLA
referencia
referencia
referencia
referencia
Concepto: 4000
Instancia Espacio: 5000
relaciona
relaciona
Instancia Espacio: 5001
Relación: ES UN
15/02/2004
16/02/2004
1 día
15/02/2004
16/02/2004
tiene
Atributo: 3000TEMPERATURA
tiene
acotado
contiene
Valor: 20
DominioNumérico: TEMPERATURA
actualiza
Concepto: 3000
Valor: 25
- 20
+ 50
grados
Fuente: WWW.UC3M.ES
referencia
Término: PARTE METEOROLÓGICO

Figura 2.6

2.5 FASE DE IMPLEMENTACIÓN

Una vez definido el diagrama entidad relación, para poder comenzar a escribir código se necesita un modelo implementable. Por lo tanto, el siguiente paso es la transformación del diagrama entidad relación en un diagrama relacional (véase la figura 2.7).

En este apartado se explica cada una de las decisiones tomadas en la transformación del diagrama entidad relación en el diagrama relacional. Principalmente, las restricciones de borrado y actualización de las distintas tablas y las claves primarias de cada una de ellas junto con sus atributos:

- **Término**: Su clave primaria, 'cod_termino', es una cadena de caracteres. En el campo 'lenguaje' se almacena la lengua o conjunto de símbolos a la que pertenece la palabra utilizada como término. Se considera que no se puede borrar un término cuando exista un concepto que sea referenciado por dicho término. Las modificaciones se efectúan en cascada.

- **Relación**: Consta de dos campos, 'tipo', que es una cadena de caracteres y constituye la clave primaria de la tabla, y 'descripción', que explica brevemente el significado de la relación. No se puede borrar una relación mientras exista, al menos, una pareja de conceptos relacionados a través de dicha relación. Las modificaciones se realizan en cascada.

- **Concepto**: Tiene dos campos, 'cod_concepto', que es un entero y la clave primaria de la tabla; y el atributo 'tipo', que es utilizado para la gestión de la jerarquía genérico-instancia. Un concepto puede ser eliminado o modificado en cualquier momento, sin ninguna restricción para el editor de conocimiento.

- **Fuente**: Tiene dos atributos que son cadenas de caracteres. Su clave primaria es el atributo 'web'. El segundo atributo es 'descripción', que explica brevemente el contenido de la web. Se permite borrar una fuente en cualquier momento, cuando no sea representativa e importante para ningún atributo. Esa semántica no puede ser "controlada" por la ontología y es tarea de los agentes y del editor de conocimiento decidir si la fuente puede ser borrada. Las actualizaciones se realizan en cascada.

- **Atributo**: Su clave primaria es la unión de los campos ‘nombre’ y ‘cod_concepto’, el cual es clave ajena de la tabla concepto. Tiene un campo numérico, ‘caducidad’, en el que se almacena la validez de un valor para dicho atributo. Tiene otra clave ajena a la tabla dominio donde se almacena el dominio que restringe a un atributo. Se puede eliminar y actualizar un atributo de un concepto sin ninguna restricción.

- **Dominio**: Al igual que concepto, dominio forma una jerarquía con las tablas numérico y alfanumérico. Su clave primaria, ‘identificador’, es una cadena de caracteres. El atributo ‘tipo’ gestiona la jerarquía. No se permite borrar un dominio si existe algún atributo que está acotado por dicho dominio. Las modificaciones se realizan en cascada.

- **Numérico**: Su clave primaria es una clave ajena a dominio. Tiene dos campos donde se almacena las cotas numéricas inferior y superior que acotan dicho dominio. Opcionalmente se puede almacenar la unidad (métrica) en que están expresados los valores del dominio.

- **Alfanumérico**: Su clave primaria la constituyen la unión de un ‘cod_concepto’ de la tabla genérico junto con una clave ajena a dominio. Tiene un tercer campo, ‘descripción’, que explica brevemente el significado del dominio.

- **Espacio**: La tabla espacio contiene aquellos conceptos que se usan para dar una valoración espacial a un atributo. Se permite eliminar y modificar un espacio en cualquier momento sin ninguna restricción.

- Todas las jerarquías “es un” tienen borrado y modificación en cascada.

Respecto a las tablas **tiene1** y **tiene2**, es la representación elegida para el paso al diagrama relacional desde la interrelación “tiene” del diagrama ER. Como es obvio, existían diferentes alternativas pero se ha optado por ésta, por claridad y suficiencia. Desde el punto de vista conceptual (diagrama ER), se podría haber añadido una entidad “momento” que aglutinara espacio y tiempo (inicio y fin) para evitar redundancias. Estudiando esta solución, se llega a la conclusión de que es difícil la aparición de redundancias ya que la existencia de un espacio para un concepto es limitada y los atributos inicio y fin son opcionales y con muchas posibilidades de no aparecer como se ve en los ejemplos desarrollados. Desde el punto de vista lógico

(diagrama relacional) se ha decidido que el tiempo (‘inicio’ y ‘fin’) no pertenezca a la clave porque no se pretende generar un histórico de valores de atributos. Por este motivo, tampoco ‘valor’ forma parte de la clave primaria de las tablas tiene1 y tiene2.

Se han utilizado 3 tecnologías para la implementación de este proyecto. Con la ayuda de la herramienta Case Erwin se han desarrollado únicamente los *scripts* de creación de la base de datos, es decir, la creación de tablas. Esto incluye la definición de los tipos de datos de cada atributo así como los *checks* que restringen algunos valores y las restricciones de borrado y actualización. Todos los disparadores creados, los procedimientos implementados y el resto de componentes necesarios para el funcionamiento de la ontología se han realizado directamente a través de SQL Plus. Por último, el desarrollo del interfaz gráfico se ha implementado con la aplicación Oracle9i Jdeveloper, que utiliza tecnología java y permite ejecutar tanto procedimientos java como acceder y ejecutar procedimientos contenidos en un paquete definido en Oracle.

Diagrama Relacional

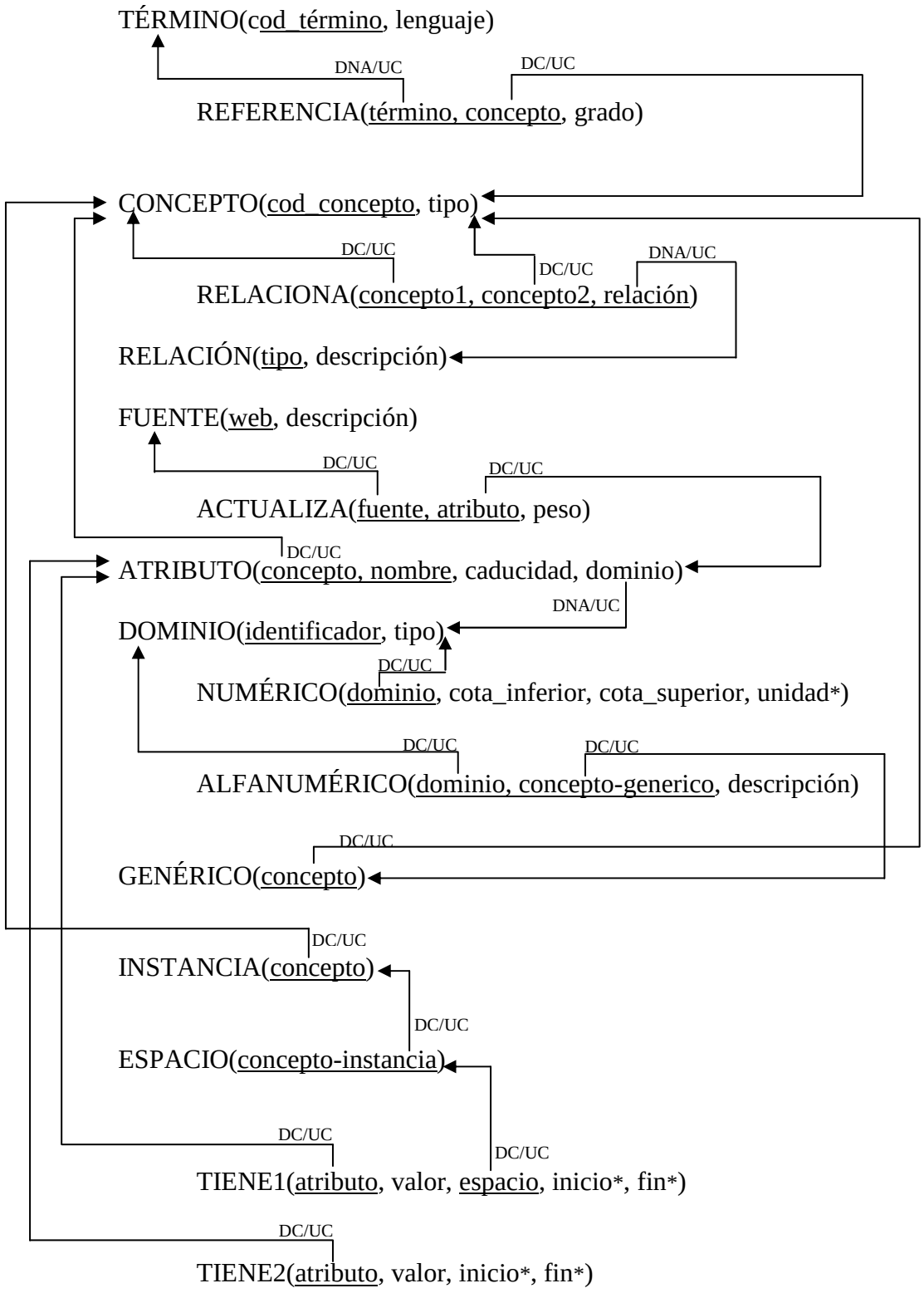


Figura 2.7

2.5.1 DESCRIPCIÓN GENERAL DE LA ONTOLOGÍA (BASE DE DATOS)

La jerarquía formada por las entidades concepto, genérico e instancia, se ha implementado en Oracle en tres tablas, donde el atributo tipo (NOT NULL) se encarga de controlar la totalidad/parcialidad de la jerarquía y la exclusividad/solapamiento se controla mediante un disparador, donde para cada inserción que se efectúa en concepto, automáticamente inserta en el correspondiente subtipo (instancia, genérico). La cardinalidad mínima (igual a uno) que existe entre las tablas concepto y término, se controla a través del procedimiento de inserción de un concepto, en el que para cada nuevo concepto se relaciona obligatoriamente con un término. Con el mismo fin de mantener la consistencia, el interfaz del editor de conocimiento no permite insertar directamente en cada uno de los subtipos (genérico e instancia).

La jerarquía dominio es controlada de manera similar a la jerarquía concepto. El atributo tipo, al igual que en la otra jerarquía, controla la totalidad/parcialidad, mientras que la exclusividad/solapamiento se controla mediante disparadores y procedimientos que se describirán más adelante. Hay que resaltar que no se permiten sentencias SQL sobre Oracle, sino que todas las operaciones se realizarán a través de un interfaz (sólo se pueden utilizar las operaciones dadas en el paquete de cada perfil).

El resto de jerarquías no controladas (alfanumérico y espacio) no necesitan la implementación de disparadores puesto que su clave primaria “es un” genérico e instancia respectivamente, y la integridad referencial controla que un alfanumérico "es un" genérico y que un espacio "es un" instancia.

En la implementación, se han definido algunos valores por defecto. El atributo grado de la tabla referencia, por defecto, indica que un término referencia a un concepto con un grado de calidad del 50%. Por otro lado, el atributo peso de la tabla actualiza tiene un valor por defecto de 5, en una escala de [0..10]. Para el atributo caducidad de la tabla atributo se ha definido un valor por defecto de 1 día.

El control del contenido de algunos de los campos se ha implementado con *checks*. De este modo, el atributo tipo de la tabla concepto sólo admite dos valores (genérico e instancia) y el atributo tipo de la tabla dominio, otros dos (numérico y alfanumérico). El atributo grado de la tabla referencia expresa porcentajes de calidad entre parejas de término-concepto y su rango

permitido es [0..100]. Por su parte, el atributo peso de la tabla actualiza admite valores del rango [0..10].

La herramienta ERwin, por defecto, genera todas las restricciones de borrado y modificación como RESTRICT (NO ACTION) y tan sólo permite definir los borrados en cascada. El resto de posibilidades hay que implementarlas mediante disparadores (*triggers*). En este caso, solo se ha necesitado implementar con disparadores las restricciones de modificación en cascada.

2.5.2 DESCRIPCIÓN DE LOS DISPARADORES IMPLEMENTADOS

Para controlar el buen funcionamiento y consistencia de la ontología, se han implantado los siguientes controles, implementados con disparadores:

- Existe un conjunto de disparadores que ejecutan todas las restricciones “update cascade” de las diferentes tablas.

- **Disparador T_concepto**

Se encarga de insertar en el correspondiente subtipo (genérico, instancia) cuando se inserta una nueva tupla en la tabla ‘concepto’.

- **Disparadores T_validar1 y T_validar2**

Antes de insertar una fila en las tablas ‘tiene1’ y ‘tiene2’, respectivamente, se comprueba si el valor introducido para un atributo cumple con las condiciones del dominio que le restringe.

- **Disparador T_numérico**

Controla que en la tabla ‘numérico’ la cota superior sea estrictamente mayor que la cota inferior.

- **Disparadores T_tiene1 y T_tiene2**

Controlan que en las tablas ‘tiene1’ y ‘tiene2’, respectivamente, la fecha inicio sea menor estricto que la fecha fin.

- **Disparador T_fechas1 y T_fechas2**

Controlan en las tablas ‘tiene1’ y ‘tiene2’, respectivamente, que no se almacene una fecha de inicio sin una fecha de fin. Si se introduce una fecha de fin sin un inicio, se mostrará un mensaje de error. Si se introduce una fecha de inicio sin una fecha de fin, se calcula, con la ayuda del atributo caducidad, la fecha de fin.

- **Disparador T_referencia**

Controla que tras borrar en la tabla ‘referencia’, no exista en la tabla ‘término’, alguno que no referencie a ningún concepto. Si es así, se elimina.

- **Disparador T_ref**

Controla que no se borre una fila de la tabla ‘referencia’ si el concepto involucrado no tiene otro término que lo referencie. En ese caso, previamente habría que borrar de la tabla ‘concepto’ y luego de ‘referencia’.

- **Disparador T_actualiza**

Controla que no exista una fuente que no actualice a ningún atributo. Cuando se borra de la tabla ‘actualiza’, se eliminan de la tabla ‘fuente’ aquellas que no sean utilizadas por ningún atributo.

- **Disparador T_alfanumérico**

En los casos en que se permita borrar una fila de la tabla ‘alfanumérico’ (véase el apartado “comentarios de los disparadores”), el disparador borra automáticamente de ‘dominio’ el identificador de dominio correspondiente. No puede existir un dominio alfanumérico sin un dominio “padre”.

En la siguiente tabla se recogen los errores (raise_application_error) definidos para estos disparadores:

Disparador	Nº de error	Explicación
T_numerico	-20000	La cota superior debe ser mayor que la cota inferior
T_tiene1	-20001	La fecha inicio debe ser anterior a la fecha fin
T_tiene2	-20002	La fecha inicio debe ser anterior a la fecha fin
T_fechas1	-20003	No se puede introducir fecha fin sin fecha inicio
T_fechas2	-20004	No se puede introducir fecha fin sin fecha inicio
T_validar1	-20007	Valor no valido para la tabla tiene1
T_validar2	-20008	Valor no valido para la tabla tiene2

Comentarios de los disparadores

Durante la implementación de los distintos disparadores, han surgido diversos problemas que se detallan a continuación junto a las soluciones adoptadas y sus correspondientes justificaciones.

Para mantener la integridad y la semántica de la ontología, se vio necesario introducir un disparador que, antes de borrar un concepto, comprobara si se trataba de un dominio alfanumérico utilizado por algún atributo (véase figura 2.1). En ese caso, no se debería eliminar el concepto. Si ningún atributo utiliza ese dominio, se puede eliminar el concepto deseado. Para implementar este control hay dos soluciones:

1. Control a nivel de disparador

   A nivel de disparadores existen varias posibilidades. Para este caso se han barajado las tres siguientes:

   - Mediante vistas y *triggers instead of*: es el más elegante y el más complejo de implementar. Es la mejor solución para sistemas grandes pero para el problema planteado no es una solución eficiente.
   - Deshacer la operación de borrado que se haya efectuado, volviendo a insertar aquello que no se debería haber borrado: es limpio y práctico pero poco elegante y eficiente para este problema particular debido a que el borrado afecta a un porcentaje muy alto de las tablas y rehacer la operación sería muy complejo y laborioso.
   - Tras desechar las dos primeras opciones, se plantea la de levantar un *RAISE_APPLICATION_ERROR*: es menos elegante pero para este caso, la más eficiente. Tras estudiar dicha solución, aparecen los siguientes problemas:

a) al tratarse de la jerarquía concepto (genérico, alfanumérico) todo *trigger* que compruebe algo desde concepto a alfanumérico, a nivel de fila, dará error de tabla mutante.

b) para solucionar el problema del apartado “a”, se haría a nivel de sentencia. Haciéndolo de este modo, no se puede acceder a la variable :old para saber qué concepto se está borrando y a partir de ese valor efectuar las comprobaciones.

c) para poder solucionar el apartado “b”, se hicieron dos *triggers*, un 'before' para almacenar en una tabla auxiliar el valor del concepto que se quiere borrar, y un 'after' para hacer las operaciones. Esta solución es aún peor, porque al utilizar el 'after delete' ya se ha borrado y no se puede disparar el *RAISE_APPLICATION_ERROR* a tiempo.

d) la única solución que queda sería rehacer todos los borrados e insertar de nuevo en concepto. Pero, como ya se explica anteriormente, no es una buena solución para este problema.

e) A nivel de disparadores, existe una única solución factible y eficiente para este caso. Se trataría de implementar dos *triggers* ‘before’, uno a nivel de sentencia, que se ejecuta primero, y otro a nivel de fila, que se ejecuta después. Con el *trigger* a nivel de sentencia se guardarían en una tabla auxiliar aquellos conceptos que se desean borrar, y posteriormente, para cada fila se comprobaría si es un dominio alfanumérico utilizado por algún atributo; y, en caso afirmativo, se dispararía el *RAISE_APPLICATION_ERROR*. Sin embargo, se ha desechado esta solución por ser poco elegante.

2. Control a nivel de procedimiento de borrado

   Tras haber estudiado todos los puntos anteriores, y dado que todas las operaciones del editor de conocimiento se realizarán a través de un interfaz que le ofrece un conjunto de operaciones que ejecutar, para este caso, se ha elegido esta opción como la mejor solución, puesto que es sencilla de implementar, eficiente, fácil de mantener y de añadir cualquier modificación.

Por otro lado, al implementar otros dos disparadores, se ha encontrado una debilidad de Oracle. Se pretendía controlar que tras borrar de la tabla actualiza (véase figura 2.7), se comprobase en la tabla fuente si existía alguna página web que ya no modificara a ningún

atributo. En ese caso, para mantener la semántica que aporta la cardinalidad (1:N) entre las tablas fuente y atributo (véase figura 2.1) se realizó el siguiente disparador:

```
CREATE OR REPLACE TRIGGER T_actualiza
AFTER DELETE ON actualiza
BEGIN
  DELETE FROM fuente
  WHERE web NOT IN (SELECT web
                    FROM actualiza);
END T_actualiza;
```

Con un disparador de estas características Oracle se mete en un bucle infinito y genera un error ("se ha excedido el número de llamadas recursivas permitidas"). El problema es el siguiente:

- tras borrar de la tabla actualiza, se dispara el *trigger* anterior.
- la ejecución del *trigger* provoca un borrado en la tabla fuente.
- todo borrado en la tabla fuente provoca un "delete cascade" en la tabla actualiza.
- ese "delete cascade" dispara de nuevo el *trigger* y, a pesar de que no hay ninguna web que borrar, Oracle entra y de nuevo provoca un "delete cascade" en la tabla actualiza.

Para superar esta limitación de Oracle, se ha implementado un disparador menos elegante pero que funciona correctamente:

```
CREATE OR REPLACE TRIGGER T_actualiza
AFTER DELETE ON actualiza
DECLARE
 obsoletas   INTEGER := 0;
BEGIN
  SELECT COUNT(*) INTO obsoletas
  FROM fuente WHERE web NOT IN (SELECT web
                                FROM actualiza);
  IF obsoletas > 0 THEN
     DELETE FROM fuente
     WHERE web NOT IN (SELECT web
                       FROM actualiza);
  END IF;
END T_actualiza;
```

Este mismo problema apareció en el *trigger* T_alfanumerico, entre las tablas dominio y alfanumérico. La solución adoptada fue la misma.

2.5.3 DESCRIPCIÓN DE LOS PROCEDIMIENTOS IMPLEMENTADOS

Todas las operaciones que ofrece este sistema se han implementado mediante procedimientos que han sido empaquetados para poder ser asignados a cada perfil. Como se dijo en la especificación de requisitos, el sistema podrá dar cobertura a cuatro perfiles bien delimitados: el editor de conocimiento, los dos agentes software (agente síncrono y agente asíncrono) y un usuario estándar. Cada uno de ellos sólo tiene acceso a las operaciones asignadas a su perfil.

Tanto el editor de conocimiento como el usuario estándar acceden al sistema S.I.B.O. a través de un interfaz diferente en el cual están las operaciones disponibles para cada tipo de usuario. Llegado este punto hay que recordar que la funcionalidad del usuario estándar está implementada pero no el interfaz correspondiente a este perfil. Del mismo modo, la funcionalidad de los agentes también queda implementada pero no los agentes que serán añadidos en proyectos futuros.

Agentes software

Los procedimientos implementados para los agentes software son los siguientes y están agrupados en el paquete “agentes”:

- **pr_caducadas1 y pr_caducadas2**

Comprueban en las tablas ‘tiene1’ y ‘tiene2’, respectivamente, si existen valores caducados para un atributo (consulta el sistema de información).

- **pr_modifpeso**

Permite actualizar el valor del atributo peso de la tabla ‘actualiza’ para una pareja atributo-web (actualiza el conocimiento ontológico).

- **pr_fuente**

Obtiene las páginas web que actualizan a un atributo (consulta el sistema de información).

- **pr_modiftiene1 y pr_modiftiene2**

Permiten actualizar de las tablas ‘tiene1’ y ‘tiene2’, respectivamente, el atributo valor, así como la fecha de inicio y fin del nuevo valor asignado (actualiza el sistema de información).

- **pr_inserttiene1 y pr_inserttiene2**

Permiten insertar tuplas nuevas para atributos existentes (actualiza el sistema de información).

- **pr_consulta1**

Obtiene el valor de un atributo en un espacio determinado (consulta el sistema de información).

- **pr_consulta2**

Obtiene el valor de un atributo, almacenado sin espacio (consulta el sistema de información).

- **pr_qtermino**

Obtiene los términos que referencian a un concepto (consulta ontológica).

- **pr_ambito**

Obtiene el ámbito de un concepto (véase anexo E), es decir, obtiene el “padre” de un concepto. Ejecutando “n” veces este procedimiento se obtiene el concepto “padre” de una jerarquía. Todos los descendientes del concepto “padre” pertenecen al ámbito de la raíz de una jerarquía (consulta ontológica).

- **pr_contexto**

Obtiene el contexto de un concepto (véase explicación en el anexo E), es decir, obtiene el ámbito del concepto: el “padre”, y todos los conceptos pertenecientes a ese ámbito: sus “hermanos” (consulta ontológica).

- **pr_qconcepto0**

Obtiene la lista de conceptos referenciados por un término (consulta ontológica).

- **pr_qconcepto1**

Obtiene el concepto referenciado por un término dentro de un contexto introducido para afinar el resultado. El contexto será otro término, que puede referenciar uno o más conceptos (contextos reales). Para calcular qué concepto se aproxima mejor a una pareja término-contexto se ha diseñado una heurística. El cálculo de la distancia semántica entre dos conceptos es básico para que funcionen eficientemente los procedimientos (qconcepto1 y qconcepto2). Para este proyecto, el cálculo para hallar la "distancia semántica" se ha simplificado a aquellos conceptos relacionados a través de relaciones 'es un'. Sin embargo, cabe resaltar la escalabilidad de dicha función ya que para el cálculo se puede ir añadiendo el resto de relaciones existentes y aquellas que se vayan introduciendo posteriormente a la ontología. Simplemente habría que asignar a cada relación un peso, para que al hallar la distancia, cada tipo de relación tuviera una importancia real respecto a la información semántica que aporta al relacionar dos conceptos. Es decir, una relación 'es un' aporta mucha más información semántica que una relación de 'antonimia', por lo tanto, los conceptos unidos por dicha relación tienen una mayor distancia que los conceptos unidos por la relación 'es un'.

En este proyecto, la heurística se ha implementado del siguiente modo: se asume que la distancia real "física" entre dos conceptos, a través de relaciones 'es un', se aproxima a la distancia semántica o importancia de un concepto dentro de un contexto, es decir, la distancia entre dos conceptos colocados jerárquicamente es el número de saltos de un concepto a otro. Adoptando esta heurística, se producirían muchas "colisiones", muchos empates entre conceptos candidatos. Para mejorar la calidad del cálculo y ofrecer un resultado más afinado, la distancia final utilizada se ajusta en el procedimiento qconcepto1 y será el producto de la distancia "física" por un valor que se ha denominado 'encaje', donde encaje representa la "distancia semántica" entre ambos conceptos.

Distancia = distancia física * encaje

Encaje = (1- media grados)

Media grados = ((grado termino + grado contexto)/2)/100

A mayor encaje, mayor será la distancia total y menos posibilidades de ser el concepto deseado. Por el contrario, a menor encaje, menor será la distancia total y mayor posibilidad de ser el concepto buscado. La media de grados se corresponde con la media aritmética del valor del atributo grado para el término introducido y sus conceptos candidatos referenciados, junto con el valor del atributo grado para el contexto

introducido (término) con sus conceptos referenciados. Como se observa, el encaje es inversamente proporcional a la media aritmética de los grados de los términos implicados; de este modo, aquellos conceptos que sean referenciados por un término con un alto porcentaje de calidad generarán una media alta y por lo tanto el encaje será bajo, lo que implicará que su aportación a la distancia total será mínima, lo que se traducirá en que, en caso de "empate" de distancias físicas, un encaje pequeño (una calidad buena de referencia entre término y concepto) inclinará la balanza hacia uno u otro lado.

- **pr_qconcepto2**

Obtiene el concepto que más se aproxima a un término, dada una lista de contextos para afinar la búsqueda. Dicha lista de contextos son términos, donde cada término puede referenciar un concepto diferente, o todos al mismo.

Editor de Conocimiento

Los procedimientos implementados para el editor de conocimiento están agrupados en el paquete "editor". El editor de conocimiento sólo podrá acceder a la ontología a través del interfaz que le ofrece las operaciones contenidas en el paquete "editor":

Procedimientos de inserción

- **pr_concp_termino**

Inserta un nuevo concepto en la ontología junto con un término que lo referencia. Inserta en las tablas concepto, término y referencia.

- **pr_insertalfanumerico**

Inserta un dominio alfanumérico. Esto implica insertar una tupla en la tabla dominio y otra en la tabla alfanumérico.

- **pr_insertnumérico**

Inserta un dominio numérico. Esto implica insertar una tupla en la tabla dominio y otra en la tabla numérico.

- **pr_fuenteactualiza**

Inserta una fuente nueva para un atributo existente. Inserta en la tablas fuente y actualiza.

- **pr_insertatributo**

Inserta un atributo para un dominio y un concepto existente.

- **pr_insertespacio**

Inserta un nuevo espacio para un concepto existente.

- **pr_insertrelacion**

Inserta una nueva relación.

- **pr_relaciona**

Relaciona dos conceptos a través de una relación.

- **pr_insertttermino**

Inserta nuevos términos que referencian a conceptos que ya existen. Inserta en las tablas término y referencia.

- **pr_inserttiene1**

Inserta una tupla en la tabla tiene1.

- **pr_inserttiene2**

Inserta una tupla en la tabla tiene2.

Procedimientos de borrado

- **pr_borraractualiza**

Borra una tupla de la tabla actualiza.

- **pr_borraratributo**

Borra una tupla de la tabla atributo.

- **pr_borrarconcepto**

Es el procedimiento más importante de todos. Se encarga de gestionar un borrado en la tabla concepto y de mantener la consistencia de la ontología. Antes de borrar un concepto comprueba si se trata de un dominio alfanumérico utilizado por algún atributo. En caso afirmativo aborta el borrado; en caso negativo, borra el concepto deseado.

- **pr_borrardominio**

Borra una tupla de la tabla dominio.

- **pr_borrarespacio**

Borra una tupla de la tabla espacio.

- **pr_borrarfuente**

Borra una tupla de la tabla fuente.

- **pr_borrarrerferencia**

Borra una tupla de la tabla referencia.

- **pr_borrarrelacion**

Borra una tupla de la tabla relación.

- **pr_borrarrelaciona**

Borra una tupla de la tabla relaciona.

- **pr_borrartermino**

Borra una tupla de la tabla término.

- **pr_borrartiene1**

Borra una tupla de la tabla tiene1.

- **pr_borrartiene2**

Borra una tupla de la tabla tiene2.

Usuario Estándar

Los procedimientos asignados a un posible usuario estándar son, básicamente, los mismos del paquete de los agentes y por ese motivo no se explican ni se incluye su código en este documento.

Explicación del funcionamiento e implementación de los procedimientos "qconcepto0", "qconcepto1" y "qconcepto2"

- Procedimiento qconcepto0

Recibe como entrada un término y devuelve una lista de todos los conceptos referenciados por dicho término.

```
PROCEDURE pr_qconcepto0 (p_termino        IN termino.cod_termino%TYPE,
                         p_tablaout       OUT ttabla5,
                         p_numentradas IN OUT BINARY_INTEGER) IS
```

Recorre la tabla referencia, donde se encuentran todas las parejas término-concepto y devuelve todos aquellos conceptos referenciados por el término introducido.

```
BEGIN
  -- se inicializa el índice de la tabla.
  p_numentradas := 0;

  FOR v_concept IN (SELECT cod_concepto,grado
                    FROM referencia
                    WHERE (cod_termino = p_termino)) LOOP

     -- guarda la fila obtenida en la tabla auxiliar.
     p_numentradas := p_numentradas + 1;
     p_tablaout(p_numentradas).cod_concepto := v_concept.cod_concepto;
     p_tablaout(p_numentradas).grado := v_concept.grado;
  END LOOP;
END pr_qconcepto0;
```

- Procedimiento qconcepto1

Recibe como entrada un término del que se quiere conocer los conceptos que referencia y otro término, que representa un contexto, para afinar en la búsqueda de dicho concepto. Devuelve como salida el/los concepto/s que mejor es/son referenciado/s por el término en el contexto deseado.

```
PROCEDURE pr_qconcepto1 (p_termino      IN termino.cod_termino%TYPE,
                         p_contexto     IN termino.cod_termino%TYPE,
                         p_tablaresult  IN OUT ttabla8,
                         p_numentradas  IN OUT BINARY_INTEGER) IS
```

En primer lugar comprueba, llamando a qconcepto0, si el término introducido referencia un único concepto o más de uno. Si sólo referencia uno, el cálculo está acabado. Si referencia más de uno, obtiene todos los conceptos referenciados por el término que entra como segundo parámetro. Para cada concepto obtenido del primer término, y para cada concepto obtenido del segundo término, se calcula, con la ayuda del procedimiento distancia, la distancia "física" que existe entre cada pareja de conceptos. Como ya se ha explicado anteriormente, este valor de la distancia pudiera no ser suficiente y por eso en el procedimiento qconcepto1, se calcula la distancia total = distancia*encaje.

Una vez que se tienen las distancias totales de todas las posibles combinaciones concepto 1 (primer parámetro) y concepto 2 (segundo parámetro), se devuelve como concepto resultado aquel cuya distancia total respecto al contexto introducido sea menor. Si hay varios conceptos en esta situación, devuelve a todos ellos como resultado.

- Procedimiento distancia

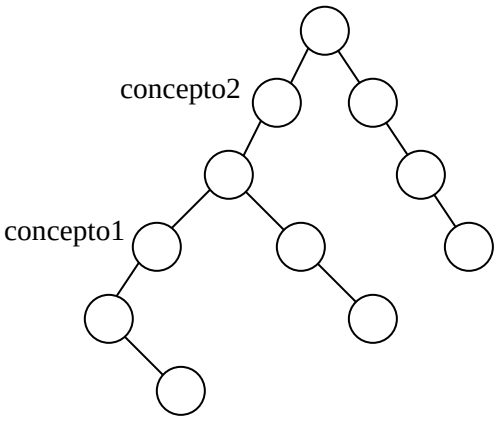
Recibe como entrada dos conceptos y devuelve la distancia "física" entre ellos, es decir, el número de saltos que hay que dar en una jerarquía para llegar de un concepto a otro.

```
PROCEDURE pr_distancia (p_concepto1   IN concepto.cod_concepto%TYPE,
                        p_concepto2   IN concepto.cod_concepto%TYPE,
                        p_dist       OUT INTEGER) IS
```

Como ya se ha explicado anteriormente, el algoritmo del cálculo de la distancia se basa en relaciones 'es un', es decir, en relaciones jerárquicas entre conceptos. Si son conceptos diferentes, procesa el cálculo, si no la distancia es cero. El procesamiento de la distancia se

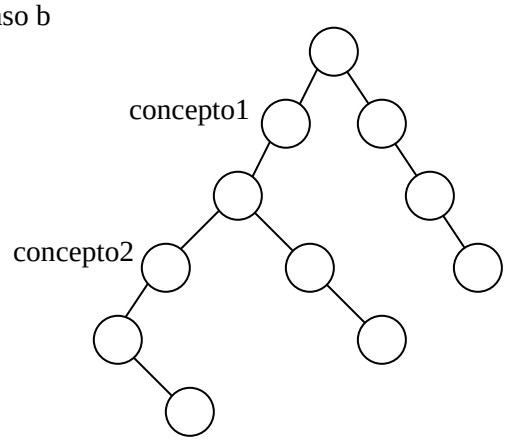
divide en dos bucles “while”, uno para cada concepto. Se pueden dar varios casos que se detallan a continuación:

- Caso a



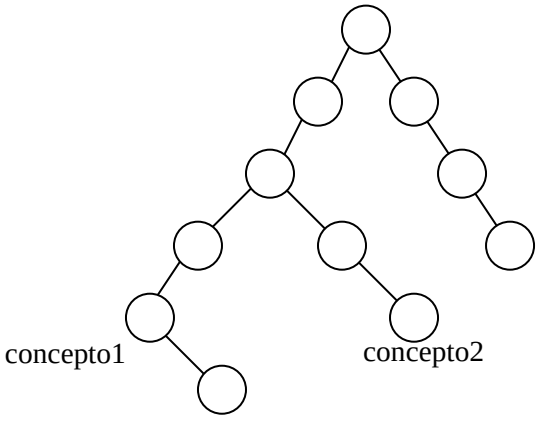
En este caso la distancia sería igual a dos. Número de saltos de un concepto a otro.

- Caso b



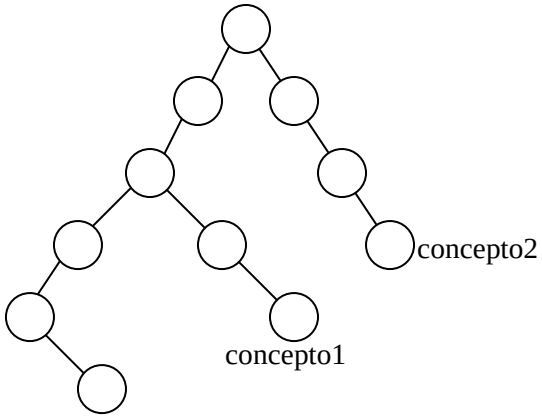
Al igual que en el caso anterior la distancia sería igual a dos.

- Caso c



En situaciones así, hay que ver cuál es el primer nodo común y sumar la distancia desde ese nodo a cada concepto. En este caso la distancia sería igual a cuatro.

- Caso d



Es un caso particular del ejemplo anterior, donde el único nodo común es la raíz del árbol. En este caso la distancia sería igual a 7.

- Caso e

Por último, se puede dar el caso en el que dos conceptos estén en jerarquías inconexas. La distancia en situaciones así es infinita y se ha implementado asignando a la distancia el símbolo NULL.

El procesamiento de la distancia se divide en dos bloques, uno para cada concepto. El bucle del concepto 1 se basa en ir ascendiendo por la jerarquía hasta que se llega a la raíz o hasta que se encuentre con el concepto 2. En cada iteración, en cada salto hacia un nivel superior, se suma uno al contador de distancia. Si cuando se acaba de procesar el concepto 1 todavía no se tiene la distancia, se entra en el bucle del concepto 2. El bucle del concepto 2 se basa en ir ascendiendo por la jerarquía hasta la raíz o hasta que se encuentre con el concepto 1 o con un concepto común a ambos. Una vez finalizados los dos bucles, si aún no se tiene calculada la distancia y no tienen "antepasados" comunes, significa que pertenecen a jerarquías inconexas. El procedimiento distancia se ayuda de otros dos procedimientos: "padre" y "pasado". El procedimiento 'padre' devuelve el concepto padre de un concepto introducido como parámetro de entrada. El procedimiento 'pasado' es capaz de verificar si un "antepasado" del concepto 2 es "antepasado" del concepto 1.

- <u>Procedimiento qconcepto2</u>

Recibe como entrada un término del que se quiere conocer los conceptos que referencia, y como segundo parámetro una lista de términos, que representa uno o varios contextos, para facilitar la búsqueda del concepto deseado.

```
PROCEDURE pr_qconcepto2 (p_termino        IN termino.cod_termino%TYPE,
                         p_tablacontexto  IN ttabla9,
                         p_numcontextos   IN OUT INTEGER,
                         p_tablaresult    OUT ttabla8,
                         p_numentradas    IN OUT BINARY_INTEGER) IS
```

El procedimiento qconcepto2 funciona de forma similar al qconcepto1. Primero comprueba si el término introducido referencia a más de un concepto. En caso negativo, el procedimiento acaba. Si referencia más de un concepto, se llama a qconcepto1 para cada pareja término (primer parámetro) y contexto (un elemento de la lista que forma el segundo parámetro). Una vez ha calculado la distancia para cada pareja término-contexto, al igual que en el procedimiento qconcepto1, se devuelve como resultado aquel concepto o conceptos cuya distancia es la mínima respecto al contexto introducido.

2.5.4 OTROS ELEMENTOS IMPLEMENTADOS

Implementación de un job

Con el fin de mantener la coherencia con las especificaciones y el diseño, así como tener un sistema de información útil y consistente, se ha implementado un job que cada día se encarga de eliminar, si existen, valores caducados para un atributo en las tablas ‘tiene1’ y ‘tiene2’.

```
DECLARE v integer;
BEGIN
  DBMS_JOB.SUBMIT(v,'paqueteagentes.pr_Deletecaducadas;',
                  SYSDATE,'SYSDATE +1',true);
END;
/
```

Implementación de una secuencia

Se ha creado una secuencia que es utilizada por el interfaz del editor de conocimiento para asignar códigos de concepto consecutivos a cada nuevo concepto que se introduce en la ontología.

2.6 FASE DE PRUEBAS

Explicación práctica de un ejemplo para las funciones "qconceptoX"[1]

Antes de leer esta explicación, se recomienda al lector ver el anexo B donde se da una visión global del significado de este ejemplo.

Para explicar el funcionamiento de las funciones qconcepto0, qconcepto1 y qconcepto2, a continuación se muestra un ejemplo, con diferentes llamadas a dichas funciones, los resultados obtenidos en cada llamada y el motivo de los distintos resultados. Como lo importante para entender estas funciones está en los diferentes términos que referencian un concepto, a continuación, partiendo del ejemplo de la jerarquía de vehículos, marítimo, etc., se describen todos los términos y conceptos relevantes para esta explicación.

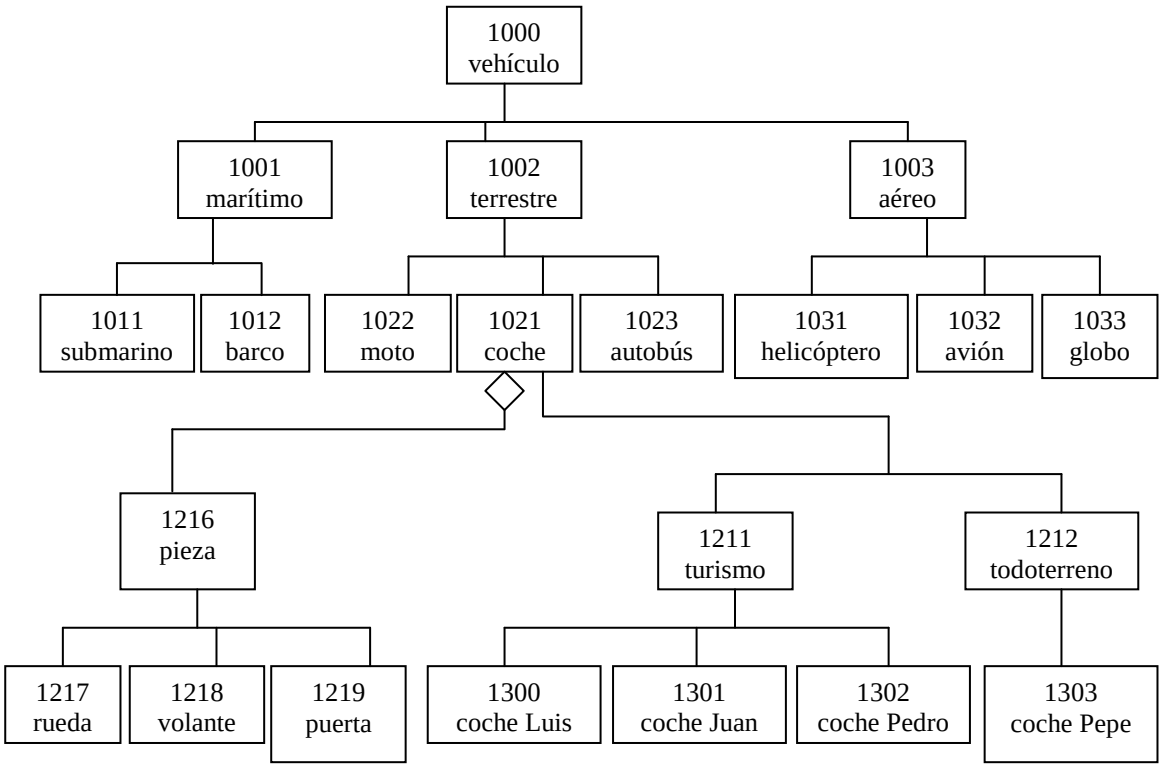


Figura 2.8

[1] X= 0,1,2

Para dicha explicación, sólo se va a trabajar con la rama del concepto 1002 y 1003. Las consultas, para ver los diferentes resultados en función de pequeñas variantes en los parámetros de entrada, se harán con los conceptos 1023 y 1032.

Como se ha podido ver en el anexo señalado anteriormente, cada uno de estos conceptos son referenciados por 1..N términos y con un porcentaje de calidad en cada caso. Para un buen funcionamiento de dichas funciones la información almacenada en la ontología tiene que ser fiel al contexto sobre el que se quiere tener conocimiento. De este modo, valores mal afinados de la calidad de referencia entre un término y un concepto, cambian totalmente el resultado dado por las funciones qconceptoX. El conocimiento almacenado en la ontología para este caso es:

- Concepto 1002 es referenciado por los términos “terrestre” (100% de calidad), “terrestrial” (90%) y “no acuático” (90%).
- Concepto 1003 es referenciado por los términos “aéreo” (100%), “volador” (80%), “airy” (70%), “flying” (90%) y “no acuático” (90%).
- Concepto 1023 es referenciado por los términos “autobús” (100%) y “transporte público” (95%).
- Concepto 1032 es referenciado por los términos “avión” (100%), “avioneta” (80%), “aircraft” (80%), “plane” (80%) y “transporte público” (90%).

Una vez que se conoce el conocimiento almacenado en la ontología, se muestra un conjunto de llamadas a las funciones tratadas.

- **qconcepto0**: recibe como entrada un término y devuelve todos los conceptos referenciados por dicho término y la calidad de cada respuesta.
    - ejemplo1:

      qconcepto0(volador) => resultado: 1003 (calidad 80%)

    - ejemplo2

      qconcepto0(no acuático) => resultado: 1002 (90%), 1003 (90%)

- **qconcepto1**: recibe como entrada dos términos, el primero es aquel del que se desea conocer qué conceptos referencia y el segundo representa un contexto (una rama de la jerarquía). Devuelve el concepto que más se acerca al término introducido.
    - ejemplo1:

      qconcepto1(autobús, terrestre) => resultado: 1023

      Cuando el término que se introduce en primer lugar, sólo referencia un único concepto, el procedimiento no realiza ningún cálculo con el contexto introducido (segundo término que se introduce).

    - ejemplo 2:

      qconcepto1(transporte público, aéreo) => resultado: 1032

      Cuando el término que se introduce en primer lugar referencia dos o más conceptos, hay que realizar un cálculo con el contexto introducido (segundo término) para intentar devolver la respuesta que espera el usuario. El término “transporte público” referencia tanto el concepto 1023 como el 1032. Únicamente con esa información no se podría ajustar más la búsqueda del concepto deseado, pero al introducir un segundo término que identifica el concepto 1003 (contexto aéreo), se calcula cuál de los conceptos candidatos está “más cerca” o mejor se ajusta a la petición. En este caso, es evidente viendo la jerarquía, que la mejor respuesta es el concepto 1032 por ser el que más cerca física y semánticamente se encuentra.

    - ejemplo 3:

      qconcepto1(transporte público, no acuático) => resultado: 1023

      En esta llamada se ve la gran potencia de la ontología implementada. El término “transporte público” referencia los conceptos 1023 y 1032. Por lo tanto, para dar un resultado hay que utilizar el segundo término. Sin embargo, en este caso, el segundo término “no acuático” también representa dos conceptos 1002 y 1003. El funcionamiento del procedimiento qconcepto1 en estas situaciones es el siguiente: calcula la distancia entre cada uno de los conceptos candidatos y cada contexto, y da como resultado aquel concepto que tiene menor distancia con uno de los contextos candidatos. Hay casos, como ya se ha visto en la explicación del procedimiento “distancia”, en que la distancia física entre conceptos candidatos puede ser la misma. En este ejemplo ha ocurrido eso, es decir, el concepto

candidato 1023 está a una distancia física = 1 respecto al contexto (concepto 1002). Del mismo modo, el otro concepto candidato 1032, también está a una distancia física = 1 respecto al contexto (concepto 1003). El buen diseño de la función heurística definida, junto con la calidad de los datos almacenados en la ontología ha sido capaz de superar este empate y dar como resultado el concepto esperado. Como se ha señalado anteriormente, el término "transporte público" referencia el concepto 1023 con una calidad del 95%, y el concepto 1032 con una calidad del 90%. Esos valores del atributo grado han sido los que han inclinado la balanza a favor del término deseado, porque cuando a una persona se le pregunta por la "idea" que tiene en su cabeza de lo que es un transporte público, en general, pensará antes en un autobús (o en el metro), que en un avión. Por el resultado obtenido en este ejemplo, se puede llegar a la conclusión de que la ontología diseñada es capaz de aportar el conocimiento que se buscaba al comienzo de este proyecto.

- **qconcepto2**: recibe como entrada dos parámetros. El primero es el término del que se desea conocer qué conceptos referencia y el segundo es una lista de términos que representan 1..N contextos (1..N ramas de la jerarquía). Devuelve el concepto que más se acerca al término introducido.
    - ejemplo1:

      lista_contextos := "no acuático";

      qconcepto2(transporte público, lista_contextos) => resultado: 1023

      Exactamente la misma explicación que el ejemplo 3 del procedimiento qconcepto1. El procedimiento qconcepto2 se limita a llamar a la función qconcepto1 para cada uno de los contextos que hay en la lista. Por lo tanto, este ejemplo es exactamente igual al ejemplo 3 explicado anteriormente, puesto que la lista sólo contiene un término.

    - ejemplo2:

      lista_contextos := "no acuático", "aereo";

      qcontexto2(transporte público, lista_contextos) => resultado: 1032

      Éste es otro de los ejemplos en los que se demuestra la potencia de la ontología. Como se ha visto en ejemplos anteriores, para el término "transporte público" hay varios conceptos candidatos. Al introducirle el contexto "no acuático" como se ve

en el ejemplo 1 del procedimiento qconcepto2, el resultado obtenido es el concepto 1023. Pero en este caso se añade una lista de términos (contextos) para ayudar al procedimiento a afinar más la búsqueda del concepto deseado. Por lo tanto, al añadir como contexto “no acuático” y “aéreo”, el concepto que más se ajusta a dichas condiciones es el 1032.

Por otro lado, se han ejecutado todas los procedimientos de inserción, modificación y borrado y funcionan correctamente en conjunción con los disparadores implementados para mantener la consistencia de la ontología. Las funciones del paquete del editor de conocimiento pueden ser fácilmente probadas y ejecutadas desde el interfaz que usará este perfil.

CAPÍTULO 3

Conclusiones y líneas futuras

1. Sistema Completo
    - 1.1 Introducción: agentes web
    - 1.2 Propósito del sistema completo
    - 1.3 Arquitectura del sistema
    - 1.4 Líneas futuras
2. Conclusiones y tendencias actuales

3.1 SISTEMA COMPLETO

El objeto de este apartado no es describir de nuevo el sistema o sus necesidades de almacenamiento sino el de realizar una breve descripción del sistema completo donde se va a utilizar la ontología, para que el lector termine con una idea global del sistema desarrollado, ya que este proyecto no tiene sentido si se entiende como una ontología aislada.

3.1.1 INTRODUCCIÓN: AGENTES WEB

Un agente, en general, es "alguien" que te ayuda a realizar determinadas operaciones. En la vida real existen agentes de bolsa, agentes de viaje, etc.

La gran telaraña mundial que es internet no para de crecer y cada día ofrece servicios nuevos más completos pero más complejos de utilizar. Por esta razón se hace necesario, en algunos casos, la ayuda de determinados agentes (web) que nos faciliten las distintas tareas (consultas, búsquedas, etc). A una persona le puede resultar más sencillo especificar el objetivo a alcanzar que conocer previamente la secuencia de pasos para llegar a dicho objetivo.

Un agente software es un programa de computadora que se comporta de manera análoga a un agente humano. Los investigadores han propuesto las siguientes características como cualidades deseables en los agentes software:

- **Autonomía**: toma la iniciativa y ejerce el control sobre sus propias acciones en la siguiente forma:
    - Orientado a objetivo: acepta requerimientos de alto nivel y decide cómo y dónde satisfacerlos.
    - Colaborativo: no obedece ciegamente los comandos debido a que puede modificar las solicitudes, realizar preguntas de clarificación, o rehusar satisfacer ciertas solicitudes.
    - Flexible: es capaz de escoger dinámicamente qué acciones invocar, y en qué secuencia, en respuesta al estado de su entorno.
    - Autoiniciable: puede sentir los cambios de su entorno y decidir cuándo actuar.
- **Continuidad temporal**: significa que está continuamente corriendo, no es un programa que transforma entradas simples en salidas simples y luego termina.

- **Personalidad**: tiene una personalidad creíble, bien definida, que facilita la interacción con usuarios humanos.
- **Habilidad de comunicación**: puede estar en comunicación compleja con otros agentes, incluyendo personas, para obtener información o conseguir ayuda para llevar a cabo sus objetivos.
- **Adaptabilidad**: automáticamente se adapta a las preferencias de su usuario, basándose en la experiencia previa. También se adapta automáticamente a los cambios en su entorno.
- **Movilidad**: puede transportarse de una máquina a otra y a través de diferentes arquitecturas y plataformas.

Un agente aporta dos **beneficios** claros:

- **Abstracción**: el agente oculta los detalles de la tecnología subyacente. La persona plantea qué información requiere y el agente determina dónde buscar la información y cómo recuperarla.
- **Distracción**: el carácter o la personalidad del agente ayuda a distraer a la persona de lo que podría ser una tarea tediosa o compleja. En los agentes software inteligentes, la abstracción ganará sobre la distracción.

En la actualidad ya existe una amplia variedad de agentes web, cada uno de ellos especializados en una tarea. A continuación se enumeran y describen los más extendidos:

- **Agentes indexados**: llevan a cabo una búsqueda autónoma, masiva en la web y almacenan un índice de palabras de títulos de documentos y de textos de documentos. El usuario puede consultar al agente preguntando por documentos que contengan ciertas palabras clave. Los agentes indexados actuales no son selectivos en su búsqueda. Por el contrario, intentan ser tan exhaustivos como sea posible. Ejemplos de tales agentes son Lycos, WebCrawler e Infoseek.
- **Guías de viaje**: son un tipo de agentes que ayuda al usuario a responder la pregunta "¿Dónde voy después?". Por ejemplo, WebWatcher aconseja interactivamente al usuario de la web que hipervínculo seguir después y aprende observando la reacción del usuario a su aviso.
- **FAQ-finders**: guía a las personas a respuestas de preguntas frecuentemente realizadas (FAQ, Frequently Asked Questions). Como las personas tienden a hacer las mismas

preguntas una y otra vez, distintas organizaciones han desarrollado archivos de preguntas, frecuentemente hechas, junto con sus respuestas.

- **Buscadores expertos**: intentan determinar lo que el usuario quiere y comprender los contenidos de los servicios de información. Un ejemplo de este tipo de agentes es el Softbot, que aprovecha que la información está estructurada, y no necesita un lenguaje natural o técnicas de recuperación de información, para "entender" la información suministrada por un servicio. En lugar de eso, el Softbot confía en un modelo del servicio para entender la semántica precisa asociada con la información suministrada por el servicio. Como resultado, el Softbot puede responder consultas con confiabilidad relativamente alta.

Los sistemas basados en agentes inteligentes precisan de los dos siguientes elementos:

- **base de conocimientos**: contiene los conocimientos relativos a la tarea. Se utilizan formalismos para representar en ella los conocimientos.
- **motor de inferencias**: es el medio por el cual se controlan y aplican los conocimientos. Los mecanismos de inferencia permiten que el sistema razone a partir de datos y conocimientos de entrada para producir los resultados de salida.

Los sistemas expertos son aquellos cuyas prestaciones (toma de decisiones, resolución de problemas complejos, transferencia de conocimiento, etc) son similares a la de los expertos humanos. Se basan en un modelo de representación de conocimiento y en la adquisición del conocimiento en un área de uno o más expertos en la misma, ayudados por ontologías y metadatos.

Con la ayuda de las ontologías los agentes web no sólo encontrarán la información de forma precisa, sino que podrán realizar inferencias automáticamente. Estos agentes de conocimiento serán capaces de interpretar los esquemas ontológicos y axiomas de diferentes dominios.

Habitualmente el agente se basa en una ontología; eventualmente la ontología se ayuda de agentes para su actualización (el caso de este proyecto). Estos agentes mantendrán la consistencia de las instancias que se inserten (en las páginas web, en bases de datos, etc). Siguiendo los esquemas ontológicos definidos, realizarán una búsqueda con inferencias

utilizando los axiomas situados en los esquemas, y podrán realizar ligaduras de los árboles taxonómicos de varias ontologías. Las relaciones semánticas que se definen entre clases de una ontología pueden considerarse predicados, que junto con la colección de reglas de inferencia, permiten construir agentes software que realicen razonamientos automáticos.

3.1.2 PROPÓSITO DEL SISTEMA COMPLETO

Desde una perspectiva global, este proyecto forma parte de otros dos proyectos fin de carrera que se están desarrollando paralelamente por sendos alumnos de Ingeniería Técnica en Informática de Gestión en la Universidad Carlos III de Madrid.

Una ontología, en sí misma, no tiene ningún sentido si no puede ofrecer sus servicios a un agente o una comunidad de agentes. Este sistema fue concebido como un proyecto de investigación de ámbito universitario y, de momento, no tiene perspectivas de aplicarse a un entorno distinto.

La funcionalidad que ofrece dicho sistema, en grandes pinceladas, es la siguiente. En la ontología se anotará conocimiento sobre un dominio, o varios, para que los agentes software puedan acceder y consultarlo. Estos agentes, inicialmente, tendrán la tarea de recibir consultas, es decir, palabras clave a partir de las cuales poder realizar una búsqueda por internet y recuperar, tanto las páginas que más se acercan a la consulta dada, como actualizar, si procede, los valores anotados en la ontología. Dicha búsqueda por internet, con la ayuda del conocimiento anotado en la ontología, tiene que ofrecer mejores resultados finales que los obtenidos en una consulta tradicional de un buscador web, como por ejemplo Google.

3.1.3 ARQUITECTURA DEL SISTEMA

A continuación se muestra la arquitectura del sistema, no sólo la ontología, sino una visión global del sistema completo (véase figura 3.1).

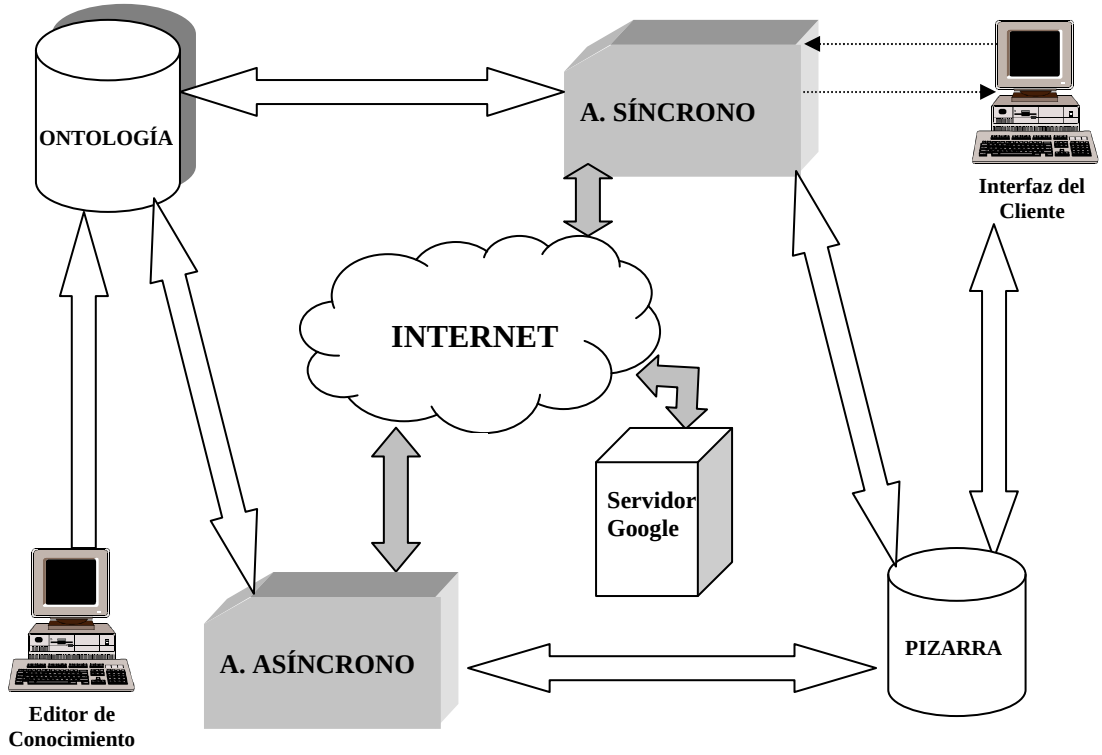


Figura 3.1 Arquitectura del sistema

El sistema, en esta primera versión, cuenta con dos agentes (síncrono y asíncrono) y con la ontología desarrollada en este proyecto.

El editor de conocimiento se limita a la gestión de la ontología (inserciones y borrados). Por su parte, los agentes se limitan a consultar la pizarra, efectuar la correspondiente búsqueda por internet y, finalmente, almacenar los resultados en la pizarra para que puedan ser consultados por el usuario que originó la consulta. La "pizarra" es el lugar donde están anotadas todas la peticiones que se han realizado, así como, los resultados generados por los agentes. Es el vehículo de comunicación entre los agentes software y los clientes.

Existen dos diferencias básicas entre ambos agentes. En primer lugar, el agente asíncrono se ayudará de Google para obtener sus resultados, mientras que el agente síncrono, únicamente, buscará los resultados en las fuentes (direcciones web) anotadas en la ontología. En segundo lugar, el agente asíncrono "despierta" y se ejecuta cada cierto tiempo, configurable en función de las necesidades, mientras que el agente síncrono se ejecuta, se "despierta" cuando recibe una petición de búsqueda. La arquitectura diseñada también permite al agente síncrono, cuando sea necesario, un comportamiento asíncrono.

3.1.4 LÍNEAS FUTURAS

La construcción del sistema SIBO, junto con el desarrollo que se está haciendo en paralelo de los agentes software, abre una interesante vía de estudio para futuros alumnos que quieran realizar su proyecto fin de carrera sobre esta apasionante e incipiente tecnología.

Desde el punto de vista del conocimiento ontológico, sería interesante que se fusionaran los agentes con la ontología para poder actualizarla en tiempo real y aplicarla a un dominio actual para poder aprovechar toda su potencia. De este modo, los valores de los atributos contendrían datos actuales, las fuentes de un atributo se renovarían continuamente y su calificación variaría en función de los resultados de cada búsqueda. La calidad de los datos introducidos por el editor de conocimiento, especialmente en la denominación de los términos y su grado de calidad, debería ser cada vez más precisa para potenciar las funciones utilizadas por los agentes, debiendo efectuar incluso, un estudio previo sobre dicho dominio para disponer del conocimiento necesario para anotar información en la ontología.

Por otro lado, el diseño de esta ontología ha sido elaborado por una única persona, por lo cual, con el trabajo de otros alumnos, tomando como base la ontología actual, se llegase a otra mucho más general, con mayores posibilidades de aportar conocimiento, y como es el fin de todo esto, tan estandarizada que pudiera ser entendida por el mayor número de agentes posible. Del mismo modo, actualmente hay un número limitado de funciones, tales como ‘qconcepto1’, ‘ámbito’, ..., cuya ampliación aportaría nuevos horizontes en el conocimiento de los agentes, en la posibilidad de aprender de la experiencia, etc.

Desde el punto de vista de la aplicación también habría varias vías de investigación. Como se ha dicho a lo largo del desarrollo de este proyecto, el interfaz para un usuario estándar, un agente humano, no ha sido implementado. Actualmente existe un conjunto de operaciones básicas disponibles, como consultas de los valores de los atributos, de las fuentes, de los ámbitos, ..., que podrían ejecutarse desde ese interfaz. Pero se podrían añadir otras muchas y conseguir, con la ayuda de los agentes, mejorar considerablemente los resultados de una búsqueda “Google”. Se podría diseñar el interfaz del usuario estándar como una página web, similar a “Google”, de la que se obtendrían resultados mucho más cercanos a la necesidad de quien formuló la consulta.

3.2 CONCLUSIONES Y TENDENCIAS ACTUALES

El uso de ontologías para dotar de semántica a los datos todavía se encuentra limitado a dominios muy específicos. Actualmente es inabordable hacer una ontología general que abarque un gran número de dominios al mismo tiempo. Además, las ontologías son muy subjetivas y dependen mucho de su diseñador. Otra pequeña dificultad es la falta de estandarización para definir ontologías con un único lenguaje.

La motivación del uso de ontologías se centra en dos puntos básicos: intentar dar semántica a la información, debido a la evolución que están sufriendo los sistemas actuales, y la necesidad de reutilización de conocimiento. Teniendo un entendimiento compartido que unifique los diferentes puntos de vista de cada desarrollador (suposiciones, contextos, etc.) facilitará:

- la comunicación (entre máquinas, entre desarrolladores, etc.).
- la interoperabilidad entre sistemas.
- la reutilización.
- la confiabilidad.
- la especificación.

Algunos de los principales beneficios que aportan las ontologías son [2]:

- proporcionan una forma de representar y compartir el conocimiento utilizando un vocabulario (terminología) común.
- permiten usar un formato de intercambio de conocimiento.
- proporcionan un protocolo específico de comunicación.
- permiten una reutilización del conocimiento.

Expectativas / Tendencias Actuales

1. Se aprecia una progresiva sustitución de los modelos convencionales del proceso de recuperación de la información por modelos cognitivos que aplican el modelo humano de memoria y aprendizaje a los agentes inteligentes de recuperación de la información, proponiendo las ontologías para estructurar las bases del conocimiento [1].

2. Los servicios de ontologías ofrecerán un marco contextualmente rico y moderno para elaborar modelos, prestar servicios y gestionar la terminología en cualquier materia. Cuando se integre con instrumentos de búsqueda basados en internet, facilitará mucho la recuperación de recursos, no sólo proporcionando acceso a los documentos específicos que una determinada persona está buscando, sino también ofreciendo sugerencias relativas a otros recursos conexos que son potencialmente pertinentes para el tema de interés. Esta funcionalidad adicional no sólo aumentará espectacularmente el alcance de los motores de búsqueda basados en internet, sino también revolucionará la forma en que los usuarios interactuarán con la web.
3. Las ontologías, al describir los conceptos y sus relaciones, pueden ser empleadas de manera más general que los tradicionales *thesauros* pudiendo establecer un mayor número de relaciones distintas.
4. Respecto a la web semántica, ésta proporcionará un salto cualitativo sobre el potencial en la web. Las principales ventajas de esta nueva revolución en internet serán:
    - El desarrollo de aplicaciones con esquemas de grafos comunes.
    - El fomento de las transacciones entre empresas por comercio electrónico.
    - La búsqueda de información con inferencias.

¿Adónde se encamina esta tecnología?

Actualmente hay muchos caminos hacia los que se puede orientar el uso de una ontología. A continuación, a modo de ejemplo, se enumeran algunas de las posibles aplicaciones futuras de esta tecnología:

- Desarrollo de ontologías como soporte de interlingua e interoperabilidad entre herramientas en algún dominio (traductores e integración).
- Desarrollo de herramientas para apoyar el diseño y evaluación de ontologías.
- Desarrollo de bibliotecas de ontologías.
- Desarrollo e integración de nuevas ontologías.
- Metodologías de diseño y evaluación de ontologías.

Como se puede ver, va a ser muy importante que haya herramientas que estandaricen la construcción de ontologías para que, tras haber sido organizadas en bibliotecas, puedan ser utilizadas e integradas fácilmente en otros proyectos.

CAPÍTULO 4

Bibliografía

1. Bibliografía consultada
2. Referencias bibliográficas

4.1 BIBLIOGRAFÍA CONSULTADA

- **ONTOLOGÍAS**
  - **Estudio de ontologías para representación de contenidos** [Lidia Silvia Muñoz. Porto Alegre, Septiembre 2002]
  - **Ingeniería del conocimiento** [A. Gómez et al., Editorial Centro de Estudios Ramón Areces S.A]
  - **Ontologías** [A. Gómez-Pérez. Universidad Politécnica de Madrid, 2001]
  - **Ontologies: State of the art** [D. Sabater, 2000]
  - http://www.iua.upf.es/~jblat/material/doctorat/students/jccbis/Ontologias.htm
  - http://www.info-ab.uclm.es/asignaturas/42551/trabajos-anteriores/ /TrabajoOntologias.pdf
- **DOCUMENTO ESPECIFICACIÓN DE REQUISITOS**
  - **IEEE Recommended Practice for Software Requirements Specification ANSI/IEEE 830** [1998]
  - **Modelo de proceso software IEEE 1074-1991**
- **PL/SQL (ORACLE)**
  - **Oracle9i PL/SQL Programming** [Scott Urman, 2002]
  - **Oracle7 Server Language Quick Reference** [Eric Armstrong,1992]
- **JAVA**
  - http://www.java.sun.com
  - Tutoriales de la aplicación jDeveloper

4.2 REFERENCIAS BIBLIOGRÁFICAS

Nº Ref.	Fuente	Autor/es
1	http://ttt.upv.es/ccarrasc/doc/2001-2002/ Ontologias/ /inicio.htm	Laura Grela, Elena Sauri, Alicia Sellés
2	http://www.info-ab.uclm.es/asignaturas/42551/trabajos-anteriores/Trabajo-Ontologias.pdf	Fco. Javier Honrubia López
3	http://www.ksl.stanford.edu/	
4	http://webonto.open.ac.uk/ http://kmi.open.ac.uk/projects/webonto/	
5	http://smi-web.stanford.edu/projects/prot-nt/	
6	http://www.isi.edu/isd/ontosaurus.html	
7	http://webode.dia.fi.upm.es/	
8	http://www.swi.psy.uva.nl/projects/kads22	
9	http://apollo.open.ac.uk/	
10	http://www.automatas.org/schneider/flink_7.htm http://www.landc.be/	
11	http://oiled.man.ac.uk/building/	
12	http://www.ontoprise.de/ http://www.ontoprise.de/com/start_downlo.htm	
13	http://www.topthing.com/openknome.html	
14	http://protege.stanford.edu/	
15	http://www.symontox.org/ http://www.culture.gouv.fr/culture/mrt/numerisation/fr/ /eeurope/documents/SymOntoX.ppt	
16	**PFC** Estudio y desarrollo de una arquitectura para la creación de portales semánticos (uc3m)	Silvestre Losada Alonso [Universidad Carlos III]
17	http://www.sema.es/SP/mkbeem.htm	
18	MESIA: Modelo computacional para extracción selectiva de información de textos cortos. XVI Congreso de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN 2000). Vigo, 26 al 28 de Septiembre de 2000 http://www.sepln.org/revistaSEPLN/revista/27/27-demostracion1.pdf	Paloma Martínez [Universidad CarlosIII] Paloma Gonzalez [Universidad Politécnica] Pablo Sánchez [Universidad Politécnica] Ana García-Serrano [Universidad Politécnica]
19	El desarrollo de una ontología a base de conocimiento enciclopédico parcialmente estructurado. Actas de las II Jornadas de Tratamiento y Recuperación de la Información (Jotri 2003). Universidad Carlos III de Madrid, Leganés. http://www.fundacion.uc3m.es/jotri2003/ ('dentro de ediciones previas')	Rafael Marín, Begoña Martínez, Josep M. Merenciano, David Miramón, GemaPérez, Lluís Valentín
20	Un sistema de presentación dinámica hipermedia para representaciones personalizadas del conocimiento. II Congreso Internacional de Interacción Persona-Ordenador (Interacción 2001). Salamanca (Spain), May 2001. http://www.ii.uam.es/~castells/publications/interaccion01.pdf	Pablo Castells, José Antonio Macías [Universidad Autónoma de Madrid]
21	Generación Automática de una Base de Datos desde Documentos de la Web. Congreso Argentino de Ciencias de la Computación, CACIC 2000, Ushuahia, October 2000.	Jaime Ferreiro, Regina Motz, Fernando Perelló, Dina Wonsever, [Universidad de la República, Montevideo, -Uruguay-]

Nº Ref.	Fuente	Autor/es
22	KeyConcept: A Conceptual Search Engine. Information and Telecommunication Technology Center. Technical Report: ITTC-FY2004-TR-8646-37. http://www.ittc.ku.edu/keyconcept/	Juan Manuel Madrid Molina [Univ. Icesi-I2T –Colombia-] Susan Gauch [Univ. de Kansas –USA-]
23	Estudo de ontologías para representação de conteídos de engino baseado na www	Lydia Silva Muñoz – Porto Alegre, Septiembre 2002
24	Análisis y diseño estructurado y orientado a objetos de sistemas informáticos [McGraw-Hill, 2003]	Antonio de Amescua Seco, Juan José Cuadrado Gallego, Emilio Ernica Lafuente, Javier García Gumán, Luis García Sánchez, Paloma Martínez Fernández, Mª Isabel Sánchez Segura

NOTA: El alto grado de movilidad existente en internet puede tener como resultado que algunas de estas direcciones dejen de existir y que aparezcan otras nuevas; no puedo, por tanto, hacerme responsable del contenido de las mismas.

CAPÍTULO 5

Anexos

a. Documento de especificación de requisitos
b. Datos de un ejemplo completo
c. Manual de la aplicación
d. Importancia de la evaluación de los sistemas basados en conocimiento
e. Glosario
f. Código SQL

ANEXO A

Documento de Especificación de Requisitos Software (E.R.S.)

Índice Relativo al E.R.S.

1. INTRODUCCIÓN

Este documento es una Especificación de Requisitos Software (ERS) para el Sistema de Información Basado en Ontologías (SIBO). En él, se encuentran las especificaciones funcionales del sistema, es decir, todo lo necesario para construir el software. Además, contiene una descripción exacta de todos los pasos previos, muy importantes en el proyecto, como puede ser la especificación de necesidades para la construcción del diagrama entidad relación o las herramientas y técnicas de análisis y diseño que se utilizarán. Esta especificación se ha estructurado inspirándose en las directrices dadas por el estándar "IEEE Recommended Practice for Software Requirements Specification ANSI/IEEE 830 1998".

1.1 Propósito

El objeto de la especificación es definir de manera clara y precisa el trabajo a desarrollar en este proyecto, así como todas las funcionalidades y restricciones del sistema que se desea construir. Todo su contenido ha sido elaborado en colaboración con Javier Calle, tutor del proyecto. Esta especificación está sujeta a revisiones, que se recogerán por medio de sucesivas versiones del documento, hasta alcanzar su aprobación por las distintas partes implicadas. Una vez aprobado, servirá de base al equipo de desarrollo (en este caso Ignacio Herrero) para la construcción del nuevo sistema, y será el canal de comunicación entre los miembros del proyecto.

1.2 Ámbito del sistema

El motor que impulsa el desarrollo del sistema es la evidencia de una creciente complicación y dificultad en almacenar, mantener y recuperar datos de las diferentes fuentes de información que cada día aumentan en todo el mundo.

Se ha constatado la necesidad de un sistema informático que ayude a tales labores, de forma que se garantice la fácil localización de información y una rápida puesta a disposición de los usuarios. Puede generalizarse el uso y aplicación de este sistema a cualquier tipo de información susceptible de ser clasificada.

El futuro sistema será capaz de almacenar datos, con contenido semántico, del dominio deseado y de proporcionar la información que se le solicite. Deberá minimizar la información "basura" que proporcione, típico de los buscadores basados en palabras clave. Además, será

capaz de entender, en algún grado, el “significado” de la consulta que le es requerida, y podrá recuperar sinónimos, etc.

La carga del sistema puede ser estimada como baja, sin embargo, es necesario contemplar un eventual crecimiento en el uso del sistema por el aumento de distintos usuarios.

1.3 Definiciones, acrónimos y abreviaturas

1.3.1 Definiciones

- **Entorno**: conjunto de condiciones extrínsecas que necesita un sistema informático para funcionar:
    - entorno lógico: conjunto de condiciones y elementos lógicos (agentes, herramientas lógicas, etc).
    - entorno físico: conjunto de condiciones y elementos físicos (hardware).
- **Agente**: entidad independiente presente en un entorno capaz de interactuar con el propio entorno y con otros agentes.
- **Interactuar**: ejercer acción mutua (recíproca) entre dos agentes o un agente y su entorno || 2. proceso compuesto por: a) adquisición de conocimiento; b) proceso de datos; c) consecución de una acción (o secuencia de acciones).
    a) proceso por el cual un agente percibe eventos del entorno y los interpreta, actualizando su conocimiento (base de hechos) sobre el entorno.
    b) fase inmediatamente posterior a la adquisición de información en la que los hechos obtenidos son procesados (junto con el resto de hechos almacenados) por el agente. Al final de esta fase, se pueden producir cambios en las bases de hechos del agente (incluyendo aquellos que determinen sus próximas acciones)
    c) fase última de una interacción mediante la cual el agente pretende ejercer cierta influencia en su interlocutor o, más generalmente, en su entorno.
- **Agente software**: proceso en un entorno informático, de comportamiento autónomo, dotado de inteligencia artificial || 2. La porción de código cuya ejecución

da lugar a un proceso como el descrito || 3. La instanciación de una porción de código con una base de conocimiento inicial, y que va recopilando nuevos hechos que lo diferencia de otras instanciaciones con una base de conocimiento distinta (aún cuando su base de conocimiento inicial fuese idéntica) || 4. Agente no humano.

- **Web-Crawler**: agente software cuyo entorno lo componen uno o más usuarios y todos los ficheros accesibles en internet.
- **Usuario**: cualquier agente que se dirige a otro con el propósito de solicitar un servicio.
- **Servicio (cliente-servidor)**: funcionalidad que ofrece un servidor a un cliente; servidor: máquina o programa que atiende la petición de un cliente; cliente: aplicación que utiliza un usuario final cuando accede a un servicio a través de la red.
- **Ontología**: especificación explícita y formal sobre una conceptualización compartida.
- **Sistema de información**: permite, en general, almacenar, manipular y recuperar información (información que se puede convertir en conocimiento).

1.3.2 Acrónimos

ERS: Especificación de Requisitos Software.

SIBO: Sistema de Información Basado en Ontologías.

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

ANSY: American National Standars Institute (Instituto Nacional de Estándares Americanos).

ISO: International Organization for Standardization (Organización Internacional para Estandarización).

ODBC: Open DataBase Connectivity.

JDBC: Java DataBase Connectivity.

WWW: World Wide Web.

SI: Sistema de Información.

URL: Uniform Resource Locator (Localizador uniforme de recursos).

SGBD: Sistema Gestor de Base de Datos.

1.3.3 Abreviaturas

No se han definido

1.4 Referencias

IEEE Recommended Practice for Sotfware Requirements Specification. ANSI/IEEE std. 830, 1998.

Modelo de proceso software IEEE 1074-1991.

1.5 Visión general del documento

Este documento consta de 5 secciones. Esta primera sección es la introducción y proporciona una visión general del ERS. En la sección 2 se da una descripción general del sistema, con el fin de conocer las principales funciones que debe realizar, los datos asociados y los factores, restricciones, supuestos y dependencias que afectan al desarrollo, sin entrar en excesivos detalles. En la sección 3 se definen detalladamente los requisitos que debe satisfacer el sistema. En la sección 4 se realiza un pequeño análisis de la viabilidad del proyecto para determinar si es abordable. Por último, la sección 5 trata sobre las funciones de validación y verificación (pruebas).

2. DESCRIPCIÓN GENERAL

En esta sección se presenta una descripción de los pasos importantes del proyecto, además de una descripción a alto nivel del sistema. Se presentarán las principales áreas de negocio a las cuales el sistema debe dar soporte, las funciones que el sistema debe realizar, la información utilizada, las restricciones y otros factores que afecten al desarrollo del mismo.

2.1 Perspectiva del producto

Se desea diseñar una ontología que sirva de soporte para la construcción de un sistema capaz de resolver consultas sobre términos basándose en búsquedas síncronas y asíncronas sobre la world wide web. Éste será un sistema multiagente, no siendo el objeto de este proyecto el diseño e implementación de tales agentes, sino constituir la base de otros proyectos basados en la implementación de agentes que necesitan de la ontología para su realización. Se pretende construir un mecanismo de representación de conceptos que conecte semánticamente la

solicitud de información de un usuario con la búsqueda de información en sus respectivos recursos.

Se trata de diseñar e implementar una base de datos que recoja y estructure toda la información que contará con un significado para el sistema. Esta base de datos constituirá la ontología.

La definición de los conceptos quedará a cargo de un operador (editor de conocimiento), mientras que otros usuarios tendrán acceso para actualizar y consultar conocimiento. Este proyecto, por tanto, versa sobre la descripción del metaconocimiento.

El sistema, en esta primera versión, no interactuará con ningún otro sistema informático (fuera de los ya descritos).

2.2 Objetivos generales del proyecto

Para lograr el objetivo y poner en funcionamiento el sistema, se han definido los siguientes pasos a seguir:

- Definir las necesidades de información de un mecanismo de representación de conceptos genéricos.
- Describir las características del sistema desde el enfoque del almacenamiento.
- Describir las características del sistema desde el enfoque de la utilización.
- Implementar y documentar el sistema.

Como ya se ha dicho en apartados anteriores, las expectativas de uso del sistema, en esta primera fase, se prevé que serán bajas. Utilizarán el sistema dos agentes software, un editor de conocimiento y un usuario estándar de consulta. Este sistema será diseñado e implementado para que pueda satisfacer las posibilidades de crecimiento que se esperan, ya que el futuro de cualquier sistema basado en conocimiento tendrá su base en una ontología. El objetivo de este proyecto es construir una ontología lo más general posible (dentro de unos límites de esfuerzo) para que pueda ser adaptada e implantada en diferentes dominios.

2.3 Necesidades de información

El objetivo de este proyecto es el estudio, diseño e implementación de una ontología como base de conocimiento en el dominio acotado del sistema. Tras dicho estudio, se han obtenido las siguientes conclusiones:

- El sistema tiene que ser capaz de representar conceptos (y todas sus características).
- El sistema tiene que ser capaz de contemplar varios términos para cada concepto, y los matices que los hagan diferentes si se da el caso.
- Debe permitir una cantidad de instanciaciones ilimitadas para cada concepto. Cada instancia de un concepto ha de poder identificarse de forma unívoca, aun cuando sus características/atributos sean similares.
- El sistema tiene que ser capaz de representar relaciones entre conceptos. Estas relaciones también podrán ser de tipo jerárquico (clase-superclase), para poder conocer los vínculos que existen entre los conceptos, ya que aporta un mayor contenido semántico.
- La ontología debe poder recoger las restricciones definidas sobre las relaciones (axiomas).
- El sistema debe contemplar las características de un concepto, como otros conceptos cuya conjunción define el concepto primero (relación de ámbito).
- Las características deberán ser susceptibles de valuación, teniendo ese valor ciertos atributos tales como tipo, métrica, validez (espacial/temporal), etc.
- Las características deben contar con medios para su actualización, ubicación de información (fuentes), etc. La selección de las mejores fuentes en base a la experiencia es un punto a valorar.
- Los conceptos podrán tener relaciones, con otros conceptos, de ‘vecindad’, es decir, un concepto será vecino de otro si puede o suele acompañar a éste en determinados discursos del dominio referente al ámbito del concepto).

2.4 Funciones del sistema

2.4.1 Funciones de consulta

Se debe poder acceder al sistema para realizar consultas. Existen dos perfiles de consulta: consulta de la ontología y consulta del sistema de información.

El usuario estándar consulta en el sistema de información, proporcionando para ello los elementos necesarios para cada búsqueda. Entre otras cosas, se podrá consultar el ámbito o el contexto de un concepto, así como el valor de un determinado atributo.

Los agentes software consultan en la ontología (conceptos, términos, etc) para obtener la información que necesitan para realizar búsquedas en la world wide web.

2.4.2 Funciones de inserción

Como en cualquier sistema basado en bases de datos es necesaria la figura de un operador, en este caso el editor de conocimiento, que se encargue de introducir nuevos conceptos, términos, relaciones, etc., así como de actualizar el conocimiento disponible. Es el encargado de gestionar los metadatos.

2.4.3 Funciones de actualización

Parte del sistema de información debe ser habitualmente actualizado debido a la volatilidad de su contenido. Para ello, ha de permitir y facilitar el acceso al sistema de información a los agentes software, quienes actualizan los valores con información obtenida de internet.

2.5 Características del sistema desde el enfoque del almacenamiento

Por las características de permanencia y compartición de la información en este sistema, se hará necesario el uso de almacenes de información. En este caso, se requiere explícitamente que estos almacenes sean soportados por un sistema de base de datos relacional. Para ello, el primer paso será obtener un modelado del universo conceptual de la información que interviene en este sistema: elementos, relaciones entre elementos, restricciones.

El diseño de bases de datos (BD) consiste en describir la estructura de la BD de forma que se represente fielmente la parcela del mundo real que se quiere almacenar. Ello se realiza mediante un proceso de abstracción, denominado modelado, que se apoya en un modelo de datos [24]. Un modelo de datos es el instrumento que se aplica a un universo del discurso (UD) para obtener una estructura de datos que se denomina esquema de la BD. A lo largo del desarrollo de la BD se utilizan varios modelos de datos que nos permiten representar la realidad según las distintas fases de una metodología y según distintos niveles de abstracción. El modelo E/R consiste en un conjunto de conceptos, reglas y notaciones que se utiliza en la fase de

análisis de requisitos. Mediante los constructores del modelo E/R se recoge toda la semántica que puede obtenerse mediante la observación del UD. Nótese que los esquemas conceptuales no son directamente implementables, por ello no tienen ninguna connotación física. Por tanto, una vez obtenido el modelo lógico de datos, se transforma en un modelo físico de datos, modelo relacional, para que pueda serle útil al desarrollador del proyecto.

2.6 Características del sistema desde el enfoque de la utilización (características de los usuarios)

En esta sección se describen los esquemas externos necesarios para la utilización del sistema, es decir, qué tipo de usuarios tiene el sistema (y sus características), qué operaciones efectúan y qué interfaz necesita cada tipo de usuario.

- **Operador (editor de conocimiento)**: es el encargado de actualizar la ontología (conceptos, relaciones, etc.). Efectúa operaciones de creación, modificación y borrado. *Consulta y actualiza la ontología*. Por otro lado, también es el encargado de insertar los valores para cada atributo. *Actualiza el sistema de información*. Es necesario realizar el análisis y diseño de la interfaz que será usada por el editor de conocimiento, así como decidir la herramienta con la que se implementará.

- **Agentes software**

  - Agente software (síncrono)

    Es un usuario de actualización del sistema de información, actualiza los valores de las instancias y puede insertar valores nuevos para atributos existentes. Cuando llega una petición “despierta” y comienza su ejecución, buscando en la world wide web los datos necesarios. Para realizar esta búsqueda, consulta las URL fuentes correspondientes a las distintas instancias, pudiendo dar una valoración sobre el uso de las fuentes (qué fuentes han sido útiles, y cuáles no lo han sido, para añadir conocimiento a la información almacenada). Por tanto, *Consulta el sistema de información (fuentes), consulta la ontología (conceptos, términos, etc) y actualiza el S.I*.

  - Agente software (asíncrono)

    Es un usuario de consulta de la ontología (términos, conceptos, etc.) que le va a permitir realizar una búsqueda en Google con la información obtenida. *Consulta la ontología*.

También podrá actualizar el sistema de información, si el atributo que intenta valorar pertenece a la ontología, podría actualizarlo. *Actualiza el S.I*.

- **Usuario estándar**

  Persona que hará una consulta y el sistema le devolverá una respuesta. *Consulta el sistema de información*.

La descripción de los procedimientos e interfaces de los que hará uso cada clase de usuario se encuentran en el apartado 3.2.1 Requisitos interfaces externos de usuario.

2.7 Restricciones

- Los agentes accederán a la base de datos mediante el JDBC de Java.
- Los administradores utilizarán el sistema sin la ayuda de ninguna interfaz. Accederán directamente a ORACLE.
- El operador de conocimiento accederá al sistema a través de un interfaz para introducir los conceptos, relaciones, etc.
- Los agentes software sólo podrán ejecutar aquellas funciones que se le faciliten a través del paquete implementado para ellos.

2.8 Suposiciones y dependencias

2.8.1 Suposiciones

Se asume que los requisitos descritos en este documento son estables una vez que sea aprobado por las partes implicadas. Cualquier petición de cambios en la especificación debe ser aprobada por todos los interesados y gestionada por el director de proyecto.

2.8.2 Dependencias

Desde el punto de vista de la ontología, este sistema es independiente y puede ser reutilizado en todos aquellos sistemas alineados con su propósito.

Desde el punto de vista de sistema de información, este sistema SIBO, necesita comunicarse con otros sistemas externos (agentes, web-crawler), para poder disponer de una información fiable y actualizada. Del mismo modo, los agentes software comparten el conocimiento ontológico proporcionado por esta ontología, aunque cada uno de ellos dispone de una base de conocimiento propia, así pues, éstos tienen una dependencia funcional fuerte con este proyecto.

3. REQUISITOS ESPECÍFICOS

En este apartado se presentan los requisitos funcionales que deberán ser satisfechos por el sistema. Todos los requisitos aquí expuestos son esenciales, es decir, no sería aceptable un sistema que no satisfaciese alguno de los requisitos aquí presentados.

3.1 Requisitos funcionales

3.1.1 Acceso al sistema

Req(01) Cada tipo de usuario accederá al sistema a través del interfaz diseñado para su perfil o a través de las operaciones que le han sido asignadas.

Req(02) Cada tipo de usuario sólo podrá efectuar operaciones permitidas a su perfil.

3.1.2 Manejo de los metadatos (ontología)

Req(03) El editor de conocimiento debe poder insertar los datos necesarios a través del interfaz de inserción. Los datos de una inserción son conceptos, relaciones, etc.

Req(04) El editor de conocimiento ha de poder modificar la información existente en la base de datos (ontología).

Req(05) El sistema debe permitir al editor de conocimiento dar de baja todos aquellos conceptos, términos, relaciones, etc., que considere obsoletos.

Req(06) Los agentes software deben poder acceder a la ontología para realizar consultas.

3.1.3 Gestión de la información (instancias)

Req(07) El usuario estándar ha de poder realizar consultas (consultas del sistema de información). Se le proporcionará un interfaz de acceso, que no es objeto de este proyecto desarrollar, y tan sólo se implementará el paquete con las funciones necesarias.

Req(08) El agente software síncrono ha de poder consultar el sistema de información para determinar qué información (instancias) está "caducada".

Req(09) Los agentes software han de poder acceder al sistema de información para actualizarlo (crear instancias).

Req(10) Los resultados de una consulta (usuario estándar) se mostrarán por pantalla sin la posibilidad de imprimir los listados.

Req(11) El sistema debe almacenar información para conocer la utilidad y calidad de fuentes y términos.

3.2 Requisitos de interfaces externos

3.2.1 Interfaces de usuario

Req(12) Las interfaces de usuario estarán orientadas a ventanas y cuadros de texto desde donde solicitar las consultas o introducir nuevos datos.

Req(13) Debe existir un interfaz gráfico para el editor de conocimiento que le proporcione todas las operaciones de este perfil.

Req(14) Organizar en un paquete todos los procedimientos desarrollados para los agentes software y poderles ofrecer una API con las funciones disponibles.

3.2.2 Interfaces software

Req(15) Los agentes software interactuarán con el sistema SIBO a través de JDBC de Java.

Req(16) El sistema operativo utilizado será Windows 2000.

Req(17) Se utilizará SGBD Oracle9i.

3.2.3 Interfaces de comunicación

Req(18) La conexión a la red se establecerá a través de un módem, a la red LAN correspondiente. Esto será transparente para la aplicación y se utilizarán los protocolos estándares en internet.

3.3 Requisitos de rendimiento

Req(19) Sólo debe dar servicio simultáneamente a un único puesto de gestión de meta-datos (editor de conocimiento).

Req(20) Debe dar servicio a un usuario de actualización (agente software asíncrono).

Req(21) Debe dar servicio a cuatro usuarios de consulta (los dos agentes software, el editor de conocimiento y el usuario estándar)

Req(22) El tiempo de respuesta de una consulta debe ser inferior a “2 segundos”.

3.4 Requisitos de desarrollo

Req(23) El ciclo de vida elegido para desarrollar el producto será el de prototipo evolutivo, de manera que se puedan incorporar fácilmente cambios y nuevas funciones.

3.5 Requisitos tecnológicos

Req(24) La configuración mínima recomendada es la siguiente:

Procesador: Pentium III

Memoria: 128 Megabytes

Tarjeta Ethernet o Módem o Tarjeta RDSI (conexión superior a 56Kb/seg)

Sistema Operativo: Windows2000

Req(25) Sería preferible que la base de datos estuviera en la misma máquina y tuviera JDBC de Java.

3.6 Atributos

3.6.1 Seguridad

Req(26) El sistema debe cumplir el requisito de perdurabilidad de la información.

Req(27) El sistema debe cumplir el requisito de confidencialidad de la información. Cada perfil sólo tiene acceso a la parte del sistema que le está permitido.

3.6.2 Mantenibilidad

Req(28) El sistema es susceptible de ser ampliado. Por tanto deberá diseñarse fácilmente mantenible, aplicando para su desarrollo las metodologías que para ello sean precisas.

4. ANÁLISIS DE VIABILIDAD

En este apartado se estudia si la consecución de los requisitos definidos es posible.

1. La definición del problema hace necesaria la utilización de un SGBDR.
2. Debido a que el equipo de desarrollo ya conoce PL/SQL, la implementación de la ontología se hará en este lenguaje. De este modo, se acotarán los tiempos de aprendizaje y familiarización con el SGBD Oracle9i.
3. Los requisitos 26 y 27 están asegurados gracias al buen uso que se hará del SGBD.

4. Será precisa la utilización de un servidor donde se encuentre la base de datos. En este caso se trata de un servidor Apache (por ser de libre distribución) instalado en la máquina con dirección IP fija 163.117.129.179.
5. La aplicación funcionará sobre un sistema operativo Windows. No se espera la implantación en un futuro sobre otra plataforma (Linux), por lo que no se tienen que realizar esfuerzos en esa dirección.
6. Para facilitar la tarea del editor de conocimiento se implementará un interfaz con JDeveloper. La utilización de tecnología java proporciona al sistema mayor independencia y capacidad de crecimiento.
7. Toda la documentación que exige el proyecto se realizará con la herramienta Microsoft Word.
8. Puesto que java es la tecnología desde la que se accede a la base de datos se necesitará disponer del JDBC, pues aporta un conjunto de primitivas para la comunicación con una base de datos.

5. VERIFICACIÓN Y VALIDACIÓN

En este apartado se explicitarán las pruebas a que será sometido el sistema para comprobar, por un lado, su correcto funcionamiento y, por otro lado, que su funcionalidad cubre los requisitos establecidos. No sólo requisitos funcionales, sino también los requisitos de interfaces externos, de rendimiento, de desarrollo, tecnológicos, etc., que se han definido en el proyecto.

Se han de desarrollar las siguientes tareas en la etapa de verificación y validación, según el modelo de proceso software IEEE 1074-1991:

- Planificar la verificación y validación.
- Ejecutar las tareas de verificación y validación.
- Recoger y analizar los datos de las métricas.
- Planificar las pruebas.
- Desarrollar especificaciones de las pruebas.
- Ejecutar las pruebas.

5.1 Pruebas

5.1.1 Pruebas de requisitos funcionales

(P1) Introducir en el sistema una serie de conceptos prueba, relaciones, etc., (introducir metadatos).

(P2) Analizar la validez de los conceptos y la posible pérdida de semántica respecto a lo que se pretendía representar.

(P3) Modificar y eliminar elementos de la ontología y analizar si el resultado es el esperado.

(P4) Ejecutar todas los procedimientos de consulta, borrado y actualización, y verificar su comportamiento.

5.1.2 Pruebas de requisitos de interfaces externos

(P5) Comprobar el funcionamiento de la interfaz gráfica del editor de conocimiento (ventanas, botones, etc).

(P6) Comprobar la velocidad y funcionamiento de la conexión.

5.1.3 Pruebas de requisitos de rendimiento

(P7) Analizar si el tiempo de respuesta cumple con los requisitos.

(P8) Determinar si la calidad del servicio es aceptable.

5.1.4 Pruebas de los atributos definidos

(P9) Asegurar el buen uso del SGBD para garantizar la perdurabilidad y confidencialidad.

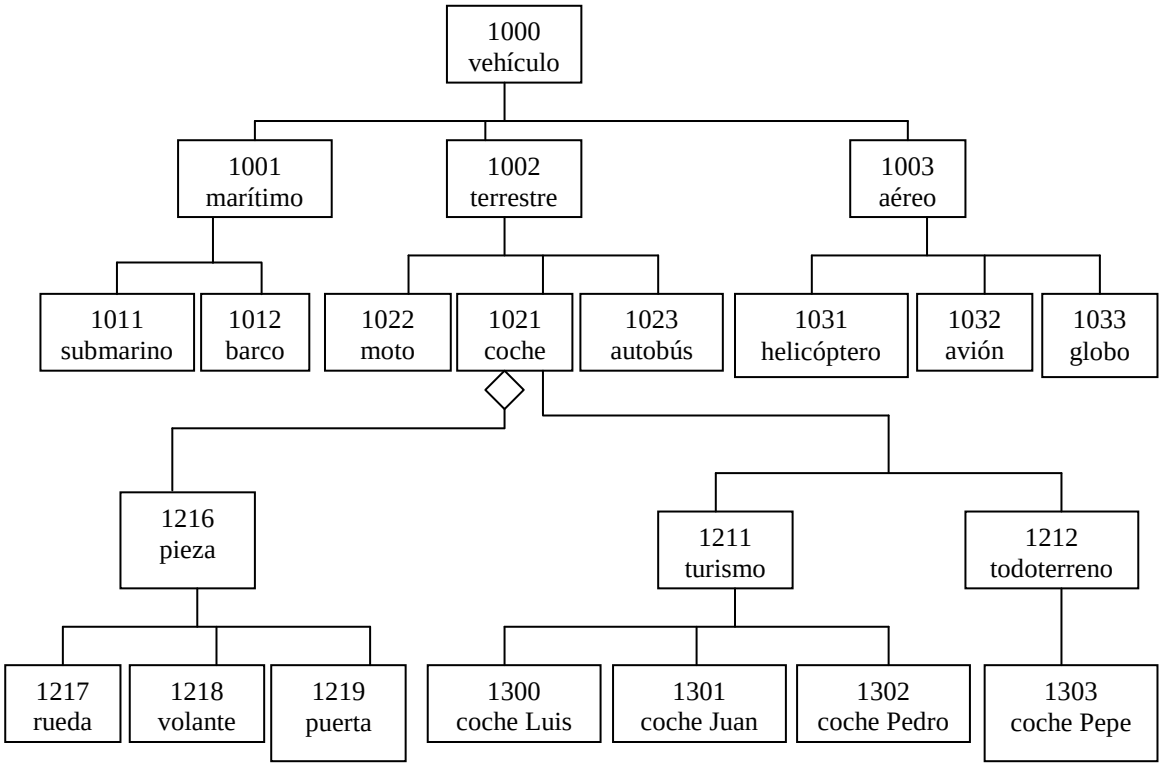
5.2 Matriz de traceabilidad

Requisitos	Pruebas
R3,R4,R5	P1,P2,P3,P4,P5
R6,R7,R8,R9	P4
R12,R13	P5
R15,R16,R17	P8
R18	P6
R22	P7
R26,R27	P9

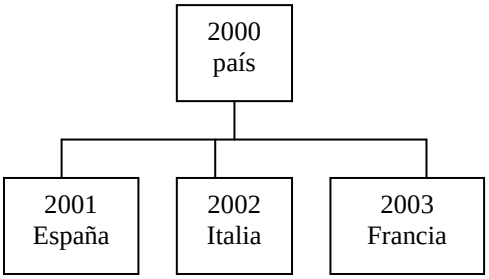
ANEXO B

Datos de un ejemplo completo

Para ofrecer una visión global de la funcionalidad y de la potencia de la ontología realizada se ha diseñado un ejemplo completo que se describe a continuación. Para simplificar, a diferencia de los otros ejemplos (validación apartado 2.4), no se han representado de forma gráfica las relaciones, atributos, etc. Sin embargo, los siguientes organigramas muestran la estructura jerárquica de los conceptos, a un alto nivel de abstracción, con el fin de hacer más fácil la comprensión del ejemplo representado. Cada recuadro contiene el 'cod_concepto' con el que se almacena en la ontología y un término significativo que describe a dicho concepto para facilitar la tarea del lector. (Nota: no representa un modelo ER, el rombo representa una agregación de UML y el resto de líneas una simple jerarquía)

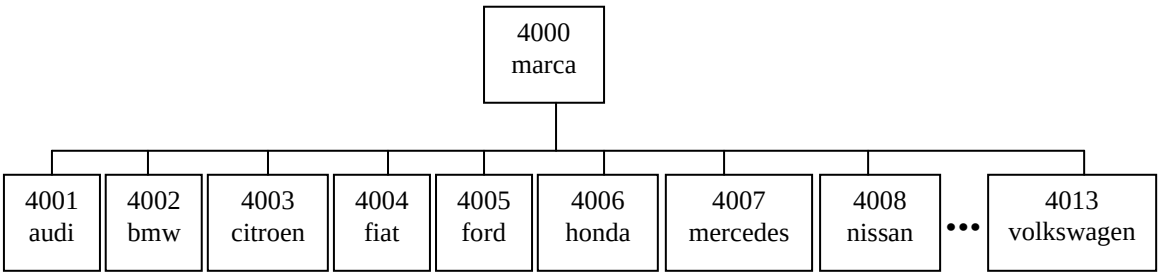


Aparte de la jerarquía anterior, existen unos conceptos independientes que se utilizarán en la tabla 'espacio', 'tiene1' y 'tiene2' para especificar el precio de un coche en cada país (expresado en euros).

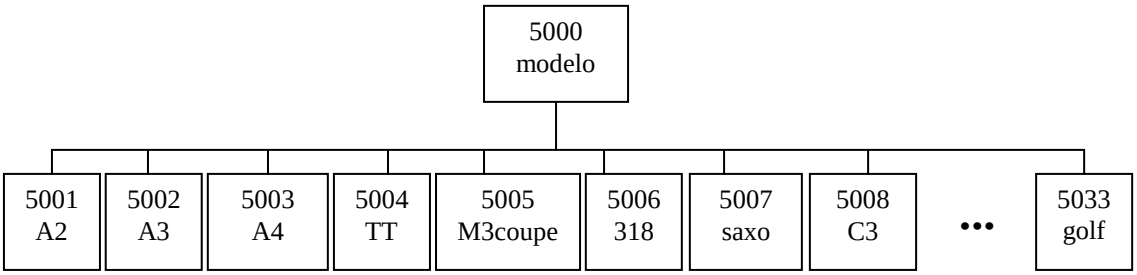


El resto de conceptos de la ontología son los siguientes:

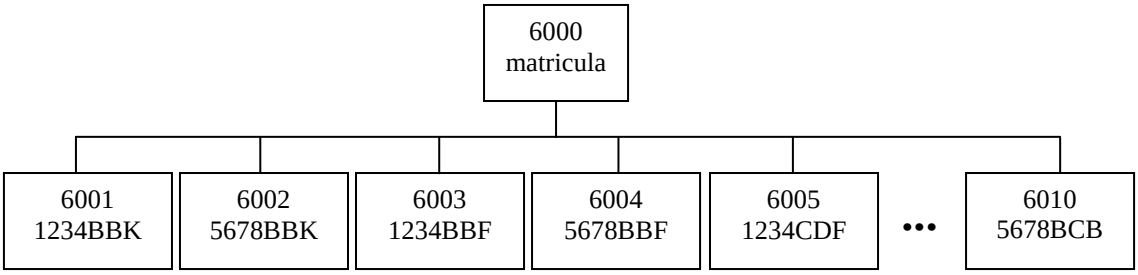
- Para formar el dominio "marca" de coche



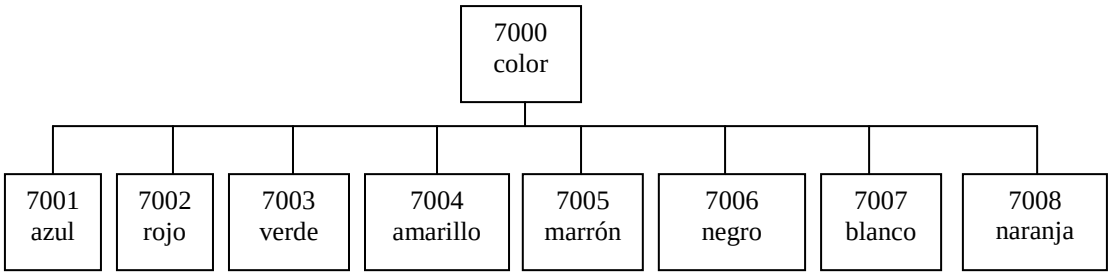
- Para constituir el dominio "modelo" de coche



- Para crear el dominio "matrícula"



- Para generar el dominio "color"



Las siguientes tablas muestran el contenido real de la base de datos, y en ellas se pueden ver los términos y relaciones de cada concepto, atributos, valor de los atributos, etc.

Tabla concepto

cod_concepto	tipo
1000	genérico
1001	genérico
1002	genérico
1003	genérico
1011	instancia
1012	instancia
1021	genérico
1022	instancia
1023	instancia
1031	instancia
1032	instancia
1033	instancia
1211	genérico
1212	genérico
1216	genérico
1217	instancia
1218	instancia
1219	instancia
1300	instancia
1301	instancia
1302	instancia
1303	instancia
2000	genérico
2001	instancia
2002	instancia
2003	instancia
4000	genérico
4001	instancia
4002	instancia
4003	instancia
4004	instancia
4005	instancia
4006	instancia
4007	instancia
4008	instancia
4009	instancia
4010	instancia

cod_concepto	tipo
4011	instancia
4012	instancia
4013	instancia
5000	genérico
5001	instancia
5002	instancia
5003	instancia
5004	instancia
5005	instancia
5006	instancia
5007	instancia
5008	instancia
5009	instancia
5010	instancia
5011	instancia
5012	instancia
5013	instancia
5014	instancia
5015	instancia
5016	instancia
5017	instancia
5018	instancia
5019	instancia
5020	instancia
5021	instancia
5022	instancia
5023	instancia
5024	instancia
5025	instancia
5026	instancia
5027	instancia
5028	instancia
5029	instancia
5030	instancia
5031	instancia
5032	instancia
5033	instancia

cod_concepto	tipo
6000	genérico
6001	instancia
6002	instancia
6003	instancia
6004	instancia
6005	instancia
6006	instancia
6007	instancia
6008	instancia
6009	instancia
6010	instancia
7000	genérico
7001	instancia
7002	instancia
7003	instancia
7004	instancia
7005	instancia
7006	instancia
7007	instancia
7008	instancia

Tabla Genérico

cod_concepto
1000
1001
1002
1003
1021
1211
1212
1216
2000
4000
5000
6000
7000

Tabla Instancia

cod_concepto		5011
1011		5012
1012		5013
1022		5014
1023		5015
1217		5016
1218		5017
1219		5018
1031		5019
1032		5020
1033		5021
1300		5022
1301		5023
1302		5024
1303		5025
2001		5026
2002		5027
2003		5028
4001		5029
4002		5030
4003		5031
4004		5032
4005		5033
4006		6001
4007		6002
4008		6003
4009		6004
4010		6005
4011		6006
4012		6007
4013		6008
5001		6009
5002		6010
5003		7001
5004		7002
5005		7003
5006		7004
5007		7005
5008		7006
5009		7007
5010		7008

Tabla Término

cod_término	lenguaje
vehículo	castellano
vehicle	inglés
marítimo	castellano
acuático	castellano
maritime	inglés
aquatic	inglés
terrestre	castellano
terrestrial	inglés
aéreo	castellano
volador	castellano
airy	inglés
flying	inglés
submarino	castellano
submarine	inglés
barco	castellano
yate	castellano
bote	castellano
ship	inglés
yacht	inglés
boat	inglés
helicóptero	castellano
helicopter	inglés
avión	castellano
avioneta	castellano
aircraft	inglés
plane	inglés
globo aerostático	castellano
globe	inglés
autobús	castellano
coche	castellano
automóvil	castellano
car	inglés
moto	castellano
ciclomotor	castellano
motor-cycle	inglés
turismo	castellano
todoterreno	castellano
pieza	castellano
piece	inglés

cod_término	lenguaje
rueda	castellano
wheel	inglés
volante	castellano
steering wheel	inglés
puerta	castellano
door	inglés
coche de Luis	castellano
coche de Juan	castellano
coche de Pedro	castellano
coche de Pepe	castellano
país	castellano
country	inglés
españa	castellano
spain	inglés
italia	castellano
italy	inglés
francia	castellano
france	inglés
marca	castellano
audi	ascii
bmw	ascii
citroen	ascii
fiat	ascii
ford	ascii
honda	ascii
mercedes	ascii
nissan	ascii
opel	ascii
jeep	ascii
renault	ascii
seat	ascii
volkswagen	ascii
modelo	castellano
A2	ascii
A3	ascii
A4	ascii
TT	ascii
M3coupe	ascii
318	ascii

cod_término	lenguaje
saxo	ascii
C3	ascii
xsara	ascii
punto	ascii
stilo	ascii
ka	ascii
fiesta	ascii
focus	ascii
mondeo	ascii
civic	ascii
accord	ascii
claseA	ascii
sportcoupe	ascii
corsa	ascii
astra	ascii
vectra	ascii
cherokee	ascii
grand cherokee	ascii
clio	ascii
twingo	ascii
laguna	ascii
scenic	ascii
ibiza	ascii
toledo	ascii
leon	ascii
polo	ascii
golf	ascii
matricula	castellano
1234BBK	ascii
5678BBK	ascii
1234BBF	ascii
5678BBF	ascii
1234CDF	ascii
5678CDF	ascii
1234CSK	ascii
5678CSK	ascii
1234BCB	ascii
5678BCB	ascii
color	castellano
colour	inglés
azul	castellano

cod_término	lenguaje
blue	inglés
rojo	castellano
red	inglés
verde	castellano
green	inglés
amarillo	castellano
yellow	inglés
marrón	castellano
brown	inglés
negro	castellano
black	inglés
blanco	castellano
white	inglés
naranja	castellano
orange	inglés
transporte publico	castellano
no acuático	castellano

Tabla Referencia

cod_término	cod_concepto	grado
vehículo	1000	100
vehicle	1000	90
marítimo	1001	100
acuático	1001	80
maritime	1001	90
aquatic	1001	70
terrestre	1002	100
terrestrial	1002	90
aéreo	1003	100
volador	1003	80
airy	1003	70
flying	1003	90
submarino	1011	100
submarine	1011	90
barco	1012	100
yate	1012	50
bote	1012	60
ship	1012	90
yacht	1012	40
boat	1012	50
helicóptero	1031	100
helicopter	1031	90
avión	1032	100
avioneta	1032	80
aircraft	1032	80
plane	1032	80
globo aerostático	1033	100
globe	1033	60
autobús	1023	100
coche	1021	100
automóvil	1021	90
car	1021	90
moto	1022	100
ciclomotor	1022	70
motor-cycle	1022	80
turismo	1211	100
todorreno	1212	100
pieza	1216	90
piece	1216	80

cod_término	cod_concepto	grado
rueda	1217	100
wheel	1217	90
volante	1218	100
steering wheel	1218	90
puerta	1219	100
door	1219	60
coche de Luis	1300	100
coche de Juan	1301	100
coche de Pedro	1302	100
coche de Pepe	1303	100
país	2000	100
country	2000	90
españa	2001	100
spain	2001	90
italia	2002	100
italy	2002	90
francia	2003	100
france	2003	90
marca	4000	100
audi	4001	100
bmw	4002	100
citroen	4003	100
fiat	4004	100
ford	4005	100
honda	4006	100
mercedes	4007	100
nissan	4008	100
opel	4009	100
jeep	4010	100
renault	4011	100
seat	4012	100
volkswagen	4013	100
modelo	5000	100
A2	5001	100
A3	5002	100
A4	5003	100
TT	5004	100
M3coupe	5005	100
318	5006	100

cod_término	cod_concepto	grado
saxo	5007	100
C3	5008	100
xsara	5009	100
punto	5010	100
stilo	5011	100
ka	5012	100
fiesta	5013	100
focus	5014	100
mondeo	5015	100
civic	5016	100
accord	5017	100
claseA	5018	100
sportcoupe	5019	100
corsa	5020	100
astra	5021	100
vectra	5022	100
cherokee	5023	100
grand cherokee	5024	100
clio	5025	100
twingo	5026	100
laguna	5027	100
scenic	5028	100
ibiza	5029	100
toledo	5030	100
leon	5031	100
polo	5032	100
golf	5033	100
matricula	6000	100
1234BBK	6001	100
5678BBK	6002	100
1234BBF	6003	100
5678BBF	6004	100
1234CDF	6005	100
5678CDF	6006	100
1234CSK	6007	100
5678CSK	6008	100
1234BCB	6009	100
5678BCB	6010	100
color	7000	100
colour	7000	90
azul	7001	100

cod_término	cod_concepto	grado
blue	7001	90
rojo	7002	100
red	7002	90
verde	7003	100
green	7003	90
amarillo	7004	100
yellow	7004	90
marrón	7005	100
brown	7005	90
negro	7006	100
black	7006	90
blanco	7007	100
white	7007	90
naranja	7008	100
orange	7008	90
transporte publico	1023	95
transporte publico	1032	90
no acuático	1002	90
no acuático	1003	90

Tabla Relación

tipo	descripción
es un	Utilizada para representar jerarquías
compuesto de	La unión de "n" conceptos compone otro concepto
antonimia	Expresa que dos conceptos son antónimos entre sí

Tabla Espacio

cod_concepto
2001
2002
2003

Tabla Relaciona

concepto1	concepto2	relación
1001	1000	es un
1002	1000	es un
1003	1000	es un
1001	1002	antonimia
1001	1003	antonimia
1002	1003	antonimia
1011	1001	es un
1012	1001	es un
1021	1002	es un
1022	1002	es un
1023	1002	es un
1031	1003	es un
1032	1003	es un
1033	1003	es un
1211	1021	es un
1212	1021	es un
1021	1216	compuesto de
1217	1216	es un
1218	1216	es un
1219	1216	es un
1300	1211	es un
1301	1211	es un
1302	1211	es un
1303	1212	es un
2001	2000	es un
2002	2000	es un
2003	2000	es un
4001	4000	es un
4002	4000	es un
4003	4000	es un
4004	4000	es un
4005	4000	es un
4006	4000	es un
4007	4000	es un
4008	4000	es un
4009	4000	es un
4010	4000	es un
4011	4000	es un
4012	4000	es un

concepto1	concepto2	relación
4013	4000	es un
5001	5000	es un
5002	5000	es un
5003	5000	es un
5004	5000	es un
5005	5000	es un
5006	5000	es un
5007	5000	es un
5008	5000	es un
5009	5000	es un
5010	5000	es un
5011	5000	es un
5012	5000	es un
5013	5000	es un
5014	5000	es un
5015	5000	es un
5016	5000	es un
5017	5000	es un
5018	5000	es un
5019	5000	es un
5020	5000	es un
5021	5000	es un
5022	5000	es un
5023	5000	es un
5024	5000	es un
5025	5000	es un
5026	5000	es un
5027	5000	es un
5028	5000	es un
5029	5000	es un
5030	5000	es un
5031	5000	es un
5032	5000	es un
5033	5000	es un
6001	6000	es un
6002	6000	es un
6003	6000	es un
6004	6000	es un
6005	6000	es un

concepto1	concepto2	relación
6006	6000	es un
6007	6000	es un
6008	6000	es un
6009	6000	es un
6010	6000	es un
7001	7000	es un
7002	7000	es un
7003	7000	es un
7004	7000	es un
7005	7000	es un
7006	7000	es un
7007	7000	es un
7008	7000	es un

Tabla Alfanumérico

dominio	cod_concepto	descripción
marca	4000	(...)
modelo	5000	(...)
matricula	6000	(...)
color	7000	(...)

Tabla Fuente

web	descripción
www.seguros-uc3m.com	(...)
www.tasación-uc3m.com	(...)

Tabla Dominio

identificador	tipo
marca	alfanumérico
modelo	alfanumérico
matricula	alfanumérico
color	alfanumérico
precio	numérico
año	numérico
booleano	numérico
km	numérico

Tabla Numérico

dominio	cota inferior	cota superior	unidad
precio	0	360000	euros
año	1900	2004	años
km	0	500000	kms
booleano	0	1	

Respecto a los dominios definidos anteriormente, como ya se explica en otros apartados, un dominio alfanumérico está formado por todas las instancias (conceptos) de un concepto genérico. En este ejemplo, existen cuatro dominios alfanuméricos marca, modelo, matrícula y color que se corresponden con los conceptos 4000, 5000, 6000 y 7000 respectivamente. El dominio marca incluye a firmas como Ford, Opel, Renault, Seat, etc. y se corresponde con el rango de instancias [4001-4013]. El dominio modelo incluye alguno de los modelos pertenecientes a las marcas del dominio anterior y el rango de instancias que componen este dominio es [5000-5033]. Por su parte, el dominio matrículas debe tener explícitamente en forma de instancias todas las matrículas sobre las que se quiere almacenar conocimiento. Para este ejemplo sólo se han definido diez [6001-6010]. Por último, el dominio colores está formado por el rango de instancias [7001-7008].

Para la representación de los atributos se ha elegido la “rama” coche de la jerarquía para extender el ejemplo desde el punto de vista de los atributos. Para el resto de los conceptos se haría de forma análoga, pero por claridad y simplificar sólo se ha implementado lo descrito a continuación. Se pretende disponer de un conocimiento actualizado y eficiente que podría ser útil para cualquier empresa que trabaje en el mundo de la automoción. Para el atributo ‘km recorridos’, se supone que cada automóvil, mediante alguna tecnología, puede ofrecer a una página web diariamente su kilometraje. A esta página podrán acceder los agentes de actualización para modificar los valores almacenados en la ontología y disponer de información actualizada de cada vehículo. Por otro lado, también se almacena y actualiza el precio actual estimado de cada coche, apoyándose en otra página web.

Tabla Atributo

cod_concepto	nombre	caducidad	dominio
1021	marca	0	marca
1021	modelo	0	modelo
1021	precio	30	precio
1021	matricula	0	matricula
1021	color	0	color
1021	año compra	0	año
1021	km recorridos	1	km
1211	marca	0	marca
1211	modelo	0	modelo
1211	precio	30	precio
1211	matricula	0	matricula
1211	color	0	color
1211	año compra	0	año
1211	km recorridos	1	km
1212	marca	0	marca
1212	modelo	0	modelo
1212	precio	30	precio

cod_concepto	nombre	caducidad	dominio
1212	matricula	0	matricula
1212	color	0	color
1212	año compra	0	año
1212	km recorridos	1	km
1212	reductora	0	booleano
1300	marca	0	marca
1300	modelo	0	modelo
1300	precio	30	precio
1300	matricula	0	matricula
1300	color	0	color
1300	año compra	0	año
1300	km recorridos	1	km
1301	marca	0	marca
1301	modelo	0	modelo
1301	precio	30	precio
1301	matricula	0	matricula
1301	color	0	color
1301	año compra	0	año
1301	km recorridos	1	km
1302	marca	0	marca
1302	modelo	0	modelo
1302	precio	30	precio
1302	matricula	0	matricula
1302	color	0	color
1302	año compra	0	año
1302	km recorridos	1	km
1303	marca	0	marca
1303	modelo	0	modelo
1303	precio	30	precio
1303	matricula	0	matricula
1303	color	0	color
1303	año compra	0	año
1303	km recorridos	1	km
1303	reductora	0	booleano

Tabla Actualiza

web	cod_concepto	nombre	peso
www.seguros-uc3m.com	1021	km recorridos	10
www.seguros-uc3m.com	1211	km recorridos	10
www.seguros-uc3m.com	1212	km recorridos	10
www.seguros-uc3m.com	1300	km recorridos	10
www.seguros-uc3m.com	1301	km recorridos	10
www.seguros-uc3m.com	1302	km recorridos	10
www.seguros-uc3m.com	1303	km recorridos	10
www.tasación-uc3m.com	1021	precio	8
www.tasación-uc3m.com	1211	precio	8
www.tasación-uc3m.com	1212	precio	8
www.tasación-uc3m.com	1300	precio	8
www.tasación-uc3m.com	1301	precio	8
www.tasación-uc3m.com	1302	precio	8
www.tasación-uc3m.com	1303	precio	8

Tabla Tiene1

En la tabla 'tiene1' se puede observar que la ontología almacena conocimiento (tres valores para un mismo atributo) dependiendo del 'espacio' en que se instancie un concepto. En este caso, se recoge que el atributo precio para una instancia del concepto 1021 (coche) tiene un valor diferente en función del espacio (España, Italia y Francia). Destacar que el precio se refiere a un único coche ("el coche de Pedro") que instantáneamente, en tres lugares diferentes, tiene un valor distinto para el atributo precio. Dicho atributo es actualizado en función de la información obtenida en www.tasación-uc3m.com.

cod_concepto	nombre	espacio	valor	inicio	fin
1302	precio	2001	18000	15/06/04	15/07/04
1302	precio	2002	17000	15/06/04	15/07/04
1302	precio	2003	17500	15/06/04	15/07/04

Tabla Tiene2

El atributo ‘km recorridos’ será actualizado con la información que cada vehículo aporta a www.seguros-uc3m.com.

cod_concepto	nombre	valor	inicio	fin
1300	marca	opel		
1301	marca	ford		
1302	marca	volkswagen		
1303	marca	jeep		
1300	modelo	corsa		
1301	modelo	focus		
1302	modelo	golf		
1303	modelo	cherokee		
1300	matricula	5678BBK		
1301	matricula	1234CSK		
1302	matricula	1234BCB		
1303	matricula	1234BBF		
1300	color	azul		
1301	color	blanco		
1302	color	negro		
1303	color	verde		
1300	año compra	1999		
1301	año compra	2002		
1302	año compra	2003		
1303	año compra	2000		
1300	km recorridos	100000	12/07/04	13/07/04
1301	km recorridos	75000	12/07/04	13/07/04
1302	km recorridos	25000	12/07/04	13/07/04
1303	km recorridos	85000	12/07/04	13/07/04
1303	reductora	1		

Para finalizar, hay que destacar una de las principales características de este ejemplo. Todos los conceptos relevantes están representados en dos idiomas (español e inglés). Esto aporta a la ontología mucha potencia, ya que dos profesionales del campo de la automoción, en este caso, podrían referirse a un mismo concepto, cada uno en su idioma, sin perder ninguna semántica en este dominio, ya que cada concepto está referenciado directamente por los términos correspondientes en los respectivos idiomas.

ANEXO C

Manual de la aplicación

PARTE I: Instalación y Administración

Para la instalación de esta aplicación, es necesario tener en cuenta dos partes: servidor y cliente.

Instalación del servidor: el sistema está soportado por el SGBDR Oracle 9i. Para su instalación, el administrador deberá crear una instancia de nombre *prueba*, y ejecutar en ella los *scripts* de creación de tablas que se facilitan en este documento en el Anexo F. Además, deberá crear el usuario de actualización *nherrero*, y todos los usuarios generales que sean precisos para acceder al sistema.

Instalación del cliente: todos los clientes deberán configurar el SID *prueba* en el archivo *tnsnames.ora*. Además, el cliente de actualización necesitará tener instalada la máquina virtual de java jdk1.3, incluyendo la biblioteca *classes12.zip* que proporciona Oracle®. La aplicación 'Editor de Conocimiento' no necesita ninguna instalación especial, siendo únicamente necesario copiar la carpeta proporcionada en el directorio que tenga disponible cada usuario. Posteriormente, basta con especificar esa ruta en el fichero *sibo2.bat* para ejecutar la aplicación.

PARTE II: Manual del usuario de la aplicación 'Editor de Conocimiento'

Para ayudar a las tareas del editor de conocimiento, se ha implementado una aplicación software que se encarga de facilitar las inserciones y borrado de datos de la ontología. Es una aplicación de uso muy intuitivo y fácil de aprender. A continuación se expone brevemente un pequeño manual de uso de la aplicación.

La primera pantalla (figura 5.1), que se ve al ejecutar la aplicación, ofrece dos opciones al editor de conocimiento: operaciones de inserción y operaciones de borrado. Si se elige operaciones de inserción, transita a una nueva pantalla (figura 5.2) donde se encuentran todas las operaciones de inserción disponibles para el editor de conocimiento. Si por el contrario elige las operaciones de borrado, transita a una nueva pantalla (figura 5.3) donde aparece un listado completo con las operaciones de borrado autorizadas.

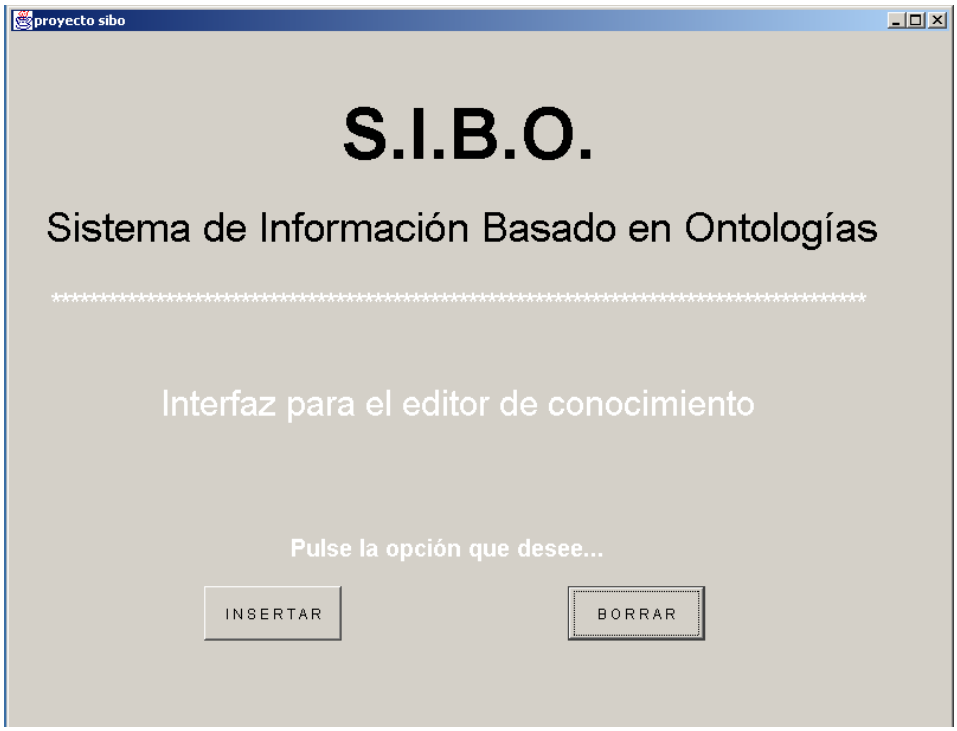


Figura 5.1

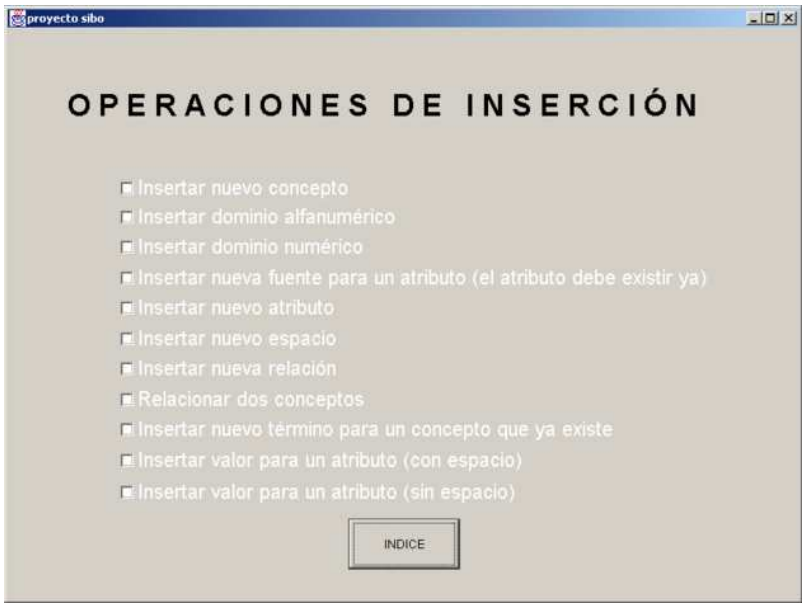


Figura 5.2

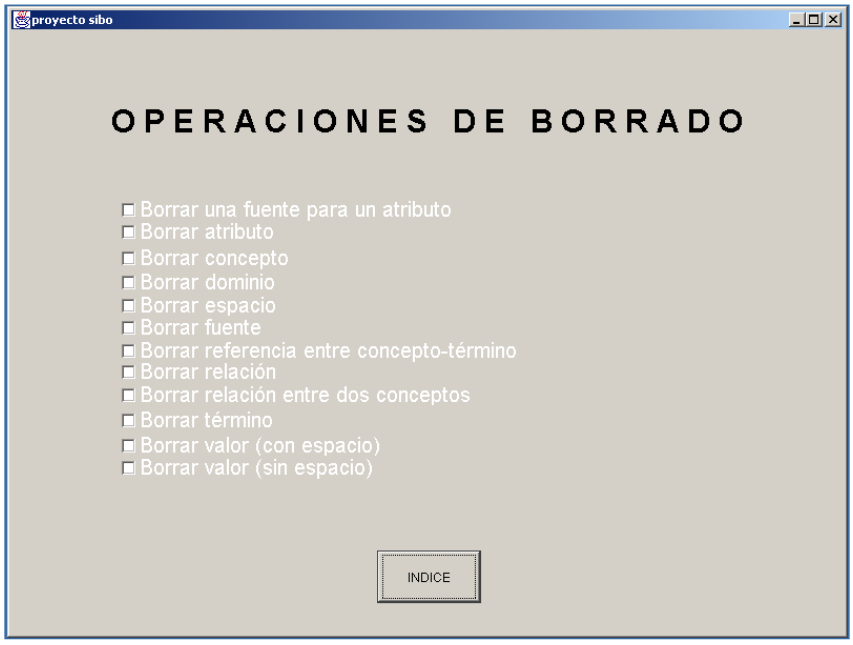


Figura 5.3

Para ejecutar una operación basta con elegir una opción en cualquiera de las pantallas anteriores (figura 5.2 y figura 5.3). A modo de ejemplo se muestran dos casos, uno de inserción (figura 5.4) y otro de borrado (figura 5.5).

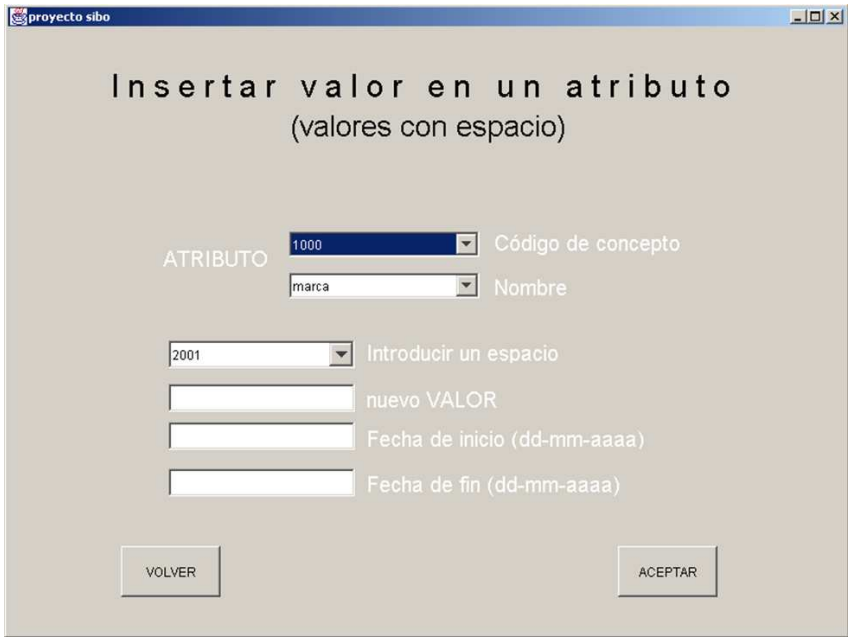


Figura 5.4

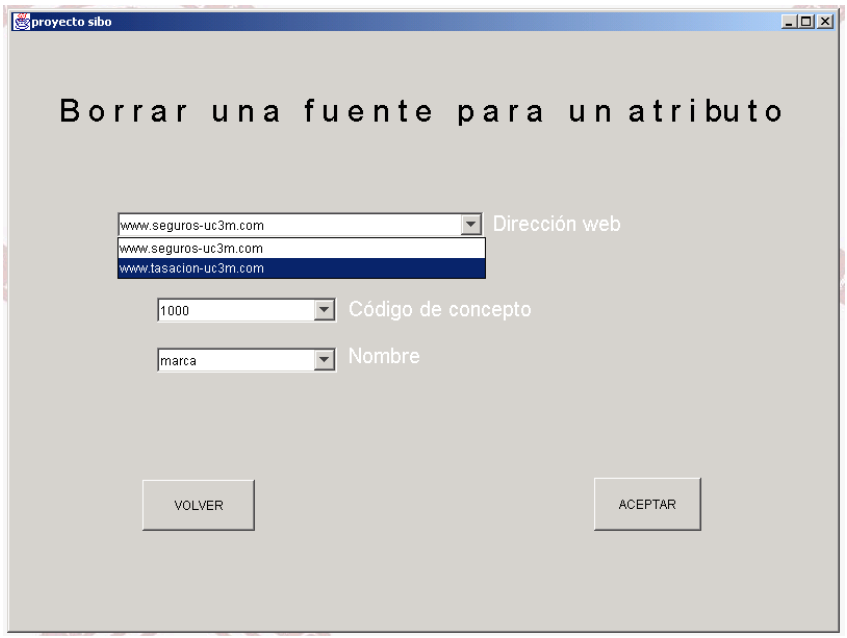


Figura 5.5

Para finalizar la ejecución, al igual que cualquier aplicación ejecutada sobre Windows, se debe hacer *clic* en el aspa del vértice superior derecho de la ventana.

ANEXO D

Importancia de la evaluación de los sistemas basados en conocimiento

El futuro crecimiento de los sistemas basados en conocimiento se verá gravemente entorpecido si los desarrolladores no encaran con firmeza el problema de la fiabilidad.

Para los sistemas en uso crítico (vehículos espaciales, centrales nucleares, etc.) la fiabilidad debe estar asegurada antes de usarlos para esas tareas. Adicionalmente, dado que a los SS.BB.CC. se les supone un comportamiento "inteligente", las conclusiones erróneas derivadas por el sistema podrían provocar una pérdida de credibilidad por parte del usuario. Esto puede conducir a un decremento del uso de estos sistemas y, en definitiva, a su fracaso final.

Existen múltiples criterios de evaluación, de los cuales se pueden destacar los siguientes:

- La **corrección** del modelo (en este caso E/R y relacional).

  Si posee una estructura y sintaxis adecuada. Si están bien expresados los conocimientos que se pretendían modelar. A la evaluación del aspecto de la corrección de un modelo se le ha llamado *verificación*.

- La **validez** del modelo.

  Un modelo es válido si responde a una semántica adecuada. Por lo tanto, este parámetro depende totalmente del dominio que se esté modelando. A la evaluación del aspecto de validez de un sistema se le llama *validación*. Se validan criterios referentes a la calidad de las respuestas del sistema de información y criterios referentes a la satisfacción de los requisitos.

- La **usabilidad** (amigabilidad de uso) del sistema.

  Si al usuario le resulta agradable la interacción con él. El aspecto de la usabilidad es muy importante puesto que el nuevo sistema se va a introducir en la rutina de una organización y de unas personas. Se evalúa si el usuario se siente cómodo al tratar con el sistema, si el uso del sistema es agradable. Los criterios a tener en cuenta son:

  - Interfaz.
  - Carga impuesta al usuario: valorar si la carga es soportable por el usuario (el estrés que el sistema crea sobre el usuario).
  - Adaptación del sistema a la tarea: el modo en el que el sistema realiza la tarea no debe resultar oscuro ni rebuscado.

- Coste de aprendizaje: intuitivo y fácil de aprender a utilizar el sistema para los usuarios.
- Adaptación a los estándares: normas de las organizaciones o estándares externos.
- Documentación: facilidad de uso de la documentación que lo acompaña.
- Aspectos subjetivos: que el software sea atractivo, divertido, innovador, etc.

- La **utilidad** del sistema por haber alcanzado en la organización las metas perseguidas con el desarrollo del mismo: disminuir la tasa de errores, ampliar las prestaciones, mejorar la eficiencia (tiempo de respuesta), eficacia (fiabilidad y disponibilidad) y la capacidad del sistema (resolviendo nuevos tipos de problemas).

ANEXO E

Glosario

Clustering

En términos generales un cluster es un grupo de sistemas independientes que trabajan juntos como un sistema único. El cliente interactúa con un cluster como si fuera un servidor único. Las configuraciones de cluster se utilizan para tener disponibilidad y capacidad de escalabilidad:

- Disponibilidad: Cuando un sistema falla en el cluster, el software de cluster responde distribuyendo el trabajo del sistema con fallo a los sistemas que quedan en el cluster.
- Capacidad de escalabilidad: Cuando la carga general excede las capacidades de los sistemas en el cluster, es posible agregar sistemas adicionales al mismo. En la actualidad, los clientes que planean ampliar la capacidad de su sistema deben considerar servidores *high end* costosos que proporcionan espacio para CPUs, controladores y memoria adicionales. Al utilizar la tecnología de clustering, los clientes podrán agregar gradualmente sistemas estándar más pequeños, según sea necesario, para satisfacer los requerimientos generales de potencia de procesamiento.

Conceptualización

Conceptualización es la tarea de entender el dominio del problema y la terminología usada. Consiste en hacer explícitos los conceptos clave y las relaciones relevantes, así como tener los conocimientos de una forma estructurada. Formalmente, la conceptualización es una terna que incluye los conceptos, sus relaciones y sus funciones.

Interoperatividad

Capacidad por la cual una máquina se puede comunicar con las demás.

Metadatos

Información acerca de la información (datos sobre los propios datos) para dotarlos de una semántica y estructurar su contenido.

Parser

Analizador sintáctico

Thesauro

Diccionario, catálogo, antología. Desde el punto de vista de su estructura, el *thesaurus* es un vocabulario controlado y dinámico de términos que tienen entre sí relaciones semánticas (referentes al significado: sinonimia, antonimia, homonimia) y genéricas (referentes a su organización, sólo a nivel jerárquico, organización de términos, pero no a nivel conceptual: madrileño implica español) y que se aplica a un campo particular del conocimiento. Desde el punto de vista funcional, un *thesaurus* es un instrumento de control de la terminología, utilizado para trasladar a un lenguaje más estricto (lenguaje documental) el lenguaje natural utilizado en los documentos.

Según García Gutiérrez, el *thesaurus* es un lenguaje documental de estructura combinatoria, normalizado y normativo, de carácter especializado, que unifica terminológicamente los conceptos expresados en los documentos con el fin de permitir su análisis y recuperación. Dado que se trata de un subgrupo estructurado del lenguaje natural, describe el contenido analítico de los documentos, de los objetos o del conjunto de datos. La terminología utilizada para describir las características formales del documento y que corresponde a términos no descriptores del contenido no tiene necesidad de formar parte de un *thesaurus*. Por otra parte, debe reflejar con precisión las informaciones contenidas en el conjunto de documentos u otros elementos de la colección a la que se aplica el *thesaurus* y contener los términos y reenvíos apropiados al tema, teniendo en cuenta a la vez el lenguaje de los documentos y el lenguaje de los usuarios. El *thesaurus* es un diccionario que muestra la equivalencia entre los términos del lenguaje natural y los normalizados y preferentes del lenguaje documental: en él quedan incluidos los términos descriptores y no descriptores, con las relaciones explícitas que faciliten el análisis del documento y la formulación de las necesidades del usuario. Participan, pues, de la estructura jerárquica y de la asociativa

Un *thesaurus* es menos potente que una ontología, ya que sólo aporta una normalización terminológica frente a la normalización conceptual que aportan las ontologías (véase capítulo 1 de este libro)

COMPONENTES BÁSICOS DE UNA ONTOLOGÍA

Asignación: términos que se usan para asignar un valor a una propiedad en un contexto.

Atributos / Slot: propiedades de cada concepto que describen varias características y atributos de un concepto.

Axiomas: son proposiciones, principios o restricciones que se asumen ciertas para un universo del discurso y que deben cumplir los elementos de la ontología. Se usan para modelar verdades que se cumplen siempre en la realidad modelada. Los axiomas definidos en una ontología pueden ser estructurales o no estructurales.

- Un axioma estructural establece condiciones relacionadas a las jerarquías de la ontología, conceptos y atributos definidos.

  Un ejemplo de axioma estructural es, "*si A y B son de la clase C, entonces A no es subclase de B*".

- Los axiomas no estructurales establecen relaciones entre atributos de un concepto, y son específicos de cada dominio.

  Un ejemplo de axioma no estructural es la relación **F=m*a** que debe cumplirse siempre entre los atributos **F** (fuerza), **m** (masa) y **a** (aceleración) de un determinado concepto.

Cardinalidad: define cuántas valoraciones simultáneas admite cierto concepto individual (máximo y mínimo). Cuántas instanciaciones simultáneas permite un concepto.

Casos / Instancias: se utilizan para representar objetos determinados de un concepto, es decir, ejemplos específicos pertenecientes a alguna clase. Es una ocurrencia de un caso particular de un concepto.

Clase jerárquica (jerarquía de taxonomías): una colección de clases conectadas por relaciones "es un tipo de".

Conceptos (clases): son ideas básicas que se intentan formalizar mediante su descripción y asociación a símbolos. Se corresponden con conceptos en el dominio de un discurso. En otras palabras, son pensamientos o ideas y sus características, que son susceptibles de ser utilizados (exportados a un interlocutor). Son únicos y son los elementos más primitivos de una ontología.

Contexto: conceptos que pueden y suelen acompañar a otro concepto en una circunstancia.

Dominio: valores que puede tomar un atributo. Un rango de valores.

Herencia: proceso en el que las subclases heredan propiedades y valores (e incluso términos) definidos en alguno de sus "antecesores" en la jerarquía.

Medidas: en algunos casos esos valores estarán expresados en una unidad de medida.

Métrica: valores que puede tomar un atributo (dominio).

Objeto Sujeto: algunos conceptos se pueden particularizar hasta el extremo de que su **instancia** se relaciona con un **término**, que es nombre propio, de modo que ese concepto se puede etiquetar como 'sujeto'. Por ejemplo, el concepto "región" (que es una instancia de concepto general) puede ser un concepto sujeto porque puede particularizarse a instancias con nombre propio (es decir, se relaciona con de 0 a n ocurrencias de particularizaciones) tales como Madrid, Segovia, etc. En realidad, en una ontología genérica no suelen representarse 'sujetos', aunque, el sujeto también es conocimiento. Pero los conceptos, en realidad, existen independientemente de la existencia de los sujetos.

Relaciones: representan la interacción y enlace entre los conceptos del dominio. Suelen formar la taxonomía del dominio (clasificación y descripción de modo formal de un dominio). Ejemplos de relaciones: *subclase-de*, *parte-de*, *conectado-a*, *sinonimia*, *jerarquías*, *relaciones conceptuales*, etc.

Relaciones derivadas

- **Funciones**: son un tipo concreto de relación donde se identifica un elemento mediante el cálculo de una función que considera varios elementos de la ontología. Por ejemplo, *precio = valor + ganancia + iva, asignar-fecha*, etc.
- **Otras relaciones**: "compuesto de", sinonimia, "es un", antonimia, etcétera.

Relación subclase: es una clase normal pero que depende de alguna otra clase (concepto). Se utiliza para representar conceptos más específicos. A la relación que existe entre una subclase y su clase padre se le denomina, entre otras muchas, "es una" o "es un tipo de".

Relación ámbito: relacionar dos conceptos por concepto es parte de ámbito.

- un concepto es un concepto.
- un ámbito, que puede abarcar varios conceptos, también es un concepto.
- debe existir una relación entre conceptos que enlace un concepto y su ámbito.

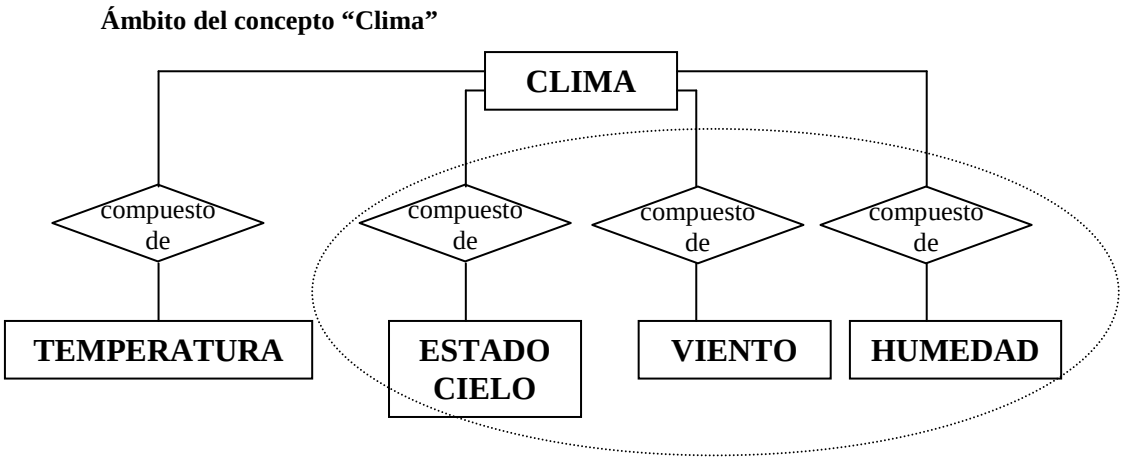


Figura 5.6

Término: es la representación simbólica de un concepto. Puede referenciar uno o más conceptos, y viceversa.

Valor: es una cuantificación dentro del dominio de un concepto referido a una métrica (o escala de valores) y a un momento, que podríamos definir como delimitación espacio-tiempo.

Valor por defecto: valor que toma un determinado atributo en todas las instancias de una clase, cuando no se ha especificado otro valor y este valor es requerido.

ANEXO F

Código SQL

Script de creación de la base de datos

```
/* ============================================================ */
/*          FICHERO DE CONSTRUCCIÓN DE LA BASE DE DATOS         */
/* ============================================================ */


/* Crear tablas y con sus correspondientes restricciones */

CREATE TABLE ACTUALIZA (
       peso                INTEGER DEFAULT 5 NOT NULL
                                  CHECK (peso IN (0, 1, 2, 3, 4, 5, 6, 7, 8,
9, 10)),
       nombre              VARCHAR2(20) NOT NULL,
       web                 VARCHAR2(200) NOT NULL,
       cod_concepto        INTEGER NOT NULL
);


ALTER TABLE ACTUALIZA
       ADD  ( PRIMARY KEY (nombre, cod_concepto, web) ) ;


CREATE TABLE ALFANUMERICO (
       identificador       VARCHAR2(20) NOT NULL,
       descripcion         VARCHAR2(400) NOT NULL,
       cod_concepto        INTEGER NOT NULL
);


ALTER TABLE ALFANUMERICO
       ADD  ( PRIMARY KEY (identificador, cod_concepto) ) ;


CREATE TABLE ATRIBUTO (
       nombre              VARCHAR2(20) NOT NULL,
       caducidad           INTEGER DEFAULT 1 NOT NULL,
       cod_concepto        INTEGER NOT NULL,
       identificador       VARCHAR2(20) NOT NULL
);


ALTER TABLE ATRIBUTO
       ADD  ( PRIMARY KEY (nombre, cod_concepto) ) ;


CREATE TABLE CONCEPTO (
       cod_concepto        INTEGER NOT NULL,
       tipo                VARCHAR2(20) NOT NULL
                                  CHECK (tipo IN ('generico', 'instancia'))
);


ALTER TABLE CONCEPTO
       ADD  ( PRIMARY KEY (cod_concepto) ) ;


CREATE TABLE DOMINIO (
       identificador       VARCHAR2(20) NOT NULL,
       tipo                VARCHAR2(20) NOT NULL
                                  CHECK (tipo IN ('numerico',
'alfanumerico'))
);
```

```
ALTER TABLE DOMINIO
       ADD  ( PRIMARY KEY (identificador) ) ;


CREATE TABLE ESPACIO (
       cod_concepto         INTEGER NOT NULL
);


ALTER TABLE ESPACIO
       ADD  ( PRIMARY KEY (cod_concepto) ) ;


CREATE TABLE FUENTE (
       web                  VARCHAR2(200) NOT NULL,
       descripcion          VARCHAR2(400) NOT NULL
);


ALTER TABLE FUENTE
       ADD  ( PRIMARY KEY (web) ) ;


CREATE TABLE GENERICO (
       cod_concepto         INTEGER NOT NULL
);


ALTER TABLE GENERICO
       ADD  ( PRIMARY KEY (cod_concepto) ) ;


CREATE TABLE INSTANCIA (
       cod_concepto         INTEGER NOT NULL
);


ALTER TABLE INSTANCIA
       ADD  ( PRIMARY KEY (cod_concepto) ) ;


CREATE TABLE NUMERICO (
       cota_inferior        INTEGER NOT NULL,
       cota_superior        INTEGER NOT NULL,
       unidad               VARCHAR2(20) NULL,
       identificador        VARCHAR2(20) NOT NULL
);


ALTER TABLE NUMERICO
       ADD  ( PRIMARY KEY (identificador) ) ;


CREATE TABLE REFERENCIA (
       grado                INTEGER DEFAULT 50 NOT NULL
                                   CHECK (grado BETWEEN 0 AND 100),
       cod_concepto         INTEGER NOT NULL,
       cod_termino          VARCHAR2(30) NOT NULL
);


ALTER TABLE REFERENCIA
       ADD  ( PRIMARY KEY (cod_concepto, cod_termino) ) ;
```

```
CREATE TABLE RELACION (
       tipo                 VARCHAR2(20) NOT NULL
       descripcion          VARCHAR2(400) NOT NULL
);


ALTER TABLE RELACION
       ADD  ( PRIMARY KEY (tipo) ) ;


CREATE TABLE RELACIONA (
       concepto1            INTEGER NOT NULL,
       concepto2            INTEGER NOT NULL,
       tipo                 VARCHAR2(20) NOT NULL
                                   CHECK (tipo IN ('es un', 'compuesto de',
'antonimia', 'sinonimia'))
);


ALTER TABLE RELACIONA
       ADD  ( PRIMARY KEY (concepto1, concepto2, tipo) ) ;


CREATE TABLE TERMINO (
       cod_termino          VARCHAR2(30) NOT NULL,
       lenguaje             VARCHAR2(20) NOT NULL
);


ALTER TABLE TERMINO
       ADD  ( PRIMARY KEY (cod_termino) ) ;


CREATE TABLE TIENE1 (
       inicio               DATE NULL,
       fin                  DATE NULL,
       valor                VARCHAR2(20) NOT NULL,
       concepto_espacio     INTEGER NOT NULL,
       nombre               VARCHAR2(20) NOT NULL,
       cod_concepto         INTEGER NOT NULL
);


ALTER TABLE TIENE1
       ADD  ( PRIMARY KEY (concepto_espacio, nombre, cod_concepto) ) ;


CREATE TABLE TIENE2 (
       inicio               DATE NULL,
       fin                  DATE NULL,
       valor                VARCHAR2(20) NOT NULL,
       nombre               VARCHAR2(20) NOT NULL,
       cod_concepto         INTEGER NOT NULL
);


ALTER TABLE TIENE2
       ADD  ( PRIMARY KEY (nombre, cod_concepto) ) ;


ALTER TABLE ACTUALIZA
       ADD  ( FOREIGN KEY (nombre, cod_concepto)
                           REFERENCES ATRIBUTO
                           ON DELETE CASCADE ) ;
```

```
ALTER TABLE ACTUALIZA
       ADD  ( FOREIGN KEY (web)
                           REFERENCES FUENTE
                           ON DELETE CASCADE ) ;


ALTER TABLE ALFANUMERICO
       ADD  ( FOREIGN KEY (cod_concepto)
                           REFERENCES GENERICO
                           ON DELETE CASCADE ) ;


ALTER TABLE ALFANUMERICO
       ADD  ( FOREIGN KEY (identificador)
                           REFERENCES DOMINIO
                           ON DELETE CASCADE ) ;


ALTER TABLE ATRIBUTO
       ADD  ( FOREIGN KEY (identificador)
                           REFERENCES DOMINIO ) ;


ALTER TABLE ATRIBUTO
       ADD  ( FOREIGN KEY (cod_concepto)
                           REFERENCES CONCEPTO
                           ON DELETE CASCADE ) ;


ALTER TABLE ESPACIO
       ADD  ( FOREIGN KEY (cod_concepto)
                           REFERENCES INSTANCIA
                           ON DELETE CASCADE ) ;


ALTER TABLE GENERICO
       ADD  ( FOREIGN KEY (cod_concepto)
                           REFERENCES CONCEPTO
                           ON DELETE CASCADE ) ;


ALTER TABLE INSTANCIA
       ADD  ( FOREIGN KEY (cod_concepto)
                           REFERENCES CONCEPTO
                           ON DELETE CASCADE ) ;


ALTER TABLE NUMERICO
       ADD  ( FOREIGN KEY (identificador)
                           REFERENCES DOMINIO
                           ON DELETE CASCADE ) ;


ALTER TABLE REFERENCIA
       ADD  ( FOREIGN KEY (cod_termino)
                           REFERENCES TERMINO ) ;


ALTER TABLE REFERENCIA
       ADD  ( FOREIGN KEY (cod_concepto)
                           REFERENCES CONCEPTO
                           ON DELETE CASCADE ) ;


ALTER TABLE RELACIONA
       ADD  ( FOREIGN KEY (tipo)
                           REFERENCES RELACION ) ;
```

```
ALTER TABLE RELACIONA
       ADD  ( FOREIGN KEY (concepto2)
                           REFERENCES CONCEPTO
                           ON DELETE CASCADE ) ;


ALTER TABLE RELACIONA
       ADD  ( FOREIGN KEY (concepto1)
                           REFERENCES CONCEPTO
                           ON DELETE CASCADE ) ;


ALTER TABLE TIENE1
       ADD  ( FOREIGN KEY (concepto_espacio)
                           REFERENCES ESPACIO
                           ON DELETE CASCADE ) ;


ALTER TABLE TIENE1
       ADD  ( FOREIGN KEY (nombre, cod_concepto)
                           REFERENCES ATRIBUTO
                           ON DELETE CASCADE ) ;


ALTER TABLE TIENE2
       ADD  ( FOREIGN KEY (nombre, cod_concepto)
                           REFERENCES ATRIBUTO
                           ON DELETE CASCADE ) ;

/* ---------------------------------------------------------------- */


/* Añadir los 'update cascade' que no genera ERwin */

CREATE OR REPLACE TRIGGER TUpdate_Atributo
AFTER UPDATE ON atributo
FOR EACH ROW
BEGIN
   UPDATE actualiza
   SET nombre = :new.nombre,
       cod_concepto = :new.cod_concepto
   WHERE (
          (nombre = :old.nombre) AND
          (cod_concepto = :old.cod_concepto)
         );

   UPDATE tiene1
   SET nombre = :new.nombre,
       cod_concepto = :new.cod_concepto
   WHERE (
          (nombre = :old.nombre) AND
          (cod_concepto = :old.cod_concepto)
         );

   UPDATE tiene2
   SET nombre = :new.nombre,
       cod_concepto = :new.cod_concepto
   WHERE (
          (nombre = :old.nombre) AND
          (cod_concepto = :old.cod_concepto)
         );
END TUpdate_Atributo;
/
```

```
CREATE OR REPLACE TRIGGER TUpdate_Concepto
AFTER UPDATE ON concepto
FOR EACH ROW
BEGIN
   UPDATE referencia
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;

   UPDATE relaciona
   SET concepto1 = :new.cod_concepto
   WHERE concepto1 = :old.cod_concepto;

   UPDATE relaciona
   SET concepto2 = :new.cod_concepto
   WHERE concepto2 = :old.cod_concepto;

   UPDATE atributo
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;

   UPDATE generico
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;

   UPDATE instancia
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;

END TUpdate_Concepto;
/

CREATE OR REPLACE TRIGGER TUpdate_Dominio
AFTER UPDATE ON dominio
FOR EACH ROW
BEGIN
   UPDATE atributo
   SET identificador = :new.identificador
   WHERE identificador = :old.identificador;

   UPDATE numerico
   SET identificador = :new.identificador
   WHERE identificador = :old.identificador;

   UPDATE alfanumerico
   SET identificador = :new.identificador
   WHERE identificador = :old.identificador;
END TUpdate_Dominio;
/

CREATE OR REPLACE TRIGGER TUpdate_Espacio
AFTER UPDATE ON espacio
FOR EACH ROW
BEGIN
   UPDATE tiene1
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;
END TUpdate_Espacio;
/

CREATE OR REPLACE TRIGGER TUpdate_Fuente
AFTER UPDATE ON fuente
FOR EACH ROW
BEGIN
   UPDATE actualiza
   SET web = :new.web
   WHERE web = :old.web;
END TUpdate_Fuente;
/
```

```
CREATE OR REPLACE TRIGGER TUpdate_Generico
AFTER UPDATE ON generico
FOR EACH ROW
BEGIN
   UPDATE alfanumerico
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;
END TUpdate_Generico;
/

CREATE OR REPLACE TRIGGER TUpdate_Instancia
AFTER UPDATE ON instancia
FOR EACH ROW
BEGIN
   UPDATE espacio
   SET cod_concepto = :new.cod_concepto
   WHERE cod_concepto = :old.cod_concepto;
END TUpdate_Instancia;
/

CREATE OR REPLACE TRIGGER TUpdate_Relacion
AFTER UPDATE ON relacion
FOR EACH ROW
BEGIN
   UPDATE relaciona
   SET tipo = :new.tipo
   WHERE tipo = :old.tipo;
END TUpdate_Relacion;
/

CREATE OR REPLACE TRIGGER TUpdate_Termino
AFTER UPDATE ON termino
FOR EACH ROW
BEGIN
   UPDATE referencia
   SET cod_termino = :new.cod_termino
   WHERE cod_termino = :old.cod_termino;
END TUpdate_Termino;
/

/* ---------------------------------------------------------------- */

create sequence seqconcepto START WITH 1;

/* ---------------------------------------------------------------- */


/* Inserta en el correspondiente subtipo tras insertar un concepto */

CREATE OR REPLACE TRIGGER T_concepto
AFTER INSERT ON concepto
FOR EACH ROW
BEGIN
    IF (:new.tipo = 'generico') THEN
      INSERT INTO generico VALUES (:new.cod_concepto);
    ELSE
        IF (:new.tipo = 'instancia') THEN
           INSERT INTO instancia VALUES (:new.cod_concepto);
        ELSE  -- Nunca deberia llegar a este else
           DBMS_OUTPUT.put_line('DISPARADOR T_dominio: Error: tipo no
valido');
        END IF;
    END IF;
END T_concepto;
/
```

Script de creación de los disparadores

```
/* ================================================================ */
/*          FICHERO CON TODOS LOS DISPARADORES DE CONTROL           */
/* ================================================================ */

/* Valida el valor insertado en 'tiene1' */

CREATE OR REPLACE TRIGGER T_validar1
BEFORE INSERT ON tiene1
FOR EACH ROW
DECLARE
excp_validar1      EXCEPTION;
v_dominio          atributo.identificador%TYPE;
v_tipo             dominio.tipo%TYPE;
v_inf              numerico.cota_inferior%TYPE;
v_sup              numerico.cota_superior%TYPE;
v_padre            alfanumerico.cod_concepto%TYPE;
v_concepto         alfanumerico.cod_concepto%TYPE;
v_test             alfanumerico.cod_concepto%TYPE;
v_valido           referencia.cod_termino%TYPE;
num                INTEGER;
BEGIN
    -- obtiene el dominio del atributo
    SELECT DISTINCT identificador INTO v_dominio
    FROM atributo
    WHERE (nombre=:new.nombre);

    -- obtiene el tipo de dominio (numerico vs alfanumerico)
    SELECT tipo INTO v_tipo
    FROM dominio
    WHERE (identificador = v_dominio);

    -- si numerico, obtiene los valores validos (cotas)
    IF (v_tipo = 'numerico') THEN
       SELECT cota_inferior INTO v_inf
       FROM numerico
       WHERE (identificador=v_dominio);

       SELECT cota_superior INTO v_sup
       FROM numerico
       WHERE (identificador=v_dominio);

    -- si alfanumerico, comprueba si el valor pertenece al dominio
    -- es decir, comprueba si existe una instancia con ese valor
    ELSE
       -- inicializa la variable
       v_valido := 'no';

       -- obtiene el codigo del dominio
       SELECT cod_concepto INTO v_padre
       FROM alfanumerico
       WHERE (identificador=v_dominio);

       num := 0;
       -- comprueba si existe una instancia con el valor introducido
       SELECT COUNT (*) INTO num
       FROM referencia
       WHERE (cod_termino = :new.valor);


       IF (num>0) THEN
           SELECT cod_concepto INTO v_concepto
           FROM referencia
           WHERE (cod_termino = :new.valor);

           -- si existe una instancia = valor introducido
           -- comprueba si dicha instancia pertenece al dominio  afectado
```

```
           SELECT concepto1 into v_test
           FROM relaciona
           WHERE ((concepto2=v_padre) AND
                  (concepto1=v_concepto) AND
                  (tipo='es un'));

           -- si existe una instancia con el nombre del valor
           -- ademas esa instancia 'es_un' del concepto que define al
dominio(v_padre)
           IF (v_test = v_concepto) THEN
               v_valido := 'SI';
           END IF;
      END IF;
    END IF;

    -- comprueba si el valor introducido es valido
    IF (v_tipo='numerico') THEN
       IF ((:new.valor <= v_sup) AND (:new.valor >= v_inf)) THEN
          DBMS_OUTPUT.put_line('Valor valido para tabla tiene1');
       ELSE
          RAISE excp_validar1;
       END IF;
    ELSE
       IF (v_valido = 'no') THEN
          RAISE excp_validar1;
       END IF;
    END IF;


EXCEPTION
WHEN excp_validar1 THEN
   RAISE_APPLICATION_ERROR(-20007,'Error => valor incorrecto tabla tiene1');
END T_validar1;
/

/* ---------------------------------------------------------------- */


/* Valida el valor insertado en 'tiene2' */

CREATE OR REPLACE TRIGGER T_validar2
BEFORE INSERT ON tiene2
FOR EACH ROW
DECLARE
excp_validar2      EXCEPTION;
v_dominio          atributo.identificador%TYPE;
v_tipo             dominio.tipo%TYPE;
v_inf              numerico.cota_inferior%TYPE;
v_sup              numerico.cota_superior%TYPE;
v_padre            alfanumerico.cod_concepto%TYPE;
v_concepto         alfanumerico.cod_concepto%TYPE;
v_test             alfanumerico.cod_concepto%TYPE;
v_valido           referencia.cod_termino%TYPE;
num                INTEGER;
BEGIN

    -- obtiene el dominio del atributo
    SELECT DISTINCT identificador INTO v_dominio
    FROM atributo
    WHERE (nombre=:new.nombre);

    -- obtiene el tipo de dominio (numerico vs alfanumerico)
    SELECT tipo INTO v_tipo
    FROM dominio
    WHERE (identificador = v_dominio);
```

```
    -- si numerico, obtiene los valores validos (cotas)
    IF (v_tipo = 'numerico') THEN
       SELECT cota_inferior INTO v_inf
       FROM numerico
       WHERE (identificador=v_dominio);

       SELECT cota_superior INTO v_sup
       FROM numerico
       WHERE (identificador=v_dominio);

    -- si alfanumerico, comprueba si el valor pertenece al dominio
    -- es decir, comprueba si existe una instancia con ese valor
    ELSE
       v_valido := 'no';

       -- obtiene el codigo del dominio
       SELECT cod_concepto INTO v_padre
       FROM alfanumerico
       WHERE (identificador=v_dominio);

       num := 0;
       -- comprueba si existe una instancia con el valor introducido
       SELECT COUNT (*) INTO num
       FROM referencia
       WHERE (cod_termino = :new.valor);

       IF (num>0) THEN
           SELECT cod_concepto INTO v_concepto
           FROM referencia
           WHERE (cod_termino = :new.valor);

           -- si existe una instancia = valor introducido
           -- comprueba si dicha instancia pertenece al dominio afectado
           SELECT concepto1 into v_test
           FROM relaciona
           WHERE ((concepto2=v_padre) AND
                  (concepto1=v_concepto) AND
                  (tipo='es un'));

           -- si existe una instancia con el nombre del valor
           -- ademas esa instancia 'es_un' del concepto que define al
dominio(v_padre)
           IF (v_test = v_concepto) THEN
               v_valido := 'SI';
           END IF;
      END IF;
    END IF;

    -- comprueba si el valor introducido es valido
    IF (v_tipo='numerico') THEN
       IF ((:new.valor <= v_sup) AND (:new.valor >= v_inf)) THEN
          DBMS_OUTPUT.put_line('Valor valido para tabla tiene2');
       ELSE
          RAISE excp_validar2;
       END IF;
    ELSE
       IF (v_valido = 'no') THEN
          RAISE excp_validar2;
       END IF;
    END IF;


EXCEPTION
WHEN excp_validar2 THEN
   RAISE_APPLICATION_ERROR(-20008,'Error => valor incorrecto tabla tiene2');
END T_validar2;
/
```

```
/* ---------------------------------------------------------------- */


/* En la tabla numerico hay que controlar que cota_inferior < cota_superior */

CREATE OR REPLACE TRIGGER T_numerico
BEFORE INSERT ON numerico
FOR EACH ROW
DECLARE
excp_numerico EXCEPTION;
BEGIN
     IF (:new.cota_superior - :new.cota_inferior) <= 0 THEN
      RAISE excp_numerico;
     END IF;
EXCEPTION
WHEN excp_numerico THEN
   RAISE_APPLICATION_ERROR(-20000,'Error => cota superior debe ser mayor que
cota inferior');
END T_numerico;
/

/* ---------------------------------------------------------------- */


/* Controla que en la tabla tiene1 que inicio < fin */

CREATE OR REPLACE TRIGGER T_tiene1
BEFORE INSERT ON tiene1
FOR EACH ROW
DECLARE
excp_tiene1 EXCEPTION;
BEGIN
    IF (:new.inicio - :new.fin) > 0  THEN
       RAISE excp_tiene1;
    END IF;
EXCEPTION
WHEN excp_tiene1 THEN
     RAISE_APPLICATION_ERROR(-20001,'Error => fecha inicio debe ser anterior a
fecha fin');
END T_tiene1;
/

/* ---------------------------------------------------------------- */


/* Controla que en la tabla tiene2 que inicio < fin */

CREATE OR REPLACE TRIGGER T_tiene2
BEFORE INSERT ON tiene2
FOR EACH ROW
DECLARE
excp_tiene2 EXCEPTION;
BEGIN
    IF (:new.inicio - :new.fin) > 0  THEN
       RAISE excp_tiene2;
    END IF;
EXCEPTION
WHEN excp_tiene2 THEN
     RAISE_APPLICATION_ERROR(-20002,'Error => fecha inicio debe ser anterior a
fecha fin');
END T_tiene2;
/

/* ---------------------------------------------------------------- */
```

```
/* En la tabla tiene1 se calcula fin, si no se ha introducido, con la  */
/* ayuda del campo caducidad (expresado en dias). Si solo se introduce */
/* fecha de fin da un error porque no se puede calcular el inicio      */

CREATE OR REPLACE TRIGGER T_fechas1
BEFORE INSERT ON tiene1
FOR EACH ROW
DECLARE
excp_fechas1 EXCEPTION;
aux          INTEGER;
BEGIN
    SELECT caducidad INTO aux
    FROM atributo
    WHERE ((cod_concepto=:new.cod_concepto) AND
           (nombre=:new.nombre));

    IF ((:new.inicio IS NOT NULL) AND (:new.fin IS NULL)) THEN
       :new.fin := (:new.inicio + aux);
    ELSE
        IF ((:new.inicio IS NULL) AND (:new.fin IS NOT NULL)) THEN
           RAISE excp_fechas1;
        END IF;
    END IF;
EXCEPTION
WHEN excp_fechas1 THEN
   RAISE_APPLICATION_ERROR(-20003,'Error => falta fecha inicio');
END T_fechas1;
/

/* ---------------------------------------------------------------- */


/* En la tabla tiene2 se calcula fin, si no se ha introducido, con la  */
/* ayuda del campo caducidad (expresado en dias). Si solo se introduce */
/* fecha de fin da un error porque no se puede calcular el inicio      */

CREATE OR REPLACE TRIGGER T_fechas2
BEFORE INSERT ON tiene2
FOR EACH ROW
DECLARE
excp_fechas2 EXCEPTION;
aux          INTEGER;
BEGIN

    SELECT caducidad INTO aux
    FROM atributo
    WHERE (cod_concepto=:new.cod_concepto AND
           nombre=:new.nombre);

    IF (:new.inicio IS NOT NULL) AND (:new.fin IS NULL) THEN
       :new.fin := (:new.inicio + aux);
    ELSE IF (:new.inicio IS NULL) AND (:new.fin IS NOT NULL) THEN
           RAISE excp_fechas2;
        END IF;
    END IF;
EXCEPTION
WHEN excp_fechas2 THEN
   RAISE_APPLICATION_ERROR(-20004,'Error => falta fecha inicio');
END T_fechas2;
/

/* ---------------------------------------------------------------- */
```

```
/* Si se borra en referencia=> borrar todos los terminos que se han */
/* quedado sin conceptos */

CREATE OR REPLACE TRIGGER T_referencia
AFTER DELETE ON referencia
BEGIN
  DELETE FROM termino
  WHERE cod_termino NOT IN (SELECT cod_termino
                            FROM referencia);
END T_referencia;
/

/* ---------------------------------------------------------------- */


/* Si se borra en referencia,comprobar que ningun concepto se queda sin
termino */

-- paquete para definir las "variables globales"
CREATE OR REPLACE PACKAGE paqueteref AS
  TYPE tref IS TABLE OF referencia%ROWTYPE
                INDEX by BINARY_INTEGER;

v_tref          tref;
v_numentradas   BINARY_INTEGER := 0;
END paqueteref;
/
-- se guarda en una tabla todas las referencias borradas
CREATE OR REPLACE TRIGGER T_auxref
BEFORE DELETE ON referencia
FOR EACH ROW
BEGIN
    paqueteref.v_numentradas := paqueteref.v_numentradas + 1;
    paqueteref.v_tref(paqueteref.v_numentradas).cod_termino :=
:old.cod_termino;
    paqueteref.v_tref(paqueteref.v_numentradas).cod_concepto :=
:old.cod_concepto;
    paqueteref.v_tref(paqueteref.v_numentradas).grado := :old.grado;
END T_auxref;
/
-- si algun concepto se ha quedado sin referencia => insertar referencia
CREATE OR REPLACE TRIGGER T_ref
AFTER DELETE ON referencia
DECLARE
 total         INTEGER := 0;
 numconceptos  INTEGER := 0;
 conceptoaux   concepto.cod_concepto%TYPE;
BEGIN

  FOR v_indexbucle IN 1..paqueteref.v_numentradas LOOP
    conceptoaux := paqueteref.v_tref(v_indexbucle).cod_concepto;

    SELECT COUNT (*) INTO total
    FROM referencia
    WHERE (cod_concepto = conceptoaux);

    SELECT COUNT (*) INTO numconceptos
    FROM concepto
    WHERE (cod_concepto = conceptoaux);

     -- se ha borrado una tupla no permitida para un concepto existente
    IF (total = 0) AND (numconceptos > 0) THEN
          -- rehacer operacion
          INSERT INTO REFERENCIA (cod_termino,cod_concepto,grado) VALUES
          (paqueteref.v_tref(v_indexbucle).cod_termino,
          paqueteref.v_tref(v_indexbucle).cod_concepto,
          paqueteref.v_tref(v_indexbucle).grado);
```

```
          DBMS_OUTPUT.put_line('Error: No se puede borrar la referencia');
    END IF;
  END LOOP;

  paqueteref.v_numentradas := 0;
END T_ref;
/

/* ---------------------------------------------------------------- */


/* Controlar que una fuente no se queda sin atributo que actualizar */

CREATE OR REPLACE TRIGGER T_actualiza
AFTER DELETE ON actualiza
DECLARE
 obsoletas   INTEGER := 0;
BEGIN
  SELECT COUNT(*) INTO obsoletas
  FROM fuente WHERE web NOT IN (SELECT web FROM actualiza);

  IF obsoletas > 0 THEN
     DELETE FROM fuente
     WHERE web NOT IN (SELECT web FROM actualiza);
  END IF;
END T_actualiza;
/

/* ---------------------------------------------------------------- */

/* Tras borrar un concepto que es dominio alfanumerico => borrar dominio
implicado */

CREATE OR REPLACE TRIGGER T_alfanumerico
AFTER DELETE ON alfanumerico
DECLARE
 borrados   INTEGER := 0;
BEGIN

  SELECT COUNT(*) INTO borrados
  FROM dominio
  WHERE (identificador NOT IN (SELECT identificador
                               FROM alfanumerico) AND
         tipo = 'alfanumerico');

   IF borrados > 0 THEN
      DELETE FROM dominio
      WHERE (identificador NOT IN (SELECT identificador
                                   FROM alfanumerico) AND
             tipo = 'alfanumerico');
  END IF;
END T_alfanumerico;
/
```

Script de creación del paquete de los agentes

```
/* ================================================================ */
/*          C a b e c e r a    d e l     p a q u e t e              */
/* ================================================================ */
CREATE OR REPLACE PACKAGE paqueteagentes AS


   TYPE tiporegistro1 IS RECORD (Idconcepto   concepto.cod_concepto%TYPE,
                                 nombre       atributo.nombre%TYPE,
                                 espacio      tiene1.concepto_espacio%TYPE);

   TYPE ttabla2 IS TABLE OF tiporegistro1 INDEX BY BINARY_INTEGER;


   PROCEDURE pr_caducadas1 (p_tablacaducadas OUT ttabla2,
                            p_numentradas IN OUT BINARY_INTEGER);


   TYPE tiporegistro2 IS RECORD (Idconcepto   concepto.cod_concepto%TYPE,
                                 nombre       atributo.nombre%TYPE);

   TYPE ttabla6 IS TABLE OF tiporegistro2 INDEX BY BINARY_INTEGER;


   PROCEDURE pr_caducadas2 (p_tablacaducadas OUT ttabla6,
                            p_numentradas IN OUT BINARY_INTEGER);


   PROCEDURE pr_modifpeso (p_web           actualiza.web%TYPE,
                           p_concepto      actualiza.cod_concepto%TYPE,
                           p_nombre        actualiza.nombre%TYPE,
                           p_peso          actualiza.peso%TYPE);


   TYPE ttabla1 IS TABLE OF actualiza.web%TYPE
                   INDEX BY BINARY_INTEGER;


   PROCEDURE pr_fuente (p_concepto        IN  concepto.cod_concepto%TYPE,
                        p_nombre          IN  atributo.nombre%TYPE,
                        p_mitabla         OUT ttabla1,
                        p_numentradas  IN OUT BINARY_INTEGER);


   PROCEDURE pr_modiftiene1 (p_concepto      concepto.cod_concepto%TYPE,
                             p_nombre        atributo.nombre%TYPE,
                             p_espacio       tiene1.concepto_espacio%TYPE,
                             p_valor         tiene1.valor%TYPE,
                             p_inicio        tiene1.inicio%TYPE,
                             p_fin           tiene1.fin%TYPE);


   PROCEDURE pr_modiftiene2 (p_concepto      concepto.cod_concepto%TYPE,
                             p_nombre        atributo.nombre%TYPE,
                             p_valor         tiene1.valor%TYPE,
                             p_inicio        tiene1.inicio%TYPE,
                             p_fin           tiene1.fin%TYPE);


   PROCEDURE pr_inserttiene1 (p_concepto      concepto.cod_concepto%TYPE,
                              p_nombre        atributo.nombre%TYPE,
                              p_espacio       concepto.cod_concepto%TYPE,
                              p_valor         tiene1.valor%TYPE,
                              p_inicio        tiene1.inicio%TYPE,
                              p_fin           tiene1.fin%TYPE);
```

```
PROCEDURE pr_inserttiene2 (p_concepto      concepto.cod_concepto%TYPE,
                           p_nombre        atributo.nombre%TYPE,
                           p_valor         tiene2.valor%TYPE,
                           p_inicio        tiene2.inicio%TYPE,
                           p_fin           tiene2.fin%TYPE);

PROCEDURE pr_Deletecaducadas;



TYPE ttabla3 IS TABLE OF concepto.cod_concepto%TYPE
                INDEX BY BINARY_INTEGER;

TYPE ttabla4 IS TABLE OF termino.cod_termino%TYPE
                INDEX BY BINARY_INTEGER;


TYPE ttabla5 IS TABLE OF referencia%ROWTYPE
                INDEX BY BINARY_INTEGER;



PROCEDURE pr_consulta1 (p_concepto      tiene1.cod_concepto%TYPE,
                        p_nombre        tiene1.nombre%TYPE,
                        p_espacio       espacio.cod_concepto%TYPE,
                        p_output IN OUT tiene1.valor%TYPE);


PROCEDURE pr_consulta2 (p_concepto      tiene1.cod_concepto%TYPE,
                        p_nombre        tiene1.nombre%TYPE,
                        p_output IN OUT tiene1.valor%TYPE);


PROCEDURE pr_qtermino (p_concepto        IN concepto.cod_concepto%TYPE,
                       p_tablaterminos  OUT ttabla4,
                       p_numentradas IN OUT BINARY_INTEGER);


PROCEDURE pr_ambito (p_concepto        IN concepto.cod_concepto%TYPE,
                     p_tablaout    IN OUT ttabla3,
                     p_numentradas IN OUT BINARY_INTEGER);


PROCEDURE pr_contexto (p_concepto        IN concepto.cod_concepto%TYPE,
                       p_tablaout    IN OUT ttabla3,
                       p_numentradas IN OUT BINARY_INTEGER);



PROCEDURE pr_qconcepto0 (p_termino        IN termino.cod_termino%TYPE,
                         p_tablaout       OUT ttabla5,
                         p_numentradas IN OUT BINARY_INTEGER);



TYPE tiporegistro IS RECORD (IdConcepto concepto.cod_concepto%TYPE,
                             encaje NUMBER);


TYPE ttabla8 IS TABLE OF tiporegistro INDEX BY BINARY_INTEGER;


PROCEDURE pr_padre (p_hijo           IN concepto.cod_concepto%TYPE,
                    p_padre      IN OUT concepto.cod_concepto%TYPE,
                    p_tienepadre    OUT INTEGER);
```

```
   PROCEDURE pr_pasado(p_padre            IN concepto.cod_concepto%TYPE,
                       p_tconcepto    IN OUT ttabla5,
                       p_numfilas     IN OUT BINARY_INTEGER,
                       p_valor           OUT referencia.grado%TYPE,
                       p_pasado          OUT INTEGER);


   PROCEDURE pr_distancia (p_concepto1    IN concepto.cod_concepto%TYPE,
                           p_concepto2    IN concepto.cod_concepto%TYPE,
                           p_dist        OUT INTEGER);


   PROCEDURE pr_existe(p_concepto       IN concepto.cod_concepto%TYPE,
                       p_tabla          IN ttabla8,
                       p_numfilas       IN BINARY_INTEGER,
                       p_existe        OUT INTEGER);


   PROCEDURE pr_qconcepto1 (p_termino          IN termino.cod_termino%TYPE,
                            p_contexto         IN termino.cod_termino%TYPE,
                            p_tablaresult  IN OUT ttabla8,
                            p_numentradas  IN OUT BINARY_INTEGER);



   PROCEDURE pr_qconcepto2 (p_termino          IN termino.cod_termino%TYPE,
                            p_tablacontexto    IN ttabla4,
                            p_numcontextos IN OUT INTEGER,
                            p_tablaresult     OUT ttabla8,
                            p_numentradas  IN OUT BINARY_INTEGER);



END paqueteagentes;
/

/* ================================================================ */
/*           C u e r p o    d e l     p a q u e t e                 */
/* ================================================================ */
CREATE OR REPLACE PACKAGE BODY paqueteagentes AS

/* Obtiene todas las tuplas caducadas de tiene1 */
   PROCEDURE pr_caducadas1 (p_tablacaducadas  OUT ttabla2,
                            p_numentradas  IN OUT BINARY_INTEGER) IS

      -- variables locales
      v_fecha             DATE;

   BEGIN
      /* fecha (SELECT SYSDATE FROM DUAL;)*/
      SELECT SYSDATE INTO v_fecha
      FROM DUAL;

      p_numentradas := 0; -- inicializa la tabla

      FOR v_fila IN (SELECT cod_concepto,nombre,concepto_espacio,fin
                     FROM TIENE1
                     WHERE (fin IS NOT NULL)) LOOP

        -- guarda las tuplas caducadas en la tabla auxiliar.
        IF (v_fila.fin < v_fecha) THEN
           p_numentradas := p_numentradas + 1;
           p_tablacaducadas(p_numentradas).Idconcepto := v_fila.cod_concepto;
           p_tablacaducadas(p_numentradas).nombre := v_fila.nombre;
           p_tablacaducadas(p_numentradas).espacio := v_fila.concepto_espacio;
        END IF;
      END LOOP;
   END pr_caducadas1;
```

```
/* ---------------------------------------------------------------- */


/* Obtiene todas las tuplas caducadas de tiene2 */
   PROCEDURE pr_caducadas2 (p_tablacaducadas  OUT ttabla6,
                            p_numentradas  IN OUT BINARY_INTEGER) IS

      -- variables locales
      v_fecha             DATE;

   BEGIN
      /* fecha (SELECT SYSDATE FROM DUAL;)*/
      SELECT SYSDATE INTO v_fecha
      FROM DUAL;

      p_numentradas := 0; -- inicializa la tabla

      FOR v_fila IN (SELECT cod_concepto,nombre,fin
                     FROM TIENE2
                     WHERE (fin IS NOT NULL)) LOOP

        -- guarda las tuplas caducadas en la tabla auxiliar.
        IF (v_fila.fin < v_fecha) THEN
           p_numentradas := p_numentradas + 1;
           p_tablacaducadas(p_numentradas).Idconcepto := v_fila.cod_concepto;
           p_tablacaducadas(p_numentradas).nombre := v_fila.nombre;
        END IF;
      END LOOP;
   END pr_caducadas2;

/* ---------------------------------------------------------------- */

/* Actualiza el peso de las fuentes de un atributo */

   PROCEDURE pr_modifpeso (p_web           actualiza.web%TYPE,
                       p_concepto      actualiza.cod_concepto%TYPE,
                       p_nombre        actualiza.nombre%TYPE,
                       p_peso          actualiza.peso%TYPE) AS
   BEGIN
     UPDATE actualiza
     SET peso=p_peso
     WHERE ((cod_concepto=p_concepto) AND
            (nombre=p_nombre) AND
            (web=p_web));
   END pr_modifpeso;

/* ---------------------------------------------------------------- */


/* Procedimiento FUENTE: Obtiene todas las fuentes web que
                         actualizan a un atributo */

   PROCEDURE pr_fuente (p_concepto        IN  concepto.cod_concepto%TYPE,
                        p_nombre          IN  atributo.nombre%TYPE,
                        p_mitabla         OUT ttabla1,
                        p_numentradas  IN OUT BINARY_INTEGER) IS

   BEGIN
      -- se inicializa el indice de la tabla.
      p_numentradas := 0;

      FOR v_web IN (SELECT web
                    FROM actualiza
                    WHERE cod_concepto=p_concepto AND
                          nombre=p_nombre) LOOP
        -- guarda la fila obtenida en la tabla auxiliar.
        p_numentradas := p_numentradas + 1;
        p_mitabla(p_numentradas) := v_web.web;
```

```
      END LOOP;
   END pr_fuente;

/* ---------------------------------------------------------------- */


/* Actualiza los valores y fechas caducadas de la tabla tiene1 */

   PROCEDURE pr_modiftiene1 (p_concepto      concepto.cod_concepto%TYPE,
                         p_nombre        atributo.nombre%TYPE,
                         p_espacio       tiene1.concepto_espacio%TYPE,
                         p_valor         tiene1.valor%TYPE,
                         p_inicio        tiene1.inicio%TYPE,
                         p_fin           tiene1.fin%TYPE) AS
   BEGIN
      UPDATE tiene1
      SET  valor=p_valor,inicio=p_inicio,fin=p_fin
      WHERE ((cod_concepto=p_concepto) AND
             (nombre=p_nombre) AND
             (concepto_espacio=p_espacio));
   END pr_modiftiene1;

/* ---------------------------------------------------------------- */


/* Actualiza los valores y fechas caducadas de la tabla tiene2 */

   PROCEDURE pr_modiftiene2 (p_concepto      concepto.cod_concepto%TYPE,
                         p_nombre        atributo.nombre%TYPE,
                         p_valor         tiene1.valor%TYPE,
                         p_inicio        tiene1.inicio%TYPE,
                         p_fin           tiene1.fin%TYPE) AS
   BEGIN
      UPDATE tiene2
      SET  valor=p_valor,inicio=p_inicio,fin=p_fin
      WHERE ((cod_concepto=p_concepto) AND
             (nombre=p_nombre));
   END pr_modiftiene2;

/* ---------------------------------------------------------------- */


/* Insertar una tupla en la tabla tiene1 */

   PROCEDURE pr_inserttiene1 (p_concepto      concepto.cod_concepto%TYPE,
                              p_nombre        atributo.nombre%TYPE,
                              p_espacio       concepto.cod_concepto%TYPE,
                              p_valor         tiene1.valor%TYPE,
                              p_inicio        tiene1.inicio%TYPE,
                              p_fin           tiene1.fin%TYPE) AS
   BEGIN
      INSERT INTO tiene1
(cod_concepto,nombre,concepto_espacio,valor,inicio,fin) VALUES
(p_concepto,p_nombre,p_espacio,p_valor,p_inicio,p_fin);
   END pr_inserttiene1;

/* ----------------------------------------------------------------  */
```

```
/* Insertar una tupla en la tabla tiene2 */

   PROCEDURE pr_inserttiene2 (p_concepto      concepto.cod_concepto%TYPE,
                              p_nombre        atributo.nombre%TYPE,
                              p_valor         tiene2.valor%TYPE,
                              p_inicio        tiene2.inicio%TYPE,
                              p_fin           tiene2.fin%TYPE) AS
   BEGIN
      INSERT INTO tiene2 (cod_concepto,nombre,valor,inicio,fin) VALUES
(p_concepto,p_nombre,p_valor,p_inicio,p_fin);
   END pr_inserttiene2;

/* ----------------------------------------------------------------  */


/* Borra todas las tuplas caducadas de tiene1 y tiene2 */
   PROCEDURE pr_Deletecaducadas IS

      -- variables locales
      v_fecha             DATE;

   BEGIN
      /* fecha (SELECT SYSDATE FROM DUAL;)*/
      SELECT SYSDATE INTO v_fecha
      FROM DUAL;


      /* Procesando tabla tiene1 */
      FOR v_fila IN (SELECT cod_concepto,nombre,concepto_espacio,fin
                     FROM TIENE1
                     WHERE (fin IS NOT NULL)) LOOP

        IF (v_fila.fin < v_fecha) THEN
           DELETE FROM tiene1 WHERE (cod_concepto = v_fila.cod_concepto AND
                                     nombre = v_fila.nombre AND
                                     concepto_espacio =
v_fila.concepto_espacio);
        END IF;
      END LOOP;


      /* Procesando tabla tiene2 */
      FOR v_fila IN (SELECT cod_concepto,nombre,fin
                     FROM TIENE2
                     WHERE (fin IS NOT NULL)) LOOP


        IF (v_fila.fin < v_fecha) THEN
           DELETE FROM tiene2 WHERE (cod_concepto = v_fila.cod_concepto AND
                                     nombre = v_fila.nombre);

        END IF;
      END LOOP;


   END pr_Deletecaducadas;


/*
================================================================================
== */
```

```
/* Procedimiento CONSULTA1: obtiene un valor de un atributo de la tabla tiene1
(un espacio) */

   PROCEDURE pr_consulta1 (p_concepto      tiene1.cod_concepto%TYPE,
                     p_nombre        tiene1.nombre%TYPE,
                     p_espacio       espacio.cod_concepto%TYPE,
                     p_output IN OUT tiene1.valor%TYPE) IS

    num   INTEGER := 0;
   BEGIN

      SELECT COUNT(*) INTO num
      FROM tiene1
      WHERE ((cod_concepto=p_concepto) AND
             (nombre=p_nombre) AND
             (concepto_espacio=p_espacio));

      IF num > 0 THEN
          SELECT valor INTO p_output
          FROM tiene1
          WHERE ((cod_concepto=p_concepto) AND
                 (nombre=p_nombre) AND
                 (concepto_espacio=p_espacio));
      ELSE
        p_output := 'variable sin valor';
      END IF;
   END pr_consulta1;

/* ----------------------------------------------------------------  */


/* Procedimiento CONSULTA2: Obtiene un valor de un atributo de la tabla tiene2
*/

   PROCEDURE pr_consulta2 (p_concepto      tiene1.cod_concepto%TYPE,
                     p_nombre        tiene1.nombre%TYPE,
                     p_output IN OUT tiene1.valor%TYPE) IS

     num     INTEGER := 0;
   BEGIN

     SELECT COUNT(*) INTO num
     FROM tiene2
     WHERE ((cod_concepto=p_concepto) AND
            (nombre=p_nombre));

     IF num > 0 THEN
         SELECT valor INTO p_output
         FROM tiene2
         WHERE ((cod_concepto=p_concepto) AND
                (nombre=p_nombre));
     ELSE
        p_output := 'variable sin valor';
     END IF;
   END pr_consulta2;

/* ----------------------------------------------------------------  */
```

```
/* Procedimiento AMBITO: Obtiene el ambito de un concepto */

   PROCEDURE pr_ambito (p_concepto        IN concepto.cod_concepto%TYPE,
                        p_tablaout    IN OUT ttabla3,
                        p_numentradas IN OUT BINARY_INTEGER) IS

   BEGIN
      FOR v_num IN (SELECT concepto2
                    FROM relaciona
                    WHERE ((concepto1=p_concepto) AND
                           (tipo='es un'))) LOOP

          p_numentradas := p_numentradas + 1;
          p_tablaout(p_numentradas) := v_num.concepto2;
      END LOOP;
   END pr_ambito;

/* ----------------------------------------------------------------  */


/* Procedimiento CONTEXTO: Obtiene el contexto de un concepto */

   PROCEDURE pr_contexto(p_concepto        IN concepto.cod_concepto%TYPE,
                         p_tablaout    IN OUT ttabla3,
                         p_numentradas IN OUT BINARY_INTEGER) IS

   BEGIN
      FOR v_concepto IN (SELECT concepto1
                         FROM relaciona
                         WHERE ((concepto2=(SELECT concepto2
                                            FROM relaciona
                                            WHERE (concepto1=p_concepto AND
tipo='es un'))) AND
                                (tipo='es un') AND
                                (concepto1<>p_concepto))) LOOP

         -- guarda la fila obtenida en la tabla auxiliar.
         p_numentradas := p_numentradas + 1;
         p_tablaout(p_numentradas) := v_concepto.concepto1;
      END LOOP;

     -- el contexto también está formado por su ámbito
     pr_ambito (p_concepto,p_tablaout,p_numentradas);
   END pr_contexto;

/* ----------------------------------------------------------------  */
/* ----------------------------------------------------------------  */


/* Procedimiento QTERMINO: Obtiene lOs terminos de un concepto */

   PROCEDURE pr_qtermino (p_concepto        IN  concepto.cod_concepto%TYPE,
                          p_tablaterminos   OUT ttabla4,
                          p_numentradas  IN OUT BINARY_INTEGER) IS

   BEGIN
      -- se inicializa el indice de la tabla
      p_numentradas := 0;

      FOR v_termino IN (SELECT cod_termino
                        FROM referencia
                        WHERE cod_concepto=p_concepto) LOOP

        -- guarda la fila obtenida en la tabla auxiliar
        p_numentradas := p_numentradas + 1;
        p_tablaterminos(p_numentradas) := v_termino.cod_termino;
      END LOOP;
   END pr_qtermino;
```

```
/* ---------------------------------------------------------------  */


/* Procedimiento QCONCEPTO0: Obtiene lOs conceptos de un termino */

   PROCEDURE pr_qconcepto0 (p_termino        IN termino.cod_termino%TYPE,
                            p_tablaout       OUT ttabla5,
                            p_numentradas IN OUT BINARY_INTEGER) IS

   BEGIN
      -- se inicializa el indice de la tabla.
      p_numentradas := 0;

      FOR v_concept IN (SELECT cod_concepto,grado
                        FROM referencia
                        WHERE (cod_termino = p_termino)) LOOP

         -- guarda la fila obtenida en la tabla auxiliar.
         p_numentradas := p_numentradas + 1;
         p_tablaout(p_numentradas).cod_concepto := v_concept.cod_concepto;
         p_tablaout(p_numentradas).grado := v_concept.grado;
      END LOOP;
   END pr_qconcepto0;

/* ---------------------------------------------------------------  */


/*  Procedimientos necesarios en la ejecucion del qconcepto1          */
/* ==========================================================         */



/* Procedimiento PADRE: Obtiene el "padre" de un concepto en una     */
/*                      jerarquia                                    */

   PROCEDURE pr_padre (p_hijo               IN concepto.cod_concepto%TYPE,
                       p_padre          IN OUT concepto.cod_concepto%TYPE,
                       p_tienepadre        OUT INTEGER) IS
   BEGIN

     p_tienepadre := 0;

     SELECT COUNT(*) INTO p_tienepadre
     FROM relaciona
     WHERE (concepto1 = p_hijo AND
            tipo = 'es un');

     -- si existe un padre
     IF (p_tienepadre > 0) THEN
        SELECT concepto2 INTO p_padre
        FROM relaciona
        WHERE (concepto1 = p_hijo AND
               tipo = 'es un');
     END IF;
   END pr_padre;

/* ---------------------------------------------------------------  */
```

```
/* Procedimiento PASADO: Comprueba si el concepto1 y concepto2 tienen
antecesores comunes */

   PROCEDURE pr_pasado(p_padre            IN concepto.cod_concepto%TYPE,
                       p_tconcepto    IN OUT ttabla5,
                       p_numfilas     IN OUT BINARY_INTEGER,
                       p_valor           OUT referencia.grado%TYPE,
                       p_pasado          OUT INTEGER) IS

   BEGIN
     p_pasado := 0; -- booleano (0 = false)

      -- recorre la tabla de los antecesores de concepto1
      FOR i IN 1..p_numfilas LOOP
          -- si concepto1 y concepto2 comparten antecesores
          IF p_tconcepto(i).cod_concepto = p_padre THEN
             p_valor := p_tconcepto(i).grado;
             p_pasado := 1; -- booleano (1 = true)
          END IF;
      END LOOP;
   END pr_pasado;

/* ----------------------------------------------------------------  */


/* Procedimiento DISTANCIA: Calcula la distancia entre dos conceptos, donde
distancia es el  */
/*                          número de "saltos" desde un concepto a otro en una
jerarquía     */
/*                          definida por relaciones 'es un' entre conceptos
*/

   PROCEDURE pr_distancia (p_concepto1   IN concepto.cod_concepto%TYPE,
                         p_concepto2   IN concepto.cod_concepto%TYPE,
                            p_dist       OUT INTEGER) IS

   -- Declaracion de variables locales
   v_calculado       INTEGER := 0; -- booleano (0 = false)
   v_hijo            concepto.cod_concepto%TYPE;
   v_continuar       INTEGER := 1; --booleano (1 = true)
   v_padre           concepto.cod_concepto%TYPE;
   v_tienepadre      INTEGER;
   v_tconcept1       ttabla5;
   v_numfilast1      BINARY_INTEGER := 0;
   v_tconcept2       ttabla5;
   v_numfilast2      BINARY_INTEGER := 0;
   v_valor           referencia.grado%TYPE;
   v_pasado          INTEGER;
   v_pos             INTEGER;


   BEGIN

     p_dist := 0;

     -- si los dos conceptos son el mismo, distancia = 0
     IF p_concepto1 = p_concepto2 THEN
        p_dist := 0;
        v_calculado := 1;  -- booleano (1 = true)

     ELSE -- concepto1<>concepto2
        v_hijo := p_concepto1;

        -- ascender hasta la raiz de la jerarquia o hasta que encuentre a
concepto2
        WHILE v_continuar = 1 LOOP
           pr_padre (v_hijo,v_padre,v_tienepadre);
```

```
            -- si tiene padre
            IF v_tienepadre > 0 THEN
               IF v_padre = p_concepto2 THEN -- si el padre es concepto2
                  v_continuar := 0; -- booleano (0 = false)
                  -- si el primer padre es concepto2
                  IF v_numfilast1 = 0 THEN
                     p_dist := 1;
                  ELSE
                     -- en la variable grado se guarda el recorrido parcial
                     p_dist := v_tconcept1(v_numfilast1).grado + 1;
                  END IF;
                  v_calculado := 1; -- booleano (1 = true)
               ELSE -- seguir ascendiendo en la jerarquia
                  v_numfilast1 := v_numfilast1 + 1;
                  v_tconcept1(v_numfilast1).cod_concepto := v_padre;
                  if (v_numfilast1 - 1) = 0 then -- tabla vacia
                      v_tconcept1(v_numfilast1).grado := 1;
                  else
                      v_tconcept1(v_numfilast1).grado :=
v_tconcept1(v_numfilast1 - 1).grado + 1;
                  end if;

                  -- nuevo hijo para la siguiente iteracion
                  v_hijo := v_padre;
               END IF; -- padre=concepto2
            ELSE -- si no tiene padre
               v_continuar := 0; -- booleano (0 = false)
            END IF; -- si tiene padre
         END LOOP; -- bucle concepto1

         -- procesamiento del concepto2

         -- si no se ha calculado la distancia aun
         IF v_calculado = 0 THEN
            v_hijo := p_concepto2;
            v_continuar := 1; -- booleano (1 = true)

            -- ascender hasta la raiz de la jerarquia o hasta que encuentre a
concepto1
            WHILE v_continuar = 1 LOOP
               pr_padre (v_hijo,v_padre,v_tienepadre);

               -- si tiene padre
               IF v_tienepadre > 0 THEN
                  IF v_padre = p_concepto1 THEN
                     v_continuar := 0; -- booleano (0 = false)
                     -- si el primer padre es concepto1
                     IF v_numfilast2 = 0 THEN
                        p_dist := 1;
                     ELSE
                       -- en la variable grado se guarda el recorrido parcial
                       p_dist := v_tconcept2(v_numfilast2).grado + 1;
                     END IF;
                     v_calculado := 1; -- booleano (1 = true)
                  ELSE -- seguir ascendiendo en la jerarquia
                     -- si concepto1 has pasado por el padre

pr_pasado(v_padre,v_tconcept1,v_numfilast1,v_valor,v_pasado);

                     IF v_pasado = 1 THEN -- si tienen antecesores comunes
                        v_continuar := 0; -- booleano (0 = false)
                        -- si es el primer padre de concepto2
                        IF v_numfilast2=0 THEN
                           p_dist := v_valor + 1;
                        ELSE
                           p_dist := v_valor + v_tconcept2(v_numfilast2).grado
+ 1;
                        END IF; -- asignar distancia
```

```
                     v_calculado := 1; -- booleano (1 = true)
                  ELSE -- sin antecesores comunes; seguir ascendiendo por la
jerarquia
                     v_numfilast2 := v_numfilast2 + 1;
                     v_tconcept2(v_numfilast2).cod_concepto := v_padre;
                     -- almacena en la variable grado el recorrido parcial
                     if (v_numfilast2 - 1) = 0 then -- tabla vacia
                        v_tconcept2(v_numfilast2).grado := 1;
                     else
                         v_tconcept2(v_numfilast2).grado :=
v_tconcept2(v_numfilast2 - 1).grado + 1;
                     end if;
                     -- nuevo hijo para la siguiente iteracion
                     v_hijo := v_padre;
                  END IF; -- si concepto1 ha pasado por padre
               END IF; -- si padre = concepto1
            ELSE -- no tiene padre
               v_continuar := 0; -- booleano (0 = false)
            END IF; -- si tiene padre
         END LOOP; -- bucle concepto2

      END IF; -- tratamiento concepto2


      -- si todavia no se tiene la distancia entre los conceptos
      IF v_calculado = 0 THEN
         -- si tienen superpadre comun
         IF (v_numfilast1>0 AND v_numfilast2>0) AND
            (v_tconcept1(v_numfilast1).cod_concepto =
v_tconcept2(v_numfilast2).cod_concepto)  THEN
            -- distancia = suma desde superpadre a cada concepto
             p_dist := v_tconcept1(v_numfilast1).grado +
v_tconcept2(v_numfilast2).grado;
         ELSE -- pertenecen a grafos inconexos
            p_dist := NULL;
         END IF; -- superpadre comun
      END IF; -- calculando


   END IF; -- concepto1=concepto2

END pr_distancia;

/* ----------------------------------------------------------------  */


/* Procedimiento EXISTE: Comprueba si un concepto pertenece a una tabla */

   PROCEDURE pr_existe(p_concepto       IN concepto.cod_concepto%TYPE,
                       p_tabla          IN ttabla8,
                       p_numfilas       IN BINARY_INTEGER,
                       p_existe        OUT INTEGER) IS

   -- Declaracion de variables locales
   v_continuar         INTEGER; -- booleano (1 = true)
   v_indexloop         INTEGER;
   BEGIN

      v_continuar := 1;
      v_indexloop := 0; -- para que compruebe correctamente la condicion del
while
      p_existe := 0; -- boolean (0 = false)

      WHILE (v_continuar = 1 AND v_indexloop <= p_numfilas) LOOP
        v_indexloop := 1; -- para que acceda correctamente a la tabla

        -- si tabla vacia
        IF p_numfilas = 0 THEN
```

```
           v_continuar := 0; -- booleano (0 = false)
           p_existe := 0;  -- booleano (0 = false)
         ELSE
            IF p_tabla(v_indexloop).IdConcepto = p_concepto THEN
              p_existe := 1; -- booleano (1 = true)
              v_continuar := 0; -- booleano (0 = false)
           END IF;
         END IF;
           v_indexloop := v_indexloop + 1;
      END LOOP;
   END pr_existe;

/* ----------------------------------------------------------------  */


/* Procedimiento QCONCEPTO1: Obtiene un codigo de concepto para un termino y
un contexto dado */
/* ATENCION: la llamada debe tener p_numentradas = 0*/

   PROCEDURE pr_qconcepto1 (p_termino          IN termino.cod_termino%TYPE,
                            p_contexto         IN termino.cod_termino%TYPE,
                            p_tablaresult  IN OUT ttabla8,
                            p_numentradas  IN OUT BINARY_INTEGER) IS

   -- Declaracion de variables locales
   v_tconceptos        ttabla5;
   v_numtconceptos     BINARY_INTEGER;
   v_taux1             ttabla8;
   v_numfilasaux1      BINARY_INTEGER;
   v_tcontextos        ttabla5;
   v_numtcontextos     BINARY_INTEGER;
   v_distancia         INTEGER;
   v_media             NUMBER;
   v_encaje            NUMBER;
   v_minimo            NUMBER;
   v_existe            INTEGER;

   BEGIN

       -- inicializa la tabla
       v_numfilasaux1 := 0;

       -- obtiene el/los cod_concepto referenciados por p_termino
       pr_qconcepto0 (p_termino,v_tconceptos,v_numtconceptos);

       -- si solo referencia a un concepto => fin
       IF v_numtconceptos = 1 THEN
          -- almacena el resultado en una tabla auxiliar
          v_numfilasaux1 := v_numfilasaux1 + 1;
          v_taux1(v_numfilasaux1).IdConcepto := v_tconceptos(1).cod_concepto;
          v_taux1(v_numfilasaux1).encaje := v_tconceptos(1).grado;
       ELSE -- si el termino referencia a mas de un concepto
          -- obtiene el/los cod_concepto (contextos) referenciados por
p_contexto
          pr_qconcepto0 (p_contexto,v_tcontextos,v_numtcontextos);

          -- para cada contexto
          FOR i IN 1..v_numtcontextos LOOP
              -- para cada concepto
              FOR j IN 1..v_numtconceptos LOOP

                  -- obtiene la distancia entre cada pareja termino-contexto
                  pr_distancia
(v_tconceptos(j).cod_concepto,v_tcontextos(i).cod_concepto,v_distancia);

                  IF v_distancia IS NULL THEN
                     -- se le asigna un valor grande que se comporta como
distancia infinita
```

```
                    v_encaje := 900;
                  ELSE
                    -- cálculo del encaje
                    v_media := ((v_tconceptos(j).grado +
v_tcontextos(i).grado)/2)/100;
                    v_encaje := v_distancia * (1 - v_media);
                  END IF;

                  -- almacena el resultado en una tabla auxiliar
                  v_numfilasaux1 := v_numfilasaux1 + 1;
                  v_taux1(v_numfilasaux1).IdConcepto :=
v_tconceptos(j).cod_concepto;
                  v_taux1(v_numfilasaux1).encaje := v_encaje;

              END LOOP; -- conceptos
          END LOOP; -- contextos

       END IF; -- si termino solo referencia a un concepto


       -- Obtiene el valor minimo de encaje para todos los pares termino-
contexto tratados
       v_minimo := 32000; -- valor suficientemente grande
       FOR i IN 1..v_numfilasaux1 LOOP
           IF v_taux1(i).encaje < v_minimo THEN
              v_minimo := v_taux1(i).encaje;
           END IF;
       END LOOP;

       -- Seleccionar el/los concepto/s con un menor valor de "encaje"
       -- (encaje equivale a una menor distancia pero influye tambien el grado
de cómo referencia)

       FOR j IN 1..v_numfilasaux1 LOOP
           IF v_taux1(j).encaje = v_minimo THEN
             -- antes de insertar se comprueba si ya esta ese concepto como
resultado

pr_existe(v_taux1(j).IdConcepto,p_tablaresult,p_numentradas,v_existe);
             IF v_existe = 0 THEN
                p_numentradas := p_numentradas + 1;
                p_tablaresult(p_numentradas).IdConcepto :=
v_taux1(j).IdConcepto;
                p_tablaresult(p_numentradas).encaje := v_taux1(j).encaje;
             END IF; -- si existe
           END IF;
       END LOOP;

   END pr_qconcepto1;

/* ----------------------------------------------------------------  */


/*  Procedimientos necesarios en la ejecucion del qconcepto2          */
/* ==========================================================         */
```

```
/* Procedimiento QCONCEPTO2: Obtiene un codigo de concepto para un termino y
un conjunto de terminos-contexto */


   PROCEDURE pr_qconcepto2 (p_termino          IN termino.cod_termino%TYPE,
                            p_tablacontexto    IN ttabla4,
                            p_numcontextos IN OUT INTEGER,
                            p_tablaresult     OUT ttabla8,
                            p_numentradas  IN OUT BINARY_INTEGER) IS

   -- Declaracion de variables locales
   v_tconceptos        ttabla5;
   v_numtconceptos     BINARY_INTEGER;
   v_contexto          termino.cod_termino%TYPE;
   v_taux2             ttabla8;
   v_numfilasaux2      BINARY_INTEGER;
   v_minimo            NUMBER;
   v_existe            INTEGER;

   BEGIN

       -- inicializa las tablas
       p_numentradas := 0;
       v_numfilasaux2 := 0;

       -- obtiene el/los cod_concepto referenciados por p_termino
       pr_qconcepto0 (p_termino,v_tconceptos,v_numtconceptos);

       -- si solo referencia a un concepto => fin
       IF v_numtconceptos = 1 THEN
          -- almacena el resultado en una tabla auxiliar
          v_numfilasaux2 := v_numfilasaux2 + 1;
          v_taux2(v_numfilasaux2).IdConcepto := v_tconceptos(1).cod_concepto;
          v_taux2(v_numfilasaux2).encaje := v_tconceptos(1).grado;

       ELSE -- si el termino referencia a mas de un concepto
          -- llama a qconcepto1 para cada "contexto" introducido
          FOR i IN 1..p_numcontextos LOOP
             v_contexto := p_tablacontexto(i);
             pr_qconcepto1 (p_termino,v_contexto,v_taux2,v_numfilasaux2);
          END LOOP;
       END IF;

       -- Obtiene el valor minimo de encaje para todos los pares termino-
contexto tratados
       v_minimo := 32000; -- se inicializa a un numero suficientemente grande

       FOR i IN 1..v_numfilasaux2 LOOP
           IF v_taux2(i).encaje < v_minimo THEN
              v_minimo := v_taux2(i).encaje;
           END IF;
       END LOOP;

       -- Seleccionar el/los concepto/s con un menor valor de "encaje"
       -- (encaje equivale a una menor distancia pero influye tambien el grado
de cómo referencia)
       FOR j IN 1..v_numfilasaux2 LOOP
           IF v_taux2(j).encaje = v_minimo THEN
             -- antes de insertar se comprueba si ya esta ese concepto como
resultado

pr_existe(v_taux2(j).IdConcepto,p_tablaresult,p_numentradas,v_existe);
             IF v_existe = 0 THEN
                p_numentradas := p_numentradas + 1;
                p_tablaresult(p_numentradas).IdConcepto :=
v_taux2(j).IdConcepto;
                p_tablaresult(p_numentradas).encaje := v_taux2(j).encaje;
             END IF; -- si existe
```

```
            END IF;
        END LOOP;

   END pr_qconcepto2;


END paqueteagentes;
/
```

Script de creación del paquete del editor

Por razones de espacio y por la sencillez del código implicado, no se incluye en este libro el código que ejecuta las operaciones de editor de conocimiento.