José Andrés Asensio

Un Sistema de Representación del Conocimiento Basado en Mediación Ontológica

Tesis Doctoral

Directores

Dr. Luis Iribarne
Dr. Nicolás Padilla

Departamento de Informática
Grupo de Informática Aplicada

UNIVERSIDAD DE ALMERÍA

Departamento de Informática
Universidad de Almería

Un Sistema de Representación del Conocimiento Basado en Mediación Ontológica

TESIS DOCTORAL

Presentada por

José Andrés Asensio Cortés

para optar al grado de
Doctor Ingeniero en Informática

Dirigida por

Dr. D. Luis Iribarne Martínez
Profesor Titular de Universidad
Departamento de Informática
Universidad de Almería

Dr. D. Nicolás Padilla Soriano
Profesor Titular de Universidad
Departamento de Informática
Universidad de Almería

Almería, a 21 de Junio de 2013

Editado por:

ISBN: 978-84-616-4597-8
Depósito legal: AL 486-2013
Autoedición: Autor
Portada: José Andrés Asensio
Impresión: Murex Factoría de Color

Impreso en España

Mayo de 2013

Tesis Doctoral

Un Sistema de Representación del Conocimiento Basado en Mediación Ontológica

José Andrés Asensio Cortés

Este documento ha sido generado con LaTeX.

Todas las *Figuras* y *Tablas* contenidas en
el presente documento son originales.

Un Sistema de Representación del Conocimiento
Basado en Mediación Ontológica

José Andrés Asensio Cortés
Departamento de Informática
Grupo de Investigación de Informática Aplicada (TIC-211)
Universidad de Almería
Almería, a 21 de Junio de 2013

http://www.ual.es/personal/jacortes/
http://acg.ual.es/

A mis padres, a mis abuelos,
a mi compañera de viaje...

Agradecimientos

Sin duda, para mí esta es la página más compleja. Me resulta inevitable esbozar una sonrisa cada vez que recuerdo aquellas tardes de reunión, aquellas amenas discusiones... El tiempo no era un impedimento, como tampoco lo era el lugar y los recursos: una pequeña servilleta de papel durante un café era capaz de vislumbrar una idea que más tarde se traduciría en una nueva línea de trabajo. El motivo, muy sencillo: estábamos haciendo algo que realmente nos satisfacía y con lo que disfrutábamos. Claro está que no todo ha sido un camino exento de dificultades, también han existido momentos menos buenos, que ahora aparecen difuminados en mi memoria. Finaliza una etapa y comienza otra con nuevas ilusiones, nuevas perspectivas, nuevos retos..., no antes sin mostrar mi agradecimiento a todos los que de un modo u otro han estado presentes en ella.

En primer lugar, me gustaría mencionar a Luis Iribarne y Nicolás Padilla, mis amigos, compañeros y directores de este trabajo, por haberme dado la oportunidad de ser partícipe de sus proyectos, por tantas cosas que me han aportado a lo largo de estos años, ya no sólo desde el punto de vista académico y profesional, sino también desde el punto de vista humano, así como por su impulso motivador y dedicación desinteresada.

Además quisiera dar las gracias a Cristina Vicente-Chicote, por su entrega y colaboración, sin olvidar los buenos momentos que hemos pasado; a Juan Francisco Inglés por la ayuda que ha brindado; a Rafael Corchuelo y a su grupo, por habernos recibido, escuchado y también aportado su granito de arena a este trabajo; y a Antonio Vallecillo por sus sabios consejos y reflexiones.

A Javi, Jesús, José Francisco y al resto de amigos y compañeros del Grupo de Informática Aplicada, a los compañeros del antiguo Departamento de Lenguajes y Computación, ahora Departamento de Informática, y a los compañeros de los proyectos en los que he participado, por el tiempo que hemos pasado juntos trabajando y disfrutando.

Un reconocimiento muy especial a mis padres, por su apoyo para alcanzar siempre las metas que me he fijado, ya que sin ellos no hubiera sido posible; a mis abuelos, que aunque ya no pueden ver esta realidad, unos porque nos cuidan y ayudan desde el cielo y la abuela por su enfermedad, yo de buena tinta sé que les hubiera hecho mucha ilusión ver este logro de su nieto. A mis tíos Pedro, Elena y Juan, y a mi prima Alda, que me han alentado a seguir adelante. A los padres de mi compañera de viaje, Pedro y Chelo, que en la distancia también han creído en mí y en lo que estaba haciendo. Y a los demás familiares que de un modo u otro me han ayudado.

No, no me he olvidado de ti, Patricia, la persona que más intensamente ha vivido conmigo esta etapa. Por tu infinita paciencia y comprensión, por tu dedicación a mí, por tu confianza y apoyo en los momentos delicados... Por todo eso que sólo tú y yo sabemos.

A mis compañeros de trabajo en la Unidad, sin olvidar a Antonio Becerra,

por infundirme ánimos; a Mercedes y a John por su paciencia con el Inglés, y a la Universidad, en general, por su labor.

Y como no, gracias también a esos pequeños infatigables y anónimos que, en realidad, son los que más horas han pasado junto a mí.

Por último, mencionar a los proyectos de investigación "*SOLERES: un sistema de información espacio-temporal para la gestión medioambiental basado en redes neuronales, agentes y componentes software*" (TIN2006-06698/TIN2007-61497), "*Una metodología para la recuperación y explotación de información medioambiental mediante interfaces de usuario evolutivas y cooperativas*" (TIN2010-15588) y "Desarrollo de un agente web inteligente de información" (P10-TIC-6114), que han subvencionado en su mayor parte este trabajo, y a la *Red de Información Ambiental de Andalucía (REDIAM)/EGMASA* por facilitarnos la información medioambiental en la que nos hemos apoyado.

A todos vosotros, de corazón, mi más profundo y sincero agradecimiento...

José Andrés Asensio Cortés
Departamento de Informática
Grupo de Informática Aplicada
Universidad de Almería
Almería, 2013

Contenido

Índice de Figuras

Índice de Tablas

Prólogo

Uno de los mayores retos que plantea esta *sociedad de la información* en la que nos encontramos inmersos es hacer frente a la ingente cantidad de información que se maneja en casi cualquier ámbito. Los *Sistemas de Información* (*Information Systems*, *IS*) tratan de dar solución a este problema. Concretamente, los *Sistemas de Información basados en la Web* (*Web-based Information Systems*, *WIS*) favorecen la eliminación de las barreras espacio-temporales y facilitan una comunicación ubicua y asíncrona a la amplia variedad de usuarios (administradores, técnicos, políticos, etc.) que hacen uso de ellos, proporcionándoles información interactiva y recursos gráficos que facilitan la resolución de problemas y dan soporte a sus procesos de toma de decisiones. Aún así, estos usuarios sufren con bastante frecuencia una sobrecarga de información, viéndose obligados a discriminar entre los contenidos.

La dificultad que plantean los procesos de búsqueda y recuperación de la información es uno de los principales escollos que presentan estos sistemas debido fundamentalmente al gran volumen que gestionan, además de la heterogeneidad de la misma y la integración de múltiples fuentes o repositorios donde se almacena. Para abordar este problema se han desarrollado y aplicado multitud de técnicas desde diferentes áreas: representación de información, almacenamiento y recuperación de información, filtrado de información, estudios de comportamiento en la búsqueda de información, interacción persona-ordenador, etc. Por otro lado, el cumplimiento de determinados aspectos a los que deben dar soporte, como integridad, escalabilidad, accesibilidad, adaptabilidad, extensibilidad, etc., obligan al empleo de normas y estándares para su construcción. En este contexto, el objetivo principal del presente trabajo de investigación consiste en el planteamiento de un marco de trabajo que facilite el diseño y desarrollo de un sistema que aporte una solución a los problemas planteados. Este sistema ha sido bautizado con el nombre de *Sistema de Representación del Conocimiento basado en Mediación*, representado con el acrónimo *TKRS* procedente de su traducción al inglés (*Trading-based Knowledge Representation System*).

Este marco se ha construido siguiendo las directrices tanto de la *Ingeniería Dirigida por Modelos* (*Model-Driven Engineering*, *MDE*) como de la *Ingeniería Dirigida por Ontologías* (*Ontology-Driven Engineering*, *ODE*), y permite la separación de la arquitectura del sistema de su implementación. Un editor gráfico implementado específicamente para el diseño de la arquitectura va a facilitar la obtención de un modelo del sistema en unos sencillos pasos. Por otro lado, se va a definir el modelo de un repositorio de implementaciones de los componentes del sistema con el objeto de que un tercer mo-

delo, el modelo de configuración del sistema —para el cual también se ha desarrollado un *Lenguaje Específico del Dominio* (*Domain Specific Language*, *DSL*)—, relacione cada componente de la arquitectura diseñada con su correspondiente implementación. Para que los modelos de la arquitectura y del repositorio puedan ser referenciados y utilizados por los modelos de configuración, se les van a aplicar transformaciones *Modelo-A-Modelo* (*Model-To-Model*, *M2M*). Finalmente, a partir del modelo de configuración se va a obtener una instancia concreta del sistema (código ejecutable) para la plataforma específica seleccionada, mediante una transformación *Modelo-A-Texto* (*Model-To-Text*, *M2T*).

En relación a la arquitectura de *TKRS*, está constituida por un conjunto de nodos cada uno de ellos formado, a su vez, por una serie de módulos con una funcionalidad determinada. De entre estos módulos destaca uno en torno al cual gira el diseño del sistema y que basa su funcionamiento en un modelo de mediación ontológico, denominado *Onto-Trader* (otra aportación de la propuesta). Este modelo extiende la *función de mediación* tradicional de *ODP* [ITU-T and ISO/IEC, 1998], sustituyendo las especificaciones de los servicios ofrecidos por los objetos o componentes por *metadatos* de la información del dominio del sistema. Estos metadatos, que se definen mediante una ontología de datos y son almacenados en un repositorio gestionado por el servicio de mediación, representan el tercer nivel de la estructura de datos del sistema: en un primer nivel reside la información de origen; en un segundo nivel se encuentran metadatos de la información anterior, representados también mediante una ontología de datos; y en un tercer nivel figuran metadatos de los metadatos anteriores, es decir, meta-metadatos (utilizados por el servicio de mediación). El cometido de este servicio de mediación consiste en proporcionar la integración de las fuentes de información disponibles, así como de los componentes del propio sistema, y mejorar los procesos de búsqueda y recuperación de dicha información (en este sentido, sigue el modelo *Query-Searching/Recovering-Response*, *QS/RR*, una extensión del modelo tradicional *Query/Response*). La interacción con las diferentes interfaces del modelo de mediación se ha definido mediante una serie de ontologías de servicio/proceso —del uso de estas ontologías junto con la de datos es de donde procede el calificativo de modelo de mediación *ontológico*—.

La propuesta presentada a lo largo del documento está acompañada de una serie de escenarios para verificar los procesos de gestión y, sobre todo, consulta de información según los modelos operativos de mediación identificados: reflexión, delegación y federación. También, se analiza el caso de estudio de un sistema, *SOLERES-KRS*, diseñado y desarrollado bajo el marco presentado. *SOLERES-KRS* es el subsistema responsable de la gestión del conocimiento en *SOLERES*, un *Sistema de Información para la Gestión Medioambiental* (*Environmental Management Information System*, *EMIS*) específicamente centrado en el estudio de mapas ecológicos, imágenes de satélite y sus clasificaciones, que surge de un proyecto multidisciplinar del Plan Nacional de I+D+i. Este proyecto —soportado por la aplicación, integración y desarrollo de trabajos en imágenes de satélite, redes neuronales, sistemas cooperativos basados en arquitecturas multi-agente y sistemas software basados en componentes comerciales— tenía como objetivo estudiar la posibilidad y viabilidad de correlacionar variables medioambientales, usadas en la creación de mapas ecológicos, con la información de satélite o de vuelos, que finalmente quedó demostrado. La correlación permitiría crear mapas ecológicos automáticos con un coste temporal, económico y humano muy inferior a como se venían

realizando hasta el momento: mapas que se generaban en intervalos de tiempo muy distanciados (años), mediante trabajo de campo y con un alto coste económico y humano.

Y sin más preámbulos, a continuación, se recogen los objetivos generales de este trabajo, junto con las aportaciones que se realizan y se describe la estructura que sigue el documento para dar paso a su contenido.

Objetivos y contribución

Como se ha indicado, el resultado de esta investigación persigue la obtención de un marco que permita el diseño y desarrollo de un sistema de representación del conocimiento basado en un modelo de mediación ontológico. Los objetivos generales del trabajo son los siguientes:

(1) El diseño de un modelo de mediación ontológico y la definición de una ontología de servicio/proceso por cada una de las interfaces que implementa para la interacción con él.

(2) La integración de este modelo de mediación en un sistema de representación del conocimiento, este último modelado también mediante ontologías de datos.

(3) La definición de un marco de trabajo desarrollado bajo la *Ingeniería Dirigida por Modelos* (*MDE*) y la *Ingeniería Dirigida por Ontologías* (*ODE*) que permita:

   (a) La especificación y el modelado de la arquitectura del sistema.

   (b) El modelado de un repositorio de implementaciones de los componentes del sistema.

   (c) El modelado de la configuración del sistema y, a partir de él, su despliegue en el entorno dado.

(4) El análisis de un caso de estudio desarrollado según el marco de trabajo anterior y empleando la tecnología de agentes software y *Sistemas Multi-Agente* (*Multi-Agent System*, *MAS*).

(5) La evaluación y validación de la propuesta.

La contribución de este trabajo plantea una propuesta de solución que ofrece básicamente un marco para el diseño y desarrollo de sistemas que gestionan grandes volúmenes de información —cuyo procesamiento no se termina de ajustar a las herramientas de procesamiento y análisis actuales— y necesitan la integración de múltiples fuentes o repositorios, sin dejar de lado aspectos de integridad, disponibilidad, accesibilidad, etc., y tratando de aislar su diseño de la tecnología utilizada para su implementación.

Las principales aportaciones de este trabajo aparecen a continuación:

— Se ha realizado el diseño de un modelo de mediación ontológico denominado *OntoTrader* con una arquitectura *stand-alone* que implementa las interfaces *Lookup*,

Register y *Admin*. Utiliza un repositorio de metadatos obtenidos a partir de la información del dominio definido en base a una ontología de datos (que sustituye las tradicionales especificaciones de los objetos o componentes y de los servicios que ofrecen), para facilitar la integración de diferentes fuentes de información, componentes o sistemas, y mejorar los procesos de búsqueda y recuperación de dicha información. La interacción con este modelo de mediación se ha definido también mediante ontologías de servicio/proceso —una por cada interfaz implementada—.

— Se ha integrado el modelo *OntoTrader* en el diseño de la arquitectura distribuida de un sistema —denominado *Sistema de Representación del Conocimiento basado en Mediación* (*TKRS*)—, compuesta por nodos con diferentes módulos y con una estructura de datos formada por tres niveles. Además, se ha propuesto un marco que sigue en mayor o menor medida las líneas propuestas tanto por la *Ingeniería Dirigida por Modelos* como por la *Ingeniería Dirigida por Ontologías* y que permite, mediante modelos, la definición de esta arquitectura de forma independiente de su implementación. También se ha creado un modelo de repositorio de implementaciones de los componentes del sistema y otro modelo de configuración que permite asociar cada componente con su correspondiente implementación y establecer los parámetros de operación.

— Se ha realizado un análisis del modelo *OntoTrader* en diferentes escenarios básicos de los procesos de gestión y consulta de información de *TKRS* (basados en el mecanismo *QS/RR*), y se ha desarrollado un entorno de experimentación *on-line* para la evaluación y verificación de la propuesta. También, se ha presentado el análisis del caso de estudio del subsistema *SOLERES-KRS* —diseñado y desarrollado bajo este marco mediante la tecnología de agentes software—, que refuerza la validez de la propuesta.

Estructura del documento

Este documento se encuentra organizado, por este orden, en cinco *Capítulos*, dos *Anexos*, un listado de *Acrónimos* y un listado con las *Referencias bibliográficas* utilizadas.

El *Capítulo 1* pretende dar una visión general de la *Ingeniería Dirigida por Modelos* y de la *Ingeniería Dirigida por Ontologías*, realizando un breve repaso por los principales pilares sobre los que se sustenta el trabajo, los *Sistemas de Información* y los *servicios de mediación*, además de la tecnología de *Agentes software*. El *Capítulo 2* describe la propuesta de un modelo de mediación ontológico (*OntoTrader*), presentando sus características o propiedades, además de los detalles del diseño de su arquitectura y de las ontologías asociadas a sus interfaces. La integración de este modelo en un *Sistema de Representación del Conocimiento basado en Mediación* (*Trading-based Knowledge Representation System*, *TKRS*) aparece en el *Capítulo 3*; en este *Capítulo* se muestra un marco con los detalles del diseño de la arquitectura del sistema, junto con el diseño de un repositorio de implementaciones de *TKRS* y un modelo de configuración capaz de relacionar la arquitectura del sistema con una implementación concreta. En el *Capítulo*

4 se analiza el desarrollo e implementación del sistema *SOLERES-KRS* siguiendo las directrices marcadas en el *Capítulo* anterior y se realiza una evaluación y validación de la propuesta. Para finalizar, el *Capítulo 5* recoge los resultados y conclusiones de este trabajo, en forma de aportaciones, limitaciones y líneas de trabajo futuras.

Todos los *Capítulos* presentan una estructura similar. Cada uno de ellos comienza con una *Sección* de *Introducción y conceptos relacionados* (salvo el *Capítulo 5*); a continuación, se desarrollan las *Secciones* propias del mismo y finaliza con las de *Trabajos relacionados* (a excepción del *Capítulo 1*) y *Resumen y conclusiones*. Como complemento a ciertas *Secciones*, se han incluido dos *Anexos*: el *Anexo A*, que contiene el desarrollo completo de la especificación de la arquitectura de *TKRS*, y el *Anexo B*, que incluye el modelo conceptual de los metadatos utilizados por el sistema *SOLERES-KRS*. Al final del documento aparece un listado de *Acrónimos* usados en la redacción y un apartado de *Bibliografía* con las referencias bibliográficas del texto.

Capítulo 1

Paradigmas de la Ingeniería del Software

Capítulo 1

Paradigmas de la Ingeniería del Software

Contenidos

El presente trabajo de investigación tiene como meta principal el establecimiento de un marco inspirado en los paradigmas de la *Ingeniería Dirigida por Modelos* (*Model-Driven Engineering*, *MDE*) y la *Ingeniería Dirigida por Ontologías* (*Ontology-Driven Engineering*, *ODE*) para el diseño y desarrollo de *Sistemas de Representación del Conocimiento basados en Mediación ontológica*, apuntado ya en el *Prólogo*. Dadas las características observadas en los *Sistemas de Información basados en la Web* (*Web-based Information Systems*, *WIS*) y las dificultades que presentan fundamentalmente en cuanto a la integración y a la búsqueda y recuperación de la información, de forma muy resumida, el planteamiento se basa, por un lado, en una arquitectura alrededor de un *modelo de mediación* y, por otro, en la representación tanto del conocimiento como de la funcionalidad del mismo mediante *ontologías*. Para ello, este *Capítulo* pretende realizar un breve recorrido por los fundamentos de ambas ingenierías, los sistemas de representación del conocimiento y los servicios de mediación para sentar las bases de la propuesta desarrollada en los siguientes *Capítulos*. También, se describen los *Agentes Software* como una tecnología útil para la implementación de este tipo de sistemas.

El *Capítulo* se organiza en siete secciones. Comienza realizando una presentación general del trabajo e introduciendo los principales elementos en los que se sustenta (analizados con más detalle en las siguientes *Secciones*) dentro de la *Sección 1.1*. La *Sección 1.2* trata los *Sistemas de Información*, su clasificación y la representación del conocimiento en ellos. Seguidamente, en la *Sección 1.3*, se ofrece una visión general de la *Ingeniería Dirigida por Modelos*, se describe una propuesta concreta de arquitectura y se especifican algunos aspectos de las transformaciones de modelos. De forma análoga, la *Sección 1.4* muestra una perspectiva general de la *Ingeniería Dirigida por Ontologías* y presenta algunas herramientas para la especificación y el tratamiento de ontologías. Los detalles del servicio de mediación (roles, interfaces, etc.) y sus limitaciones se analizan en la *Sección 1.5*. En la *Sección 1.6* se presenta la tecnología de *Agentes Software* y *Sistemas Multi-Agente* —utilizada en la implementación del caso de uso del *Capítulo 4*—, y se realiza un breve recorrido por los principales lenguajes y entornos de desarrollo relacionados. Para finalizar, en la *Sección 1.7*, aparece un breve resumen junto con las conclusiones del presente *Capítulo*.

1.1. Introducción y conceptos relacionados

La *sociedad de la información* necesita sistemas capaces de dar una respuesta adecuada a sus necesidades y a la problemática que plantean la integración no sólo de la información sino también de las aplicaciones que la sustentan, en primer lugar, y los procesos de búsqueda y recuperación de dicha información, en segundo lugar. Aquí es donde juegan un papel fundamental los *Sistemas de Información basados en la Web*, que presentan en muchos casos como características más destacadas la gestión de grandes volúmenes de información y el soporte a un amplio número de usuarios, por lo general, con diferentes perfiles y diferentes niveles de acceso a la información, que esperan de ellos una herramienta en la que sustentar su toma de decisiones, en numerosas ocasiones, crítica.

Ante tal desafío, el complejo diseño de estos sistemas requiere el empleo de mecanismos, métodos y técnicas estandarizados para desarrollar un producto final de calidad. En este sentido, los servicios de mediación tradicionales han realizado una gran aportación en lo que a integración de componentes software se refiere. El trabajo de investigación que aquí se presenta apuesta por utilizar los servicios de mediación por un lado, para proporcionar esta interoperabilidad entre los objetos o componentes del propio sistema e, incluso, externos a él y, por otro, adaptarlos con el objetivo de facilitar la integración de información y mejorar el citado proceso de búsqueda y recuperación de la misma. Para dar soporte a esta doble perspectiva, se propone una extensión de la *función de mediación* de *ODP* [ITU-T and ISO/IEC, 1998] en la que se reemplazan las especificaciones de servicios por un conjunto de metadatos de la información que gestiona el sistema en el que media.

Este modelo de mediación se va a calificar con el adjetivo *ontológico* por dos razones: la primera de ellas, por el uso de ontologías para representar dichos metadatos (concretamente, los conceptos y las relaciones existentes entre ellos) —que se van a denominar *ontologías de datos*— y, la segunda, por el manejo de otro tipo de ontologías —en este caso denominadas de *servicio/proceso*—, para modelar la funcionalidad de las interfaces implementadas por el servicio de mediación y, por extensión, la funcionalidad del sistema en su conjunto. La utilización de las ontologías viene justificada por las facilidades que presentan, respectivamente, para la integración de información heterogénea y para definir tanto el comportamiento como los protocolos de interacción de los componentes del sistema. Este *modelo de mediación ontológico* (que se va a denominar *OntoTrader*) se va a convertir en el núcleo de la estructura distribuida de un *Sistema de Representación del Conocimiento* que se va a conocer con el nombre de *TKRS*. Como se analizará en la *Sección 3.2.1*, este sistema plantea una arquitectura de datos constituida por tres niveles de información compatible con este modelo de mediación: (1) un primer nivel formado por toda la información del dominio del sistema; (2) un segundo nivel de *metadatos* obtenidos a partir de la información del nivel anterior y representados también mediante una ontología, para agilizar los procesos de búsqueda y recuperación de la información; y (3) un tercer nivel de *meta-metadatos* sobre el nivel anterior que recoge la información gestionada por el servicio de mediación.

Además, se establece un marco para el diseño y desarrollo de este tipo de sistemas según las directrices de la *Ingeniería Dirigida por Modelos*, más concretamente siguiendo el enfoque *MDA* (*Model-Driven Architecture*) propuesto por *OMG* [OMG, 2003]. Se van a definir unos *modelos independientes de la plataforma* (*Platform Independent Model*, *PIM*) para representar, por una parte, la arquitectura del sistema y, por otra, un repositorio con implementaciones de sus componentes para, mediante una serie de transformaciones de estos modelos a un modelo de configuración del sistema ya *específico de la plataforma* (*Platform Specific Model*, *PSM*), obtener finalmente una instancia real del mismo en un entorno dado. Todo esto con la ayuda de herramientas diseñadas específicamente para facilitar la definición de los modelos, como por ejemplo, un editor gráfico para modelar la arquitectura en unos sencillos pasos o un *Lenguaje Específico del Dominio* (*Domain Specific Language*, *DSL*) para especificar la configuración del sistema. Y, aunque no se puede afirmar en un sentido estricto que el trabajo sigue el enfoque de la *Ingeniería Dirigida por Ontologías*, en un sentido amplio se puede decir que se "inspira"

en ella, pues las diferentes ontologías que se van a definir proporcionan un vocabulario común a los desarrolladores del sistema favoreciendo una adecuada interpretación y uso de los conceptos a lo largo del ciclo de vida del desarrollo del sistema.

La *Figura 1.1* trata de sintetizar gráficamente todo este marco. En la base, se encuentran la *Ingeniería Dirigida por Modelos* y la *Ingeniería Dirigida por Ontologías* sobre las que se construyen, en primer lugar, el modelo de mediación ontológico *OntoTrader* y, en segundo lugar, el *Sistema de Representación del Conocimiento TKRS* fundamentado en el anterior.

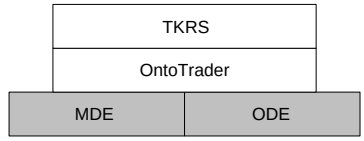


Figura 1.1: Marco general de la propuesta.

Una vez introducida la propuesta de un modo general y antes de entrar en detalles, se pretende realizar un breve repaso de los conceptos principales de la *Representación del Conocimiento*, de la *Ingeniería Dirigida por Modelos*, de la *Ingeniería Dirigida por Ontologías* y de los *servicios de mediación*, con el objetivo de clarificar esta idea general y sentar las bases del contenido de los siguientes *Capítulos*. También se dedica una *Sección* a presentar una visión general sobre los *Agentes Software* que, por las características que poseen y las aportaciones que pueden realizar a un sistema, se han utilizado para la implementación del caso de estudio analizado en el *Capítulo 4*.

1.2. Sistemas de Información y representación del conocimiento

De forma tradicional los *Sistemas de Información* (*Information Systems*, *IS*) se han definido desde dos puntos de vista, uno estructural y otro funcional [Hirschheim et al., 1995]: (i) desde una perspectiva estructural, "... un *Sistema de Información* consiste en una colección de personas, procesos, datos, modelos, tecnología y lenguaje parcialmente formalizado, formando una estructura cohesiva que cubre algún propósito organizacional o función..."; (ii) desde una perspectiva funcional, "... un *Sistema de Información* es un medio implementado tecnológicamente para el propósito de grabar, almacenar y difundir expresiones lingüísticas, así como para dar soporte al proceso de inferencia. A través de la realización de estas funciones elementales, los *Sistemas de Información* facilitan la creación y el intercambio de significados para cubrir propósitos definidos socialmente como el control, la toma de decisiones y la argumentación...". Para el propósito final de esta investigación, que es el diseño de un *Sistema de Representación del Conocimiento*, es preciso mencionar que no se van a abordar aspectos relacionados directamente con las personas (como las interfaces de usuario) ni con la inferencia de conocimiento —aspecto soportado por las ontologías—.

Existe una amplia tipología de *Sistemas de Información*, muchos de ellos *basados en la Web* (*Web-based Information Systems*, *WIS*). Por ejemplo en las grandes organizaciones se pueden encontrar *Sistemas de Información* según el nivel organizacional: a *nivel operativo* figuran los *Sistemas de Procesamiento de Transacciones* (*Transaction Processing Systems*, *TPS*) —que automatizan tareas operativas de la organización—; a *nivel administrativo* destacan los *Sistemas de Apoyo a la Toma de Decisiones* (*Decision Support Systems*, *DSS*) [Lukasheh et al., 2001] —que proporcionan información interactiva a sus usuarios para dar soporte a los procesos de toma de decisiones— o los *Sistemas de Información Gerencial* (*Management Information Systems*, *MIS*); a *nivel estratégico* los *Sistemas de Apoyo a Ejecutivos* (*Executive Support Systems*, *ESS*) —que ayudan a adquirir ventajas competitivas sobre la competencia—, etc.

Un aspecto importante a tener presente en el desarrollo de *Sistemas de Información* es la *representación del conocimiento*. Ésta es un área de la *Inteligencia Artificial* que guarda una importante relación con la inferencia o el razonamiento (*Knowledge Representation and Reasoning*, *KRR*). Surge ante la necesidad de lograr comportamientos inteligentes por parte de los ordenadores, siendo capaces de obtener por sí mismos conclusiones del conocimiento almacenado. Para tal fin, en sus orígenes [Sowa, 2000] se utilizaron variedad de métodos y técnicas, como heurísticas, redes neuronales, sistemas expertos, etc. Más adelante comenzaron a surgir lenguajes formales de representación como *Prolog* (basado en lógica formal) o *KL-ONE* [Brachman and Schmolze, 1985], hasta llegar a *SGML* (*Standard Generalized Markup Language*) o *XML* (*eXtensible Markup Language*). El desarrollo de la web semántica, más reciente, ha favorecido el desarrollo de nuevos lenguajes de representación y estándares, como *RDF* (*Resource Description Framework*), *RDF Schema*, *OIL* (*Ontology Interchange Language*), *DAML+OIL* (*DARPA Agent Markup Language + OIL*), *OWL* (*Web Ontology Language*), etc. (comentados en la *Sección 1.4.1*). De nuevo, es preciso destacar que la propuesta no entra en aspectos relacionados con la inferencia o el razonamiento, sino que se centra en una forma de entender la estructura y representación de la información.

Como se podía intuir, otro aspecto a tener en cuenta vinculado a la representación del conocimiento consiste en la organización y gestión del mismo, especialmente en aquellos sistemas que manejan información a gran escala y requieren la coordinación e integración de múltiples fuentes de conocimiento (normalmente, repositorios de datos distribuidos por alguna causa, como geolocalización, disponibilidad o tolerancia a fallos, costos...), sumado a las restricciones de los propios recursos. Por ejemplo, los *Sistemas de Información Geográfica* (*Geographic Information Systems*, *GIS*) [ISO, 2004] —una rama de los sistemas *DSS*, que hacen uso de información geográfica (explotaciones ganaderas, cultivos, acuíferos, etc.) para construir mapas y otros recursos gráficos que faciliten la resolución de problemas de planificación y gestión geográfica—, a menudo operan con multitud de bases de datos que manejan información muy heterogénea y se encuentran localizadas en diferentes emplazamientos).

Esto ha dado pie a que surjan conceptos como *Big Data* [Chaudhuri, 2012], que se refiere a grandes colecciones de datos que deben ser procesadas con ciertas limitaciones temporales razonables y que no se ajustan perfectamente a las herramientas de procesamiento actuales o necesitan algún tipo de análisis que las herramientas existentes no pueden proporcionar fácilmente: tanto los *Sistemas de Gestión de Bases de Datos* (*Database*

Management Systems, *DBMS*) como otros *frameworks* que dan soporte al procesamiento de grandes colecciones de datos, presentan inconvenientes en un aspecto u otro de los dos mencionados anteriormente [Madden, 2012]. Pese a esto, es notoria la evolución que han sufrido en este ámbito las herramientas y tecnologías, desde las tradicionales bases de datos distribuidas, los *Data Grids* formados por componentes de bajo nivel responsables de almacenar, mantener y recuperar los datos, y servicios de más alto nivel ofrecidos a los usuarios, que integran los componentes anteriores [Branco et al., 2010], etc., a soluciones recientes que van más allá como el *Cloud Computing*, que ofrece la infraestructura computacional necesaria (compartiendo recursos, infraestructuras, plataformas, software, procesos de negocio...) para mantener y procesar estos conjuntos de datos, así como su integración y explotación [Schadt et al., 2010].

Por lo tanto, queda patente que la representación, organización y gestión del conocimiento es una pieza clave en el diseño y desarrollo de los *Sistemas de Información* y depende mucho de la tecnología que se pretenda utilizar. En este sentido, la propuesta que aquí se plantea trata de aislar la tecnología —gracias a la separación *PIM/PSM* que proporciona *MDA*, como se verá en la siguiente *Sección*— ofreciendo un marco para el desarrollo de sistemas que contempla los aspectos anteriormente mencionados de accesibilidad, integración, disponibilidad, integridad, etc. de la información, y se basa en un modelo de mediación ontológico que extiende el estándar de mediación de *OMG* —por este motivo el sistema se ha bautizado como *Sistema de Representación del Conocimiento basado en Mediación*, (*Trading-based Knowledge Representation System*, *TKRS*)—.

El caso de estudio presentado en el *Capítulo 4* analiza el sistema *SOLERES*, un *Sistema de Información para la Gestión Medioambiental* (*Environmental Management Information System*, *EMIS*) —un caso especial de *GIS*— formado por dos subsistemas: (i) *SOLERES-KRS* (*Knowledge Representation System*), responsable de la gestión de información medioambiental, y (ii) *SOLERES-HCI* (*Human-Computer Interaction*), relacionado con las interfaces para la explotación de dicha información. Pues concretamente, el subsistema *SOLERES-KRS* se ha diseñado siguiendo el marco propuesto para el desarrollo de *TKRS*. La *Figura 1.2* muestra la relación entre los diferentes tipos de sistemas mencionados.

1.3. Ingeniería Dirigida por Modelos

La *Ingeniería Dirigida por Modelos* (*Model-Driven Engineering*, *MDE*) surge como una evolución en el desarrollo del software, que comienza con el desarrollo convencional (lenguajes de primera, segunda y tercera generación), continúa con el desarrollo estructurado y el desarrollo orientado a objetos/componentes hasta llegar al *Desarrollo de Software Dirigido por Modelos* (*DSDM*). Básicamente, la *Ingeniería Dirigida por Modelos* lo que plantea es el uso de modelos del sistema definidos con diferentes niveles de abstracción a lo largo del ciclo de vida del desarrollo de software, junto con una serie de técnicas de transformación para manipular dichos modelos. Existen dos tipos de transformaciones: (i) *transformaciones horizontales*, que se ocupan de las conversiones entre modelos con el mismo nivel de abstracción, y (ii) *transformaciones verticales*, que permiten realizar conversiones entre modelos con diferentes niveles de abstracción. Haciendo un uso com-

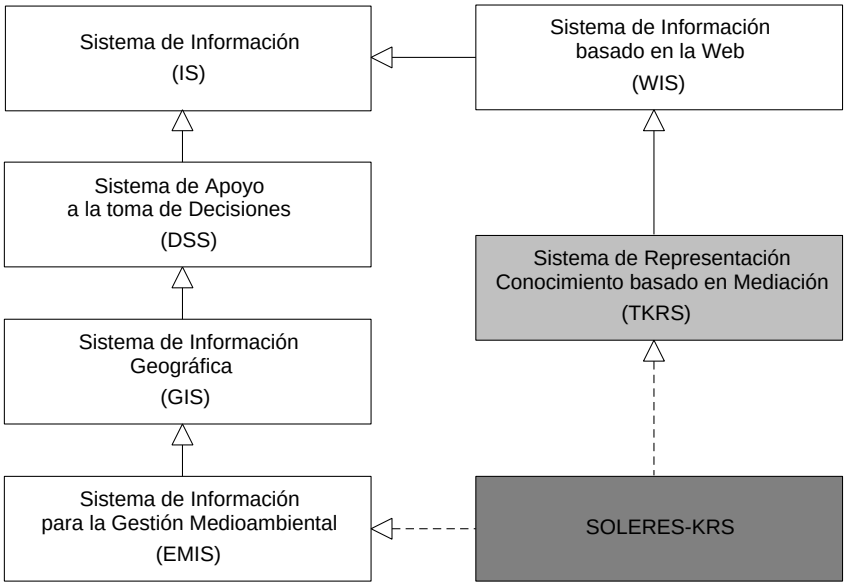


Figura 1.2: Clasificación de los *Sistemas de Información*.

binado de ambos, los modelos con un mayor grado de abstracción se van convirtiendo en modelos menos abstractos progresivamente hasta llegar a la generación casi automática del código. Con esto se consigue una reducción considerable del tiempo y del esfuerzo.

Esta visión de *MDE* aparece representada en la parte izquierda de la *Figura 1.3*. El nivel *M3* (meta-metamodelo) define los conceptos, atributos y relaciones de los elementos del nivel inferior, que serían instancias de los elementos de este nivel; *MOF* (*Meta-Object Facility*) [OMG, 2002] es el lenguaje de *OMG* (*Object Management Group*) utilizado para describir los elementos del nivel *M2* que, a su vez, se utiliza para definirse a sí mismo. Del mismo modo, el nivel *M2* (metamodelo) define los elementos del nivel *M1* que serían instancias de los elementos de *M2*; como ejemplo de metamodelo se podría citar la especificación de *UML* (*Unified Modeling Language*) [OMG, 1997]. Nuevamente, el nivel *M1* (modelos) describe los elementos del nivel *M0*, que serían instancias de los elementos de este nivel; un diagrama de clases, podría ser un ejemplo de modelo. Finalmente, el nivel *M0* representaría las instancias reales; por ejemplo, los objetos en la ejecución de una aplicación orientada a objetos.

Existen varias aproximaciones de este planteamiento como las *Factorías de Software* (*Software Factories*) [Greenfield and Short, 2003], el *Desarrollo de Software Dirigido por Modelos y Centrado en la Arquitectura* (*Architecture-Centric Model-Driven Software Development*, *AC-MDSD*) [Stahl and Völter, 2006], la *Programación Generativa* (*Generative Programming*) [Czarnecki and Eisenecker, 2000], etc., pero sobre todas destaca la propuesta *MDA* (*Model-Driven Architecture*) de *OMG* [OMG, 2003]. A continuación se describe esta última y se profundiza un poco en las transformaciones de modelos.

1.3.1. Arquitectura Dirigida por Modelos

La *Arquitectura Dirigida por Modelos* (*MDA*) propuesta por *OMG* parte de la idea de separar la especificación de un sistema de su implementación. Define el *modelo* de un sistema como una "*descripción o especificación de dicho sistema y su ambiente para algún propósito determinado*" e identifica tres puntos de vista diferentes del sistema: (i) un punto de vista independiente de la computación, centrado en el ambiente y los requisitos del sistema, que no muestra los detalles de su estructura y del procesamiento; (ii) otro punto de vista independiente de la plataforma, focalizado en la operación del sistema; y (iii) un tercer punto de vista específico de la plataforma, que proporciona los detalles necesarios para que el sistema pueda hacer uso de una plataforma concreta. Cada uno de ellos se define mediante un modelo distinto: *CIM* (*Computation Independent Model*), *PIM* (*Platform Independent Model*) y *PSM* (*Platform Specific Model*), respectivamente.

La *Figura 1.3* muestra la relación entre los tres tipos de modelos que define *MDA* y cómo encaja esta aproximación en la visión de *MDE*, concretamente en los niveles *M1* y *M2*. Es necesario aclarar que, aunque en la *Figura* pueda parecer que las transformaciones para obtener un modelo *PIM* a partir de un modelo *CIM* o un modelo *PSM* a partir de un modelo *PIM* sean transformaciones de tipo horizontal, no es así: los modelos se encuentran en diferentes niveles de abstracción, aunque todos formen parte del nivel *M1* (modelos) de *MDE*. Un ejemplo de transformación horizontal podría ser aquélla realizada para obtener un modelo *PIM* a partir de otro modelo *PIM*.

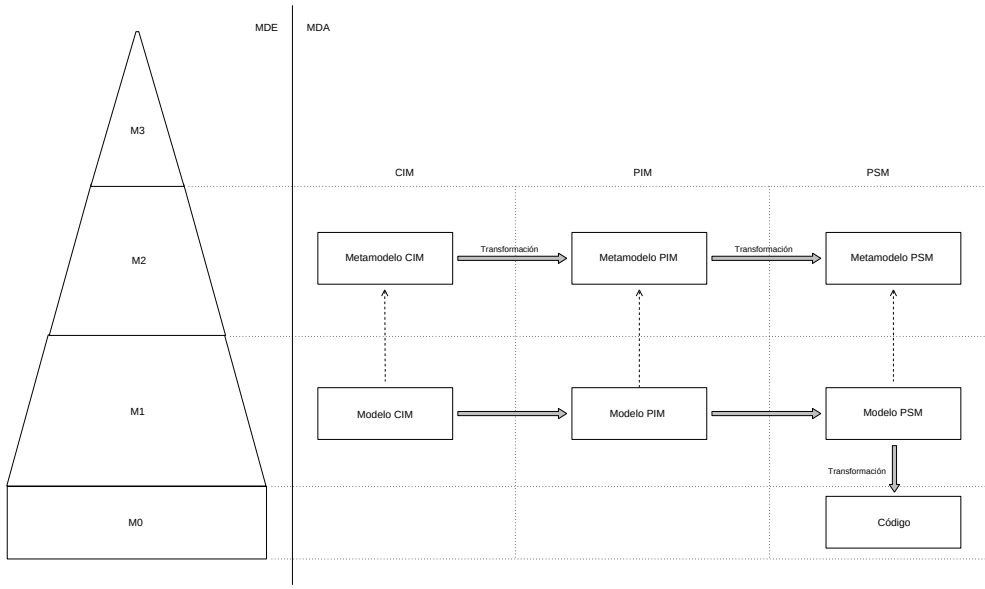


Figura 1.3: Aproximación *MDA* dentro de la visión de cuatro niveles de *MDE*.

Visto con un ejemplo como es el desarrollo de un sistema de red social (instrumento bastante cercano), el modelo *CIM* recogería todos los requerimientos del sistema sin entrar en los detalles de su estructura y funcionalidad, que sí incluiría el modelo *PIM*. Este modelo podría indicar que van a existir listas de amigos que cada usuario puede ir confeccionando mediante invitaciones enviadas a otros usuarios y aceptadas por éstos, que estos usuarios van a disponer de un espacio (denominado "muro") en el que, tanto ellos como sus amigos, van a poder escribir mensajes, que también van a poder crear sus álbumes de fotos, etc. El modelo PIM describiría la funcionalidad de todas estas características sin entrar en los detalles de implementación en una plataforma concreta, que establecería el modelo *PSM*. Podrían existir diferentes modelos *PSM* cada uno de ellos con los detalles precisos de implementación del sistema utilizando cierta tecnología para un dispositivo (ordenador, móvil, tableta...) o entorno (escritorio, web...) concretos, etc. La *Figura 1.4* muestra un fragmento de este modelo *PIM* y de un modelo *PSM* para una implementación basada en el lenguaje de programación *Objective-C*.

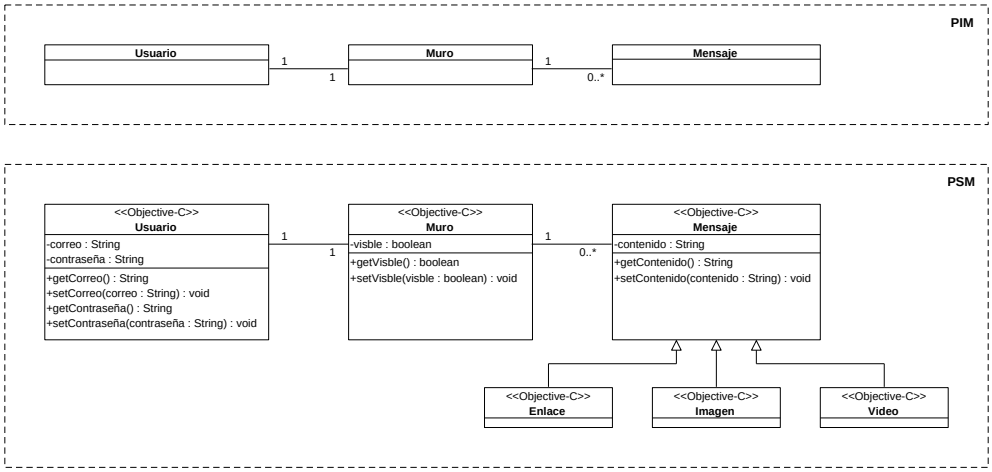


Figura 1.4: Fragmento de los modelos *PIM* y *PSM* de un sistema de red social.

El proceso básico de *MDA* comienza con la definición de un *Modelo Independiente de la Computación* (*CIM*), que recoge los requerimientos del sistema. Este modelo se transforma en otro *Modelo Independiente de la Plataforma* (*PIM*) al que se le incorpora la información correspondiente a la estructura y el funcionamiento del sistema. Por último, este otro modelo se transforma en un *Modelo Específico de la Plataforma* (*PSM*) cuando se le añaden los detalles de implementación concretos de la plataforma en la que se va a desplegar el sistema. Como se puede observar, el grado de abstracción de los modelos va decreciendo a medida que se realizan dichas transformaciones verticales. En sistemas muy complejos, lo normal es que se produzcan, además, diferentes transformaciones dentro de un mismo nivel (transformaciones horizontales) y, de este modo, un *CIM* se transforme en otro *CIM* más específico y así sucesivamente hasta que el *CIM* de menor abstracción se transforme en un *PIM*; a partir de él, se sucedería una

serie de transformaciones entre *PIMs* hasta llegar a un *PSM* y, de nuevo, otra serie de transformaciones entre *PSMs* hasta llegar al modelo final del sistema.

El *Capítulo 3* de este documento detalla la especificación de un marco de desarrollo de *Sistemas de Representación del Conocimiento basados en Mediación* (*TKRS*). Este marco se basa: (a) en la definición de unos modelos independientes de la plataforma para representar, por un lado, la arquitectura del sistema y, por otro, un repositorio de implementaciones de sus componentes, y (b) la transformación de estos modelos para que un modelo específico de la plataforma, denominado modelo de configuración del sistema, permita relacionar cada componente de la arquitectura con su correspondiente implementación.

1.3.2. Transformación de modelos

Aunque se ha mencionado a lo largo de la *Sección* anterior, se puede deducir lo que significa una transformación de modelos: sencillamente, se trata del proceso por el cual un modelo del sistema se convierte en otro modelo distinto del mismo sistema. La *Figura 1.5* (adaptada de [Czarnecki and Helsen, 2006]) muestra su esquema básico. Existe un motor que ejecuta la transformación leyendo un modelo origen especificado conforme a un metamodelo y escribiendo un modelo destino especificado también conforme a otro metamodelo; la definición de la transformación hace referencia a los elementos de ambos metamodelos para establecer la correspondencia. Atendiendo a que "*todo es un modelo*" en *MDE*, como se sugiere en [Bézivin, 2005], las transformaciones de modelos se pueden clasificar en dos tipos: (i) transformaciones *modelo-a-modelo* (*Model-2-Model*, *M2M*) que permiten convertir un modelo más abstracto en otro más específico y como ya se ha mencionado, a su vez, pueden ser horizontales (por ejemplo, siguiendo con *MDA*, de un *PIM* a otro *PIM*) o verticales (por ejemplo, de un *PIM* a un *PSM*); y (ii) transformaciones *modelo-a-texto* (*Model-2-Text*, *M2T*), que permiten transformar un modelo en código ejecutable (una instancia concreta del sistema). Además, las transformaciones se pueden etiquetar como *endógenas* si se realizan entre modelos expresados en el mismo lenguaje (por ejemplo, de un modelo expresado en *XMI* a otro también expresado en *XMI*), o *exógenas* si se realizan entre modelos expresados en diferentes lenguajes (por ejemplo, de un modelo expresado en *XMI* a otro expresado en *XML*).

Las transformaciones requieren metamodelos que permitan la abstracción de cualquier modelo instanciado, lenguajes específicos para su definición (normalmente, se expresan por medio de reglas). La literatura recoge varias propuestas de lenguajes y herramientas de transformación de modelos ([Czarnecki and Helsen, 2006] realizan una revisión). Aunque el estándar propuesto es *QVT* (*Query/View/Transforma- tion*), definido por *OMG* [Kurtev, 2007] [OMG, 2008], está formado por un conjunto de lenguajes para la transformación de modelos, existe otro lenguaje muy difundido inspirado en él y denominado *ATL* (*ATLAS Transformation Language*) [ATLAS Group, 2004], un lenguaje híbrido imperativo-declarativo. En el terreno de las transformaciones *modelo-a-texto*, se pueden encontrar soluciones basadas en plantillas, como *JET* (*Java Emmiter Templates*) [Popma, 2003] o *Velocity Templates* de la *Fundación Apache* [Apache Foundation, 2006], y soluciones basadas en lenguajes, como *MOFScript* o *Xpand* [Xpand, 2007]. Ésta última se utilizará en el *Capítulo 3* para definir una transformación de este tipo.

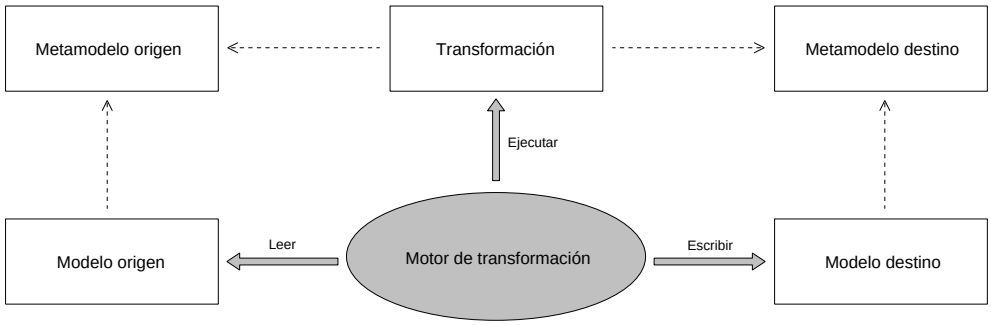


Figura 1.5: Esquema general de una transformación de modelos.

En el marco planteado, se utilizan transformaciones *M2M* con dos fines: (1) para obtener un modelo de configuración (*PSM*) a partir de los modelos de la arquitectura y del repositorio de implementaciones (*PIM*), y (2) para obtener un modelo *OWL* de las definiciones semánticas del conocimiento del sistema a partir de un modelo visual en notación *UML*, que facilita la labor de diseño. También se hace uso de una transformación *M2T* para generar el código de la implementación del sistema en la plataforma seleccionada a partir del modelo de configuración (*PSM*) correspondiente. En el *Capítulo 3* se muestran más detalles de las transformaciones mencionadas.

1.4. Ingeniería Dirigida por Ontologías

Aunque *MDA* es una propuesta bastante sólida para el planteamiento de *MDE*, su utilización de forma conjunta con ontologías permite representar vocabularios de dominio sin ambigüedades, verificar la consistencia de los modelos y validarlos, además de aportar nuevas capacidades que amplían la expresividad en la representación de restricciones —aspectos que complementan a *MDE*—, dando origen a la *Arquitectura Dirigida por Ontologías* (*Ontology-Driven Architecture*, *ODA*). La aplicación de las ontologías llevada también a las actividades que forman parte de las diferentes etapas del ciclo de vida del desarrollo de software proporciona una estandarización para: (i) la generación de modelos de dominio aplicables en estas fases; (ii) la reducción de problemas durante el desarrollo motivados por inconsistencias entre los artefactos (a su vez, ocasionados por las diferentes perspectivas de sus desarrolladores), lo que facilita su trazabilidad y redunda en una mejora de la calidad del sistema; (iii) la descripción de la arquitectura y la semántica de las interfaces, etc. [W3C, 2006]. Esta implicación de las ontologías da lugar a la denominada *Ingeniería Dirigida por Ontologías* (*Ontology-Driven Engineering*, *ODE*) [Wiebe and Chan, 2012].

Hasta este momento se ha estado empleando el término "*ontología*", pero aún no se ha explicado su significado. De todas las definiciones que aparecen en la literatura, la más consolidada es aquélla que indica que "*una ontología es una especificación explícita de una conceptualización*"; además, "*en tal ontología, las definiciones asocian*

los nombres de las entidades en el universo del discurso (por ejemplo, clases, relaciones, funciones u otros objetos) con texto legible describiendo lo que significan los nombres y axiomas formales que restringen la interpretación y el uso adecuado de estos términos" [Gruber, 1995]. Expresado de otro modo, una ontología permite la definición en un dominio de un conjunto de conceptos con sus propiedades o atributos y las relaciones existentes entre ellos, además de las restricciones asociadas.

En [van Heijst et al., 1997] se establece una doble clasificación de las ontologías, por un lado, en base a la cantidad y tipo de estructura de la conceptualización, y por otro, en base al tema de la conceptualización. En relación a la primera, se distinguen tres categorías: (i) *ontologías de términos*, que especifican los términos utilizados para representar el conocimiento en el dominio del discurso; (ii) *ontologías de información*, que especifican la estructura de registro de las bases de datos; y (iii) *ontologías de modelado de conocimiento*, que especifican las conceptualizaciones del conocimiento. En cuanto al tema de la conceptualización, se pueden diferenciar cuatro categorías: (i) *ontologías de aplicación*, que contienen las definiciones necesarias para modelar el conocimiento de una aplicación concreta; (ii) *ontologías de dominio*, que expresan conceptualizaciones específicas para un dominio; (iii) *ontologías genéricas*, similares a las ontologías de dominio, pero los conceptos que definen se consideran comunes a muchos dominios; y (iv) *ontologías de representación*, que expresan conceptualizaciones subyacentes a formalismos de representación del conocimiento.

En esta investigación se emplean las ontologías tanto para representar la información del dominio y disponer de un vocabulario común muy útil durante las diferentes etapas del desarrollo del sistema, como para definir el comportamiento y los protocolos de interacción —en definitiva, la funcionalidad del sistema—. La siguiente *Sección* realiza un breve recorrido por los lenguajes para la especificación de ontologías.

1.4.1. Especificación de ontologías

Desde sus orígenes, los lenguajes para la representación de ontologías han sufrido una constante evolución, motivada también en gran parte por la aplicación de las ontologías en la web, que originó el nacimiento de la conocida *Web Semántica*. Sin dejar en el olvido las primeras aproximaciones, a partir de *RDF* (*Resource Description Framework*) [W3C, 1999] —un *framework* para la representación de información en la web mediante expresiones de la forma "sujeto-predicado-objeto" (grafos) con una sintaxis basada en *XML* (*eXtensible Markup Language*)— y *RDF Schema* [W3C, 2004b] —una extensión semántica de *RDF* que proporciona mecanismos para la descripción de grupos de objetos o recursos y las relaciones entre ellos— y utilizándolos como base, han surgido nuevas propuestas, como *OIL* (*Ontology Interchange Language*), *DAML+OIL* (*DARPA Agent Markup Language + OIL*), etc., hasta llegar a *OWL* (*Web Ontology Language*), el lenguaje más reciente y más ampliamente utilizado para la representación de ontologías. En [Gómez-Pérez and Corcho, 2002], [Gutierrez-Pulido et al., 2006] y [Kalibatiene and Vasilecas, 2011] se realiza una revisión de estos lenguajes, mientras que la *Figura 1.6* muestra de modo gráfico esta evolución.

Centrando la atención en este último mencionado, *OWL* [W3C, 2004a] es el lenguaje estándar del *W3C* que sigue una filosofía orientada a objetos para la definición de onto-

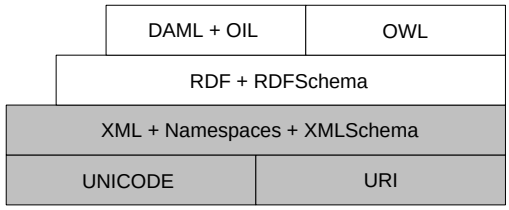


Figura 1.6: Evolución de los lenguajes de especificación de ontologías.

logías en la web, aunque hoy en día se ha extendido a cualquier dominio. *OWL* incorpora mejoras en la descripción de las clases y propiedades: destacan las relaciones entre las clases (por ejemplo, la disyunción), las clases enumeradas, la cardinalidad (por ejemplo, "exactamente uno") o ciertas características de las propiedades (como la simetría). Otra de sus características es que permite establecer relaciones entre ontologías, es decir, una ontología puede referenciar, por ejemplo, los conceptos definidos en otra, ampliarlos, etc. Además, proporciona tres sublenguajes con diferentes niveles de expresividad para adecuar su uso:

(i) *OWL Lite*, que da soporte a clasificaciones jerárquicas y restricciones simples de cardinalidad.

(ii) *OWL DL*, que aporta toda la expresividad del lenguaje con ciertas restricciones que aseguran que los razonamientos sean computacionalmente posibles.

(iii) *OWL Full*, con toda la expresividad sin restricciones, pero también sin garantías computacionales.

La definición de las ontologías utilizadas para la implementación del caso de estudio (*SOLERES-KRS*) en el *Capítulo 4* se ha realizado mediante el sublenguaje *OWL DL*. En la *Tabla 1.1*, a modo de ejemplo para hacerse una idea de su representación, aparece un fragmento de la definición de la entidad *Técnico* (`Technician`) con dos propiedades o atributos, `Technician_id` (líneas #3–#8) y `Technician_firstName` (líneas #10–#15) —cuya cardinalidad en ambos casos es 1—, y dos relaciones con otras entidades, denominadas `Technician_hasEcologicalClassification` y `Technician_hasSatelliteImageClassification` (líneas #17–#35) —cuya cardinalidad, en estos casos, es 1..∗—.

Siguiendo con el caso de estudio, las consultas realizadas —concretamente sobre las ontologías que representan los metadatos de la información del dominio— se expresan en *SPARQL* (*SPARQL Protocol And RDF Query Language*). *SPARQL* [W3C, 2008] es un lenguaje estándar también del *W3C* para la consulta de las expresiones o grafos *RDF* en las que se basa *OWL*. La estructura general de una consulta utilizando *SPARQL* es `SELECT ?z WHERE {?x ?y ?z}`. Se puede observar que consta de dos cláusulas: la cláusula `SELECT`, que identifica las variables que se desean consular y la cláusula `WHERE` que representa el patrón de búsqueda en el grafo. La *Tabla 1.2* muestra un ejemplo de consulta para "obtener el nombre de las variables utilizadas en las clasificaciones realizadas durante el año 2008", que se analizará en la *Sección 4.6.2* junto a otros. Este ejemplo concreto además incluye la cláusula `ORDER BY` que establece el orden de los resultados obtenidos.

```
<owl:Class rdf:ID="Technician">
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#Technician_id"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#Technician_firstName"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
  <rdfs:subClassOf>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Restriction>
              <owl:onProperty rdf:resource="#Technician_hasEcologicalClassification"/>
              <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
               </owl:minCardinality>
            </owl:Restriction>
            <owl:Restriction>
              <owl:onProperty rdf:resource="#Technician_hasSatelliteImageClassification"/>
              <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
               </owl:minCardinality>
            </owl:Restriction>
          </owl:unionOf>
        </owl:Class>
        ...
      </owl:unionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Tabla 1.1: Ejemplo de definición de una entidad en *OWL DL*.

```
SELECT DISTINCT ?EID__VariableName
WHERE {
 ?EID__EID eid:EID_eimId ?EID__EID_eimID .
 ?EID__EID eid:EID_hasClassification ?EID__Classification .
 ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
 ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
 ?EID__Variable eid:Variable_name ?EID__VariableName .
 ?EID__Classification eid:Classification_hasTime ?EID__Time .
 ?EID__Time eid:Time_year ?EID__TimeYear .
 FILTER (?EID__TimeYear = 2008) .
}
ORDER BY ASC(?EID__VariableName)
```

Tabla 1.2: Ejemplo de una consulta en *SPARQL*.

1.5. Servicio de mediación

El *Modelo de Referencia de Procesamiento Distribuido Abierto* (*Reference Model of Open Distributed Processing*, *RM-ODP*) [ISO et al., 1996] desarrollado por *ISO* (*International Organization for Standardization*), *IEC* (*International Electrotechnical Commission*) e *ITU-T* (*Telecommunication Standardization Sector*), proporciona un marco de coordinación para la normalización y especificación de los sistemas de procesamiento distribuido abierto con independencia de la tecnología. Consta de un conjunto de recomendaciones y estándares internacionales, entre los que se encuentra la *Función de Mediación* [ITU-T and ISO/IEC, 1998], desarrollada para proporcionar un servicio dinámico de "páginas amarillas" que permitiera a los clientes de un sistema distribuido localizar en tiempo de ejecución los servicios que necesitan a partir de unos atributos de búsqueda.

Desde el borrador de su propuesta se han desarrollado un gran número de implementaciones, conocidas como servicios de mediación o "*traders*": *DRYAD* [Kutvonen, 1996], *PC-Trader* [Beitz and Bearman, 1994], *TRADE* [Müller et al., 1995], etc. Sin embargo, más tarde, la *función de mediación* de *ODP* fue adoptada por el *OMG* (*Object Management Group*) con el nombre de *CosTrading*, el servicio de mediación de *CORBAServices* del modelo *CORBA* (*Common Object Request Broker Architecture*), un modelo que permite la comunicación entre aplicaciones sin importar su diseño ni localización. Algunas implementaciones conocidas del servicio de mediación de *CORBA* son: *AceORB* de *TAO* [Schmidt, 2001] u *OpenFusion* de *PrismTech* [PrismTech, 2001].

Las siguientes *Secciones* realizan una descripción de esta *función de mediación*, diferenciando los roles que adoptan los objetos, presentando sus interfaces y analizando las limitaciones que presenta.

1.5.1. Roles: mediador, exportador e importador

Desde el punto de vista de *ODP*, una *función de mediación* (también denominada *servicio de mediación*, *mediador* o *trader*), es el objeto software que hace de intermediario entre unos objetos que ofertan ciertas capacidades o servicios, y otros objetos que demandan su utilización de forma dinámica. Los objetos que ofrecen sus servicios se denominan "*exportadores*" y proporcionan al *trader* una descripción (aspectos extra-funcionales) y una interfaz (aspectos funcionales) de cada servicio, mientras que los objetos que demandan estos servicios se denominan "*importadores*" y solicitan al *trader* los servicios que posean una serie de características. La labor del *mediador*, por lo tanto, consiste en comprobar las características demandadas en las descripciones de los servicios ofertados —que almacena en un repositorio— e indicar al *importador* las interfaces de los servicios seleccionados para que pueda interactuar con el *exportador*. La *Figura 1.7* muestra el diagrama de secuencia básico de este proceso.

En esta secuencia, el objeto que adopta el rol de *exportador* hace uso de la operación `exportar()` (*export*) del objeto de mediación para registrar el servicio correspondiente, especificando sus características y su interfaz; el objeto de mediación almacena en su repositorio esta información del servicio. Más tarde, un segundo objeto que necesita un determinado servicio (por lo tanto, con el rol de *importador*), mediante la operación `consultar()` (*query*) del objeto de mediación, le indica a éste las características

y restricciones del servicio demandado; el objeto de mediación realiza la consulta en su repositorio y devuelve como resultado (si lo hay) al importador las referencias de los objetos exportadores que ofrecen el servicio demandado (`importar()`, *import*). Finalmente, a partir de estos resultados, el objeto *importador* selecciona la referencia que mejor se ajusta al servicio que estaba buscando y contacta directamente con el objeto *exportador* (`invocar()`, *invoke*).

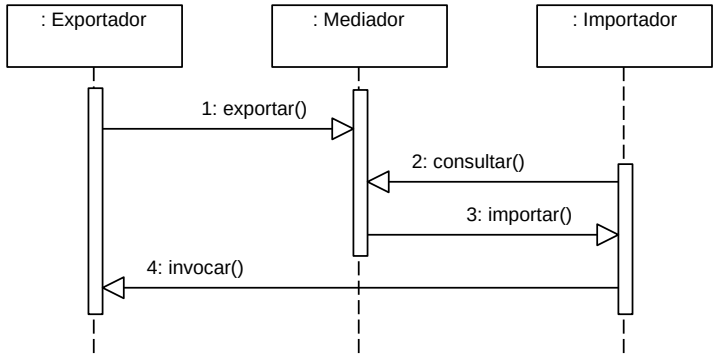


Figura 1.7: Diagrama de secuencia del proceso de mediación.

El proceso de mediación realiza las operaciones de exportación e importación de servicios normalmente sobre unos repositorios de plantillas de especificación en *XML* que almacenan las descripciones y las interfaces de cada servicio, como se ha mencionado. Sin embargo, el modelo adaptado que se propone en este trabajo utiliza un repositorio de plantillas en *OWL* —una ontología—, para dar soporte tanto a la integración de componentes e información como a la gestión y, sobre todo, consulta de esta última representada mediante ontologías, también en *OWL*. El uso de este lenguaje aparece justificado en la *Sección 4.3.2*.

Lo normal en un entorno dado es que exista más de un objeto de mediación. En este caso, los clientes (tanto objetos *exportadores* como *importadores*) pueden interactuar de dos formas totalmente distintas con los objetos de mediación: directa o indirectamente. En el primero de los casos, cuando un cliente interactúa con un objeto de mediación y éste no puede satisfacer sus necesidades, el cliente accede a otro objeto de mediación de forma directa. Sin embargo, en la interacción indirecta, es el primer objeto de mediación el que interactúa con el segundo en lugar del cliente. En esta línea va lo que se conoce como *federación* de objetos de mediación. Visto con un ejemplo, un objeto con el rol de importador que busca un determinado servicio interroga a un objeto de mediación que no puede satisfacer su demanda; en la interacción directa, es el propio importador el que continuaría buscando el servicio contactando con otro objeto de mediación, mientras que en la interacción indirecta, el primer objeto de mediación delegaría la consulta del importador a un segundo objeto de mediación con el que se encontrara federado (que, a su vez, podría continuar delegando en otro u otros) y las referencias localizadas por éste seguirían el camino inverso, es decir, serían devueltas al primer objeto de mediación y, posteriormente, al objeto *importador*.

1.5.2. Interfaces del servicio de mediación

La interacción que se produce entre exportadores e importadores y un mediador siempre se realiza a través de un conjunto de interfaces funcionales, con una serie de operaciones cada una, que posee el mediador. La función de mediación de *ODP* define cinco interfaces: (i) interfaz *Lookup*, que permite a los importadores consultar los servicios registrados en su repositorio; (ii) interfaz *Register*, que permite a los exportadores registrar sus servicios en el mediador; (iii) interfaz *Admin*, que permite la configuración de los parámetros y políticas del servicio de mediación; (iv) interfaz *Link*, que permite vincular un mediador con otro u otros dando soporte al mecanismo de federación; y (v) interfaz *Proxy*, que permite establecer una pasarela con otros servicios de mediación (herencia). En la *Figura 1.8* se representa el modelo conceptual de esta función de mediación. Se puede observar que las cinco interfaces anteriores implementan cinco "interfaces funcionales" (atendiendo al concepto *UML* de interfaz) que, a su vez, heredan de una serie de "interfaces abstractas": `TraderComponents` proporciona información sobre las interfaces que implementa el servicio, `SupportAttributes` hace alusión a la funcionalidad soportada, e `ImportAttributes` y `LinkAttributes` que almacenan los valores predeterminados de algunos parámetros referidos a las consultas y a las federaciones, respectivamente.

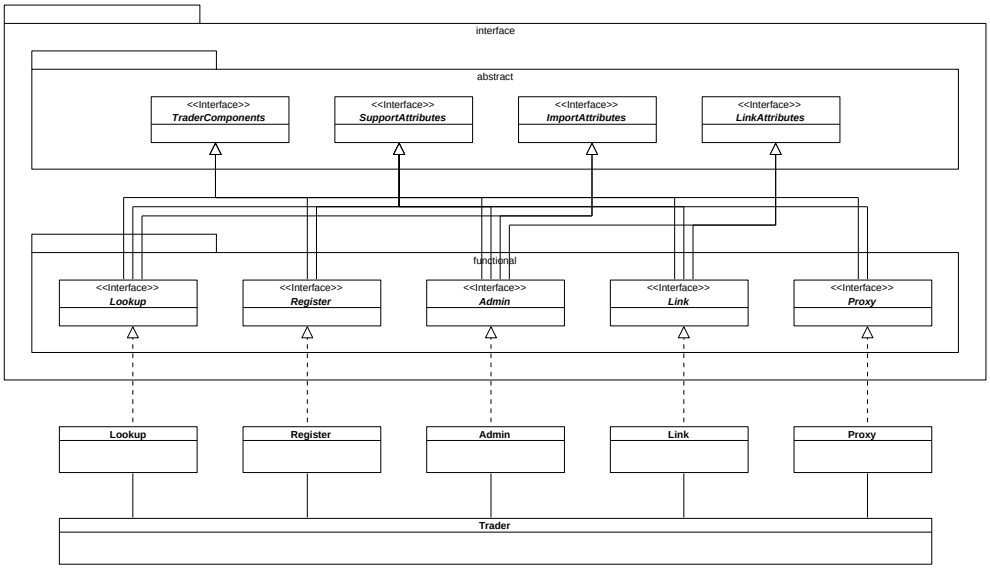


Figura 1.8: Modelo conceptual de la función de mediación de *ODP*.

Un servicio de mediación no tiene por qué implementar estas cinco interfaces. En función de las que implemente, el mediador se puede clasificar como: *Query Trader*, si únicamente cuenta con la interfaz *Lookup*; *Simple Trader*, si además implementa la interfaz *Register*; *Stand-alone Trader*, si a parte de las dos interfaces anteriores incluye la interfaz *Admin*; *Linked Trader*, si implementa las tres interfaces anteriores junto con la

interfaz *Link*; *Proxy Trader*, si se sustituye la interfaz *Link* por la interfaz *Proxy*; y *Full-service Trader*, si dispone de las cinco interfaces. La *Tabla 1.3* recoge esta clasificación.

Servicio de mediación	**Interfaces**
Query Trader	*Lookup*
Simple Trader	*Lookup* y *Register*
Stand-alone Trader	*Lookup*, *Register* y *Admin*
Linked Trader	*Lookup*, *Register*, *Admin* y *Link*
Proxy Trader	*Lookup*, *Register*, *Admin* y *Proxy*
Full-service Trader	*Lookup*, *Register*, *Admin*, *Link* y *Proxy*

Tabla 1.3: Clasificación de los servicios de mediación en base a las interfaces que implementan.

1.5.3. Limitaciones de la función de mediación

Una vez presentadas las características de la *función de mediación* de *ODP*, en esta sección se van a mostrar las limitaciones que presenta (inspiradas en las que proponen [Iribarne et al., 2004]) al extender su funcionalidad para favorecer la interoperabilidad de los componentes de un *Sistema de Información* por un lado y, por otro, ser partícipe del proceso de búsqueda y recuperación de la información mediante el uso de ontologías:

(a) **Modelo de objetos**. La especificación de la *función de mediación* de ODP está desarrollada para trabajar con objetos que siguen el modelo de *OMG* y la especificación de las interfaces de los servicios mediante su *IDL* (*Interface Definition Language*). Como uno de los principales objetivos de este trabajo consiste en diseñar y desarrollar un modelo de mediación ontológico que, además sea el núcleo de un *Sistema de Representación del Conocimiento*, se van a definir una serie de ontologías, denominadas de ontologías de servicio/proceso, una por cada una de las interfaces del modelo de mediación, que van a facilitar la interacción del resto de componentes del sistema e, incluso, de componentes externos al sistema con él.

(b) **Descripción de servicios frente a descripción de información en general**. Como se ha indicado en la *Sección 1.5.1*, cada servicio se describe mediante un conjunto de características o propiedades extra-funcionales y una interfaz que recoge sus aspectos funcionales, en definitiva, un conjunto de metadatos. Sin embargo, si se pretende seguir esta filosofía para que la función de mediación forme parte del proceso de búsqueda y recuperación de información en un sistema, es preciso realizar una abstracción y pensar que la información que va a gestionar y almacenar sería un conjunto de metadatos de la información de un dominio dado, que se va a representar también mediante una ontología, en este caso de datos.

(c) **Emparejamiento "uno a uno" frente a "uno a muchos"**. La *función de mediación* de *ODP* no soporta la selección de más de un servicio registrado con las restricciones que impone un *importador*, por lo que se produce un emparejamiento

"uno a uno" entre la demanda y el servicio. Por su parte, el servicio de mediación ontológico debería proporcionar emparejamientos del tipo "uno a muchos" entre la demanda de información y los metadatos devueltos, donde una consulta se podría satisfacer mediante la composición de dos o más instancias de metadatos.

(d) **Emparejamiento fuerte y débil**. En la línea de la limitación anterior, la búsqueda de un servicio se restringe sólo a las características extra-funcionales del mismo y no a los aspectos funcionales de la interfaz, por lo que la función de mediación realiza "emparejamientos fuertes o exactos" entre las ofertas y las demandas. En el caso del servicio de mediación ontológico que se propone, éste también debe permitir una consulta sobre la totalidad de los metadatos de la información almacenados, soportando "emparejamientos débiles o parciales", por razones obvias.

(e) **Registro bajo demanda y por extracción**. La función de mediación de *ODP* se basa en un modelo de registro bajo demanda (*push*) en el que son los *exportadores* los que registran sus servicios en el *mediador*. Sin embargo, podría ser interesante que el servicio de mediación ontológico también diera soporte a un modelo de registro por extracción (*pull*) en el que él mismo se ocupara de buscar metadatos e incorporarlos a su repositorio.

(f) **Federación en los procesos de importación y no en los de exportación**. Como se ha indicado también en la *Sección 1.5.1*, la *función de mediación* de *ODP* permite enlazar un servicio de mediación con otro u otros para favorecer la búsqueda de servicios en los procesos de importación; a esto se le conoce con el nombre de *federación directa*. La función no contempla este mecanismo en los procesos de exportación de servicios. El servicio de mediación ontológico sí debería contemplarlo para mejorar la calidad de sus actuaciones.

1.6. Agentes software

Dadas las características que poseen los *Agentes Software*, permiten dotar a un sistema de una gran modularidad, flexibilidad, extensibilidad y escalabilidad [OMG, 1998] [Alonso, 2002] [Fulbright and Stephens, 1994] [Honavar, 1999], lo que se ve reflejado en sus aplicaciones: gestión y búsqueda de información, control y supervisión, trabajo cooperativo, asistencia personal, etc. Como se pondrá de manifiesto en los *Capítulos 3* y *4*, el interés por los agentes software se centra en su uso para la implementación de un *Sistema de Representación del Conocimiento*. Por este motivo, en esta *Sección* se realiza un pequeño estudio que analiza sus principales características, muestra su tipología, describe sus asociaciones formando *Sistemas Multi-Agente* y hace una revisión de los lenguajes y entornos más destacados para su desarrollo.

1.6.1. Visión general

Existe una gran controversia con respecto a la definición de *Agente*, dados los diferentes fines con los que ha sido empleado este término. De hecho, diversos autores optan por

describirlos identificando un conjunto de propiedades que los caracterizan. Algunas de estas definiciones aparecen descritas a continuación:

(i) Según [Maes, 1994] [Maes, 1995] [Maes, 1997], "*los agentes autónomos son sistemas computacionales que habitan en entornos dinámicos complejos, percibiendo y actuando autónomamente en ese entorno, y realizan un conjunto de metas o tareas para las que han sido diseñados*".

(ii) En [Smith et al., 1994] aparece que "*un agente es un software persistente dedicado a un propósito específico*".

(iii) Mientras que en [Wooldridge and Jennings, 1995], el término agente se emplea para denotar "*un sistema hardware y/o software que disfruta de las siguientes propiedades: es autónomo, sociable, reacciona y toma la iniciativa para provocar que las cosas ocurran*".

(iv) Por otro lado, [Hayes-Roth, 1995] indica que "*los agentes inteligentes realizan continuamente tres funciones: percibir condiciones dinámicas en el entorno, actuar para afectar a las condiciones del entorno, y razonar para interpretar lo que percibe, resolver problemas, inferir y determinar las acciones*".

(v) Por último, en [Franklin and Graesser, 1997] los agentes inteligentes se describen como "*entidades software que llevan a cabo un conjunto de operaciones en beneficio de un usuario u otro programa con algún grado de independencia o autonomía y, haciendo esto, emplean algún conocimiento o representación de las metas y deseos del usuario*".

La primera definición destaca el **carácter autónomo** de los agentes —capacidad de actuar sin la intervención directa de nada o nadie más, lo que implica control sobre sus acciones y su estado interno— para la realización de una tarea o de un conjunto de tareas. En la segunda definición se resalta la **persistencia** de un agente, propiedad que lo distingue de otro tipo de software y que consiste en su continua ejecución y capacidad para decidir el momento en el que debe llevar a cabo una determinada acción. En la tercera definición surgen tres nuevas propiedades: la **sociabilidad** o capacidad de un agente de interactuar con otros utilizando un lenguaje específico; la propiedad **reactiva** de un agente le permite percibir el entorno donde se está utilizando (por ejemplo, las acciones de un usuario a través de una interfaz) y responder a los cambios que se producen en él; por último, la **pro-actividad** le proporciona la capacidad de exhibir un comportamiento dirigido a alcanzar un objetivo previamente establecido, sin actuar simplemente como respuesta a un cambio en el entorno. Sin embargo, en las dos últimas definiciones se hace referencia a la "**inteligencia**" y capacidad de razonamiento de los agentes, necesitando cierto aprendizaje.

Un *agente software* no tiene por qué incluir todas las propiedades mencionadas, sino que puede incluir algunas en función del propósito para el que sea diseñado, lo que dificulta su clasificación. En [Nwana, 1996] se puede encontrar una clasificación ampliamente referenciada en la literatura, donde el autor agrupa a los agentes en diversas categorías según:

— su **movilidad** por una red, distinguiendo entre agentes estáticos y móviles;

— su **reactividad**, agrupándolos en agentes deliberativos y agentes reactivos;

— los atributos **autonomía, cooperación y aprendizaje**, obteniendo agentes colaborativos, agentes de aprendizaje colaborativos, agentes de interfaz y agentes "inteligentes";

— sus **roles**, apareciendo agentes de información/Internet, agentes de informes, agentes de análisis y diseño, agentes de ayuda, etc.;

— por último, distingue agentes **híbridos** que combinan las características anteriores.

Aunque existen otros muchos atributos, en esta tipología se consideran secundarios los agentes versátiles, los agentes benevolentes, etc. También elimina los agentes de aprendizaje colaborativos debido a que no tiene constancia de ningún uso de éstos. Por lo tanto, finalmente se pueden identificar siete tipos de agentes: agentes colaborativos, agentes de interfaz, agentes móviles, agentes de información/Internet, agentes reactivos, agentes híbridos y agentes "inteligentes". Las aplicaciones que hacen uso de varios de los tipos de agentes mencionados reciben el nombre de sistemas de agentes heterogéneos.

1.6.2. Sistemas Multi-Agente

Normalmente, los agentes no actúan de forma independiente para la resolución de una tarea compleja, si no que se asocian para alcanzar un objetivo global mediante la resolución de tareas más simples de forma autónoma. Este tipo de asociación se denomina *Sistema Multi-Agente* (*Multi-Agent System*, *MAS*) y en ella se distinguen dos tipos de tareas: (i) las *tareas individuales* de cada agente y (ii) las *tareas globales* del sistema que se dividen, a su vez, en subtareas realizadas también por diferentes agentes de acuerdo a sus capacidades —que no tienen por qué ser totalmente independientes entre sí, por lo que necesitan mecanismos de comunicación y colaboración eficaces entre los agentes para llevarlas a cabo con éxito—. En [Jennings et al., 1998] se enuncian una serie de características inherentes a estos sistemas:

(a) Cada agente del sistema dispone de una visión o conocimiento parcial del mismo.

(b) No existe un control centralizado del sistema.

(c) La interacción entre los diferentes agentes es asíncrona.

(d) El entorno en el que se encuentra inmerso el sistema es continuamente cambiante e impredecible.

(e) Este entorno debe permitir tanto la interacción y comunicación que se establece entre los diferentes agentes, como entre dichos agentes y el propio entorno.

Estas características sugieren nuevas propiedades asociadas a los *Sistemas Multi-Agente*, más concretamente:

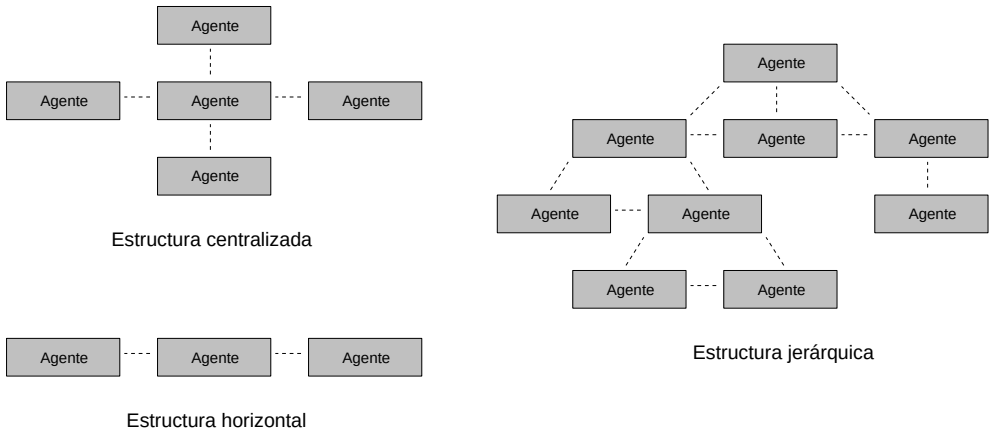


Figura 1.9: Estructuras centralizada, horizontal y jerárquica de un *MAS*.

— **Organización**. La organización hace referencia a la estructura que adoptan los agentes dentro del sistema, así como a sus responsabilidades, comunicación, etc. De forma general, se distinguen cuatro tipos de configuraciones: (i) *estructura centralizada*, donde un agente controla la interacción del resto de agentes del sistema; (ii) *estructura horizontal*, donde todos los agentes se encuentran al mismo nivel y ninguno controla la interacción de los demás; (iii) *estructura jerárquica*, donde existen diferentes niveles y, dentro de un mismo nivel, los agentes pueden seguir una estructura horizontal; y (iv) *estructura "ad hoc"*, que es una mezcla de las anteriores. La *Figura 1.9* recoge las tres configuraciones principales.

— **Cooperación**. Los mecanismos de cooperación dependen de la organización que se establezca en el sistema: (i) en una *estructura centralizada*, los agentes esclavos piden continuamente colaboración al agente maestro; (ii) en una *estructura horizontal*, la cooperación se realiza entre todos los agentes; y (iii) en una *estructura jerárquica*, la cooperación se puede establecer por niveles o bien desde los niveles superiores hacia los niveles inferiores.

— **Coordinación**. La coordinación entre los agentes del sistema se encuentra relacionada con la planificación de las acciones a realizar para la resolución de las tareas de un modo eficaz y eficiente. Los dos principales modelos son: (i) *coordinación global*, en el que el sistema determina y planifica globalmente las acciones de los diferentes agentes, y (ii) *coordinación individual*, en el que cada agente del sistema se planifica individualmente y resuelve localmente los conflictos que surjan con otros agentes.

— **Control**. El control es un mecanismo básico que sirve de apoyo al mecanismo de coordinación y que permite determinar las tareas que deben realizarse en cada instante, el tiempo estimado para su realización, etc., así como evaluar si se han realizado satisfactoriamente. El control también se puede considerar tanto a nivel

global como *local*, en función de que la toma de decisiones se realice a partir de la información global de todos los agentes del sistema o de la información local de cada uno de ellos.

— **Negociación**. Para que los mecanismos de cooperación y coordinación tengan éxito, debe existir un mecanismo adicional que ponga de acuerdo a los agentes en el momento de defender sus intereses, buscando siempre un beneficio global. Los mecanismos de negociación se rigen por un protocolo y utilizan una serie de reglas.

En el diseño e implementación de un *Sistema Multi-Agente*, a parte de estas propiedades, también resulta de especial importancia la elección de una metodología clara y precisa, así como una plataforma de desarrollo. En la siguiente *Sección* se analizan las principales tecnologías existentes para el diseño, desarrollo e implementación de este tipo de sistemas.

1.6.3. Lenguajes y entornos de desarrollo

Actualmente, existen en el mercado más de un centenar de productos software para el diseño de aplicaciones basadas en agentes, que principalmente se utilizan en entornos académicos y comerciales: *AdventNet Agent Toolkit*, *Agent Builder*, *AgentTcl*, *AgentTalk*, *AgentTool*, *AgentWare*, *Cable*, *Emorphia*, *FIPA-OS*, *Grasshopper*, *Impact*, *JADE*, *MAGE*, *MASS*, *Microsoft Agent*, *SiWalk*, *Soar*... En [Mangina, 2002] se puede encontrar una revisión de productos para el desarrollo de *Sistemas Multi-Agente* promovida por *AgentLink*, una red fundada por la Comisión Europea para la investigación y desarrollo de tecnologías basadas en agentes; su portal web [AgentLink, 1998] recoge un listado de estos productos software. Básicamente, todos ellos se pueden clasificar en *entornos de desarrollo*, *toolkits* o *servicios*.

Los *entornos de desarrollo* proporcionan un *middleware* que simplifica el desarrollo, la implementación e, incluso, la implantación de un *Sistema Multi-Agente*; un claro ejemplo es *JADE* (*Java Agent Development Framework*). Los *toolkits*, como su propio nombre indica, ofrecen un conjunto de herramientas para la gestión de mensajes intercambiados por los agentes, la administración de tareas, bases de datos, etc.; un ejemplo se puede encontrar en *FIPA-OS* basado en componentes. Por su parte, los *servicios* facilitan la interoperatividad de los agentes independientemente de la aplicación; por ejemplo, *Impact* proporciona un conjunto de servicios (registro, páginas amarillas, tesauro, etc.).

La implementación de un sistema basado en agentes tiene que cubrir las propiedades descritas en la *Sección* anterior: organización, cooperación, coordinación, control y negociación. Existen organizaciones como *OMG* (*Object Management Group*) y *FIPA* (*Foundation for Intelligent Physical Agents*), que desarrollan estándares para asegurar la interacción de los agentes, incluso pertenecientes a sistemas heterogéneos. En este sentido, se debe establecer tanto una arquitectura compatible como un lenguaje común —conocido como *Agent Communication Language* (*ACL*)—. Las arquitecturas más extendidas de *Sistemas Multi-Agente* son precisamente las que establecen los estándares de estas dos organizaciones, mientras que los lenguajes que cuentan con una mayor difusión son *FIPA-ACL* y *KQML*.

La *Arquitectura de Gestión de Objetos* (*Object Management Architecture*, *OMA*) de *OMG* [OMG, 1990] proporciona una guía para la creación de entornos de componentes software basados en tecnologías de objetos a partir de la estandarización de interfaces de componentes. La *Figura 1.10* (obtenida de [OMG, 1990]) representa esta arquitectura caracterizada por: (i) el componente *Object Request Broker* (*ORB*), que permite a los objetos comunicarse en un entorno distribuido independientemente de las plataformas e implementaciones; (ii) los *Object Services*, que proporcionan una colección de servicios (interfaces y objetos) que dan soporte a funciones básicas (creación de objetos, control de acceso a ellos, seguimiento, etc.); y (iii) las *Common Facilities*, que representan un conjunto de servicios que muchas aplicaciones pueden compartir (por ejemplo, un sistema de gestión o de correo electrónico).

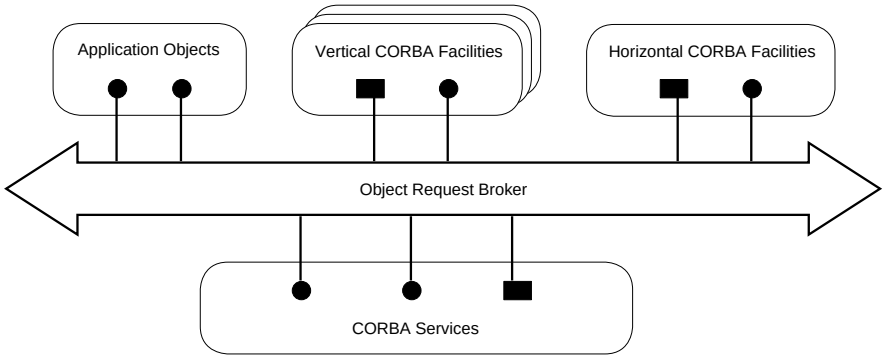


Figura 1.10: Arquitectura *OMA* de *OMG*.

La *Arquitectura Abstracta de FIPA* (*FIPA Abstract Architecture*) [FIPA, 2002a], como su propio nombre indica, define a nivel abstracto la forma en la que dos agentes pueden localizarse y comunicarse entre sí mediante el intercambio de mensajes. Esta arquitectura consta de cuatro componentes como muestra la *Figura 1.11* (adaptada de [FIPA, 2002a]): (i) el componente *Directorio de agentes*, que proporciona los servicios necesarios para que los agentes puedan registrar sus descripciones, localizar a otros agentes registrados, etc.; (ii) el *Directorio de servicios*, que permite el registro y la localización de los servicios ofrecidos por los agentes; (iii) el componente *Transporte de mensajes*, con los servicios necesarios para el envío y recepción de mensajes entre los agentes, proporcionando mecanismos de seguridad; y (iv) el *Lenguaje de Comunicación de Agentes de FIPA* (*FIPA-ACL*).

FIPA-ACL [FIPA, 2002b] es un lenguaje de comunicación mediante mensajes que incluye una serie de especificaciones de obligado cumplimiento e informativas. Las primeras están relacionadas con la integración de los agentes tanto dentro del marco de trabajo que establece *FIPA*, como con otro software (sistemas de gestión de bases de datos, gestores de dispositivos, etc.). En una de ellas se describe el lenguaje de comunicación, identificándose el conjunto de mensajes que se pueden generar; se pueden utilizar diferentes mecanismos de transporte (*TCP/IP*, *HTTP*, *SMTP*, etc.) para el envío y, aun-

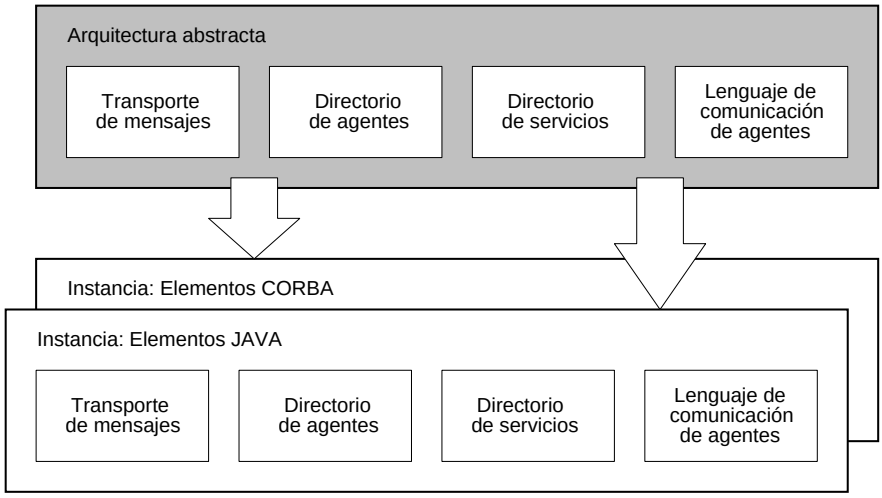


Figura 1.11: Arquitectura Abstracta de *FIPA*.

que no se impone un lenguaje concreto para el contenido, sí se establece que se puedan expresar proposiciones, objetos y acciones.

Para finalizar, *KQML* (*Knowledge Query & Manipulation Language*) [DARPA, 1993] fue originalmente concebido como una interfaz para los sistemas basados en conocimiento. Los mensajes de información (estados, requerimientos, búsquedas, suscripciones, etc.) intercambiados por los agentes siguen un formato específico —donde las expresiones en *KQML* pueden contener, a su vez, subexpresiones en otros lenguajes— y existe un protocolo de manipulación de dichos mensajes que da soporte a la compartición de conocimiento en tiempo real.

1.7. Resumen y conclusiones

En este primer *Capítulo* se ha presentado la idea global del trabajo de investigación, que consiste en el establecimiento de un marco para el diseño y desarrollo de *Sistemas de Representación del Conocimiento* con la peculiaridad de estar basados en un modelo de mediación ontológico. Para posteriormente sentar las bases de este modelo, se han dado unas pinceladas sobre los *Sistemas de Representación del Conocimiento* y se ha analizado la *función de mediación* de *ODP* descubriendo sus limitaciones. El marco general se sustenta en principios de la *Ingeniería Dirigida por Modelos* y de la *Ingeniería Dirigida por Ontologías*, por lo que se han resumido sus fundamentos, se han tratado sus principales conceptos(modelos, transformaciones, ontologías...), se han presentado algunas herramientas de trabajo, especificaciones, etc. Por último, se ha mostrado una visión general de la tecnología de *Agentes Software* y *Sistemas Multi-Agente*, utilizada en numerosas ocasiones para la implementación de este tipo de sistemas. A partir del siguiente *Capítulo* ya se entra de pleno en los detalles de la propuesta.

Capítulo 2

Modelo de mediación ontológico: OntoTrader

Capítulo 2

Modelo de mediación ontológico: OntoTrader

Contenidos

Los actuales *Sistemas de Información basados en la Web* (*Web-based Information Systems*, *WIS*) se están haciendo cada vez más imprescindibles porque favorecen el acceso a la información desde cualquier punto geográfico y dispositivo, permiten a sus usuarios analizar la información desde diferentes perspectivas, dan soporte al trabajo en equipo, ayudan a una mejor toma de decisiones, etc. El diseño de estos sistemas requiere, fundamentalmente, el empleo de métodos y técnicas estandarizados que permitan la utilización de un vocabulario común para representar todo el conocimiento subyacente. Entre otras, los servicios de mediación enriquecen la interoperabilidad de los componentes web —elementos software con una funcionalidad concreta para la web, preparados para ser ensamblados y operar junto a otros componentes mediante sus interfaces, desde las que se puede tener acceso a los servicios que ofrecen, y normalmente disponibles en repositorios— en este tipo de sistemas abiertos y distribuidos, siendo capaces de interactuar, incluso, con servicios de mediación de terceras partes. Este trabajo de investigación defiende y justifica la idea de extender su funcionalidad reemplazando las especificaciones de los servicios de los componentes (que almacena en un repositorio propio) por un conjunto de metadatos de la información gestionada por el sistema —representados mediante una ontología—, y así involucrarlos en el proceso de búsqueda y recuperación de la información. Siguiendo esta doble vertiente, a continuación se presentan las características principales que debería tener un modelo de mediación ontológico (calificado así por el empleo de ontologías tanto para representar los metadatos que gestiona como para definir la interacción con sus interfaces, como se detallará en las siguientes *Secciones*), denominado *OntoTrader*, así como los principales detalles de su diseño.

El *Capítulo* se organiza en siete secciones. Comienza analizando y justificando más detalladamente la necesidad de un modelo de mediación ontológico en la *Sección 2.1*. Seguidamente, en la *Sección 2.2*, se ofrece una perspectiva global del modelo de mediación, exponiendo las propiedades o requisitos que debe cumplir un modelo de mediación ontológico, así como las características de su repositorio. La *Sección 2.3* muestra tres modelos operativos de mediación basados en reflexión, delegación y federación. Los detalles de diseño de la arquitectura del modelo *OntoTrader*, junto con las ontologías asociadas a sus interfaces, se describen en la *Sección 2.4*. En la *Sección 2.5* se presenta una formalización matemática del modelo. Para finalizar, en la *Sección 2.6*, se discuten algunos trabajos relacionados con la propuesta presentada y, en la *Sección 2.7*, se realiza un breve resumen junto con las conclusiones del presente *Capítulo*.

2.1. Introducción y conceptos relacionados

En la actualidad, aunque los *Sistemas de Información basados en la Web* han alcanzado una gran popularidad, aún presentan algunas deficiencias. Uno de sus principales escollos se sitúa en la dificultad de los procesos de búsqueda y recuperación de la información debido fundamentalmente al gran volumen que gestionan. Sus usuarios dependen de portales web, bibliotecas digitales, motores u otros sistemas de búsqueda y recuperación de información [Goh and Foo, 2008] [Gama and May, 2011] que les ayuden y faciliten

este tedioso proceso y, aún así, se enfrentan a una sobrecarga de información donde se ven obligados a diferenciar el contenido relevante del que no lo es. En un intento de resolver este problema, se han desarrollado y aplicado una amplia variedad de técnicas basadas en diferentes ámbitos: integración de la información, recuperación de la información, filtrado de la información, interacción persona-ordenador, estudios de comportamiento en la búsqueda de información, etc.

La recuperación de información se refiere al conjunto de técnicas que facilitan a los usuarios la satisfacción de sus necesidades de información [Ramos et al., 2005]. El modelo *Query-Searching/Recovering-Response* (*QS/RR*) es una extensión del tradicional modelo *Query/Response*, el principal mecanismo de recuperación de información basado en una arquitectura cliente/servidor. Por un lado, el término *Query* se refiere a todo el proceso de creación y formulación de la consulta por parte del usuario. Por otro lado, el término *Searching* hace alusión al proceso de localización de los repositorios donde se encuentra la información, mientras que el término *Recovering* se refiere al proceso de localización, identificación y selección de los datos en las fuentes de datos (repositorios, almacenes de datos o bases de datos, independientemente del modelo). Finalmente, con el término *Response* se representa el proceso de formulación, preparación y creación de la respuesta al usuario. *Query-Searching* es un proceso que sigue la dirección cliente-servidor, mientras que *Recovering-Response* va en la dirección servidor-cliente. En la *Figura 2.1* se pueden diferenciar claramente estos cuatro conceptos.

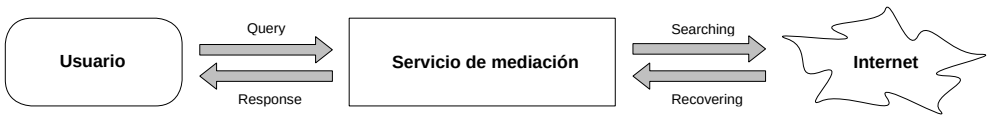


Figura 2.1: Visión general del mecanismo *QS/RR*.

Los servicios de mediación se consideran como una solución para sistemas abiertos y distribuidos [ITU-T and ISO/IEC, 1998] que extienden el mecanismo *Object Request Broker* (*ORB*) de la *Object Management Architecture* (*OMA*). Aunque tradicionalmente se han utilizado como *middleware* para proporcionar interoperabilidad entre objetos o componentes, se pueden adaptar para reemplazar las especificaciones de sus servicios por cualquier tipo de información presente en un sistema, proporcionando una doble dimensión a su funcionalidad, tal y como aparece reflejado en la *Figura 2.2*. En ella se puede apreciar cómo, por un lado, los servicios de mediación son capaces de integrar diferentes *Sistemas de Representación del Conocimiento* (*Knowledge Representation Systems*, *KRS*) y dar soporte al mecanismo de federación haciendo uso de sus interfaces *Link* (parte superior derecha de la *Figura*), mientras que por otro, pueden formar parte del proceso de búsqueda y recuperación de la información siendo capaces de resolver ciertas demandas o, en caso contrario, haciendo de intermediarios, por ejemplo, entre un módulo de consulta y diferentes módulos de procesamiento de información del sistema, y seleccionar aquellos módulos de procesamiento que disponen de la información solicitada (parte inferior izquierda). El empleo de ontologías supone un buen aporte para esta

adaptación, pudiendo ser utilizadas tanto para definir y representar la información del dominio (que el servicio de mediación almacena en un repositorio propio), como para describir la funcionalidad del servicio, definiendo el comportamiento (las reglas de operación de las funcionalidades que proporciona el servicio de mediación) y los protocolos de interacción o coreografía (qué orden deben seguir estas reglas).

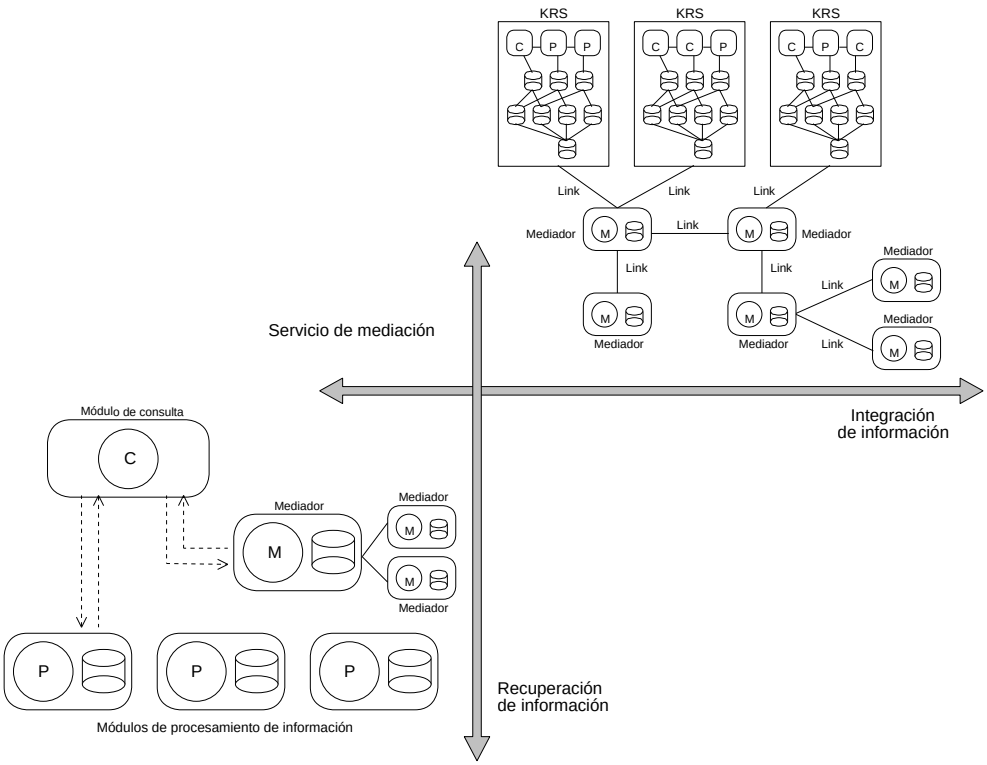


Figura 2.2: Doble dimensión en la funcionalidad de los servicios de mediación ontológicos.

Con las ideas anteriores presentes, en este *Capítulo*, se propone un modelo de mediación ontológico basado en el mecanismo *QS/RR* extendido que no implementa las cinco interfaces propuestas por *OMG* (*Lookup*, *Register*, *Admin*, *Link* y *Proxy*), sino que se centra en las tres primeras —dada la extensión del trabajo de investigación—.

2.2. Descripción del modelo

El concepto de mediación es un concepto bien conocido en el ámbito de la búsqueda y localización de servicios, donde un objeto o componente que necesita un servicio de-

terminado puede consultar a un mediador qué otros objetos o componentes de los que tiene registrados proporcionan dicho servicio. El modelo de mediación ontológico se basa en esta funcionalidad tradicional para adaptarla a la gestión de cualquier tipo de información —no sólo información sobre servicios—, mediante el uso de ontologías (por las razones expuestas en la *Sección* anterior). Concretamente, distingue ontologías de datos, que representan la información del dominio (metadatos), de ontologías de servicio/proceso, que representan las acciones que se pueden efectuar a través de cada una de las interfaces del servicio de mediación (*Acciones*), la información necesaria para realizarlas (*Conceptos*) y el resultado una vez finalizadas (*Predicados*).

La funcionalidad del servicio de mediación ontológico se ha separado en tres componentes claramente diferenciados, como se puede observar en la *Figura 2.3*:

— Componente de **comunicación**, que como su propio nombre indica, gestiona el mecanismo de comunicación del servicio.

— Componente de **análisis**, que se ocupa de la codificación y decodificación de los mensajes intercambiados, basados en cualquiera de las ontologías de servicio/proceso.

— Componente de **mediación**, encargado de realizar las labores propias del servicio de mediación.

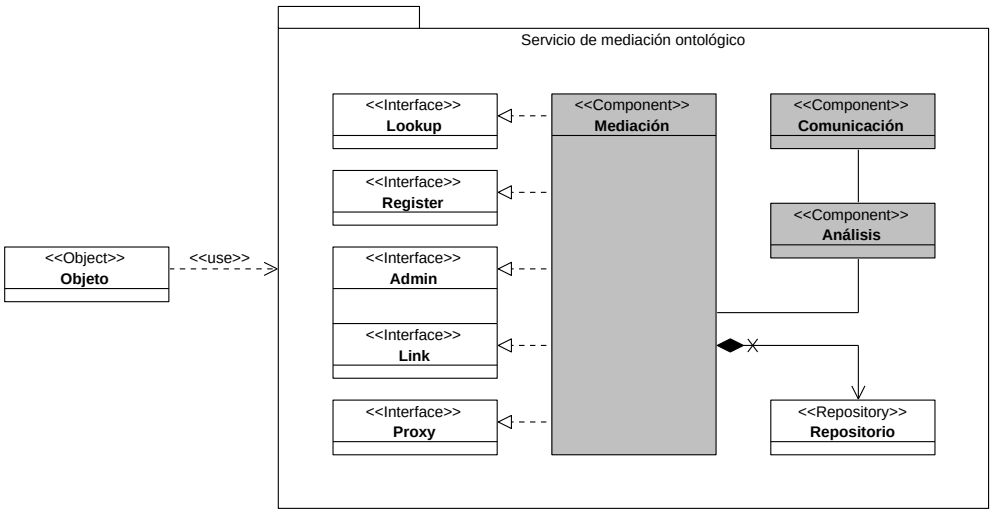


Figura 2.3: Visión general del servicio de mediación ontológico.

En las dos secciones siguientes se indican qué requerimientos debe cumplir este modelo de mediación y se describe su repositorio de metadatos.

2.2.1. Requisitos de un servicio de mediación ontológico

A continuación se presenta una lista de propiedades o requisitos que debe cumplir un servicio de mediación ontológico para entornos abiertos (una extensión de los presentados en [Iribarne et al., 2004]). Esta lista se ha elaborado para poder cubrir las limitaciones detectadas en la función de mediación estándar de *Open Distributed Processing* (*ODP*). Tanto el estudio como las limitaciones se han discutido en la *Sección 1.5*.

Propiedad 1 (Modelo de datos heterogéneo)
Un proceso de mediación no debería estar limitado a un único modelo de datos, sino que debería ser capaz de trabajar simultáneamente con distintos modelos de datos y plataformas. Los servicios de mediación heterogéneos también deberían ser capaces de mediar con múltiples protocolos de acceso a la información y de ajustarse a la evolución de los actuales modelos y a aquellos nuevos que vayan apareciendo.

Propiedad 2 (Un mediador es más que un motor de búsquedas)
Aunque los servicios de mediación podrían asemejarse a los motores de búsquedas en la web, en realidad, los primeros realizan búsquedas más estructuradas. En un servicio de mediación, las heurísticas de emparejamiento necesitan modelar el vocabulario, las funciones de distancia y las clases de equivalencia en un espacio de propiedades específicas del dominio, además del emparejamiento independiente del dominio basado en palabras claves que soportan los motores de búsqueda de documentos.

Propiedad 3 (Federación)
Para la cooperación entre mediadores se debería permitir la federación entre servicios de mediación utilizando diferentes estrategias. Por ejemplo, la estrategia tradicional de la federación "directa" obliga a que un servicio de mediación se comunique directamente con aquellos servicios de mediación con los que está federado. Si bien este esquema basado en federación es muy seguro y efectivo, la sobrecarga de la comunicación aumenta a medida que también lo hace el número de servicios de mediación federados. Por otro lado, una estrategia de federación "basada en repositorio" permite que múltiples servicios de mediación puedan leer y escribir a la vez en el mismo repositorio, desconociendo cada uno de ellos la presencia de los demás dentro de la federación y permitiendo, de esta forma, que la aproximación sea escalable. Aunque el principal problema de esta estrategia es la implementación de un repositorio comúnmente accesible, esto no parece ser un inconveniente en la web ya que los motores de búsqueda pueden operar con este tipo de repositorios.

Propiedad 4 (Composición y adaptación de servicios)
Los servicios de mediación actuales utilizan unos emparejamientos del tipo "uno a uno" entre las peticiones de servicio de los clientes y la disponibilidad real de los servicios almacenados en los repositorios a los que tienen acceso. Sin embargo, el servicio de mediación ontológico también debería proporcionar emparejamientos del tipo "uno a muchos", donde una petición del cliente se podría satisfacer mediante la composición de dos o más instancias de metadatos disponibles en los repositorios. Además de ello,

el emparejamiento "uno a uno" es también inadecuado en varias situaciones como, por ejemplo, cuando se presentan desajustes de formato entre la información demandada por el cliente y los documentos de metadatos localizados. Este problema se podría resolver componiendo las instancias de metadatos para reducir estas diferencias.

Propiedad 5 (Emparejamientos débiles)
En los procesos de mediación de servicios, sobre todo los que funcionan para sistemas abiertos independientemente extensibles (como Internet) y donde los nombres de los métodos y operaciones hacen referencia a unos servicios ofrecidos, es muy importante considerar el tipo de emparejamiento impuesto (débil o exacto), ya que los servicios seleccionados se obtienen de una forma totalmente arbitraria, no estandarizada y sin procedimientos de consenso. Por esta razón, para los procesos de búsqueda y recuperación de información, cuando se construye la lista de metadatos candidatos, un servicio de mediación debería permitir la utilización de emparejamientos parciales para seleccionar (de los repositorios) aquellos metadatos que se ajustan completamente a la demanda de información o a una parte de ella.

Propiedad 6 (Uso de heurísticas y métricas —preferencias)
Un servicio de mediación debería permitir que los usuarios pudieran especificar funciones heurísticas y métricas cuando buscan metadatos, especialmente para casos donde se realizan emparejamientos débiles. Así, por ejemplo, el servicio de mediación podría devolver los resultados ordenados según un criterio de búsqueda o descartar algunas secuencias de búsqueda en los repositorios donde se llevan a cabo, o también para evaluar los resultados obtenidos a partir de dichas heurísticas y métricas consideradas ("preferencias").

Propiedad 7 (Extensible y escalable)
Un servicio de mediación debería contemplar cualquier tipo de información acerca de los servicios (o metadatos), como datos de los creadores, información de marketing, etc. No obstante, también debería permitir que los usuarios pudieran incluir, de forma independiente, nuevos tipos de información para los metadatos que ellos exportan (registran). Y a su vez, debería ser capaz de utilizar esta nueva información como parte del metadato exportado. Además, independientemente de cómo estén federados y organizados los servicios de mediación, la escalabilidad también debería garantizarse para entornos distribuidos y abiertos a gran escala, como en Internet.

Propiedad 8 (Política "almacenamiento-y-reenvío")
Ante una petición de consulta de metadatos por parte de un cliente, un servicio de mediación debería responder con un resultado tras procesar dicha petición de consulta. Este resultado, obtenido en una acción de "petición-respuesta" y que se corresponde con un comportamiento "automático" del servicio de mediación, se puede referir a una lista de metadatos candidatos que satisfacen la petición de consulta o también puede referirse a un aviso de "fracaso" en el caso de que la búsqueda no haya encontrado ningún metadato que cumpla las características requeridas. En este último caso, también se le debería poder exigir a un servicio de mediación que una petición de consulta tenga que ser obligatoriamente satisfecha, almacenándola en el caso de que ésta no pudiera

ser satisfecha con la información disponible en ese momento y posponiendo la respuesta hasta que uno (o varios) proveedor(es) de metadatos realice(n) un registro (exportación) de un metadato que satisfaga la petición de consulta del cliente. A este comportamiento de "petición-respuesta" se le denomina comportamiento "en espera" o comportamiento de "almacenamiento-y-reenvío" (conocido también como *store and forward*).

Propiedad 9 (Delegación)
En relación con la propiedad anterior, un servicio de mediación también debería permitir la delegación de peticiones de consulta a otros servicios de mediación (conocidos) en caso de que éstas no pudieran ser satisfechas por él mismo. También se debería permitir la delegación completa o parcial de peticiones de consulta.

Propiedad 10 (Modelos de almacenamiento por demanda y extracción)
Un modelo por demanda (*push*), normalmente utilizado por los servicios de mediación basados en la función de mediación de *ODP*, es aquél donde los exportadores directamente contactan con el servicio de mediación para registrar sus metadatos. Una alternativa para el registro de metadatos, adecuada para sistemas de mediación que trabajan en entornos distribuidos y abiertos a gran escala, consiste en utilizar un modelo de almacenamiento por extracción (*pull*). En este esquema, los exportadores, en lugar de contactar directamente con los mediadores, publican los metadatos en sus sitios web para que, más tarde, los propios servicios de mediación se encarguen de "rastrear" la red en busca de nuevos metadatos. Se puede hacer uso de procesos *bots* y motores de búsqueda para mejorar el comportamiento de registro de los actuales servicios de mediación.

2.2.2. Repositorio de mediación

Del mismo modo que el servicio de mediación tradicional hace uso de un repositorio donde almacena las especificaciones de los objetos o componentes y de los servicios que ofrecen —registradas en las operaciones de exportación y consultadas en las operaciones de importación para la localización del objeto que mejor se ajuste a unos requisitos—, el servicio de mediación ontológico utiliza un repositorio, en este caso, de documentos con un conjunto de metadatos de la información del dominio. Estos metadatos favorecen, por un lado, la integración de diferentes fuentes de información (como se puede observar en la *Figura 2.4*) registradas previamente en el servicio de mediación, lo que se puede hacer extensible a los componentes o sistemas que las gestionan, y por otro, facilitan el proceso de búsqueda y recuperación de la información ya que actúan de filtro restringiendo las búsquedas a determinadas fuentes de información y no a todas. El repositorio del servicio de mediación ontológico se puede definir simplemente como una colección de instancias de una ontología que representa dichos metadatos.

A modo de ejemplo y como adelanto del caso de estudio que se analizará en el *Capítulo 4*, la *Figura 2.5* recoge el modelo conceptual de la ontología cuyas instancias maneja el servicio de mediación ontológico del sistema *SOLERES*. Esta ontología se denomina *Environmental Information metaData* (*EID*) y representa un dominio medioambiental concreto centrado en la información de mapas cartográficos, imágenes de satélite y sus clasificaciones. Como aparece indicado en el modelo, una clasificación (`Classification`)

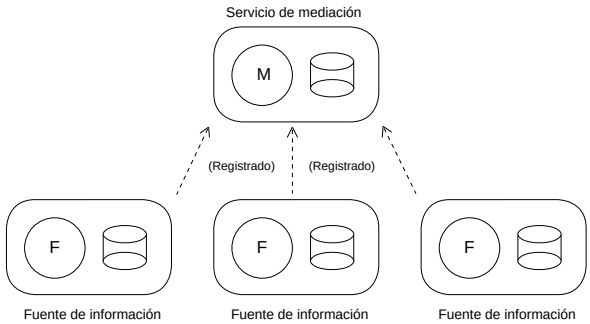


Figura 2.4: Integración de información en el servicio de mediación ontológico.

posee cierta información geográfica asociada (`Geography`) y es realizada en un determinado momento del tiempo (`Time`) por un técnico o un grupo de técnicos (`Technician`). Si se trata de la clasificación de un mapa cartográfico, entonces se almacena información de sus diferentes capas (`Layer`), cada una de las cuales agrupa un conjunto de variables (`Variable`); finalmente, si se trata de una clasificación de imágenes de satélite (`SatelliteImage`) entonces se guarda información de sus bandas (`Band`).

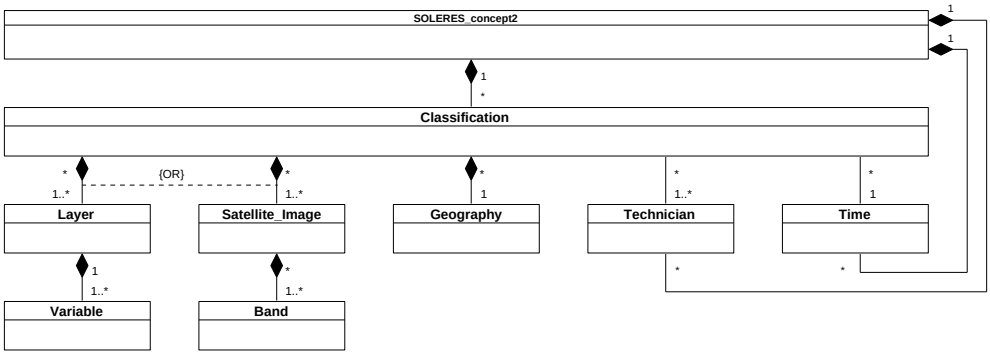


Figura 2.5: Ontología para la representación de metadatos en un dominio concreto.

2.3. Modelos de mediación: reflexión, delegación y federación

En esta *Sección* se va a describir cómo opera el servicio *OntoTrader* en el proceso de consulta, siguiendo el mecanismo *QS/RR*, desde que un objeto o componente realiza una petición de datos hasta que se devuelve el resultado. Como ya se ha mencionado en la *Sección 2.1*, este mecanismo está basado en un modelo cliente/servidor de tres niveles. Se compone básicamente de una serie de elementos $< \mathcal{O}, \mathcal{M}, \mathcal{I} >$, cada uno de

los cuales interviene en un nivel diferente, dependiendo del tratamiento de la consulta. El *Nivel 1* (*N1*) recae en el lado del cliente: las consultas son generadas y tratadas por un objeto determinado $\mathcal{O}$. El *Nivel 3* (*N3*) recae en el lado del servidor, donde reside toda la información $\mathcal{I}$ del sistema —no se debe confundir esta información con la que gestiona el servicio de mediación—. El *Nivel 2* (*N2*) es el *middleware* que permite la localización de las fuentes de información; en este nivel es donde opera el servicio de mediación $\mathcal{M}$ —y donde entra en juego su repositorio de metadatos—. Todos estos módulos utilizan una ontología de servicio/proceso, concretamente la ontología *Lookup* (que será descrita en la *Sección 2.4.2*) para interactuar. Como premisa indispensable para su funcionamiento, un objeto debe estar asociado con un servicio de mediación. También, un servicio de mediación puede estar asociado con una o más fuentes de datos externas o recursos. Por último, un servicio de mediación puede asociarse con otro u otros mediante el mecanismo de federación.

En esta arquitectura de tres niveles, a su vez, se pueden dar tres escenarios o modelos operativos de mediación diferentes, que se han denominado: reflexión, delegación y federación. La *Figura 2.6* muestra los tres niveles (*N1*, *N2*, *N3*), donde residen los tres elementos básicos ($\mathcal{O}$, $\mathcal{M}$, $\mathcal{I}$) con estos tres escenarios posibles y que se describen a continuación:

— La **reflexión** es un modelo para la mediación directa. La consulta puede ser resuelta directamente por un mediador: la consulta es generada por el objeto $\mathcal{O}$ y la información demandada se puede localizar en los documentos de metadatos que almacena el repositorio asociado al servicio de mediación $\mathcal{M}$. En este caso, interviene el par $< \mathcal{O}, \mathcal{M} >$ del modelo.

— En el modelo de **delegación** se negocia indirectamente con el servicio de mediación. La consulta es resuelta parcialmente por el mediador: es generada por un objeto $\mathcal{O}$ en el nivel *N1* y viaja al nivel *N2*, donde el mediador $\mathcal{M}$ localiza la fuente (o fuentes) de información $\mathcal{I}$ en su repositorio —infiriendo esta información de los documentos de metadatos—; seguidamente, el mediador delega la consulta a la fuente (o fuentes) de información externa $\mathcal{I}$. En este caso, la serie de elementos que intervienen es $< \mathcal{O}, \mathcal{M}, \mathcal{I} >$.

— La **federación** es el caso en el que dos o más servicios de mediación están configurados para utilizar un mecanismo de delegación. Como en los casos anteriores, la consulta procede de un objeto $\mathcal{O}$; esta consulta se transmite al servicio de mediación $\mathcal{M}$ asociado, el cual la puede propagar a otro servicio de mediación $\mathcal{M}$ con el que se encuentre federado, que localiza las fuentes de información externas $\mathcal{I}$. En este caso, intervienen los elementos de la serie $< \mathcal{O}, \mathcal{M}, \mathcal{M}, \mathcal{I} >$.

2.4. Arquitectura del modelo *OntoTrader*

El modelo de mediación ontológico que se propone, como ya se ha mencionado, se basa en la especificación *ODP* y ha sido adaptado para el manejo de documentos con metadatos de cualquier tipo de información, en lugar de especificaciones de servicios. Este modelo

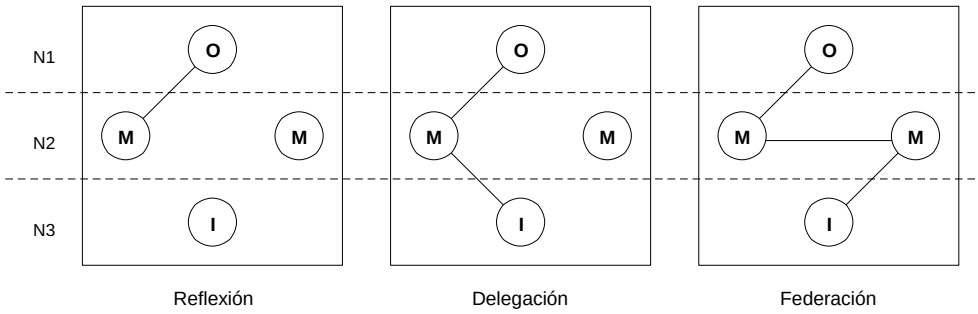


Figura 2.6: Modelos operativos de mediación.

sigue el diseño de un servicio de mediación *stand-alone* que implementa las interfaces *Lookup*, *Register* y *Admin*, como se puede apreciar en el modelo conceptual representado en la *Figura 2.7*. La interfaz *Lookup* únicamente se ocupa de la consulta de información; la interfaz *Register* permite la gestión completa de la información con la que opera el servicio; por último, a través de la interfaz *Admin* se pueden modificar los parámetros de configuración y las políticas del mediador.

Las ontologías de servicio/proceso se utilizan para definir el comportamiento y los protocolos de interacción que deben seguir los objetos o componentes que interactúen con el servicio de mediación a través de las interfaces que implementa. Todas las ontologías han sido modeladas mediante diagramas de clases *UML* en términos de conceptos (entidades del dominio), acciones realizadas en el dominio (acciones que afectan a los conceptos) y predicados (expresiones relacionadas con los conceptos), cada uno de ellos representado con el estereotipo «`Concept`», «`Action`» y «`Predicate`», respectivamente. En las siguientes secciones se analizan con más detalle cada una de ellas.

2.4.1. Ontología *Lookup*

La búsqueda de información en un repositorio específico —bajo unos criterios o políticas establecidos previamente— y recuperación de la misma se lleva a cabo mediante la ontología *Lookup*. La *Figura 2.8* representa la estructura de esta ontología. La acción `Query` hace uso de los conceptos `QueryForm` y `PolicySeq`. El primero representa una consulta expresada en un lenguaje específico y tiene varias propiedades: `id`, `uri`, `type`, `source` y `target`. La propiedad `id` es un identificador de la consulta; `uri` es la referencia a un archivo que almacena la consulta expresada en un lenguaje determinado (en la *Tabla 2.1* se puede observar un ejemplo de consulta en *SPARQL* sobre el repositorio *EID* en *SOLERES*, para obtener "el nombre de las variables utilizadas en las clasificaciones realizadas durante el año 2008"); `type` indica el tipo concreto de consulta (por ejemplo, si se trata de una consulta sobre el repositorio o sobre otra consulta anterior); `source` identifica el objeto o componente del que procede la consulta y, por último, `target` permite identificar el objeto o componente destinatario de la misma.

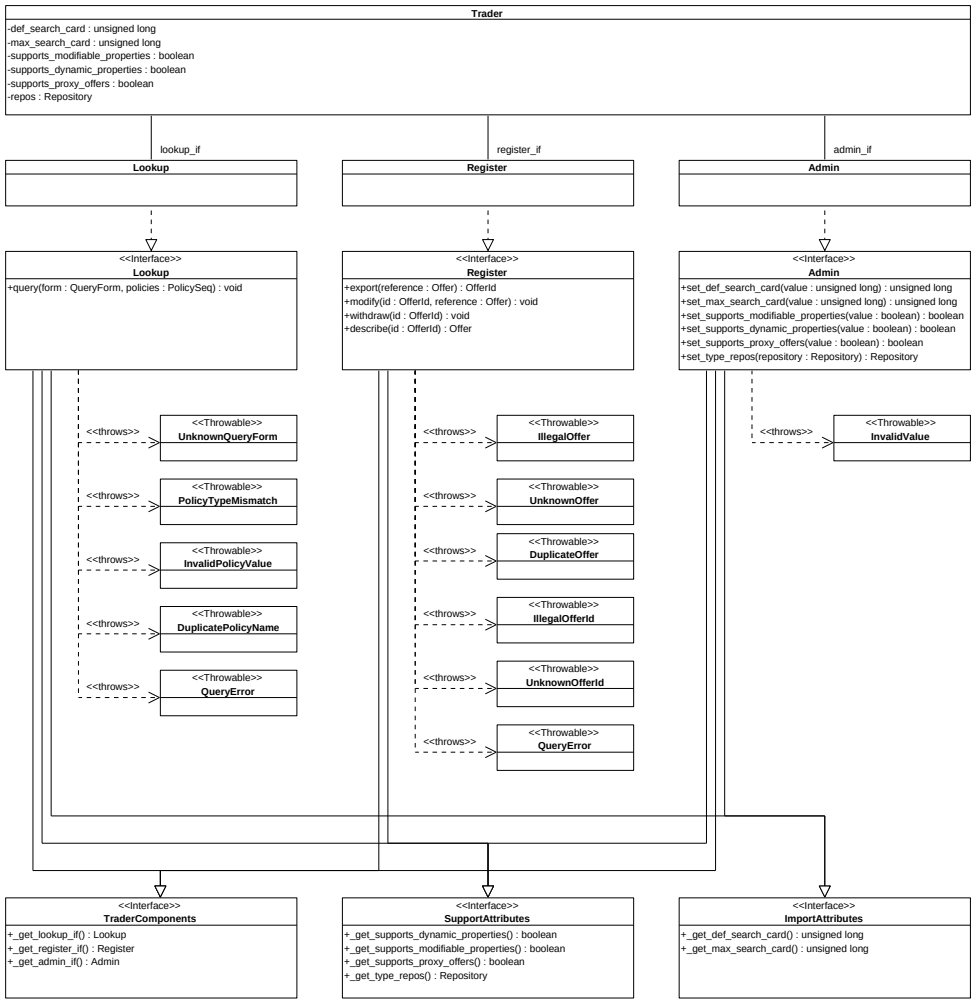


Figura 2.7: Modelo conceptual de mediación *stand-alone*.

Por otro lado, el concepto `PolicySeq` representa un conjunto de políticas (`Policies`) que se pueden establecer para realizar una consulta. El concepto `Policy` es un concepto abstracto que se utiliza para representar una política específica mediante la tupla (nombre, valor). Por este motivo, su única propiedad es `value`, que representa el valor de una `Policy` dada. Las tres políticas que se han definido son: `def_search_cardPolicy`, `max_search_cardPolicy` y `exact_type_matchPolicy`. La primera de ellas indica el número de documentos a localizar por defecto; `max_search_cardPolicy` indica el valor máximo del número de documentos a localizar en la consulta; y `exact_type_matchPolicy` establece la coincidencia exacta o no de los documentos a localizar con las condiciones

de la consulta (un valor `true` indica que los documentos que se devuelvan deben cumplir las condiciones establecidas en la consulta al 100 %, mientras que un valor `false` indica que los documentos localizados pueden no coincidir exactamente con las condiciones establecidas en ésta).

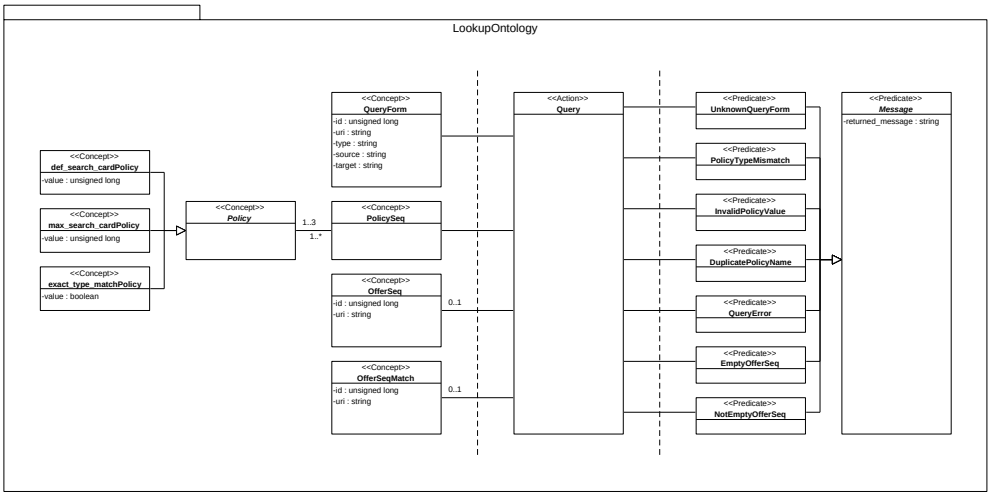


Figura 2.8: Modelo conceptual de la ontología *Lookup*.

```
1  PREFIX eid: <http://www.ual.es/acg/ont/TKRS/EIDOntology.owl#>
2  PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3  SELECT DISTINCT ?EID__VariableName
4  WHERE
5  {
6    ?EID__EID eid:EID_eimId ?EID__EID_eimID .
7    ?EID__EID eid:EID_hasClassification ?EID__Classification .
8    ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
9    ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
10   ?EID__Variable eid:Variable_name ?EID__VariableName .
11   ?EID__Classification eid:Classification_hasTime ?EID__Time .
12   ?EID__Time eid:Time_year ?EID__TimeYear .
13   FILTER (?EID__TimeYear = 2008) .
14 }
15 ORDER BY ASC(?EID__VariableName)
```

Tabla 2.1: Ejemplo de consulta expresada en *SPARQL*.

La salida de la acción `Query` puede ser una excepción controlada o el resultado en sí de la consulta, todos ellos representados mediante el predicado abstracto `Message` del que heredan. Este predicado tiene la propiedad `returned_message`, que contiene el texto

del mensaje de salida. Las excepciones que se pueden producir son: `UnknownQueryForm`, `PolicyTypeMismatch`, `InvalidPolicyValue`, `DuplicatePolicyName` y `QueryError`. La excepción `UnknownQueryForm` indica que no se puede obtener la consulta debido a que el archivo especificado en la `uri` no es accesible; `PolicyTypeMismatch` indica que el tipo del `value` especificado para la `Policy` no es el adecuado; `InvalidPolicyValue` indica que el valor especificado para la `Policy` no está dentro del rango de valores permitidos para dicha `Policy`; `DuplicatePolicyName` indica que se ha especificado más de un `value` para una misma `Policy` dentro de `PolicySeq`; por último, la excepción `QueryError` indica que se ha producido un error en la ejecución de la consulta.

Cuando la consulta se realiza con éxito, se emplea el predicado `EmptyOfferSeq`, si no devuelve ningún documento, o `NotEmptyOfferSeq`, en el caso de que sí lo haga. Este último predicado, a su vez, hace uso de los conceptos `OfferSeq` y `OfferSeqMatch`, para almacenar el conjunto de documentos localizados en la consulta y el porcentaje de coincidencia de cada documento obtenido según las condiciones de la consulta, respectivamente. El concepto `OfferSeq` tiene como propiedades un `id` de la consulta y la `uri` del archivo que almacena los documentos obtenidos, al igual que el concepto `OfferSeqMatch`, donde la `uri` ahora hace referencia a otro archivo que contiene un conjunto de documentos en forma de tuplas $<$ *identificador del registro obtenido*, *porcentaje de coincidencia* $>$.

La *Tabla 2.2* recoge, a modo de resumen, los conceptos, acciones y predicados de la ontología *Lookup*. En ella no se incluyen los predicados abstractos `Message` y `Policy`.

Ontología	**Conceptos**	**Acciones**	**Predicados**
Lookup	QueryForm PolicySeq def_search_cardPolicy max_search_cardPolicy exact_type_matchPolicy OfferSeq OfferSeqMatch	Query	UnknownQueryForm PolicyTypeMismatch InvalidPolicyValue DuplicatePolicyName QueryError EmptyOfferSeq NotEmptyOfferSeq
Register	Offer OfferId	Export Modify Withdraw Describe	IllegalOffer UnknownOffer DuplicateOffer IllegalOfferId UnknownOfferId QueryError ExportedOffer ModifiedOffer WithdrawnOffer NotDescribedOffer DescribedOffer
Admin	Def_search_card Max_search_card Offer_repos	SetDef_search_card SetMax_search_card SetOffer_repos GetDef_search_card GetMax_search_card GetOffer_repos	ModifiedDef_search_card ModifiedMax_search_card ModifiedOffer_repos ReturnedDef_search_card ReturnedMax_search_card ReturnedOffer_repos InvalidValue

Tabla 2.2: Elementos de las ontologías *Lookup*, *Register* y *Admin*.

2.4.2. Ontología *Register*

Las tareas de gestión de los documentos de metadatos que almacena el servicio de mediación en su repositorio recaen sobre la ontología *Register* y soportan la creación, modificación, eliminación y descripción de documentos. Esta ontología considera las acciones `Export`, `Modify`, `Withdraw` y `Describe`. La acción `Export` inserta un documento previamente especificado en el concepto `Offer`; este documento se almacena en un archivo referenciado por la propiedad `uri` del concepto. La acción `Modify` representa la modificación de un documento y utiliza el concepto `Offer` para referirse al nuevo documento con la información modificada y el concepto `OfferId` para representar el identificador del documento que va a ser modificado. Por último, las acciones `Withdraw` y `Describe` representan la eliminación y descripción, respectivamente, de un documento en particular especificado por el concepto `OfferId`. La *Figura 2.9* representa la estructura de la ontología `Register`.

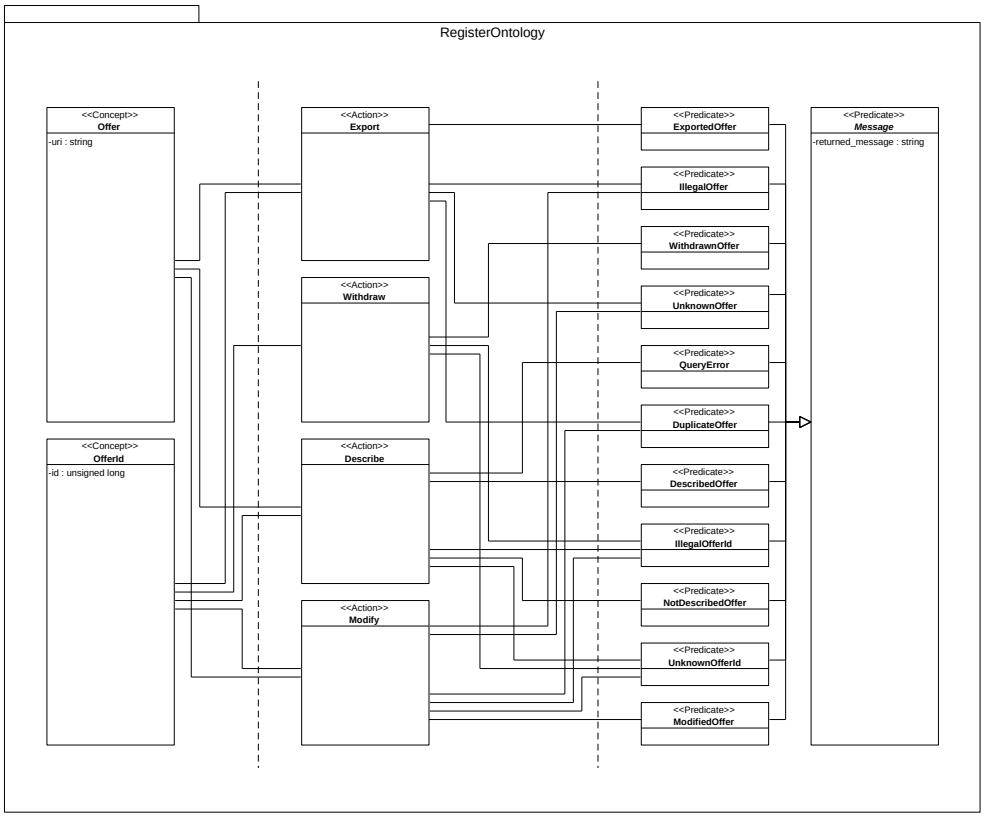


Figura 2.9: Modelo conceptual de la ontología *Register*.

El resultado de una acción puede producir una excepción controlada en el sistema o un resultado válido. Ambos se representan mediante un predicado que hereda del predicado abstracto `Message`, como ocurría en la ontología *Lookup*. Las excepciones que se pueden producir son `IllegalOffer`, `UnknownOffer`, `DuplicateOffer`, `IllegalOfferId`, `UnknownOfferId` y `QueryError`. Las tres primeras se pueden dar con las acciones `Export` y `Modify`: la excepción `IllegalOffer` indica que el documento de metadatos que se pretende insertar o modificar no es correcto; `UnknownOffer` indica que no se puede obtener el documento para su inserción o modificación debido a que el archivo especificado en la `uri` no es accesible; y la excepción `DuplicateOffer` indica que el documento ya existe en el repositorio. Las acciones `Modify`, `Withdraw` y `Describe` también pueden generar las excepciones `IllegalOfferId` y `UnknownOfferId`: la primera de ellas indica que el `id` del documento que se pretende modificar, eliminar o describir no es correcto, mientras que la segunda indica que no existe ningún documento en el repositorio con dicho `id` para su modificación, eliminación o descripción. Por último, la acción `Describe` puede generar la excepción `QueryError`, que indica que se ha producido un error al realizar la consulta del documento especificado para su descripción.

La ontología también define los predicados `ExportedOffer`, `ModifiedOffer` y `WithdrawnOffer` para indicar, respectivamente, que las acciones `Export`, `Modify` y `Withdraw` han sido realizadas con éxito. El predicado `ExportedOffer` hace uso del concepto `OfferId` para indicar el id asignado al nuevo documento. En lo que se refiere a la acción `Describe`, se emplea el predicado `NotDescribedOffer`, si no se localiza el documento, o `DescribedOffer`, en el caso de que sí se localice. Este último predicado hace uso del concepto `Offer` para devolver el documento solicitado.

La *Tabla 2.2* también recoge, a modo de resumen, los conceptos, acciones y predicados de la ontología *Register*. Tampoco se incluye el predicado abstracto `Message`.

2.4.3. Ontología *Admin*

La ontología *Admin* es utilizada para referirse a las acciones que modifican los principales parámetros de configuración y las políticas del servicio de mediación. Por ejemplo, esta ontología permite modificar el número máximo de resultados obtenidos tras la ejecución de una consulta. En la *Figura 2.10* aparece representada su estructura. Esta ontología define tres pares de acciones: (i) `SetDef_search_card`/`GetDef_search_card` permiten establecer/obtener el número de documentos de metadatos que se recuperan de forma predeterminada en una búsqueda y utiliza el concepto `Def_search_card` para almacenar esta información; (ii) `SetMax_search_card`/`GetMax_search_card` permiten establecer/obtener el número máximo de documentos que se recuperan en una búsqueda y usan el concepto `Max_search_card` para almacenar dicha información; (iii) `SetOffer_repos`/`GetOffer_ repos` pueden establecer/obtener la dirección (en formato *URI*) del repositorio con los documentos de metadatos que utiliza el servicio de mediación (el concepto `Offer_repos` se utiliza para almacenar la dirección).

Cuando las acciones `Set-` se realizan con éxito, el mediador devuelve los predicados `ModifiedDef_search_card`, `ModifiedMax_search_card` y `ModifiedOffer_repos`, para indicar que se ha establecido satisfactoriamente el valor de los parámetros `Def_search_card`, `Max_search_card` y `Offer_repos`, respectivamente. Por otro lado, puede suceder

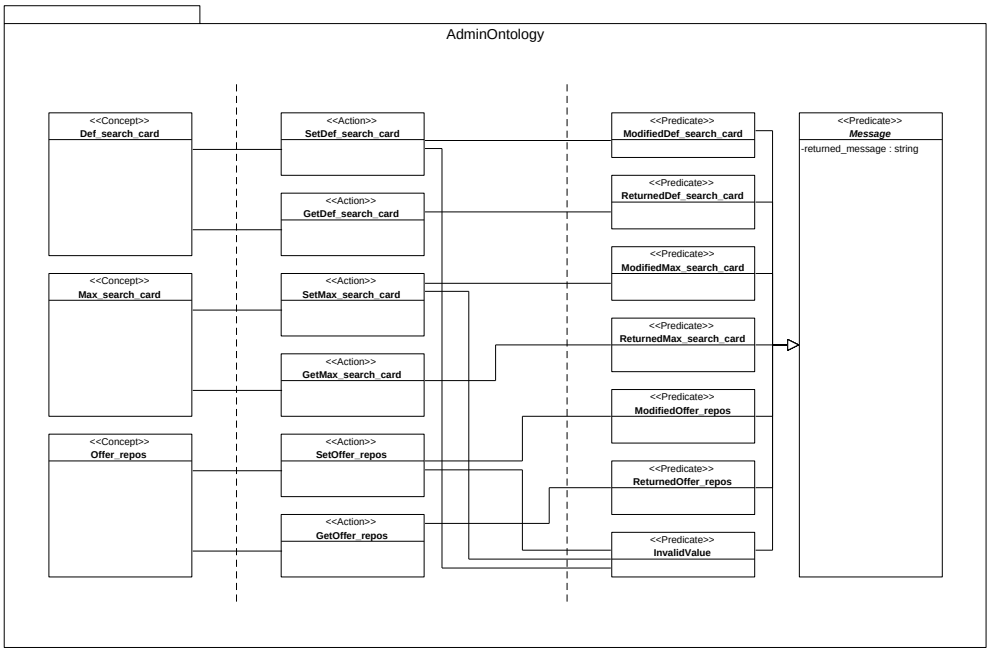


Figura 2.10: Modelo conceptual de la ontología *Admin*.

que devuelva los predicados `ReturnedDef_search_card`, `ReturnedMax_search_card` y `Returned- Offer_repos`, para indicar que se ha devuelto correctamente el valor de los parámetros correspondientes a las acciones `Get-`. La única excepción controlada que se puede producir es el predicado `InvalidValue` y puede darse con las acciones `SetDef_search_card`, `SetMax_search_card` o `SetOffer_repos`; esta excepción indica que el valor especificado para el parámetro en cuestión no es válido. Todos los predicados heredan del predicado abstracto `Message`, como ocurría en las ontologías *Lookup* y *Register*, que contiene la propiedad `returned_message` para almacenar el texto del mensaje de salida.

La *Tabla 2.2* recoge, a modo de resumen, los conceptos, acciones y predicados de la ontología *Admin*. En ella, no se incluye el predicado abstracto `Message`.

2.5. Formalización del modelo *OntoTrader*

Esta *Sección* presenta la formalización del modelo de mediación *OntoTrader* desde un punto de vista ontológico.

Definición 1 (*OntoTrader*): *OntoTrader* $\mathcal{O}$ se define como un conjunto finito formado por dos tipos de ontologías $\mathcal{O} = \{\mathcal{D}, \mathcal{S}\}$,

donde:

(i) $\mathcal{D}$ representa la ontología de datos.

(ii) $\mathcal{S}$ representa las ontologías de servicio/proceso.

En el modelo *OntoTrader*, la ontología de datos $\mathcal{D}$ se refiere al repositorio asociado al servicio de mediación que almacena los metadatos de la información de un dominio concreto. Las ontologías de servicio/proceso $\mathcal{S}$ están relacionadas con la funcionalidad del mediador.

Definición 2 (Ontologías de servicio/proceso): Las ontologías de servicio/proceso $\mathcal{S}$ se definen como un conjunto finito $\mathcal{S} = \{\mathcal{C}, \mathcal{A}, \mathcal{P}\}$,

donde:

(i) $\mathcal{C}$ representa un conjunto finito de conceptos (`Concept` en los modelos conceptuales), $\mathcal{C} = \{\mathcal{C}_1, \cdots, \mathcal{C}_i\} \ / \ i = \{1, \cdots, n\} \ , \ n \in \aleph$.

(ii) $\mathcal{A}$ representa un conjunto finito de acciones (`Action`), $\mathcal{A} = \{\mathcal{A}_1, \cdots, \mathcal{A}_j\} \ / \ j = \{1, \cdots, n\}$.

(iii) $\mathcal{P}$ representa un conjunto finito de predicados (`Predicate`), $\mathcal{P} = \{\mathcal{P}_1, \cdots, \mathcal{P}_k\} \ / \ k = \{1, \cdots, n\}$.

Las ontologías de servicio/proceso recogen las posibles acciones $\mathcal{A}$ que se pueden efectuar a través de cada interfaz del servicio de mediación, la información $\mathcal{C}$ necesaria para realizarlas y el resultado $\mathcal{P}$ una vez finalizadas (bien una excepción controlada en el transcurso de la acción o bien que ésta ha sido realizada satisfactoriamente).

Dados un conjunto de conceptos $\mathcal{C}_n \mid \mathcal{C}_n \subset \mathcal{C}$, un conjunto de acciones $\mathcal{A}_n \mid \mathcal{A}_n \subset \mathcal{A}$ y un conjunto de predicados $\mathcal{P}_n \mid \mathcal{P}_n \subset \mathcal{P}$,

entonces:

(i) $\mathcal{C}_n \times \mathcal{A}_n \times \mathcal{P}_n \Rightarrow \mathcal{S}_n$

donde $\mathcal{S}_n$ representa una ontología concreta de las cinco asociadas a las diferentes interfaces del servicio de mediación: *Lookup*, *Register*, *Admin*, *Link* y *Proxy*.

En un mediador debe estar presentes, al menos, la funcionalidad de las interfaces *Lookup* y *Register*, junto con sus correspondientes ontologías: la primera para permitir la consulta de los documentos almacenados en el repositorio y, la segunda, para la gestión de dichos documentos. La presencia del resto de interfaces no es obligatoria: la ontología *Admin* facilitaría la configuración de los parámetros y las políticas del mediador (por ejemplo, número de mediadores federados permitido, políticas de búsqueda, etc.); la ontología *Link* se refiere a la forma de interconectarse los mediadores (proceso de federación); y, finalmente, la ontología *Proxy* modela antiguas propiedades en sistemas de federación de mediadores.

Para una mejor comprensión, esta *Sección* se ha dividido en tres subsecciones, en cada una de las cuales se realiza la formalización de una ontología relacionada con la interfaz correspondiente del modelo de mediación *stand-alone* descrito en este *Capítulo*.

2.5.1. Formalización de la ontología *Lookup*

En esta *Sección* se presenta la formalización de la ontología *Lookup*.

Definición 3 (Ontología *Lookup*): Dados el conjunto de conceptos $\mathcal{C}_\mathcal{L}$, el conjunto de acciones $\mathcal{A}_\mathcal{L}$ y el conjunto de predicados $\mathcal{P}_\mathcal{L}$, todos ellos referidos a la ontología *Lookup* $\mathcal{S}_\mathcal{L}$, ésta se define como:

$$\mathcal{C}_\mathcal{L} \times \mathcal{A}_\mathcal{L} \times \mathcal{P}_\mathcal{L} \Rightarrow \mathcal{S}_\mathcal{L},$$

siendo:

(i) $\mathcal{C}_\mathcal{L} = \{\mathcal{C}^1_\mathcal{L}, \cdots, \mathcal{C}^i_\mathcal{L}\}$ / $i = \{1, \cdots, 4\}$, donde:

- (a) $\mathcal{C}^1_\mathcal{L}$ representa una consulta (`QueryForm` en el modelo conceptual).
- (b) $\mathcal{C}^2_\mathcal{L}$ representa una secuencia de políticas a seguir con la consulta (`PolicySeq`).
- (c) $\mathcal{C}^3_\mathcal{L}$ representa una secuencia de resultados (`OfferSeq`).
- (d) $\mathcal{C}^4_\mathcal{L}$ representa una secuencia con el porcentaje de coincidencia de cada resultado (`OfferSeqMatch`).

(ii) $\mathcal{A}_\mathcal{L} = \{\mathcal{A}^1_\mathcal{L}\}$, donde:

- (a) $\mathcal{A}^1_\mathcal{L}$ representa la acción de consultar metadatos (`Query`).

(iii) $\mathcal{P}_\mathcal{L} = \{\mathcal{P}^1_\mathcal{L}, \cdots, \mathcal{P}^k_\mathcal{L}\}$ / $k = \{1, \cdots, 7\}$, donde:

- (a) $\mathcal{P}^1_\mathcal{L}, \cdots, \mathcal{P}^5_\mathcal{L}$ representan una excepción controlada en el transcurso de las acciones del conjunto $\mathcal{A}_\mathcal{L}$ (`UnknownQueryForm`, `PolicyTypeMismatch`, `InvalidPolicyValue`, `DuplicatePolicyName` y `QueryError`, respectivamente).
- (b) $\mathcal{P}^6_\mathcal{L}$ y $\mathcal{P}^7_\mathcal{L}$ representan que las acciones $\mathcal{A}_\mathcal{L}$ se han realizado satisfactoriamente (`EmptyOfferSeq` y `NotEmptyOfferSeq`, respectivamente).

Definición 4 (Relación de solicitud en la ontología *Lookup*): Dados los conceptos $\mathcal{C}^1_\mathcal{L}$ y $\mathcal{C}^2_\mathcal{L}$, y la acción $\mathcal{A}^1_\mathcal{L}$ de la ontología *Lookup* $\mathcal{S}_\mathcal{L}$, se define la relación de solicitud $\mathcal{R}_\mathcal{L}$ como $\mathcal{R}_\mathcal{L} \subseteq (\mathcal{C}^1_\mathcal{L} \times \mathcal{A}^1_\mathcal{L}) \cup (\mathcal{C}^1_\mathcal{L} \times \mathcal{A}^1_\mathcal{L} \times \mathcal{C}^2_\mathcal{L})$.

Definición 5 (Relación de respuesta en la ontología *Lookup*): Dados la acción $\mathcal{A}^1_\mathcal{L}$, un predicado $\mathcal{P}^k_\mathcal{L}$ / $k = \{1, \cdots, 7\}$ y los conceptos $\mathcal{C}^3_\mathcal{L}$ y $\mathcal{C}^4_\mathcal{L}$ de la ontología *Lookup* $\mathcal{S}_\mathcal{L}$, se define la relación de respuesta $\overline{\mathcal{R}_\mathcal{L}}$ como $\overline{\mathcal{R}_\mathcal{L}} \subseteq (\mathcal{A}^1_\mathcal{L} \times \mathcal{P}^k_\mathcal{L}) \cup (\mathcal{A}^1_\mathcal{L} \times \mathcal{P}^k_\mathcal{L} \times \mathcal{C}^3_\mathcal{L}) \cup (\mathcal{A}^1_\mathcal{L} \times \mathcal{P}^k_\mathcal{L} \times \mathcal{C}^3_\mathcal{L} \times \mathcal{C}^4_\mathcal{L})$.

Definición 6 (Usos de la ontología *Lookup*): Dados la acción $\mathcal{A}^1_\mathcal{L}$, un predicado $\mathcal{P}^k_\mathcal{L}$ / $k = \{1, \cdots, 7\}$ y los conceptos $\mathcal{C}^3_\mathcal{L}$ y $\mathcal{C}^4_\mathcal{L}$ de la ontología *Lookup* $\mathcal{S}_\mathcal{L}$, se cumplen las siguientes condiciones para sus usos:

(i) $(\mathcal{A}^1_\mathcal{L} \times \mathcal{P}^k_\mathcal{L}) \Rightarrow \mathcal{P}^k_\mathcal{L} = \mathcal{P}^1_\mathcal{L} \vee \mathcal{P}^2_\mathcal{L} \vee \mathcal{P}^3_\mathcal{L} \vee \mathcal{P}^4_\mathcal{L} \vee \mathcal{P}^5_\mathcal{L} \vee \mathcal{P}^6_\mathcal{L}$

(ii) $(\mathcal{A}^1_\mathcal{L} \times \mathcal{P}^k_\mathcal{L} \times \mathcal{C}^3_\mathcal{L}) \Rightarrow \mathcal{P}^k_\mathcal{L} = \mathcal{P}^7_\mathcal{L}$

(iii) $(\mathcal{A}^1_\mathcal{L} \times \mathcal{P}^k_\mathcal{L} \times \mathcal{C}^3_\mathcal{L} \times \mathcal{C}^4_\mathcal{L}) \Rightarrow \mathcal{P}^k_\mathcal{L} = \mathcal{P}^7_\mathcal{L}$

2.5.2. Formalización de la ontología *Register*

En esta *Sección* se presenta la formalización de la ontología *Register*.

Definición 7 (Ontología *Register*): Dados el conjunto de conceptos $\mathcal{C}_\mathcal{R}$, el conjunto de acciones $\mathcal{A}_\mathcal{R}$ y el conjunto de predicados $\mathcal{P}_\mathcal{R}$, todos ellos referidos a la ontología *Register* $\mathcal{S}_\mathcal{R}$, ésta se define como:

$$\mathcal{C}_\mathcal{R} \times \mathcal{A}_\mathcal{R} \times \mathcal{P}_\mathcal{R} \Rightarrow \mathcal{S}_\mathcal{R},$$

siendo:

(i) $\mathcal{C}_\mathcal{R} = \{\mathcal{C}^1_\mathcal{R}, \mathcal{C}^2_\mathcal{R}\}$, donde:

- (a) $\mathcal{C}^1_\mathcal{R}$ representa la *URI* de un documento de metadatos (`Offer` en el modelo conceptual).
- (b) $\mathcal{C}^2_\mathcal{R}$ representa el identificador único de un documento de metadatos (`OfferId`).

(ii) $\mathcal{A}_\mathcal{R} = \{\mathcal{A}^1_\mathcal{R}, \cdots, \mathcal{A}^j_\mathcal{R}\}$ / $j = \{1, \cdots, 4\}$, donde:

- (a) $\mathcal{A}^1_\mathcal{R}$ representa la acción de añadir un documento de metadatos (`Export`).
- (b) $\mathcal{A}^2_\mathcal{R}$ representa la acción de modificar un documento de metadatos (`Modify`).
- (c) $\mathcal{A}^3_\mathcal{R}$ representa la acción de eliminar un documento de metadatos (`Withdraw`).
- (d) $\mathcal{A}^4_\mathcal{R}$ representa la acción de consultar un documento de metadatos (`Describe`).

(iii) $\mathcal{P}_\mathcal{R} = \{\mathcal{P}^1_\mathcal{R}, \cdots, \mathcal{P}^k_\mathcal{R}\}$ / $k = \{1, \cdots, 11\}$, donde:

- (a) $\mathcal{P}^1_\mathcal{R}, \cdots, \mathcal{P}^6_\mathcal{R}$ representan una excepción controlada en el transcurso de las acciones del conjunto $\mathcal{A}_\mathcal{R}$ (`IllegalOffer`, `UnknownOffer`, `DuplicateOffer`, `IllegalOfferId`, `UnknownOfferId` y `QueryError`, respectivamente).
- (b) $\mathcal{P}^7_\mathcal{R}, \cdots, \mathcal{P}^{11}_\mathcal{R}$ representan que las acciones $\mathcal{A}_\mathcal{R}$ se han realizado satisfactoriamente (`ExportedOffer`, `WithdrawnOffer`, `DescribedOffer`, `NotDescribed-Offer` y `ModifiedOffer`, respectivamente).

Definición 8 (Relación de solicitud en la ontología *Register*): Dados los conceptos $\mathcal{C}^1_\mathcal{R}$ y $\mathcal{C}^2_\mathcal{R}$, y las acciones $\mathcal{A}^1_\mathcal{R}, \cdots, \mathcal{A}^4_\mathcal{R}$ de la ontología *Register* $\mathcal{S}_\mathcal{R}$, se define la relación de solicitud $\mathcal{R}_\mathcal{R}$ como $\mathcal{R}_\mathcal{R} \subseteq (\mathcal{C}^1_\mathcal{R} \times \mathcal{A}^1_\mathcal{R}) \cup (\mathcal{C}^2_\mathcal{R} \times \mathcal{C}^1_\mathcal{R} \times \mathcal{A}^2_\mathcal{R}) \cup (\mathcal{C}^2_\mathcal{R} \times \mathcal{A}^3_\mathcal{R}) \cup (\mathcal{C}^2_\mathcal{R} \times \mathcal{A}^4_\mathcal{R})$.

Definición 9 (Relación de respuesta en la ontología *Register*): Dados las acciones $\mathcal{A}^1_\mathcal{R}, \cdots, \mathcal{A}^4_\mathcal{R}$, un predicado $\mathcal{P}^k_\mathcal{R}$ / $k = \{1, \cdots, 11\}$ y los conceptos $\mathcal{C}^1_\mathcal{R}$ y $\mathcal{C}^2_\mathcal{R}$ de la ontología *Register* $\mathcal{S}_\mathcal{R}$, se define la relación de respuesta $\overline{\mathcal{R}_\mathcal{R}}$ como $\overline{\mathcal{R}_\mathcal{R}} \subseteq (\mathcal{A}^1_\mathcal{R} \times \mathcal{P}^k_\mathcal{R}) \cup (\mathcal{A}^2_\mathcal{R} \times \mathcal{P}^k_\mathcal{R}) \cup (\mathcal{A}^3_\mathcal{R} \times \mathcal{P}^k_\mathcal{R}) \cup (\mathcal{A}^4_\mathcal{R} \times \mathcal{P}^k_\mathcal{R}) \cup (\mathcal{A}^1_\mathcal{R} \times \mathcal{P}^k_\mathcal{R} \times \mathcal{C}^2_\mathcal{R}) \cup (\mathcal{A}^2_\mathcal{R} \times \mathcal{P}^k_\mathcal{R} \times \mathcal{C}^2_\mathcal{R}) \cup (\mathcal{A}^3_\mathcal{R} \times \mathcal{P}^k_\mathcal{R} \times \mathcal{C}^1_\mathcal{R}) \cup (\mathcal{A}^4_\mathcal{R} \times \mathcal{P}^k_\mathcal{R} \times \mathcal{C}^1_\mathcal{R})$.

Definición 10 (Usos de la ontología *Register*): Dados las acciones $\mathcal{A}^1_\mathcal{R}, \cdots, \mathcal{A}^4_\mathcal{R}$, un predicado $\mathcal{P}^k_\mathcal{R}$ / $k = \{1, \cdots, 11\}$ y los conceptos $\mathcal{C}^1_\mathcal{R}$ y $\mathcal{C}^2_\mathcal{R}$ de la ontología *Register* $\mathcal{S}_\mathcal{R}$, se cumplen las siguientes condiciones para sus usos:

(i) $(\mathcal{A}^1_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^4_{\mathcal{R}} \vee \mathcal{P}^2_{\mathcal{R}} \vee \mathcal{P}^3_{\mathcal{R}}$

(ii) $(\mathcal{A}^2_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^4_{\mathcal{R}} \vee \mathcal{P}^5_{\mathcal{R}} \vee \mathcal{P}^1_{\mathcal{R}} \vee \mathcal{P}^2_{\mathcal{R}} \vee \mathcal{P}^3_{\mathcal{R}}$

(iii) $(\mathcal{A}^3_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^4_{\mathcal{R}} \vee \mathcal{P}^5_{\mathcal{R}}$

(iv) $(\mathcal{A}^4_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^6_{\mathcal{R}} \vee \mathcal{P}^4_{\mathcal{R}} \vee \mathcal{P}^{10}_{\mathcal{R}}$

(v) $(\mathcal{A}^1_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}} \times \mathcal{C}^2_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^7_{\mathcal{R}}$

(vi) $(\mathcal{A}^2_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}} \times \mathcal{C}^2_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^{11}_{\mathcal{R}}$

(vii) $(\mathcal{A}^3_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}} \times \mathcal{C}^1_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^8_{\mathcal{R}}$

(viii) $(\mathcal{A}^4_{\mathcal{R}} \times \mathcal{P}^k_{\mathcal{R}} \times \mathcal{C}^1_{\mathcal{R}}) \Rightarrow \mathcal{P}^k_{\mathcal{R}} = \mathcal{P}^9_{\mathcal{R}}$

2.5.3. Formalización de la ontología *Admin*

En esta *Sección* se presenta la formalización de la ontología *Admin*.

Definición 11 (Ontología *Admin*): Dados el conjunto de conceptos $\mathcal{C}_{\mathcal{A}}$, el conjunto de acciones $\mathcal{A}_{\mathcal{A}}$ y el conjunto de predicados $\mathcal{P}_{\mathcal{A}}$, todos ellos referidos a la ontología *Admin* $\mathcal{S}_{\mathcal{A}}$, ésta se define como:

$$\mathcal{C}_{\mathcal{A}} \times \mathcal{A}_{\mathcal{A}} \times \mathcal{P}_{\mathcal{A}} \Rightarrow \mathcal{S}_{\mathcal{A}},$$

siendo:

(i) $\mathcal{C}_{\mathcal{A}} = \{\mathcal{C}^1_{\mathcal{A}}, \cdots, \mathcal{C}^i_{\mathcal{A}}\} \ / \ i = \{1, \cdots, 3\}$, donde:

  (a) $\mathcal{C}^1_{\mathcal{A}}$ representa el número por defecto de documentos a localizar en una búsqueda (`Def_search_card` en el modelo conceptual).

  (b) $\mathcal{C}^2_{\mathcal{A}}$ representa el número máximo de documentos a localizar en una búsqueda (`Max_search_card`).

  (c) $\mathcal{C}^3_{\mathcal{A}}$ representa la *URI* del repositorio con los documentos de metadatos utilizados por el servicio de mediación (`Offer_repos`).

(ii) $\mathcal{A}_{\mathcal{A}} = \{\mathcal{A}^1_{\mathcal{A}}, \cdots, \mathcal{A}^j_{\mathcal{A}}\} \ / \ j = \{1, \cdots, 6\}$, donde:

  (a) $\mathcal{A}^1_{\mathcal{A}}$ representa la acción de establecer el número por defecto de documentos a localizar en una búsqueda (`SetDef_search_card`).

  (b) $\mathcal{A}^2_{\mathcal{A}}$ representa la acción de establecer el número máximo de documentos a localizar en una búsqueda (`SetMax_search_card`).

  (c) $\mathcal{A}^3_{\mathcal{A}}$ representa la acción de establecer la *URI* del repositorio con los documentos de metadatos utilizados por el servicio de mediación (`SetOffer_repos`).

  (d) $\mathcal{A}^4_{\mathcal{A}}$ representa la acción de devolver el número por defecto de documentos a localizar en una búsqueda (`GetDef_search_card`).

  (c) $\mathcal{A}^5_{\mathcal{A}}$ representa la acción de devolver el número máximo de documentos a localizar en una búsqueda (`GetMax_search_card`).

(d) $\mathcal{A}^6_{\mathcal{A}}$ representa la acción de devolver la *URI* del repositorio con los documentos de metadatos utilizados por el servicio de mediación (`GetOffer_repos`).

(iii) $\mathcal{P}_{\mathcal{A}} = \{\mathcal{P}^1_{\mathcal{A}}, \cdots, \mathcal{P}^k_{\mathcal{A}}\}$ / $k = \{1, \cdots, 7\}$, donde:

(a) $\mathcal{P}^1_{\mathcal{A}}$ representa una excepción controlada en el transcurso de las acciones del conjunto $\mathcal{A}_{\mathcal{A}}$ (`InvalidValue`).

(b) $\mathcal{P}^2_{\mathcal{A}}, \cdots, \mathcal{P}^7_{\mathcal{A}}$ representan que las acciones $\mathcal{A}_{\mathcal{A}}$ se han realizado satisfactoriamente (`ModifiedDef_search_card`, `ModifiedMax_search_card`, `ModifiedOffer_repos`, `ReturnedDef_search_card`, `ReturnedMax_search_card` y `ReturnedOffer_repos`, respectivamente).

Definición 12 (Relación de solicitud en la ontología *Admin*): Dados los conceptos $\mathcal{C}^1_{\mathcal{A}}, \cdots, \mathcal{C}^3_{\mathcal{A}}$, y las acciones $\mathcal{A}^1_{\mathcal{A}}, \cdots, \mathcal{A}^6_{\mathcal{A}}$ de la ontología *Admin* $\mathcal{S}_{\mathcal{A}}$, se define la relación de solicitud $\mathcal{R}_{\mathcal{A}}$ como $\mathcal{R}_{\mathcal{A}} \subseteq (\mathcal{C}^1_{\mathcal{A}} \times \mathcal{A}^1_{\mathcal{A}}) \cup (\mathcal{C}^2_{\mathcal{A}} \times \mathcal{A}^2_{\mathcal{A}}) \cup (\mathcal{C}^3_{\mathcal{A}} \times \mathcal{A}^3_{\mathcal{A}}) \cup (\mathcal{A}^4_{\mathcal{A}} \times \mathcal{C}^1_{\mathcal{A}}) \cup (\mathcal{A}^5_{\mathcal{A}} \times \mathcal{C}^2_{\mathcal{A}}) \cup (\mathcal{A}^6_{\mathcal{A}} \times \mathcal{C}^3_{\mathcal{A}})$.

Definición 13 (Relación de respuesta en la ontología *Admin*): Dados las acciones $\mathcal{A}^1_{\mathcal{A}}, \cdots, \mathcal{A}^6_{\mathcal{A}}$, un predicado $\mathcal{P}^k_{\mathcal{A}}$ / $k = \{1, \cdots, 7\}$ y los conceptos $\mathcal{C}^1_{\mathcal{A}}, \cdots, \mathcal{C}^3_{\mathcal{A}}$ de la ontología *Admin* $\mathcal{S}_{\mathcal{A}}$, se define la relación de respuesta $\overline{\mathcal{R}_{\mathcal{A}}}$ como $\overline{\mathcal{R}_{\mathcal{A}}} \subseteq (\mathcal{A}^1_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^1_{\mathcal{A}}) \cup (\mathcal{A}^2_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^2_{\mathcal{A}}) \cup (\mathcal{A}^3_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^3_{\mathcal{A}}) \cup (\mathcal{A}^4_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^1_{\mathcal{A}}) \cup (\mathcal{A}^5_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^2_{\mathcal{A}}) \cup (\mathcal{A}^6_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^3_{\mathcal{A}}) \cup (\mathcal{A}^1_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}}) \cup (\mathcal{A}^2_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}})$.

Definición 14 (Usos de la ontología *Admin*): Dados las acciones $\mathcal{A}^1_{\mathcal{A}}, \cdots, \mathcal{A}^6_{\mathcal{A}}$, un predicado $\mathcal{P}^k_{\mathcal{A}}$ / $k = \{1, \cdots, 7\}$ y los conceptos $\mathcal{C}^1_{\mathcal{A}}, \cdots, \mathcal{C}^3_{\mathcal{A}}$ de la ontología *Admin* $\mathcal{S}_{\mathcal{A}}$, se cumplen las siguientes condiciones para sus usos:

(i) $(\mathcal{A}^1_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^1_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^2_{\mathcal{A}}$

(ii) $(\mathcal{A}^2_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^2_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^3_{\mathcal{A}}$

(iii) $(\mathcal{A}^3_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^3_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^4_{\mathcal{A}}$

(iv) $(\mathcal{A}^4_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^1_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^5_{\mathcal{A}}$

(v) $(\mathcal{A}^5_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^2_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^6_{\mathcal{A}}$

(vi) $(\mathcal{A}^6_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}} \times \mathcal{C}^3_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^7_{\mathcal{A}}$

(vii) $(\mathcal{A}^1_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^1_{\mathcal{A}}$

(viii) $(\mathcal{A}^2_{\mathcal{A}} \times \mathcal{P}^k_{\mathcal{A}}) \Rightarrow \mathcal{P}^k_{\mathcal{A}} = \mathcal{P}^1_{\mathcal{A}}$

2.6. Trabajos relacionados

La mayoría de los mediadores existentes [Bearman, 1997], [OOC, 2001], [Kutvonen, 1996], [Beitz and Bearman, 1994], [Müller et al., 1995], [PrismTech, 2001] y [Schmidt, 2001] presentan características similares, ya que todos se basan en el modelo de mediación de *ODP*

[ITU-T and ISO/IEC, 1998] (adoptado también por *OMG*) analizado en la *Sección 1.5*. Uno de los trabajos en este área que parece diferenciarse de los demás es *WebTrader* [Vasudevan and Bannon, 1999], un trabajo que presenta un modelo de mediación para el desarrollo de software basado en Internet y que utiliza *XML* para describir servicios de componente. También se puede destacar el proyecto *Pilarcos* —un proyecto de investigación de la *National Technology Agency TEKES* de Finlandia, del Departamento *Computer Science* de la Universidad de Helsinki y Nokia, SysOpen y Tellabs— que reivindica, entre otras características, la necesidad de un servicio de mediación basado en *ODP/OMG* para el desarrollo de soluciones intermedias (*middleware*) con una gestión automática de aplicaciones federadas. El principal inconveniente de estas propuestas es la utilización sólo de especificaciones de componentes.

Otro trabajo relacionado es el servicio de directorios de servicios web de *UDDI* (*Universal Description, Discovery and Integration*), que utiliza *WSDL* (*Web-Services Definition Language*) para la descripción técnica de los servicios web ofrecidos. Precisamente, una de las limitaciones de *UDDI* viene de estas descripciones técnicas realizadas con *WDSL*, que sólo permiten capturar cierta información funcional acerca de los servicios descritos. En este sentido, *UDDI* proporciona un potente y completo servicio de directorio, pero no puede ser considerado realmente como un mediador. Además, como servicio de directorio que es, *UDDI* no implementa dos de las características comunes en el estándar de mediación: la federación y la propagación de consultas.

También, existen enfoques de la implementación del servicio de mediación de *ODP* que presentan deficiencias en la interacción y comunicación entre objetos o componentes, o la descripción del lenguaje de comunicación. Por ejemplo, en [Craske et al., 1999] los autores presentan un servicio de mediación denominado *DOKTrader*, que actúa sobre un sistema de base de datos federada —*Núcleo de Objetos Distribuidos* (*Distributed Object Kernel*, *DOK*)—. Otro ejemplo se encuentra en [Merz et al., 1994], un estudio que se centra en la creación de un marco para el desarrollo de aplicaciones distribuidas en un *Mercado de Servicios Abiertos Comunes* (*Common Open Service Market*, *COSM*), haciendo uso de un *Lenguaje de Descripción de Interfaces de Servicio* (*Service Interface Description Language*, *SIDL*) para describir los servicios gestionados por el mediador. En [Lamparter and Schnizler, 2006] se presenta el diseño de un mercado de servicios web dirigido por ontologías; en este sistema, se utiliza un lenguaje de comunicación ontológico para representar consultas, ofertas y acuerdos. También, en [Tsai et al., 2007] se utiliza una ontología para describir y facilitar la *Colaboración de Proceso Dinámico* (*Dynamic Process Collaboration*, *DPC*).

El modelo *OntoTrader* propuesto en este *Capítulo* trata de aportar un servicio de mediación que favorezca tanto la interoperabilidad de los objetos o componentes web en los sistemas abiertos y distribuidos, como la mejora del proceso de búsqueda y recuperación de información en ellos. Este modelo permite gestionar información de cualquier dominio de aplicación, no sólo las especificaciones de los servicios ofrecidos por los componentes, además de fijar un estándar común de comunicación con el servicio de mediación a través de sus diferentes interfaces. Para ello hace uso de una serie de ontologías, de forma más concreta, una ontología de datos y una ontología de servicio/proceso para cada una de dichas interfaces, respectivamente.

2.7. Resumen y conclusiones

El uso de los *Sistemas de Información* basados en la Web se ha incrementado en los últimos tiempos dadas las ventajas que aportan frente a otro tipo de sistemas. Esto ha hecho que el trabajo de muchos investigadores se centre en la búsqueda de soluciones que aporten mejoras significativas a los procesos que se llevan a cabo en ellos. Con ese objetivo, en este *Capítulo*, se han presentado las características de un modelo de mediación ontológico, bautizado con el nombre de *OntoTrader*, como una evolución del modelo tradicional de *ODP* —extendido a la gestión no sólo de servicios, sino de cualquier tipo de información—, que utiliza el mecanismo *QS/RR* en los procesos de búsqueda y recuperación de información. También se han analizado los diferentes modelos operativos de mediación —reflexión, delegación y federación— y se ha planteado una arquitectura en la que destacan dos novedades: la primera es la utilización de un repositorio de documentos, que podrían ser instancias de cualquier ontología que represente los metadatos de la información de un dominio específico; y la segunda novedad es un planteamiento basado en la definición, para cada una de las interfaces del servicio de mediación, de una ontología de servicio/proceso asociada que mejore el modelo de comunicación e interacción con él. Finalmente, se ha utilizado notación matemática para realizar una formalización del modelo.

Capítulo 3

Sistema de representación del conocimiento basado en mediación ontológica: TKRS

Capítulo 3

Sistema de representación del conocimiento basado en mediación ontológica: TKRS

Contenidos

Partiendo del modelo de mediación *OntoTrader*, presentado en el *Capítulo 2* como una solución que no sólo favorece la interoperabilidad de los objetos o componentes, sino que también introduce mejoras en el proceso de búsqueda y recuperación de la información, la siguiente meta consiste en ver cómo se puede integrar este modelo en los *Sistemas de Información basados en la Web* (*Web-based Information Systems*, *WIS*). Para ello, en este *Capítulo* se propone el diseño de un *Sistema de Representación del Conocimiento basado en Mediación*, que se va a denominar *TKRS* (*Trading-based Knowledge Representation System*), donde la *Ingeniería Dirigida por Modelos* (*Model-Driven Engineering*, *MDE*) va a proporcionar los mecanismos necesarios para definir un conjunto de herramientas y lenguajes que permitan la descripción de una arquitectura independiente de la plataforma. Además, se va a dar otro paso y se va a diseñar un repositorio que dé cabida a las implementaciones de *TKRS* en diferentes plataformas, junto con un modelo de configuración capaz de relacionar la arquitectura del sistema con una implementación concreta y, a partir de él, proceder, incluso, al despliegue final del mismo.

El *Capítulo* se organiza en siete secciones. Comienza justificando y presentando el marco general de trabajo para el diseño y desarrollo de sistemas *TKRS* en la *Sección 3.1*. Seguidamente, en la *Sección 3.2*, se expone la arquitectura del sistema, analizando sus componentes y cómo se ha integrado el modelo *OntoTrader*, describiendo los diferentes niveles de representación del conocimiento que se utilizan y presentando una herramienta gráfica —que se podría catalogar como una utilidad *CASE* (*Computer Aided Software Engineering*)— para el diseño de estos sistemas. La *Sección 3.3* muestra los detalles de diseño del repositorio de implementaciones mencionado y hace alusión a una implementación concreta basada en un *Sistema Multi-Agente* (*Multi-Agent System*, *MAS*). El modelo de configuración de *TKRS* se describe en la *Sección 3.4*. En la *Sección 3.5* se presenta una formalización matemática del marco desarrollado, incluyendo arquitectura, repositorio y configuración. Para finalizar, en la *Sección 3.6*, se discuten algunos trabajos relacionados con la propuesta presentada y, en la *Sección 3.7*, se realiza un breve resumen junto con las conclusiones.

3.1. Introducción y conceptos relacionados

Como ya se ha mencionado, el diseño de los *Sistemas de Información basados en la Web* y, en particular, de los *Sistemas de Gestión de Información* requiere principalmente el empleo de métodos y técnicas estandarizados que permitan la utilización, por un lado, de un vocabulario común para la representación del conocimiento y, por otro, de mecanismos de interacción en tiempo real tanto entre los usuarios y el sistema, como entre el sistema y otros sistemas de terceras partes o, incluso, entre los componentes del propio sistema. Esto ha propiciado que se utilicen nuevos paradigmas abiertos y distribuidos [Xiao-feng et al., 2006] y que conceptos como *web semántica*, *minería de datos* o *componentes de consulta* de información aparezcan relacionados con este tipo de sistemas [Taniar and Rahayu, 2004]. Algunas de las características intrínsecas a ellos son su frecuente utilización por una amplia variedad de usuarios (administradores, técnicos,

políticos, etc.) que, en la mayoría de las ocasiones, cooperan entre sí para una mejor toma de decisiones y el manejo de muy dispares fuentes de conocimiento.

En este contexto, con el objetivo de facilitar el diseño y desarrollo de *Sistemas de Representación del Conocimiento basados en Mediación* (*TKRS*), se proponen el uso de ontologías para la representación de todo el conocimiento presente en el sistema y su construcción alrededor del modelo de mediación *OntoTrader* para mejorar la gestión y, sobre todo, la consulta de información. Con ambas ideas presentes y bajo el amparo de la *Ingeniería Dirigida por Modelos*, que va a permitir aislar el diseño de la implementación del sistema, a lo largo del *Capítulo* se analizan los detalles de esta novedosa aproximación. Para ello, se van a definir un conjunto de herramientas de soporte y lenguajes de modelado que permiten la definición de la arquitectura del sistema, de un repositorio de implementaciones y de un modelo de configuración del sistema. En la *Figura 3.1* se puede observar cómo encaja la perspectiva *MDE* en la clásica *Arquitectura Dirigida por Modelos* (*Model-Driven Architecture*, *MDA*) de *OMG*, así como el resumen gráfico de todo el marco de trabajo (*framework*).

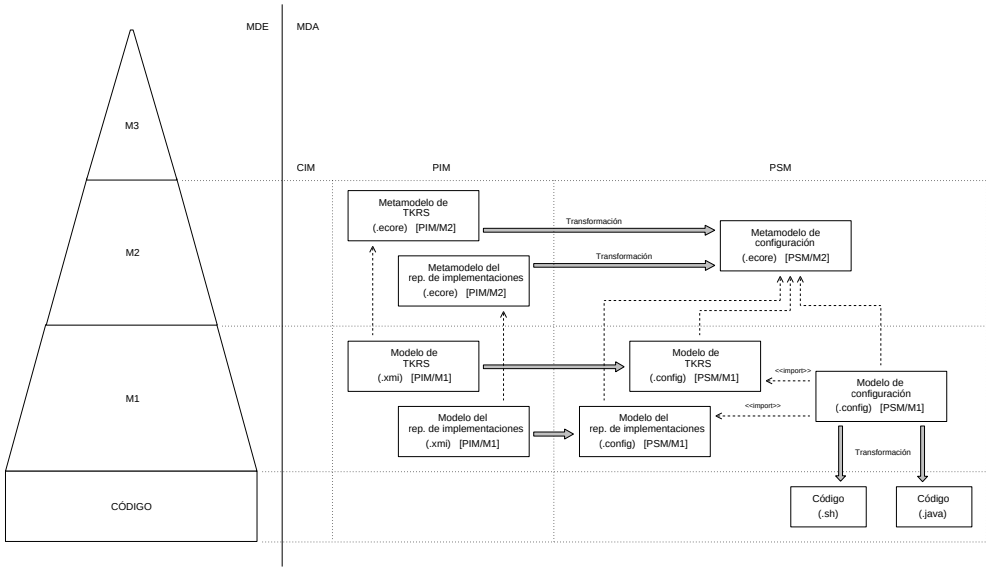


Figura 3.1: *Framework* para el diseño y desarrollo de sistemas *TKRS*.

En primer lugar, se va a realizar un modelado independiente de la plataforma (*Platform Independent Model*, *PIM*), tanto de la arquitectura del sistema (PIM/M2) como del repositorio de implementaciones (PIM/M2) mediante la definición de dos metamodelos. Posteriormente, el modelado de la configuración del sistema (PSM/M2), que sí es específico de la plataforma (*Platform Specific Model*, *PSM*), va a relacionar la arquitectura con una implementación concreta. Para ello, es necesario aplicar una transformación *Modelo-A-Modelo* (*Model-To-Model*, *M2M*) previa, tanto al modelo de la arquitectura

(PIM/M1 a PSM/M1) como al modelo del repositorio (PIM/M1 a PSM/M1). Finalmente, a partir de un modelo de configuración (PSM/M1), otra transformación, ahora *Modelo-A-Texto* (*Model-To-Text*, *M2T*), va a generar el código de la implementación para la plataforma específica. Esta aproximación utiliza una amplia variedad de tecnologías basadas en *Eclipse* como *Eclipse Modeling Framework* (*EMF*), *Graphical Modeling Framework* (*GMF*), *XText* o *Xpand*, que se verán a lo largo del *Capítulo*: los metamodelos se definen mediante un editor gráfico almacenándose en archivos "`.ecore`", los modelos creados a partir de ellos se representan en "`.xmi`", los modelos de configuración se expresan mediante un *Lenguaje Específico del Dominio* (*Domain Specific Language*, *DSL*) que genera archivos "`.config`", etc.

3.2. Arquitectura de *TKRS* (PIM/M2)

La arquitectura de *TKRS*, como ya se ha apuntado, es un tipo de arquitectura distribuida que se organiza por *Nodos* (`Node` en el metamodelo de la Figura 3.2), cada uno de los cuales consta de un conjunto de *Módulos* (`Modules`), que se describen a continuación. Cada uno de los nodos cuenta, al menos, con un *Módulo de servicios* (`ServiceModule`) que, como su propio nombre indica, proporciona una serie de servicios al resto de módulos del nodo, un *Módulo de gestión* (`ManagementModule`) que hace de nexo entre la interfaz de usuario y el resto de módulos del sistema, gestionando las demandas por parte de los usuarios, y un *Módulo de consulta* (`QueryModule`) relacionado con la resolución de las consultas de información de los usuarios. Adicionalmente, un nodo también puede disponer de uno o más *Módulos de mediación* (`TradingModule`), que permiten la búsqueda y localización de información en el sistema —aquí es donde entra en juego el modelo *OntoTrader*—, aunque no es un requisito indispensable, así como de uno o más *Módulos de procesamiento* (`ProcessingModule`), responsables de la gestión de las bases de conocimiento —éstos pueden agruparse en unidades lógicas mayores denominadas *Ambientes* (`Ambients`) atendiendo a cualquier tipo de criterio como, por ejemplo, localización—. Sí es obligatorio que el sistema, en su conjunto, posea al menos un módulo de mediación y un módulo de procesamiento en alguno de sus nodos.

El metamodelo propuesto en la *Figura 3.2* (PIM/M2 en la *Figura 3.1*) se ha definido utilizando *EMF* [EMF, 2013] y, a partir de él, se pueden crear modelos de *TKRS* que reflejen los conceptos, relaciones y restricciones descritos anteriormente. Por ejemplo, la metaclase `Node`, establece un identificador (`name`), una dirección *IP* (`ip`) y dos puertos de comunicación (`port` y `dbport`). También, se puede destacar la definición de una metaclase abstracta `Module`, con un atributo `name`, de la que heredan los diferentes tipos de módulos. El módulo de mediación, como da forma al servicio de mediación ontológico, establece cinco atributos booleanos que indican si éste implementa o no sus diferentes interfaces: *Lookup* (`lookupInterface`), *Register* (`registerInterface`), *Admin* (`adminInterface`), *Link* (`linkInterface`) y *Proxy* (`proxyInterface`). Las interfaces *Lookup* y *Register* son necesarias, ya que cada módulo de mediación está vinculado a un repositorio de metadatos. Este metamodelo define, además, la federación de módulos de mediación —a través de la relación `isFederatedWith`— y una conexión entre cada módulo de procesamiento y cada módulo de consulta —por medio de las relaciones

`usesRegisterInterface` y `usesLookupInterface`, respectivamente— con un módulo de mediación, relaciones que se analizarán en la *Sección 3.2.2*.

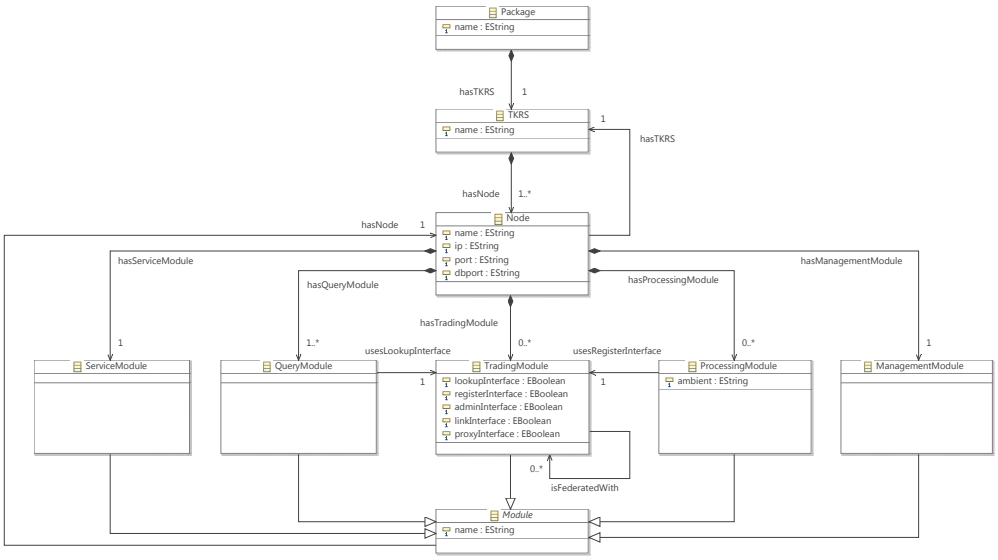


Figura 3.2: Metamodelo de la arquitectura de *TKRS*.

Con el objeto de facilitar la creación de modelos *TKRS* (PIM/M1 en la *Figura 3.1*) de acuerdo al metamodelo descrito, se ha implementado un editor gráfico de modelos utilizando *GMF* que, junto con el uso de reglas *OCL* (*Object Constraint Language*), permite representar todas las restricciones. La herramienta aparece descrita con más detalle en la *Sección 3.2.3*.

Los detalles de la funcionalidad de cada uno de los módulos de la arquitectura del sistema aparecen definidos en la *Figura 3.3* mediante un diagrama de clases *UML* (la especificación completa de la arquitectura se puede encontrar desarrollada en el *Anexo A* de este documento). En el diagrama, por ejemplo, aparece la clase `ManagementModule`, que hereda de la clase abstracta `Module` su atributo `name`, así como sus métodos `getName`, `setName`, `start` y `stop` —del mismo modo que el resto módulos del sistema—, y define sus métodos propios, como `add_metadata`, `query_information` o `set_trading_parameters`. La *Tabla 3.1* describe brevemente las operaciones de esta clase.

3.2.1. Niveles de representación del conocimiento

TKRS, como *Sistema de Gestión de Información* que es, puede llegar a manejar un gran volumen de datos cuyo origen va a estar en una o más fuentes externas a él relacionadas directamente con uno o más módulos de procesamiento. Estos datos en bruto van a constituir el primer nivel en la arquitectura de datos del sistema. Ahora bien, uno

Operación	Descripción
`add_metadata`	Añade un nuevo registro de metadatos en un módulo de procesamiento.
`modify_metadata`	Modifica un registro de metadatos específico en un mód. de procesamiento.
`remove_metadata`	Elimina un registro de metadatos específico en un mód. de procesamiento.
`describe_metadata`	Devuelve la info. de un registro de metadatos de un mód. de procesamiento.
`list_metadata`	Devuelve una lista con todos los registros de metadatos de un mód. de proc.
`query_information`	Devuelve el resultado de una consulta de información realizada al sistema.
`get_trading_parameters`	Devuelve los parámetros de configuración de un módulo de mediación.
`set_trading_parameters`	Establece los parámetros de configuración de un módulo de mediación.

Tabla 3.1: Descripción de las operaciones de la clase `ManagementModule`.

de los principales propósitos de *TKRS* es mejorar el proceso de búsqueda y recuperación de la información, para lo cual no es adecuado trabajar directamente con los datos —obviamente, dado el volumen que pueden llegar a alcanzar—, por lo que aparece un segundo nivel de metadatos, que se genera y almacena en los módulos de procesamiento. Estos metadatos ya tienen un volumen mucho menor y, aunque en ocasiones sea necesario acudir al nivel inferior de datos para resolver una consulta, la mayoría de ellas se resolverán en este otro nivel; sin embargo, si se diese el primer caso, los metadatos actuarían como un filtro, reduciendo el volumen de información y, como consecuencia, el tiempo de ejecución de la consulta.

Si el sistema no presentase la característica de ser distribuido o estuviese formado por un único módulo de procesamiento, con esta arquitectura de dos niveles sería más que suficiente —en esta situación, tampoco serían necesarios los módulos de mediación—, pero como no es lo que se pretende, se incluye un nuevo nivel en la arquitectura, denominado nivel de meta-metadatos, que recoge cierta información para relacionar los diferentes módulos de procesamiento y es utilizado por los módulos de mediación. En la definición de estos metadatos y meta-metadatos, las ontologías van a jugar un papel fundamental, como se verá en el *Capítulo 4*, además del que ya poseían en la definición de los protocolos de interacción y comunicación entre los componentes del sistema. La *Figura 3.4* refleja claramente los tres niveles de la arquitectura de datos.

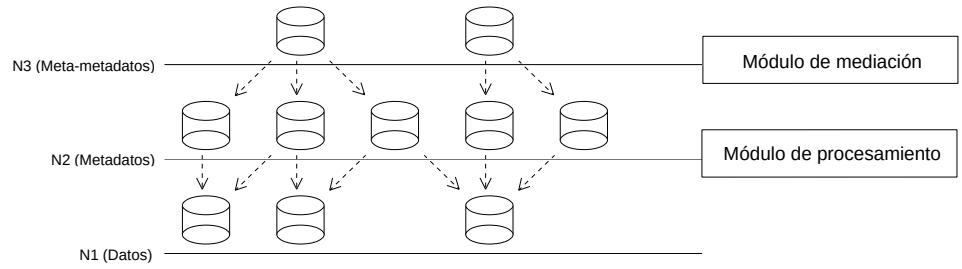


Figura 3.4: Arquitectura de datos de *TKRS* a tres niveles.

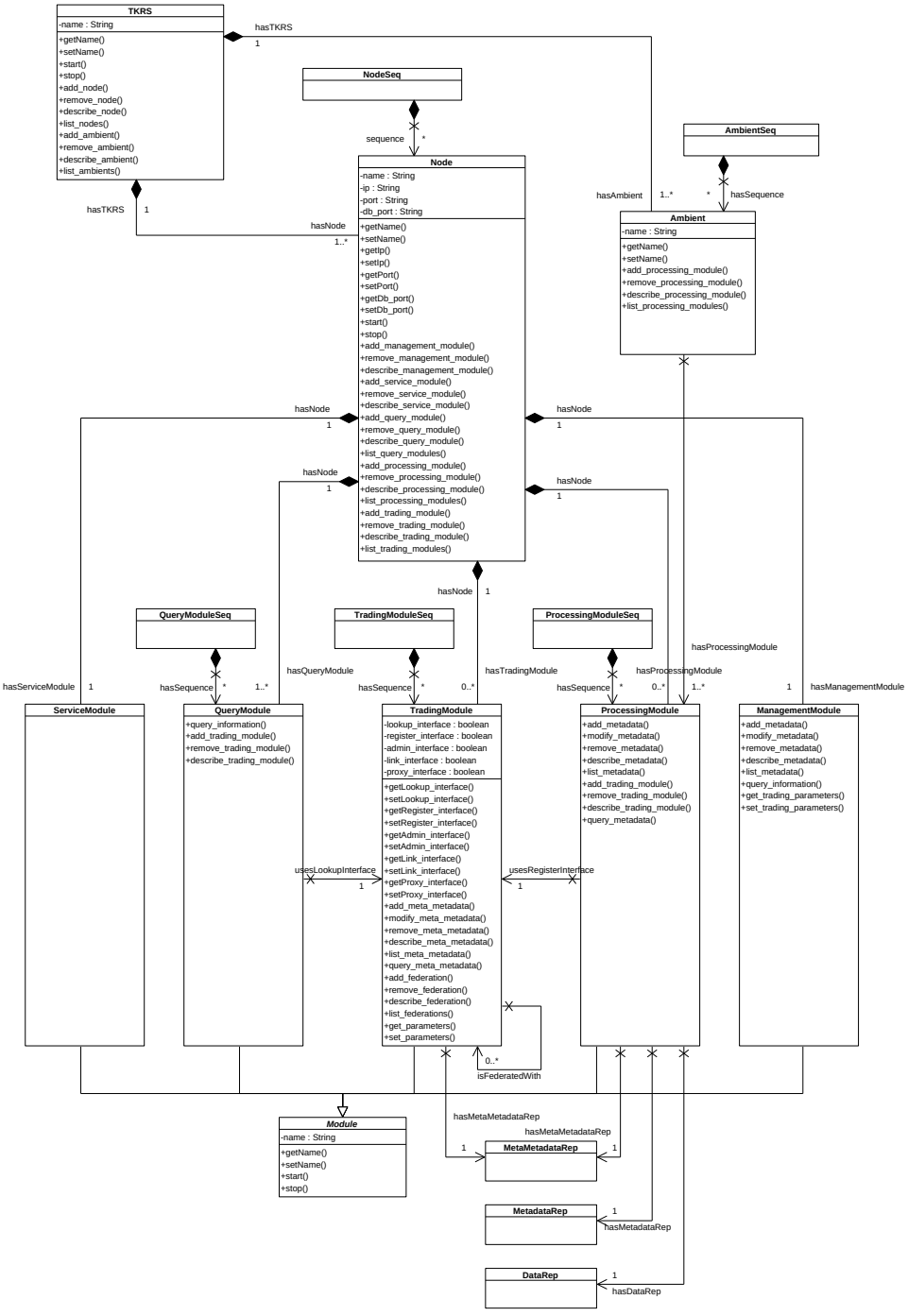


Figura 3.3: Modelo conceptual de la arquitectura de *TKRS*.

3.2.2. Integración del modelo *OntoTrader* en *TKRS*

Una de las características que se ha tenido presente para el diseño de *TKRS* es que su arquitectura gire en torno al modelo *OntoTrader*. El servicio de mediación ontológico desarrollado en el *Capítulo 2* aparece encapsulado en el módulo de mediación y son las ontologías de servicio/proceso (*Lookup*, *Register* y *Admin*) las que utilizan el resto de módulos para interactuar con él e, incluso, comunicarse entre ellos. La *Tabla 3.2* muestra las ontologías que utiliza cada uno de los módulos del sistema *TKRS*. A lo largo de esta *Sección*, se analiza la influencia del modelo *OntoTrader* en los procesos de gestión y consulta de información.

Origen	**Ontología**	**Destino**
Módulo de gestión	Ontología Lookup	Módulo de consulta
Módulo de consulta	Ontología Lookup	Módulo de mediación
Módulo de consulta	Ontología Lookup	Módulo de procesamiento
Módulo de consulta	Ontología Lookup	Módulo de gestión
Módulo de mediación	Ontología Lookup	Módulo de consulta
Módulo de procesamiento	Ontología Lookup	Módulo de consulta
Módulo de gestión	Ontología Register	Módulo de procesamiento
Módulo de procesamiento	Ontología Register	Módulo de mediación
Módulo de gestión	Ontología Admin	Módulo de mediación

Tabla 3.2: Uso de las ontologías de servicio/proceso en *TKRS*.

En el proceso de gestión de información están implícitas cuatro acciones: inserción, modificación, eliminación y descripción de datos. Todas ellas tienen como origen una petición realizada por un usuario o un grupo de usuarios a través de la interfaz, petición que recoge el módulo de gestión y traslada al módulo de procesamiento especificado por el usuario: (a) si se trata de la **inserción** de datos, este módulo almacena los metadatos en un repositorio local, genera a partir de ellos un nuevo registro de meta-metadatos que, por un lado, almacena en un repositorio local de meta-metadatos y, por otro, traslada a un módulo de mediación con el que se encuentra asociado, dando por concluida la acción —de esta manera, el módulo de mediación dispone de un repositorio global con todos los meta-metadatos de todos los módulos de procesamiento asociados a él (fundamentales en el proceso de búsqueda y recuperación de información)—; (b) si se trata de la **modificación** o **eliminación** de datos, del mismo modo que en la inserción, primero se ven afectados los repositorios locales del módulo de procesamiento y, posteriormente, se traslada la acción al módulo de mediación para que actúe en consecuencia; (c) la **descripción** de datos se resuelve directamente en el módulo de procesamiento, que devuelve la información demandada por el usuario o grupo de usuarios a partir de la que se encuentra almacenada en sus repositorios.

El proceso de búsqueda y recuperación de información del servicio de mediación *OntoTrader* descrito en la *Sección 2.3* seguía el mecanismo *QS/RR*, basado en un modelo cliente/servidor de tres niveles $< \mathcal{O}, \mathcal{M}, \mathcal{I} >$. Este modelo queda configurado de la siguiente forma en *TKRS*: en el *Nivel 1* (*N1*) aparecen las consultas generadas y tratadas

por un componente u objeto de la interfaz de usuario y recepcionadas en el sistema por los módulos de gestión $\mathcal{O}$; el *Nivel 3* (*N3*) recae en los módulos de procesamiento y sus repositorios de metadatos $\mathcal{I}$; y en el *Nivel 2* (*N2*), que era el *middleware* $\mathcal{M}$ que permitía la localización de las fuentes de información, es ahora donde operan los módulos de consulta y los módulos de mediación con sus repositorios de meta-metadatos. Si se hace un recordatorio, como premisa indispensable para su funcionamiento, cada objeto debía estar asociado con un servicio de mediación, lo que se traduce en que cada módulo de consulta y cada módulo de procesamiento deben estar asociados a un módulo de mediación —de ahí que en el metamodelo definido en la *Sección 3.1* exista una conexión entre cada módulo de procesamiento y cada módulo de consulta (por medio de las relaciones `usesRegisterInterface` y `usesLookupInterface`, respectivamente) con un módulo de mediación—; los módulo de gestión son meros organizadores y transmisores de las consultas a los módulos de consulta. De esta forma, los escenarios de reflexión, delegación y federación —resumidos de forma gráfica en la *Figura 3.5*— quedarían así:

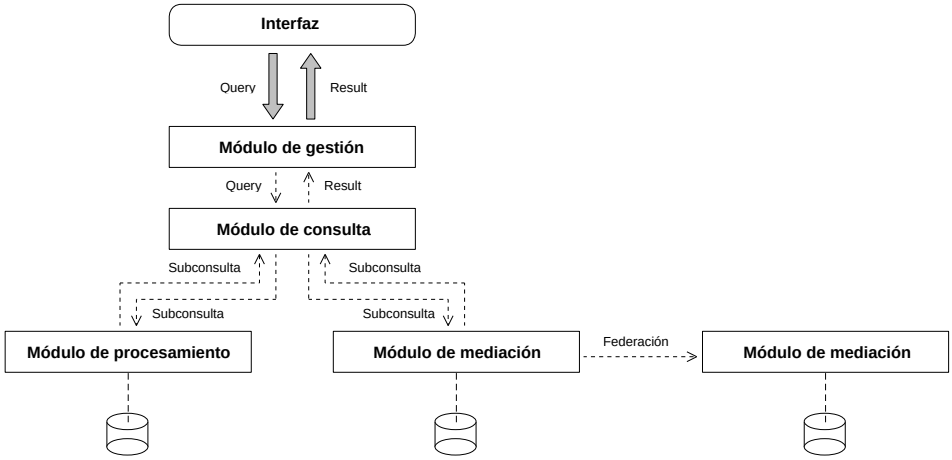


Figura 3.5: Esquema de operación de *TKRS* en un proceso de consulta.

— En el modelo de **reflexión**, la consulta generada en la interfaz $\mathcal{O}$ se puede satisfacer con los documentos de metadatos que almacena el repositorio asociado a un módulo de mediación $\mathcal{M}$; es preciso señalar que la consulta se trasladaría desde la interfaz a un módulo de gestión, de éste a un módulo de consulta, que es en realidad el que la planifica, y ya de él al módulo de mediación. En este caso, interviene el par $< \mathcal{O}, \mathcal{M} >$ del modelo.

— En el modelo de **delegación**, se negocia indirectamente con el servicio de mediación. La consulta generada en la interfaz $\mathcal{O}$ viaja hasta un módulo de consulta que la fragmenta en dos o más subconsultas; una la traslada al módulo de mediación $\mathcal{M}$ asociado para que localice el origen de la información en su repositorio y, seguidamente, el módulo de consulta reformula la subconsulta o subconsultas y

las traslada a los repositorios de los módulos de procesamiento $\mathcal{I}$ señalados por el anterior. En este caso, la serie de elementos que interviene es $< \mathcal{O}, \mathcal{M}, \mathcal{I} >$.

— En el modelo de **federación**, dos o más módulos de mediación están configurados para utilizar un mecanismo de delegación. Como en el primer caso, la consulta procedente de la interfaz $\mathcal{O}$ llegaría hasta el primer módulo de mediación con el que se encuentra asociado el módulo de consulta $\mathcal{M}$, el cual la propagaría a otro módulo de mediación $\mathcal{M}$ con el que se encuentre federado, que localizaría las fuentes de información externas $\mathcal{I}$. En este caso, intervienen los elementos de la serie $< \mathcal{O}, \mathcal{M}, \mathcal{M}, \mathcal{I} >$.

3.2.3. Herramienta gráfica para el diseño de arquitecturas *TKRS*

Con el objetivo de facilitar el diseño de modelos de *TKRS* que se ajusten al metamodelo descrito en la *Sección 3.1*, se ha implementado un editor gráfico utilizando *Eclipse GMF* [GMF, 2013]. El proceso que normalmente se sigue para implementar un editor *GMF* consta de tres etapas: en una primera etapa, se define una representación gráfica para cada elemento (conceptos y relaciones) del metamodelo; en una segunda etapa, se define una barra con herramientas que van a permitir la creación de instancias de cada uno de los elementos anteriores; y, en una última etapa, se define un mapeo entre cada elemento del metamodelo, su representación gráfica y su herramienta de creación. La *Figura 3.6* muestra una captura de pantalla del editor con un ejemplo del diseño de un sistema. Los conceptos y relaciones del metamodelo se pueden crear con la barra de herramientas situada en el margen derecho de la ventana de edición. Esta barra se encuentra divida en tres secciones: la primera sección únicamente contiene una herramienta destinada a la creación de nodos; la segunda sección agrupa las herramientas que permiten la adición de cada tipo de módulo a los distintos nodos; la tercera sección agrupa las herramientas para la creación de las relaciones definidas entre algunos de sus módulos (`usesRegisterInterface`, `usesLookupInterface` e `isFederatedWith`). Las relaciones de federación se representan mediante flechas con trazo discontinuo, mientras que las relaciones entre módulos de procesamiento o módulos de consulta y los módulos de mediación a los que se asocian se representan mediante flechas con trazo continuo. Para no hacer ilegible la vista gráfica del modelo, sólo se representan algunos atributos mediante etiquetas (por ejemplo, `ip`, `port`, `ambient`, `lookupInterface`, etc.), independientemente de que cualquier atributo puede ser visualizado y modificado usando el panel *Propiedades* de *Eclipse*.

GMF también permite extender las reglas sintácticas especificadas en el metamodelo (ver *Figura 3.2*) con restricciones definidas mediante el estándar *OCL* [OMG, 2006], como las que recoge la *Tabla 3.3*:

— Las tres primeras reglas (`rule #1`, `rule #2` y `rule #3`) se refieren a que dos nodos distintos que compartan la misma dirección *IP* no pueden poseer el mismo identificador y los mismos puertos de comunicaciones.

— La cuarta regla (`rule #4`) asegura que un módulo de mediación no pueda federarse consigo mismo.

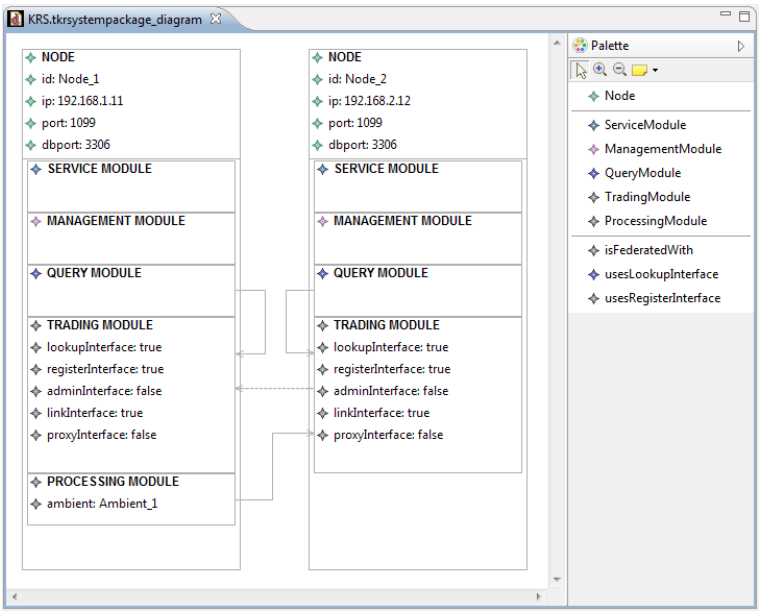


Figura 3.6: Captura de pantalla del editor gráfico de modelos de *TKRS*.

— Las reglas quinta y sexta (`rule #5` y `rule #6`) especifican que cuando dos módulos de mediación se encuentran federados, deben tener activas sus interfaces *Link*.

— Las dos últimas reglas (`rule #7` y `rule #8`) indican, respectivamente, que *TKRS* debe contar, al menos, con un módulo de mediación y un módulo de procesamiento en un algunos de sus nodos.

El proceso para diseñar un modelo de *TKRS* utilizando este editor gráfico consta de los siguientes pasos:

(1) Crear tantos nodos como sea necesario —para cada nodo, automáticamente se añaden los módulos obligatorios (el módulo de servicios, el módulo de gestión, y el módulo de consulta).

(2) Especificar el valor de los atributos `id`, `ip`, `port` y `dbport` de cada uno de los nodos.

(3) Añadir los módulos de procesamiento y los módulos de mediación en los nodos que corresponda, teniendo en cuenta que, al menos, debe existir un módulo de cada tipo en el sistema.

(4) Para cada módulo de mediación, seleccionar las interfaces que implementa estableciendo el valor de los atributos correspondientes a `true` —las interfaces *Lookup* y *Register* se activan automáticamente.

```
rule #1: context TKRS inv:
           self.hasNode -> forAll (n1, n2: Node | ((n1 <> n2) and (n1.ip = n2.ip)) implies
            (n1.id <> n2.id))
```

```
rule #2: context TKRS inv:
           self.hasNode -> forAll (n1, n2: Node | ((n1 <> n2) and (n1.ip = n2.ip)) implies
            (n1.port <> n2.port))
```

```
rule #3: context TKRS inv:
           self.hasNode -> forAll (n1, n2: Node | ((n1 <> n2) and (n1.ip = n2.ip)) implies
            (n1.dbport <> n2.dbport))
```

```
rule #4: context TradingModule inv:
           self.isFederatedWith -> forAll(tm: TradingModule | (tm <> self))
```

```
rule #5: context TradingModule inv:
           self.isFederatedWith -> notEmpty() implies (self.linkInterface = true)
```

```
rule #6: context TradingModule inv:
           self.isFederatedWith -> forAll(tm: TradingModule | (tm.linkInterface = true))
```

```
rule #7: context TKRS inv:
           self.hasNode -> exists(n: Node | n.hasTradingModule -> size() > 0)
```

```
rule #8: context TKRS inv:
           self.hasNode -> exists(n: Node | n.hasProcessingModule -> size() > 0)
```

Tabla 3.3: Restricciones *OCL* del editor gráfico de modelos de *TKRS*.

(5) Establecer la conexión entre los módulos de mediación federados (en el caso de existir federación).

(6) Establecer la conexión de cada módulo de procesamiento y cada módulo de consulta con un módulo de mediación.

A modo de ejemplo, el modelo representado en la *Figura 3.6* se corresponde con una arquitectura formada por tres nodos, cada uno de los cuales cuenta con los módulos obligatorios (módulo de servicios, módulo de gestión y módulo de consulta). Además, los nodos identificados como "Node 1" y "Node 2" contienen un módulo de procesamiento y un módulo de mediación, respectivamente, cumpliendo las exigencias mínimas de *TKRS*. También, existe federación entre los módulos de mediación de los nodos anteriores (representada con una flecha de trazo discontinuo), lo que significa que el módulo de mediación del "Node 1" puede delegar sus consultas de información en el módulo de mediación del "Node 2". Por último, se pueden observar las asociaciones obligatorias de cada módulo de procesamiento y módulo de consulta con un módulo de mediación (representadas con flechas de trazo continuo), perteneciente o no al mismo nodo.

3.3. Repositorio de implementaciones (PIM/M2)

Una vez modelada la arquitectura de *TKRS* (Metamodelo PIM/M2 en la *Figura 3.1*), se pensó en continuar avanzando en esta línea con el modelado de la implementación del sistema y, dado que puede existir más de una implementación en función de la plataforma de desarrollo utilizada, se optó por agrupar todas en ellas en un repositorio. La estructura de este repositorio de implementaciones es una estructura muy básica y, como se puede observar en la *Figura 3.7* —que representa el metamodelo del repositorio (PIM/M2)—, está formada por plataformas (metaclase `Platform`) e implementaciones de cada uno de los módulos del sistema en dichas plataformas. La metaclase `Platform` incluye un atributo requerido (`name`) para especificar el nombre de la plataforma de desarrollo utilizada. Además, como es habitual en *Ingeniería del Software*, un módulo (metaclase `CompositeModule`) puede estar formado, a su vez, por uno o más módulos (`CompositeModule` o `SimpleModule`). También, se ha incluido la metaclase abstracta `Module`, de la que heredan tanto módulos simples como compuestos, y se han definido dos atributos obligatorios, `name` y `uri`, para especificar el nombre del módulo y una referencia a la localización de los archivos de implementación, respectivamente. Por último, cada módulo posee una referencia a su módulo contenedor, si lo hubiese, y a su plataforma. La definición de este metamodelo se ha realizado nuevamente utilizando *EMF* y se puede emplear el *Eclipse Reflective Ecore Model Editor* para crear un modelo de repositorio e implementación concretos, implementación como la que se analiza a continuación basada en la tecnología de *Sistemas Multi-Agente*.

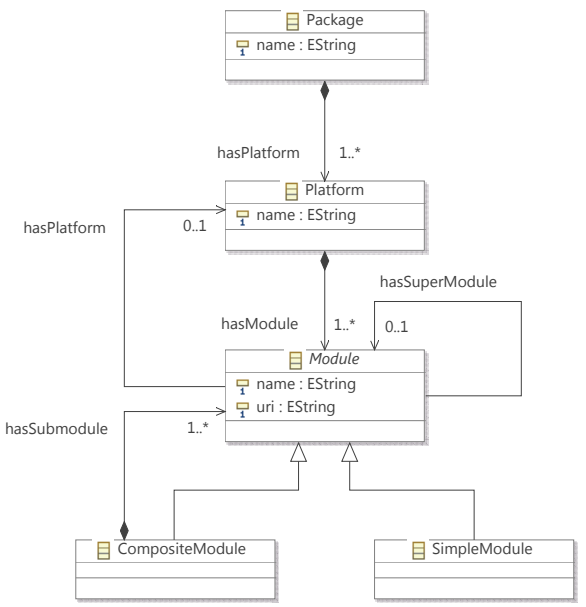


Figura 3.7: Metamodelo de un repositorio de implementaciones de *TKRS*.

La selección de esta tecnología para la implementación de *TKRS* viene dada por las características que presentan los agentes software y que ya han sido descritas en la *Sección 1.6*. Básicamente, como se puede observar en el modelo conceptual de la *Figura 3.8*, cada uno de los módulos del sistema puede ser implementado por un agente o un conjunto de agentes software. El módulo de gestión se puede implementar con un *Agente de gestión*, que haga de intermediario entre la interfaz de usuario y el resto de agentes del sistema, y sea el responsable de gestionar todas las demandas de los usuarios; el módulo de consulta, con un *Agente de consulta*, directamente encargado de resolver las consultas de información; el módulo de mediación, con un *Agente de mediación* que facilite la búsqueda y localización de esta información, filtrándola a partir de los parámetros de las consultas; y el módulo de procesamiento, con un *Agente de procesamiento*, responsable de la gestión de las fuentes de conocimiento. Sin embargo, para el módulo de servicios, por razones de especialización, puede crearse un agente para cada tipo de servicio; por ejemplo, puede existir un *Agente de ontología* cuyo cometido sea la gestión de ontologías, de tal forma que, cuando cualquier agente del sistema necesite enviar un mensaje a otro agente haciendo uso de una de las ontologías de servicio/proceso o necesite gestionar un registro de metadatos o meta-metadatos, para evitar que tenga que especializarse en estas tareas, recurre a un *Agente de ontología*.

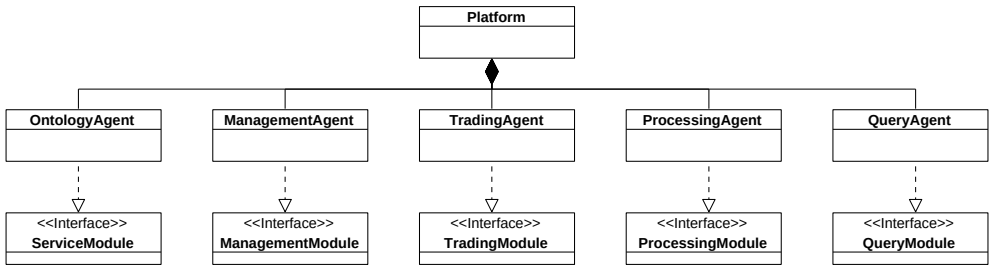


Figura 3.8: Modelo conceptual de una implementación de *TKRS* basada en *MAS*.

La *Figura 3.9* muestra el modelo de un repositorio (PIM/M1) con la implementación de *TKRS* basada en *MAS* descrito para un entorno *Java*, utilizando el *Eclipse Reflective Ecore Model Editor*. En la mitad superior de la *Figura* se puede apreciar que cada uno de los módulos del sistema se implementa como un módulo simple, mientras que en la mitad inferior, aparecen visualizadas las propiedades de la implementación del módulo de gestión: su nombre, su localización y la plataforma en la que ha sido desarrollado.

3.4. Configuración de *TKRS* (PSM/M2)

Una vez definidos, por un lado, la arquitectura de *TKRS* (Metamodelo PIM/M2 en la *Figura 3.1*) y, por otro, un repositorio de implementaciones en diferentes plataformas (Metamodelo PIM/M2), el siguiente paso consiste en establecer un mecanismo para relacionar cada elemento de la arquitectura con su correspondiente implementación, en

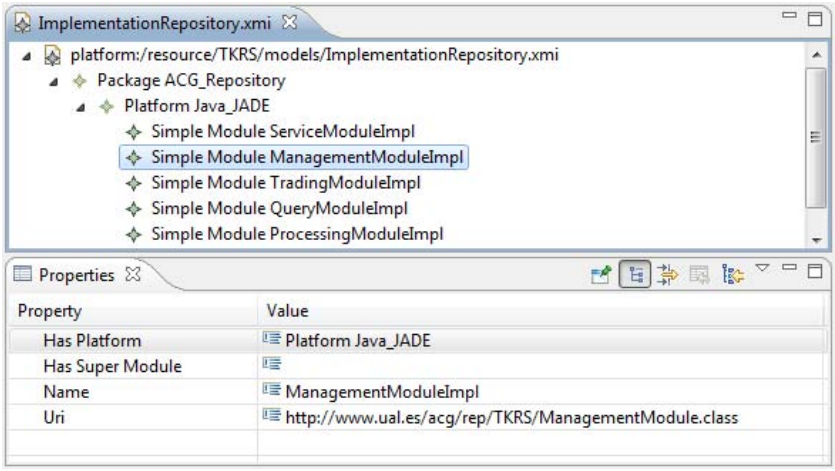


Figura 3.9: Modelo de repositorio utilizando *Eclipse Reflective Ecore Model Editor*.

este camino hacia el despliegue final del sistema en un entorno dado. Esto es lo que precisamente se plantea a continuación. Para llevarlo a cabo, se ha definido un nuevo metamodelo (PSM/M2) representado en la *Figura 3.10*. En él, la metaclase `Statement` relaciona cada módulo del metamodelo de la arquitectura de *TKRS*, a través de la metaclase abstracta `Module`, con la metaclase abstracta `Module` del metamodelo del repositorio de implementaciones. Además, para poder trabajar con ambos modelos —el de la arquitectura y el del repositorio—, incluye la metaclase `Import` con el atributo `importedNamespace` para permitir que los modelos de configuración puedan importarlos. Este metamodelo ha sido definido usando *EMF*, como en el caso de los metamodelos anteriores, pero para facilitar la creación de modelos de configuración, también se ha desarrollado un *Lenguaje Específico del Dominio* (*Domain Specific Language*, *DSL*), usando *Eclipse XText*. Esta herramienta permite definir una gramática —a partir de un metamodelo— que describa tanto la sintaxis concreta del lenguaje que se desea implementar como el procedimiento que debe seguir el *parser* para generar un modelo.

La *Tabla 3.4* representa la gramática definida para el lenguaje de configuración. La declaración de la gramática aparece en la línea #1, donde se indica su localización y el uso, a su vez, de la gramática *org.eclipse.xtext.common.Terminals* —que define reglas para terminales comunes como `ID`, `STRING` e `INT`—. De la línea #3 a la #6 se declaran los archivos *.ecore* con los metamodelos usados y, en las siguientes líneas, se define un mapeo entre el texto y el lenguaje del modelo. La regla de entrada para el *parser* aparece en la línea #8. Esta regla indica que cada entidad *Package* comienza con la palabra reservada `Package` seguida de su nombre y el símbolo "{"; a continuación, esta entidad define una cláusula para poder importar los modelos de la arquitectura (PSM/M1 en la *Figura 3.1*) y del repositorio de implementaciones (PSM/M1) —aunque previamente, es necesario aplicarles una transformación *M2M* para poder referenciar

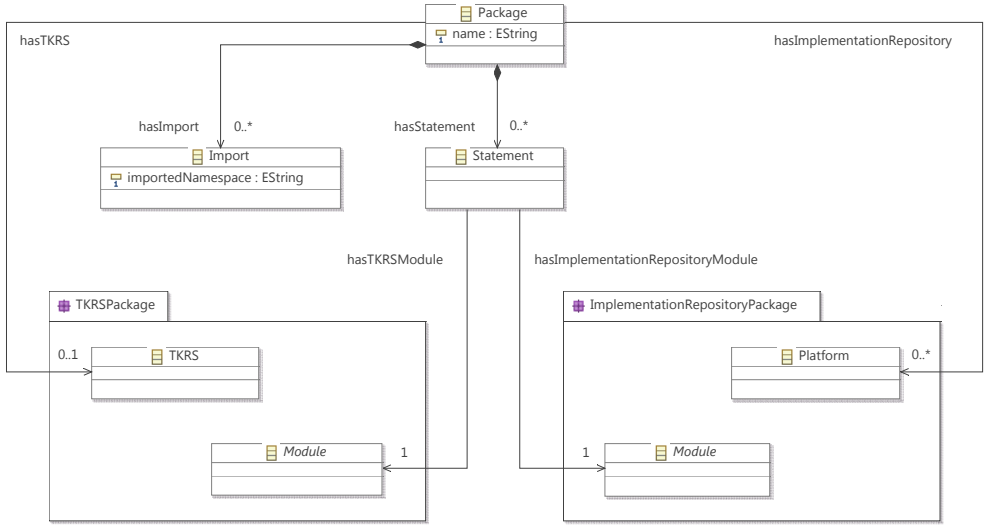


Figura 3.10: Metamodelo de configuración de *TKRS*.

sus elementos desde el modelo de configuración— y ya continúa la definición de todos los elementos de los tres metamodelos; finaliza con el símbolo "}". Como la definición se realiza del mismo modo para todos los elementos, veamos un ejemplo concreto con uno de ellos para que se comprenda mejor la idea. La regla definida a partir de la línea #31 indica que cada entidad *Node* que forma parte de *TKRS* comienza con la palabra reservada `Node` seguida nuevamente de su nombre y del símbolo "{"; entonces, se incluyen sus atributos con otra palabra reservada (como `ip`, `port`, etc.), seguida del valor del atributo; a continuación, esta entidad define los módulos que la componen (a través de las relaciones `hasServiceModule`, `hasManagementModule`, etc.), hace referencia al sistema en el que opera mediante la relación `hasTKRS` y termina con el símbolo "}".

Los modelos de configuración del sistema (PSM/M1) se crean usando un editor *Eclipse*. La *Tabla 3.5* muestra el fragmento de un modelo de configuración, siguiendo la estructura definida en la *Figura 3.6*. En cada sentencia (`Statement`) se puede distinguir claramente la asociación entre cada módulo de cada nodo del sistema (especificado en `hasTKRSModule`) y su correspondiente implementación en el repositorio (`hasImplementationRepositoryModule`).

A partir del modelo de configuración, ya es posible realizar el despliegue del sistema en un entorno real. Para mostrar un ejemplo de cómo se puede automatizar este proceso, se ha implementado una transformación *M2T* con la ayuda de la herramienta *Eclipse Xpand*, que permite la generación de los archivos necesarios para un entorno basado en *Java* a partir del modelo creado con el editor *DSL*. La *Tabla 3.6* representa la plantilla definida para la transformación. Esta plantilla describe el mapeo entre cada entidad del modelo y su correspondiente código. La sentencia descrita a partir de la línea #5 se refiere a la entidad *Package*. Esta definición genera, en la línea #8, un archivo script

```
grammar org.xtext.TKRS.config.Config with org.eclipse.xtext.common.Terminals

import "platform:/resource/TKRS/metamodels/Configuration.ecore"
import "platform:/resource/TKRS/metamodels/TKRS.ecore" as TKRSPackage
import "platform:/resource/TKRS/metamodels/ImplementationRepository.ecore" as
 ImplementationRepositoryPackage
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Package returns Package:
  {Package}
  'Package' name=EString
  ( hasImport+=Import (hasImport+=Import)* )?
  ( hasTKRS=TKRS )?
  ('ImplementationRepository' '{' hasImplementationRepository+=Platform
   (hasImplementationRepository+=Platform)* '}' )?
  ('Configuration' '{' hasStatement+=Statement (hasStatement+=Statement)* '}' )?;

Import returns Import:
  'import' importedNamespace=ImportedFQN;

ImportedFQN:
  FQN ('.' '*')?;

FQN:
  ID ('.' ID)*;

TKRS returns TKRSPackage::TKRS:
  'TKRS' name=EString
  '{'
    hasNode+=Node (hasNode+=Node)*
  '}';

Node returns TKRSPackage::Node:
  'Node' name=EString
  '{'
    'ip' ip=EString
    'port' port=EString
    'dbport' dbport=EString
    hasServiceModule=ServiceModule
    hasManagementModule=ManagementModule
    ( hasTradingModule+=TradingModule (hasTradingModule+=TradingModule)* )?
    hasQueryModule+=QueryModule (hasQueryModule+=QueryModule)*
    ( hasProcessingModule+=ProcessingModule (hasProcessingModule+=ProcessingModule)* )?
    'hasTKRS' hasTKRS=[TKRSPackage::TKRS|EString]
  '}';

TKRSModule returns TKRSPackage::Module:
  ManagementModule | QueryModule | TradingModule | ProcessingModule | ServiceModule;

ServiceModule returns TKRSPackage::ServiceModule:
  'ServiceModule' name=EString
  '{'
    'hasNode' hasNode=[TKRSPackage::Node|EString]
  '}';

ManagementModule returns TKRSPackage::ManagementModule:
  'ManagementModule' name=EString
  '{'
    'hasNode' hasNode=[TKRSPackage::Node|EString]
  '}';

QueryModule returns TKRSPackage::QueryModule:
  'QueryModule' name=EString
  '{'
```

Tabla 3.4: Definición de la gramática de configuración de *TKRS* (continúa).

```
63      'usesLookupInterface' usesLookupInterface=[TKRSPackage::TradingModule|EString]
64      'hasNode' hasNode=[TKRSPackage::Node|EString]
65    '}';
66
67  TradingModule returns TKRSPackage::TradingModule:
68    'TradingModule' name=EString
69    '{'
70      'usesLookupInterface' lookupInterface=EBoolean
71      'usesRegisterInterface' registerInterface=EBoolean
72      'usesAdminInterface' adminInterface=EBoolean
73      'usesLinkInterface' linkInterface=EBoolean
74      'usesProxyInterface' proxyInterface=EBoolean
75      ('isFederatedWith' isFederatedWith+=[TKRSPackage::TradingModule|EString]
         (isFederatedWith+=[TKRSPackage::TradingModule|EString])* )?
76      'hasNode' hasNode=[TKRSPackage::Node|EString]
77    '}';
78
79  ProcessingModule returns TKRSPackage::ProcessingModule:
80    'ProcessingModule' name=EString
81    '{'
82      'ambient' ambient=EString
83      'usesRegisterInterface' usesRegisterInterface=[TKRSPackage::TradingModule|EString]
84      'hasNode' hasNode=[TKRSPackage::Node|EString]
85    '}';
86
87  Platform returns ImplementationRepositoryPackage::Platform:
88    'Platform' name=EString
89    '{'
90      hasModule+=IRModule (hasModule+=IRModule)*
91    '}';
92
93  IRModule returns ImplementationRepositoryPackage::Module:
94    CompositeModule | SimpleModule;
95
96  CompositeModule returns ImplementationRepositoryPackage::CompositeModule:
97    'CompositeModule' name=EString
98    '{'
99      'uri' uri=EString
100     hasSubmodule+=IRModule (hasSubmodule+=IRModule)*
101     ('hasPlatform' hasPlatform=[ImplementationRepositoryPackage::Platform|EString])?
102     ('hasSuperModule' hasSuperModule=[ImplementationRepositoryPackage::Module|EString])?
103   '}';
104
105 SimpleModule returns ImplementationRepositoryPackage::SimpleModule:
106   'SimpleModule' name=EString
107   '{'
108     'uri' uri=EString
109     ('hasPlatform' hasPlatform=[ImplementationRepositoryPackage::Platform|EString])?
110     ('hasSuperModule' hasSuperModule=[ImplementationRepositoryPackage::Module|EString])?
111   '}';
112
113 Statement returns Statement:
114   'Statement'
115   '{'
116     'hasTKRSModule' hasTKRSModule=[TKRSPackage::Module|EString]
117     'hasImplementationRepositoryModule' hasImplementationRepositoryModule=
         [ImplementationRepositoryPackage::Module|EString]
118   '}';
119
120 EString returns ecore::EString:
121   STRING | ID;
122
123 EBoolean returns ecore::EBoolean:
124   'true' | 'false';
```

Tabla 3.4: Definición de la gramática de configuración de *TKRS* (continuación).

```
1  Package SOLERES_Configuration
2
3  Configuration {
4    Statement {
5      hasTKRSModule "SOLERES.KRS.Node_1.ServiceModule_1_1"
6      hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ServiceModuleImpl" }
7    Statement {
8      hasTKRSModule "SOLERES.KRS.Node_1.ManagementModule_1_1"
9      hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ManagementModuleImpl" }
10   Statement {
11     hasTKRSModule "SOLERES.KRS.Node_1.QueryModule_1_1"
12     hasImplementationRepositoryModule "ACG_Repository.Java_JADE.QueryModuleImpl" }
13   Statement {
14     hasTKRSModule "SOLERES.KRS.Node_1.TradingModule_1_1"
15     hasImplementationRepositoryModule "ACG_Repository.Java_JADE.TradingModuleImpl" }
16   Statement {
17     hasTKRSModule "SOLERES.KRS.Node_1.ProcessingModule_1_1"
18     hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ProcessingModuleImpl" }
19   ...
```

Tabla 3.5: Ejemplo de modelo de configuración de *TKRS*.

(*make.sh*) que contiene algunos comandos del sistema operativo como "`mkdir`", "`cd`" o "`wget`" para crear una estructura de carpetas que albergue los archivos necesarios. La línea #13 genera ahora un archivo *Java* por cada nodo del sistema, denominado *TKRS.java* y colocado en una carpeta que recibe su nombre del valor del atributo `name` del nodo, a su vez subcarpeta de otra con el valor del atributo `name` del sistema. El contenido de estos archivos se declara a partir de la línea #14: los atributos (líneas #20-#57), comprobando si el nodo cuenta con cada uno de los módulos definidos en *TKRS* para incluir el atributo correspondiente (por ejemplo, en las líneas #23-#29 se comprueba la presencia del `ServiceModule`); el constructor de la clase (líneas #59-#99), donde se inicializan los atributos definidos anteriormente, etc. Una vez ejecutada la transformación, se obtendría la implementación completa de *TKRS*.

3.5. Formalización del marco desarrollado

En esta *Sección* se presenta la formalización del marco desarrollado a lo largo del *Capítulo* para el diseño de *TKRS*, que incluye la definición de la arquitectura del sistema, del repositorio de implementaciones y del modelo de configuración. Para una mejor comprensión, se ha dividido en tres subsecciones.

3.5.1. Formalización de la arquitectura *TKRS*

En esta *Sección* se presenta la formalización de la arquitectura de *TKRS*.

Definición 15 (TKRS): Un *TKRS* $\mathcal{T}$ está determinado por dos conjuntos finitos $\mathcal{T} = \{\mathcal{H}_{\mathcal{T}}, \mathcal{N}\}$,

```
1  «REM»
2    «IMPORT org::xtext::tkrs::config::config»
3  «ENDREM»
4
5  «DEFINE Package FOR ConfigurationPackage::Package»
6    «IF !this.hasStatement.isEmpty»
7      «REM» ------------------------------------------------------------------------ «ENDREM»
8      «FILE "make.sh"»
9        #!/bin/bash
10       clear
11       «FOREACH this.hasStatement.first().hasISModule.hasNode.hasTKRS.hasNode AS node»
12         «REM» ------------------------------------------------------------------- «ENDREM»
13         «FILE node.hasTKRS.name + /" + node.name + /TKRS.java"»
14           package «node.hasTKRS.name».«node.name»;
15
16           import «node.hasTKRS.name».«node.name».modules.*;
17
18           public class TKRS
19           {
20             private String ip = null;
21             private int port = -1;
22             private int dbport = -1;
23             «FOREACH this.hasStatement AS statement»
24               «IF node==statement.hasTKRSModule.hasNode»
25                 «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::ServiceModule"»
26                   private ServiceModule serviceModule = null;
27                 «ENDIF»
28               «ENDIF»
29             «ENDFOREACH»
30             «FOREACH this.hasStatement AS statement»
31               «IF node==statement.hasTKRSModule.hasNode»
32                 «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::ManagementModule"»
33                   private ManagementModule managementModule = null;
34                 «ENDIF»
35               «ENDIF»
36             «ENDFOREACH»
37             «FOREACH this.hasStatement AS statement»
38               «IF node==statement.hasTKRSModule.hasNode»
39                 «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::QueryModule"»
40                   private QueryModule queryModule = null;
41                 «ENDIF»
42               «ENDIF»
43             «ENDFOREACH»
44             «FOREACH this.hasStatement AS statement»
45               «IF node==statement.hasTKRSModule.hasNode»
46                 «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::TradingModule"»
47                   private TradingModule tradingModule = null;
48                 «ENDIF»
49               «ENDIF»
50             «ENDFOREACH»
51             «FOREACH this.hasStatement AS statement»
52               «IF node==statement.hasTKRSModule.hasNode»
53                 «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::ProcessingModule"»
54                   private ProcessingModule processingModule = null;
55                 «ENDIF»
56               «ENDIF»
57             «ENDFOREACH»
58
59             public TKRS()
60             {
61               this.ip = "«node.ip»";
```

Tabla 3.6: Plantilla de una transformación *M2T* usando *Eclipse Xpand* (continúa).

```
62              this.port = «node.port»;
63              this.dbport = «node.dbport»;
64              «FOREACH this.hasStatement AS statement»
65                «IF node==statement.hasTKRSModule.hasNode»
66                  «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::ServiceModule"»
67                    this.serviceModule = new ServiceModule();
68                  «ENDIF»
69                «ENDIF»
70              «ENDFOREACH»
71              «FOREACH this.hasStatement AS statement»
72                «IF node==statement.hasTKRSModule.hasNode»
73                  «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::ManagementModule"»
74                    this.managementModule = new ManagementModule();
75                  «ENDIF»
76                «ENDIF»
77              «ENDFOREACH»
78              «FOREACH this.hasStatement AS statement»
79                «IF node==statement.hasTKRSModule.hasNode»
80                  «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::QueryModule"»
81                    this.queryModule = new QueryModule();
82                  «ENDIF»
83                «ENDIF»
84              «ENDFOREACH»
85              «FOREACH this.hasStatement AS statement»
86                «IF node==statement.hasTKRSModule.hasNode»
87                  «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::TradingModule"»
88                    this.tradingModule = new TradingModule();
89                  «ENDIF»
90                «ENDIF»
91              «ENDFOREACH»
92              «FOREACH this.hasStatement AS statement»
93                «IF node==statement.hasTKRSModule.hasNode»
94                  «IF statement.hasTKRSModule.metaType.name=="TKRSPackage::ProcessingModule"»
95                    this.processingModule = new ProcessingModule();
96                  «ENDIF»
97                «ENDIF»
98              «ENDFOREACH»
99            }
100
101           // CODE
102         }
103       «ENDFILE»
104       «REM» ------------------------------------------------------------------ «ENDREM»
105       mkdir /«this.hasStatement.first().hasTKRSModule.hasNode.hasTKRS.name»/
           «node.name»/modules
106       cd /«this.hasStatement.first().hasTKRSModule.hasNode.hasTKRS.name»/
           «node.name»/modules
107       «FOREACH this.hasStatement AS statement»
108         «IF node==statement.hasTKRSModule.hasNode»
109           wget «statement.hasImplementationRepositoryModule.uri»
110         «ENDIF»
111       «ENDFOREACH»
112       cd /«this.hasStatement.first().hasTKRSModule.hasNode.hasTKRS.name»/«node.name»
113       javac TKRS.java
114     «ENDFOREACH»
115     cd /
116   «ENDFILE»
117   «REM» ------------------------------------------------------------------ «ENDREM»
118 «ENDIF»
119 «ENDDEFINE»
```

Tabla 3.6: Plantilla de una transformación *M2T* usando *Eclipse Xpand* (continuación).

donde:

(i) $\mathcal{H}_{\mathcal{T}}$ representa un conjunto de propiedades $\mathcal{H}_{\mathcal{T}} = \{\mathcal{H}^1_{\mathcal{T}}\}$, siendo:

  (a) $\mathcal{H}^1_{\mathcal{T}}$ el nombre del sistema (`name` en el modelo conceptual).

(ii) $\mathcal{N}$ representa un conjunto de nodos (`Node`) ($\mathcal{N} \neq \varnothing$).

Definición 16 (Nodo): Un nodo (`Node` en el modelo conceptual) $\mathcal{N}$ está a su vez determinado por otros dos conjuntos finitos $\mathcal{N} = \{\mathcal{H}_{\mathcal{N}}, \mathcal{M}\}$,

donde:

(i) $\mathcal{H}_{\mathcal{N}}$ representa un conjunto de propiedades $\mathcal{H}_{\mathcal{N}} = \{\mathcal{H}^1_{\mathcal{N}}, \cdots, \mathcal{H}^l_{\mathcal{N}}\}$ / $l = \{1, \cdots, 4\}$, siendo:

  (a) $\mathcal{H}^1_{\mathcal{N}}$ el nombre del nodo (`name`).
  (b) $\mathcal{H}^2_{\mathcal{N}}$ una dirección *IP* (`ip`).
  (c) $\mathcal{H}^3_{\mathcal{N}}$ un puerto de comunicación (`port`).
  (d) $\mathcal{H}^4_{\mathcal{N}}$ un puerto secundario para la configuración y acceso al repositorio (`dbport`).

(ii) $\mathcal{M}$ representa un conjunto de módulos (`Module`) $\mathcal{M} = \{\mathcal{V}, \mathcal{G}, \mathcal{Q}, \mathcal{I}, \mathcal{E}\}$, siendo:

  (a) $\mathcal{V}$ un conjunto finito de módulos de servicio (`ServiceModule`)
  $\mathcal{V} = \{\mathcal{V}_1, \cdots, \mathcal{V}_o\}$ / $o = \{1, \cdots, n\}$ ($\mathcal{V} \neq \varnothing$).
  (b) $\mathcal{G}$ un conjunto finito de módulos de gestión (`ManagementModule`)
  $\mathcal{G} = \{\mathcal{G}_1, \cdots, \mathcal{G}_p\}$ / $p = \{1, \cdots, n\}$ ($\mathcal{G} \neq \varnothing$).
  (c) $\mathcal{Q}$ un conjunto finito de módulos de consulta (`QueryModule`)
  $\mathcal{Q} = \{\mathcal{Q}_1, \cdots, \mathcal{Q}_q\}$ / $q = \{1, \cdots, n\}$ ($\mathcal{Q} \neq \varnothing$).
  (d) $\mathcal{I}$ un conjunto finito de módulos de mediación (`TradingModule`)
  $\mathcal{I} = \{\mathcal{I}_1, \cdots, \mathcal{I}_r\}$ / $r = \{1, \cdots, n\}$.
  (e) $\mathcal{E}$ un conjunto finito de mód. de procesamiento (`ProcessingModule`)
  $\mathcal{E} = \{\mathcal{E}_1, \cdots, \mathcal{E}_s\}$ / $s = \{1, \cdots, n\}$.

En *TKRS* $\mathcal{T}$ tiene que darse que $\mathcal{I} \neq \varnothing$ y $\mathcal{E} \neq \varnothing$, al menos, en uno de sus nodos.

Definición 17 (Módulo de servicios): Un módulo de servicios (`ServiceModule` en el modelo conceptual) $\mathcal{V}_n$ se define también por un conjunto finito $\mathcal{V}_n = \{\mathcal{H}_{\mathcal{V}}\}$,

donde:

(i) $\mathcal{H}_{\mathcal{V}}$ representa un conjunto de propiedades $\mathcal{H}_{\mathcal{V}} = \{\mathcal{H}^1_{\mathcal{V}}\}$, siendo:

  (a) $\mathcal{H}^1_{\mathcal{V}}$ el nombre del módulo (`name`).

El módulo de servicios proporciona una serie de servicios a los demás módulos, incluyendo registro de módulos y componentes, verificación de estado, etc.

Definición 18 (Módulo de gestión): Un módulo de gestión (`ManagementModule` en el modelo conceptual) $\mathcal{G}_n$ se define por un conjunto finito $\mathcal{G}_n = \{\mathcal{H}_\mathcal{G}\}$,

donde:

(i) $\mathcal{H}_\mathcal{G}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{G} = \{\mathcal{H}^1_\mathcal{G}\}$, siendo:

(a) $\mathcal{H}^1_\mathcal{G}$ el nombre del módulo (`name`).

El módulo de gestión actúa como nexo entre la interfaz de usuario y el resto de módulos, permitiendo la configuración de éstos y siendo responsable de la gestión de las demandas de los usuarios.

Definición 19 (Módulo de consulta): Un módulo de consulta (`QueryModule` en el modelo conceptual) $\mathcal{Q}_n$ se define por un conjunto finito $\mathcal{Q}_n = \{\mathcal{H}_\mathcal{Q}\}$,

donde:

(i) $\mathcal{H}_\mathcal{Q}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{Q} = \{\mathcal{H}^1_\mathcal{Q}\}$, siendo:

(a) $\mathcal{H}^1_\mathcal{Q}$ el nombre del módulo (`name`).

El módulo de consulta se ocupa exclusivamente de las consultas de información que realizan los usuarios.

Definición 20 (Módulo de mediación): Un módulo de mediación (`TradingModule` en el modelo conceptual) $\mathcal{I}_r$ se define por un conjunto finito $\mathcal{I}_r = \{\mathcal{H}_\mathcal{I}\}$,

donde:

(i) $\mathcal{H}_\mathcal{I}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{I} = \{\mathcal{H}^1_\mathcal{I}, \cdots, \mathcal{H}^t_\mathcal{I}\} \ / \ t = \{1, \cdots, 6\}$, siendo:

(a) $\mathcal{H}^1_\mathcal{I}$ el nombre del módulo (`name`).

(b) $\mathcal{H}^2_\mathcal{I}$ una propiedad que indica si el módulo implementa o no la interfaz obligatoria *Lookup* del servicio de mediación (`lookupInterface`).

(c) $\mathcal{H}^3_\mathcal{I}$ una propiedad que indica si el módulo implementa o no la interfaz obligatoria *Register* del servicio de mediación (`registerInterface`).

(d) $\mathcal{H}^4_\mathcal{I}$ una propiedad que indica si el módulo implementa o no la interfaz *Admin* del servicio de mediación (`adminInterface`).

(e) $\mathcal{H}^5_\mathcal{I}$ una propiedad que indica si el módulo implementa o no la interfaz *Link* del servicio de mediación (`linkInterface`).

(f) $\mathcal{H}^6_\mathcal{I}$ una propiedad que indica si el módulo implementa o no la interfaz *Proxy* del servicio de mediación (`proxyInterface`).

El módulo de mediación permite la búsqueda y localización de la información en el sistema, estableciendo un filtro basado en los parámetros de las consultas.

Definición 21 (Módulo de procesamiento): Un módulo de procesamiento (`ProcessingModule` en el modelo conceptual) $\mathcal{E}_n$ se define por un conjunto finito $\mathcal{E}_n = \{\mathcal{H}_\mathcal{E}\}$,

donde:

(i) $\mathcal{H}_\mathcal{E}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{E} = \{\mathcal{H}^1_\mathcal{E}, \mathcal{H}^2_\mathcal{E}\}$, siendo:

  (a) $\mathcal{H}^1_\mathcal{E}$ el nombre del módulo (`name`).

  (b) $\mathcal{H}^2_\mathcal{E}$ el nombre del ambiente al que pertenece el módulo (`ambient`), ya que los módulos de procesamiento se pueden agrupar atendiendo a diferentes criterios, como, por ejemplo, su localización, la información que gestionan, etc.

El módulo de procesamiento es el responsable de la gestión de las fuentes y repositorios de conocimiento.

Definición 22 (Relación de consulta): Dados un módulo de consulta $\mathcal{Q}_q$ y un módulo de mediación $\mathcal{I}_r$, se define la relación de consulta $\mathcal{R}_\mathcal{K}$ como $\mathcal{R}_\mathcal{K} \subseteq \mathcal{Q}_q \times \mathcal{I}_r$. Su propósito es definir la relación que se establece entre ambos módulos para la consulta de información.

Definición 23 (Relación de registro): Dados un módulo de procesamiento $\mathcal{E}_s$ y un módulo de mediación $\mathcal{I}_r$, se define la relación de registro $\mathcal{R}_\mathcal{G}$ como $\mathcal{R}_\mathcal{G} \subseteq \mathcal{E}_s \times \mathcal{I}_r$. Su propósito consiste en definir la relación que se establece entre ambos módulos para la gestión de la información.

Definición 24 (Relación de federación): Dados dos módulos de mediación $\mathcal{I}_r$ e $\mathcal{I}_{r'}$ ($\mathcal{I}_r \neq \mathcal{I}_{r'}$), se define la relación de federación $\mathcal{R}_\mathcal{F}$ como $\mathcal{R}_\mathcal{F} \subseteq \mathcal{I}_r \times \mathcal{I}_{r'}$. Su propósito consiste en definir la relación que se establece entre ambos módulos para la federación. Las interfaces *Link* de ambos módulos deben estar activas (`linkInterface = true`).

3.5.2. Formalización del repositorio de implementaciones *TKRS*

En esta *Sección* se presenta la formalización del repositorio de implementaciones *TKRS*.

Definición 25 (Plataforma): Una plataforma (`Platform` en el modelo conceptual) $\mathcal{L}$ está determinada por dos conjuntos finitos $\mathcal{L} = \{\mathcal{H}_\mathcal{L}, \mathcal{W}\}$,

donde:

(i) $\mathcal{H}_\mathcal{L}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{L} = \{\mathcal{H}^1_\mathcal{L}\}$, siendo:

  (a) $\mathcal{H}^1_\mathcal{L}$ el nombre de la plataforma (`name`).

(ii) $\mathcal{W}$ representa un conjunto de implementaciones de módulos (`Module`) $\mathcal{W} = \{\mathcal{X}, \mathcal{Y}\}$ ($\mathcal{W} \neq \varnothing$), siendo:

  (a) $\mathcal{X}$ un conjunto finito de implementaciones de módulos compuestos (`CompositeModule`) $\mathcal{X} = \{\mathcal{X}_1, \cdots, \mathcal{X}_u\}$ / $u = \{1, \cdots, n\}$.

(b) $\mathcal{Y}$ un conjunto finito de implementaciones de módulos simples (`SimpleModule`) $\mathcal{Y} = \{\mathcal{Y}_1, \cdots, \mathcal{Y}_v\}$ / $v = \{1, \cdots, n\}$.

Definición 26 (Implementación de un módulo compuesto): Una implementación de un módulo compuesto (`CompositeModule` en el modelo conceptual) $\mathcal{X}_n$ se define a su vez por tres conjuntos finitos $\mathcal{X}_u = \{\mathcal{H}_\mathcal{X}, \mathcal{X}, \mathcal{Y}\}$,

donde:

(i) $\mathcal{H}_\mathcal{X}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{X} = \{\mathcal{H}^1_\mathcal{X}, \mathcal{H}^2_\mathcal{X}\}$, siendo:

(a) $\mathcal{H}^1_\mathcal{X}$ el nombre de la implementación del módulo (`name`).

(b) $\mathcal{H}^2_\mathcal{X}$ la localización de los archivos de implementación (`uri`).

(ii) $\mathcal{X}$ un conjunto finito de implementaciones de módulos compuestos (`CompositeModule`) $\mathcal{X} = \{\mathcal{X}_1, \cdots, \mathcal{X}_u\}$ / $u = \{1, \cdots, n\}$.

(iii) $\mathcal{Y}$ un conjunto finito de implementaciones de módulos simples (`SimpleModule`) $\mathcal{Y} = \{\mathcal{Y}_1, \cdots, \mathcal{Y}_v\}$ / $v = \{1, \cdots, n\}$.

Definición 27 (Implementación de un módulo simple): Una implementación de un módulo simple (`SimpleModule` en el modelo conceptual) $\mathcal{Y}_v$ se define por un conjunto finito $\mathcal{Y}_v = \{\mathcal{H}_\mathcal{Y}\}$,

donde:

(i) $\mathcal{H}_\mathcal{Y}$ representa un conjunto de propiedades $\mathcal{H}_\mathcal{Y} = \{\mathcal{H}^1_\mathcal{Y}, \mathcal{H}^2_\mathcal{Y}\}$, siendo:

(a) $\mathcal{H}^1_\mathcal{Y}$ el nombre de la implementación del módulo (`name`).

(b) $\mathcal{H}^2_\mathcal{Y}$ la localización de los archivos de implementación (`uri`).

Definición 28 (Relación de plataforma): Dadas una implementación de un módulo compuesto $\mathcal{X}_u$ ó una implementación de un módulo simple $\mathcal{Y}_v$, y una plataforma $\mathcal{L}$, se define la relación de plataforma, $\mathcal{R}_\mathcal{P}$ como $\mathcal{R}_\mathcal{P} \subseteq (\mathcal{L} \times \mathcal{X}_u) \vee (\mathcal{L} \times \mathcal{Y}_v)$. Su propósito consiste en definir la relación de pertenencia de una implementación a una plataforma.

Definición 29 (Relación de contención): Dadas una implementación de un módulo compuesto $\mathcal{X}_u$ y una implementación de un módulo simple $\mathcal{Y}_v$ u otra implementación de un módulo compuesto $\mathcal{X}_{u'}$, se define la relación de contención, $\mathcal{R}_\mathcal{C}$ como: $\mathcal{R}_\mathcal{C} \subseteq (\mathcal{X}_u \times \mathcal{Y}_v) \vee (\mathcal{X}_u \times \mathcal{X}_{u'})$. Su propósito es definir la relación de contención entre dos implementaciones de módulos.

3.5.3. Formalización de la configuración de *TKRS*

En esta *Sección* se presenta la formalización del modelo de configuración de *TKRS*.

Definición 30 (Configuración): Una configuración (`Configuration` en el modelo conceptual) $\mathcal{F}$ está determinada por un conjunto finito $\mathcal{F} = \{\mathcal{Z}\}$, donde:

(i) $\mathcal{Z}$ ($\neq \varnothing$)representa un conjunto de sentencias con las relaciones de implementación.

Definición 31 (Relación de implementación): Dados un módulo $\mathcal{M}_n$ y la implementación de dicho módulo $\mathcal{W}_n$, se define la relación de implementación $\mathcal{R}_\mathcal{I}$ como $\mathcal{R}_\mathcal{I} \subseteq \mathcal{M}_n \times \mathcal{W}_n$. Su propósito consiste en definir la relación entre un módulo y su correspondiente implementación.

3.6. Trabajos relacionados

La realidad es que existen escasos trabajos relacionados con el marco presentado en este documento para el diseño y desarrollo de *Sistemas de Gestión de Información basados en la Web*, como *TKRS*. Entre ellos, se puede mencionar [Okawa et al., 2008], donde se propone una metodología para la configuración de *Sistemas de Información* que une técnicas de *BPM* y *xUML*. También, en [Mathe et al., 2008], los autores proporcionan un marco basado en modelos que permite el desarrollo de prototipos de *Sistemas de Información Clínicos*, además de su simulación y despliegue.

Si se sitúa el punto de atención en la idea de diseñar el sistema en torno a un servicio de mediación y hacer uso de la tecnología de *Sistemas Multi-Agente*, como en la implementación descrita en la *Sección 3.3*, se pueden encontrar algunos *WIS* que, con el objetivo de lograr una mayor autonomía y operatividad, han optado por encapsular este servicio en un agente software. Por ejemplo, en [Tsai et al., 2007] se describe el agente *MinneTAC* como un agente de mediación desarrollado para participar en *Trading Agent Competition* (*TAC*). También, se destacan los beneficios de la implementación de un mediador como agente software en escenarios que requieren cooperación y negociación entre el mediador y el resto de componentes del sistema, así como en sistemas que requieren comunicación entre más de un mediador, haciendo uso de ontologías para representar la información compartida por los agentes, bien sea para describir los datos y las relaciones entre variables, como es el caso de [Collins et al., 2009], o definir las primitivas de comunicación y la interacción entre los agentes, como en [Ziming and Liyi, 2007].

3.7. Resumen y conclusiones

A partir de la idea original de diseñar un *Sistema de Gestión de Información basado en la Web* —denominado *TKRS*— en torno a un núcleo formado por el modelo de mediación *OntoTrader*, a lo largo de este *Capítulo* se ha modelado su arquitectura distribuida y compuesta de módulos con diferentes funcionalidades. Además, se ha dado otro paso y se han modelado también un repositorio de implementaciones y la configuración del sistema, con el objetivo de establecer una relación entre cada elemento de la arquitectura y su correspondiente implementación. De esta forma, se favorece la separación entre la arquitectura del sistema y la plataforma de desarrollo, y se minimiza el esfuerzo que supone el despliegue del mismo en un entorno real. Incluso, se han desarrollado una herramienta gráfica y un *DSL* para facilitar el diseño de la arquitectura y la elaboración de los modelos de configuración del sistema, respectivamente, y se ha mostrado un ejemplo de implementación de *TKRS* basada en un *Sistema Multi-Agente*. Finalmente, se ha utilizado notación matemática para realizar una formalización del marco completo.

Capítulo 4

Implementación y escenarios experimentales

Capítulo 4

Implementación y escenarios experimentales

Contenidos

Una vez planteadas las directrices para la construcción de un *Sistema de Representación del Conocimiento basado en Mediación* (*Trading-based Knowledge Representation System*, *TKRS*), ha llegado el momento de ponerlas en práctica. En este *Capítulo* se explican detalles del desarrollo e implementación del subsistema responsable de la gestión del conocimiento en *SOLERES* —denominado *SOLERES-KRS*—, un sistema que surge dentro del marco del proyecto del Plan Nacional de I+D+i (por el Ministerio de Ciencia e Innovación) titulado "*SOLERES: un sistema de información espacio-temporal para la gestión medioambiental basado en redes neuronales, agentes y componentes software*" (referencias TIN2006-06698/TIN2007-61497) y extendido por otro proyecto, "*Una metodología para la recuperación y explotación de información medioambiental mediante interfaces de usuario evolutivas y cooperativas*" (referencia TIN2010-15588), también perteneciente al Plan Nacional. Como se puede deducir, el dominio de aplicación es el medio ambiente, por lo que el sistema se puede catalogar como un *Sistema de Información para la Gestión Medioambiental* (*Environmental Management Information System*, *EMIS*) y un caso particular de *Sistema de Información basado en la Web* (*Web-based Information Systems*, *WIS*), como se ha indicado en la *Sección 1.2*.

El *Capítulo* se organiza en ocho secciones. Comienza justificando y presentando el dominio de aplicación de *SOLERES*, así como las características del subsistema *SOLERES-KRS* (objeto de implementación), en la *Sección 4.1*. Seguidamente, en la *Sección 4.2*, se ofrece una visión general de los *EMIS* y se presenta *SOLERES-KRS* como un ejemplo. Los detalles de implementación de la arquitectura del sistema, el análisis de sus componentes y la representación ontológica del conocimiento que maneja (básicamente metadatos de mapas ecológicos, imágenes de satélite y clasificaciones) se describen en la *Sección 4.3*. En la *Sección 4.4* se analiza la implementación del modelo *OntoTrader* en *SOLERES-KRS*. La *Sección 4.5* detalla el despliegue del sistema en un entorno real. La evaluación y validación de la propuesta se realiza en la *Sección 4.6*, presentando diferentes escenarios de prueba. Para finalizar, en la *Sección 4.7*, se discuten algunos trabajos relacionados y, en la *Sección 4.8*, se realiza un breve resumen junto con las conclusiones.

4.1. Introducción y conceptos relacionados

En algunas administraciones, organizaciones o instituciones de alcance medioambiental se utilizan *mapas ecológicos* para actividades de actuación en la valoración de impactos medioambientales, gestión del territorio, monitorización del impacto de determinadas actividades en espacios protegidos, etc. Estos mapas ecológicos tienen un alto coste en su elaboración, por su preparación a partir de trabajo de campo y actualización periódica (años). Sin embargo, en algunas actuaciones de gestión medioambiental, donde se necesitan estudios de la situación actual del territorio (como por ejemplo, la evaluación de la evolución del desarrollo de invernaderos en el levante de la provincia de Almería), estos mapas ecológicos no aportan la información medioambiental actualizada para la toma de decisiones, siendo necesario de nuevo un trabajo de campo. Esta información medioambiental no está disponible en otras fuentes de información espacial como en

imágenes de satélite (o de vuelos) y que, sin embargo, resulta más económica y rápida de adquirir (diariamente).

Precisamente, uno de los trabajos realizados desde el *Grupo de Investigación de Informática Aplicada* ha consistido en estudiar la posibilidad y viabilidad de establecer una correlación entre las variables medioambientales (cartografía) —usadas en la creación de los mapas ecológicos— y la información de las imágenes de satélite (correlación finalmente demostrada) que permitiese obtenerlas de forma automática con un coste temporal, económico y humano bastante inferior. Para ello, como muestra la *Figura 4.1*, por un lado, a partir de un mapa o un conjunto de mapas cartográficos de la zona de estudio y, mediante un proceso de sectorización (que además recibe otra serie de entradas), se obtiene un mapa ecológico; por otro lado, a partir de una imagen o un conjunto de imágenes de satélite de la misma zona, mediante un proceso de clasificación (que también recibe otra serie de entradas), se obtiene una imagen clasificada. Una vez obtenidos el mapa ecológico y la imagen clasificada, una red neuronal es la encargada de establecer la citada correlación. Los procesos de sectorización y clasificación serán explicados con más detalle en la *Sección 4.2*.

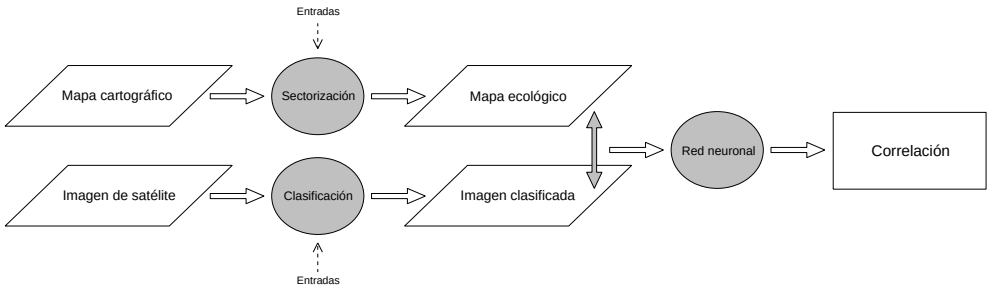


Figura 4.1: Mecanismo de correlación de mapas cartográficos e imágenes de satélite.

Toda la información utilizada durante los anteriores procesos necesita ser almacenada y gestionada de algún modo. Por este motivo, otro trabajo del *Grupo de Investigación* ha consistido en el desarrollo de un sistema de gestión y explotación de la información (en este caso medioambiental), favoreciendo la interacción hombre-máquina con interfaces de usuario que se adapten a los hábitos de los perfiles de usuario [Iribarne et al., 2010b] [Iribarne et al., 2011a] [Criado et al., 2010a] (en la línea de modelado de interfaces de usuario también se han realizado trabajos desde el Grupo en los que he participado, como [Almendros-Jiménez et al., 2009]), y con agentes software inteligentes que medien por los usuarios en los procesos de búsqueda y explotación de la información. De esta forma, se facilitan las tareas de toma de decisiones (medioambientales) y de predicción/prevención (que son las más importantes).

Este sistema se ha denominado *SOLERES* y, como se puede observar en la *Figura 4.2*, está formado, a su vez, por dos subsistemas: el *Sistema de Representación del Conocimiento* (*Knowledge Representation System*, *SOLERES-KRS* o *SKRS*), responsable de la gestión de la información medioambiental, y *SOLERES-HCI* (*Human-Computer In-*

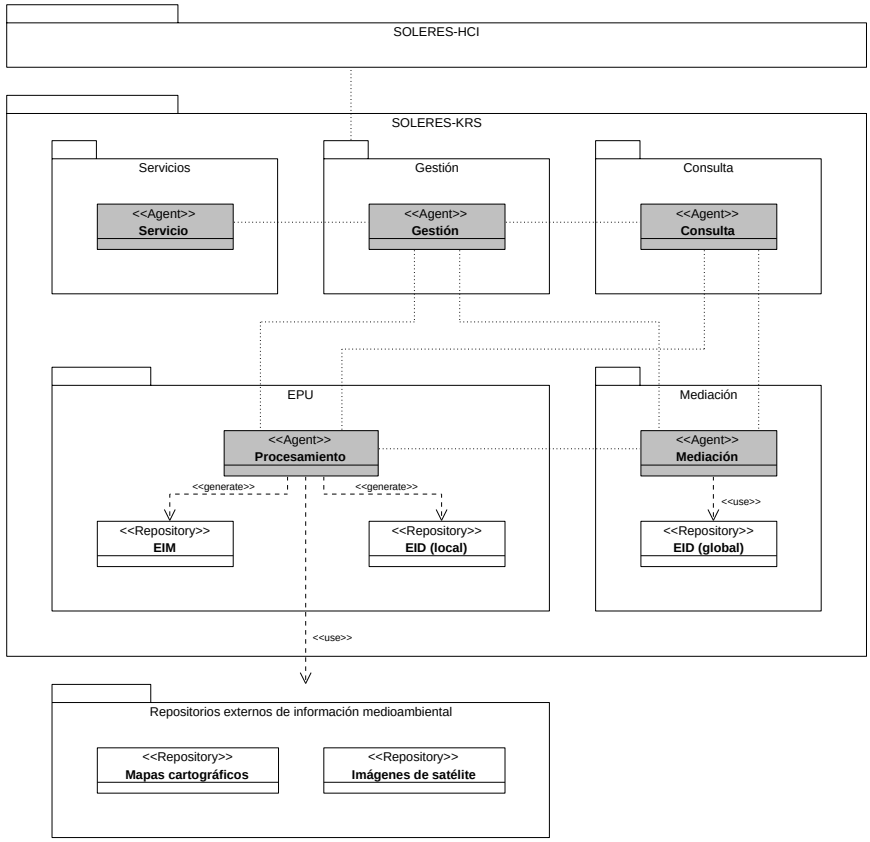


Figura 4.2: Arquitectura del sistema *SOLERES*.

teraction) [Iribarne et al., 2008] [Asensio et al., 2008a], relacionado con las interfaces de usuario que favorecen la explotación de esta información. Antes de continuar, es preciso señalar que este *Capítulo* se centra exclusivamente en el desarrollo de *SOLERES-KRS* siguiendo el marco propuesto en el *Capítulo 3* para el diseño de sistemas *TKRS*. Como dominio de la información, *SOLERES-KRS* maneja básicamente dos grandes y voluminosas fuentes de información original proporcionada en su mayoría por la *Red de Información Ambiental de Andalucía* (*REDIAM*) [REDIAM, 2007] —que tiene como objeto la integración de toda la información sobre el medio ambiente andaluz generada por todo tipo de centros en la Comunidad Autónoma—: (a) por un lado, información cartográfica y, por otro, (b) información de imágenes de satélite de la zona estudio (Granada y Almería).

Dado el volumen de información y dado que ésta puede ser proporcionada por diferentes fuentes, se ha optado por una solución distribuida en la que la información se reparte en unas denominadas *Unidades de Proceso Medioambiental* (*Environmental Pro-*

cess Units, `EPU` en la *Figura*) que trabajan con metadatos obtenidos a partir de ella: los *Mapas de Información Medioambiental* (*Environmental Information Map*, *EIM*). Por su parte, los módulos de `Mediación` realizan una doble función: (i) por un lado, integran la información de las diferentes *EPUs*, y por otro, (ii) facilitan el proceso de búsqueda y recuperación de la misma. Para ello, hacen uso de nuevos metadatos obtenidos, valga la reiteración, a partir de los *EIMs*, que almacenan localmente en un repositorio (las *EPUs* son las encargadas también de generarlos y, por este motivo, almacenan una copia en otro repositorio local); estos "meta-metadatos" se han denominado *metaDatos de Información Medioambiental* (*Environmental Information metaData*, *EID*). Los dos módulos mencionados, junto a los de `Consulta`, `Gestión` y `Servicios` que aparecen en la *Figura*, conforman la arquitectura *TKRS*.

Para finalizar, es interesante destacar que cada usuario del sistema contaría con el apoyo de un agente destinado en *SOLERES-HCI*, que operaría, por un lado, gestionando la presentación e interacción con la interfaz de usuario —mediando entre el usuario al que representa y otros usuarios del sistema por medio de sus respectivos agentes— y, por otro, gestionando las consultas medioambientales —a modo de consultor o asesor virtual que interactúa con otros agentes del subsistema *SOLERES-KRS*—.

4.2. Implementación de *SKRS* basada en *TKRS*

Los *Sistemas de Información para la Gestión Medioambiental* (*EMIS*) se clasifican dentro de los *Sistemas de Información* (*Information Systems*, *IS*) como un caso especial de *Sistemas de Información Geográfica* (*Geographic Information System*, *GIS*) con una mayor especificidad, como se indicaba en la *Sección 1.2*. Un principio básico para su desarrollo es el empleo de especificaciones y estándares —definidos y mantenidos por organizaciones dedicadas a la labor de estandarización—, que potencialmente pueden reducir los costes mediante la reducción de la dependencia de soluciones propietarias, acortando el tiempo de desarrollo y mejorando la calidad del producto [Bicocchi et al., 2010]; además, favorecen su adaptación a los continuos avances tecnológicos del mercado.

En la actualidad, existe un amplio conjunto de herramientas y métodos para el modelado de sistemas *EMIS* que cuentan con el apoyo de organizaciones como *W3C* (*World Wide Web Consortium*), *OMG* (*Object Management Group*) o *ISO* (*International Organization for Standardization*). Por ejemplo, mientras *W3C* y *OMG* dan soporte a materiales y técnicas de modelado de sistemas como *UML*, *MDA*, *OCL*, *XML*, *OWL*, etc., *ISO* enuncia normas, como la *ISO 14000*, que expresa cómo establecer un sistema de gestión ambiental efectivo, o la norma *ISO 9000* sobre calidad y gestión de calidad [Iribarne et al., 2007].

Entre los intereses de este trabajo se encuentra la mejora del diseño y desarrollo de los *Sistemas de Representación del Conocimiento* utilizando la *Ingeniería del Software*, la *Ingeniería Dirigida por Modelos* y la *Ingeniería Dirigida por Ontologías*. Su punto de confluencia se haya en la especificación de *TKRS* y se va a materializar a lo largo de las siguientes secciones en el diseño e implementación de *SOLERES-KRS* basado en un *Sistema Multi-Agente* (*Multi-Agent System*, *MAS*), con las características y singularidades que se han indicado en la *Sección* anterior.

Para ello, no sólo deben tenerse en cuenta las propiedades indicadas en la *Sección 1.6*, sino también la elección de una metodología precisa y una plataforma de desarrollo adecuada. De todos los productos mencionados también en esta *Sección*, se ha seleccionado *JADE* (*Java Agent Development Framework*) para la implementación del sistema, ya que la simplifica gracias a la incorporación de su propio *middleware*, proporcionando además un conjunto de herramientas de apoyo para la depuración de este tipo de sistemas abiertos, distribuidos y heterogéneos. También se puede destacar el uso que hace de los estándares de comunicación *FIPA* (*Foundation for Intelligent Physical Agents*) para el intercambio de mensajes *ACL* (*Agent Communication Language*) entre los agentes. Las acciones definidas en las ontologías de servicio/proceso (*Sección 2.4*) van a constituir el contenido de estos mensajes representados en *OWL*.

La creación de un agente en esta plataforma resulta tan sencilla como definir una clase *Java* que extiende la clase `jade.core.Agent` definida en *JADE* e implementar un método, denominado `setup()` donde se inicializa el agente. La *Tabla 4.1* muestra un fragmento de la clase `ManagementAgent`, que implementa al agente de gestión, donde aparece este método. Normalmente, cada una de las tareas que tiene que desarrollar un agente se encapsula en lo que se denomina un "comportamiento", que se implementa de modo similar al anterior, definiendo una nueva clase para cada comportamiento que extiende otra ya definida, como por ejemplo `jade.core.behaviours.Behaviour` (existen una serie de comportamientos predefinidos, como el comportamiento genérico, cíclico, de una única ejecución, etc.). Esta nueva clase debe implementar los métodos abstractos `action()` y `done()`: el primero define las operaciones que se realizan cuando se ejecuta el comportamiento, mientras que el segundo indica si el comportamiento se ha completado o no. Un agente puede ejecutar sus comportamientos, incluso, de forma concurrente. En relación a la comunicación entre los agentes, ésta se basa en un intercambio asíncrono de mensajes. Cada agente cuenta con un buzón o cola de mensajes, cuyo procesamiento es definido por el desarrollador. La *Tabla 4.2* incluye un fragmento de la clase que implementa el comportamiento del agente de gestión para la recepción de mensajes. Se trata de un comportamiento cíclico que se encuentra bloqueado a la espera de recibir un mensaje; cuando éste es recibido, se extrae el agente que lo envía, el contenido y su tipo, en función del cual se realiza la acción correspondiente.

```
 1  package SOLERES_KRS.ManagementAgent;
 2
 3  import jade.core.Agent;
 4
 5  public class ManagementAgent extends Agent {
 6    ...
 7    protected void setup() {
 8      ReceiveMessageBehaviour receiveMessageBehaviour;
 9      receiveMessageBehaviour = new ReceiveMessageBehaviour(this);
10      this.addBehaviour(receiveMessageBehaviour);
11    }
12    ...
13  }
```

Tabla 4.1: Fragmento de la clase `ManagementAgent`.

```
 1  package SOLERES_KRS.ManagementAgent;
 2
 3  import jade.core.AID;
 4  import jade.core.behaviours.CyclicBehaviour;
 5  import jade.lang.acl.ACLMessage;
 6
 7  public class ReceiveMessageBehaviour extends CyclicBehaviour {
 8    private ManagementAgent managementAgent;
 9    ...
10    public ReceiveMessageBehaviour(ManagementAgent managementAgent) {
11      super(managementAgent);
12      this.managementAgent = managementAgent;
13    }
14
15    public void action() {
16      ACLMessage msg = null;
17      AID sender = null;
18      String localName = null;
19      String content = null;
20
21      msg = this.imiAgent.receive();
22      if(msg != null) {
23        sender = msg.getSender();
24        localName = sender.getLocalName();
25        content = msg.getContent();
26        switch(msg.getPerformative()) {
27          case ACLMessage.QUERY_REF:
28            ...
29            break;
30          case ACLMessage.INFORM_REF:
31            ...
32            break;
33          ...
34        }
35      }
36      this.block();
37    }
38  }
```

Tabla 4.2: Fragmento de la clase `ReceiveMessageBehaviour`.

En las siguientes secciones se describe con detalle la implementación de la arquitectura multi-agente del sistema y la representación de la información que gestiona a nivel de metadatos y meta-metadatos.

4.2.1. Arquitectura multi-agente de *SOLERES-KRS*

Realizando un rápido recordatorio de la arquitectura de *TKRS* detallada en la *Sección 3.2*, ésta estaba formada por un conjunto de nodos, cada uno de los cuales a su vez constaba de una serie de módulos: **módulo de servicios**, **módulo de gestión**, **módulo de consulta**, **módulo de mediación** y **módulo de procesamiento**. Estos módulos eran o no requeridos por los nodos. Teniendo presente este modelo de arquitectura, la elección de la tecnología de agentes para la implementación de *SOLERES-KRS* y sus características descritas, la arquitectura del sistema quedaría definida como representa la *Figura 4.3* de forma general [Iribarne et al., 2010a] [Criado et al., 2010b] (este esquema no incluye los agentes relacionados con el módulo de servicios).

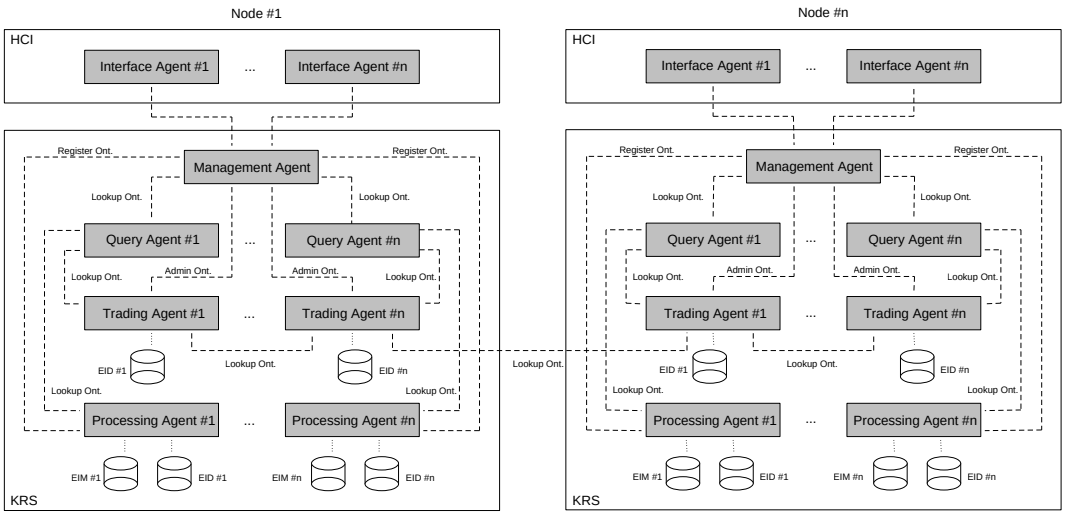


Figura 4.3: Arquitectura general de *SOLERES-KRS*.

El **módulo de gestión** se implementa con un *Agente de gestión* (`Management Agent`), que va a actuar de intermediario entre los *Agentes de interfaz* (`Interface Agents`) del subsistema *SOLERES-HCI* y el resto de agentes de *SOLERES-KRS* para gestionar todas las demandas de los usuarios; el **módulo de consulta**, con un *Agente de consulta* (`Query Agent`), directamente encargado de resolver las consultas de información ecológica; el **módulo de mediación**, con un *Agente de mediación* (`Trading Agent`) que facilita la búsqueda y localización de esta información utilizando los *EIDs* almacenados en su repositorio; y el **módulo de procesamiento**, con un *Agente de procesamiento* (`Processing Agent`) —que representa las denominadas *EPUs*—, responsable de las fuentes de conocimiento (gestión de los *EIMs*, exportación de los *EIDs* al agente de mediación asociado, etc.). En el **módulo de servicios** aparecen una serie de agentes propios del *framework JADE*, como el *Agent Management System* (*AMS*) o el *Directory Facilitator* (*DF*), que proporcionan un servicio de páginas blancas para la gestión de todos los agentes y otro de páginas amarillas, respectivamente, además de otros agentes de monitorización. Las líneas discontinuas de la *Figura* representan las ontologías que intervienen en el proceso de comunicación de los agentes: *Lookup*, *Register* y *Admin*.

La *Figura 4.4* sí representa el modelo concreto de la arquitectura de *SOLERES-KRS*, que ha sido diseñada utilizando el editor *GMF* analizado en la *Sección 3.2.3* [Asensio et al., 2011c]. Como se observa en ella, el sistema está formado por tres nodos, cada uno de los cuales cuenta con un agente de gestión (que constituye el módulo de gestión de *TKRS*), un agente de consulta (módulo de consulta) y los agentes correspondientes al módulo de servicios obligatorios. Los nodos identificados como "Node 1" y "Node 2", además contienen un agente de procesamiento (módulo de procesamiento, *EPU*) y un agente de mediación (módulo de mediación), respectivamente, cumpliendo

las exigencias mínimas de *TKRS*. Los agentes de mediación de los nodos anteriores se encuentran federados (representado con una flecha de trazo discontinuo) y sus agentes de consulta están asociados al agente de mediación de sus respectivos nodos, mientras el agente de procesamiento del "Node 1" y el agente de consulta del "Node 3" están asociados al agente de mediación del "Node 2" (todas estas asociaciones aparecen representadas con flechas de trazo continuo).

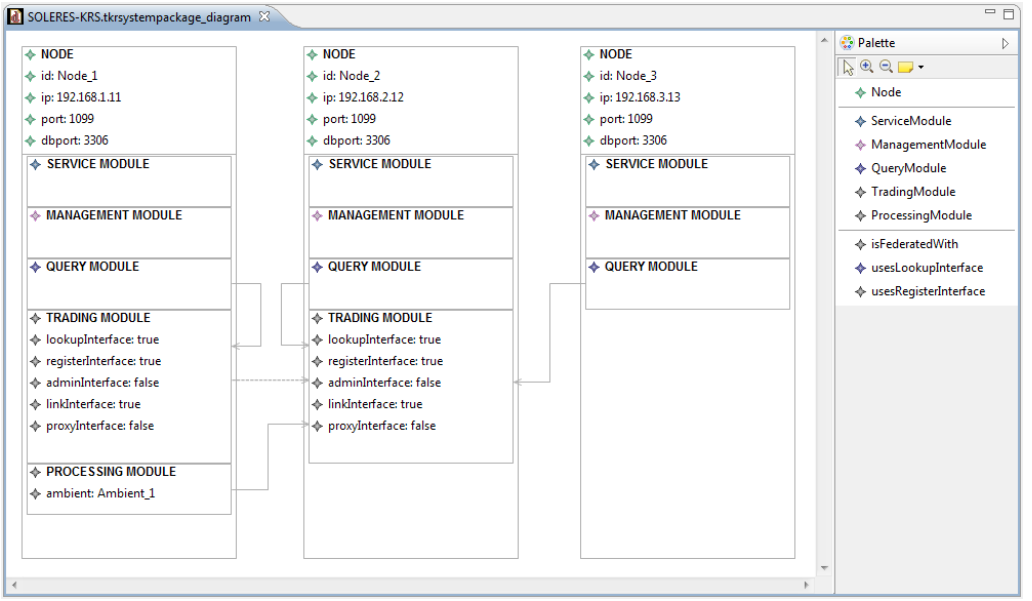


Figura 4.4: Modelo de la arquitectura de *SOLERES-KRS* usando la herramienta *GMF*.

4.2.2. Representación del conocimiento

Como se adelantaba en la *Sección 4.1*, cada *EPU* o agente de procesamiento del sistema gestiona dos repositorios locales de información medioambiental. El primero de ellos almacena metadatos de la información del dominio, es decir, básicamente metadatos de los mapas ecológicos, de las imágenes de satélite y de sus respectivas sectorizaciones o clasificaciones, y han sido denominados *Mapas de Información Medioambiental* (*Environmental Information Map*, *EIM*). Éstos representan el segundo nivel de la arquitectura de datos de *TKRS*, explicada en la *Sección 3.2.1*; el primer nivel estaría formado por la información de la que se extraen estos metadatos y que procede de fuentes externas. El segundo repositorio almacena los metadatos más relevantes de los *EIMs*, que van a ser utilizados por los agentes de mediación en los procesos de búsqueda y recuperación de información, y además incorporan nuevos metadatos necesarios para la integración de las diferentes *EPUs* y su gestión por los agentes. Estos meta-metadatos representan el tercer y último nivel de la arquitectura de datos y se denominan *metaDatos de Información Medioambiental* (*Environmental Information metaData*, *EID*). Cada agente de

procesamiento mantiene su propio conjunto de *EIDs* a nivel local y también los registra en un agente de mediación con el que se encuentra asociado; de esta forma, el agente de mediación dispone de un repositorio global con todos los *EIDs* de todos los agentes de procesamiento asociados a él. Los *EIMs* y los *EIDs* se corresponden con las ontologías de datos del sistema [Asensio et al., 2011a].

Las ontologías nacieron ante la necesidad de determinadas aplicaciones de gestionar la semántica de la información, además de su representación. Por ejemplo, en un *EMIS*, las ontologías no sólo permiten describir datos espaciales utilizando una codificación semántica comprensible por los ordenadores, sino que además permiten integrar datos geográficos procedentes de diferentes fuentes. En el contexto de los *Sistemas de Información*, por lo general, las ontologías suelen modelar el dominio del conocimiento en términos de clases, atributos y relaciones. Aunque existen diferentes lenguajes para el modelado del dominio del conocimiento, en particular, destacan *DAML+OIL* [McGuinness et al., 2002] y *OWL* [W3C, 2004a].

OWL (*Web Ontology Language*) es el lenguaje más reciente y ampliamente utilizado en la actualidad. Es el lenguaje estándar del *W3C* que sigue una filosofía orientada a objetos para la definición de ontologías en la web, aunque hoy en día, se ha extendido a cualquier dominio. Una de sus características más destacadas es que permite relacionar el contenido de una ontología con el de otra u otras, es decir, una ontología puede utilizar los términos definidos en otra, ampliarlos, etc. Además, *OWL* está soportado por otros lenguajes del *W3C* como *XML*, *RDF* o *RDF-Schema*. Las principales razones que han motivado su uso en *SOLERES-KRS* para la implementación tanto de las ontologías de servicio/proceso como de datos son las siguientes:

(a) Se trata de un lenguaje semántico diseñado específicamente para servicios basados en la web.

(b) Es un lenguaje interpretable por un ordenador, lo que posibilita a los agentes la ejecución automática de tareas de razonamiento y verificación.

(c) El nivel de expresividad de la representación del conocimiento es el adecuado para un sistema basado en agentes.

(d) El tratamiento y manipulación de las diferentes ontologías es más sencillo que con el resto de lenguajes propuestos como contenido para los mensajes intercambiados por los agentes.

Aunque las definiciones semánticas del conocimiento del sistema se realizan mediante el uso de un modelo de texto escrito en *OWL* —que los agentes software utilizan en su comunicación para interactuar y coordinarse de forma autónoma—, con el fin de facilitar la labor de diseño a los ingenieros y desarrolladores, las ontologías han sido modeladas previamente utilizando una perspectiva visual en notación *UML*: se hace uso de diagramas de clases para definir el modelo semántico subyacente en las ontologías. Por lo tanto, ha surgido la necesidad de realizar un mapeado de los modelos *UML* —una vez finalizados sus diseños— a *OWL* [Padilla et al., 2008] [Asensio et al., 2008b] [Iribarne et al., 2011b], proceso que se lleva a cabo de forma automática por medio de una transformación de

modelos denominada transformación *UML2OWL*. Existe una implementación de esta transformación como un componente (*plugin*) de código abierto [SIDo Group, 2007] para la plataforma *Eclipse*, por lo que no ha sido necesario desarrollarla. Este componente implementa la especificación *ODM* (*Ontology Definition Metamodel*) de *OMG* y su uso resulta muy útil para transformaciones simples; sin embargo, presenta ciertas deficiencias cuando se emplean tipos de datos complejos.

La *Figura 4.5* muestra un esquema de transformación *MDE* (basado en la clásica *MDA* de *OMG*) para la obtención de un modelo *OWL/XML* a partir de otro *UML*. Se puede observar que intervienen tres modelos distintos en el proceso: un modelo *UML*, un modelo *OWL* y un modelo *OWL/XML*. La transformación se realiza en dos etapas: en la primera etapa, se utiliza la transformación *ATL UML2OWL* para obtener un modelo *OWL* intermedio generado de forma temporal a partir de un modelo que representa el diagrama de clases *UML* de la ontología correspondiente; finalmente, en la segunda etapa, se obtiene el modelo final de la ontología *OWL* en formato *XML* mediante la transformación *ATL OWL2XML*.

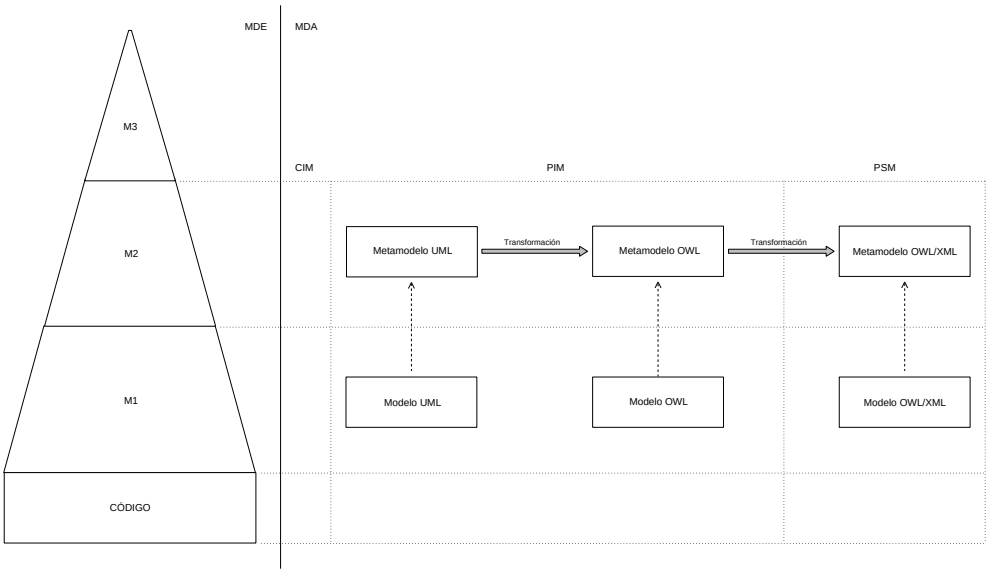


Figura 4.5: Transformación de modelos *UML* a *OWL/XML*.

En la *Tabla 4.3* aparece un fragmento del modelo *OWL/XML* obtenido como resultado del proceso de mapeo de la ontología que representa los *EIMs*, con la definición de la clase `SatelliteImageClassification`. Se puede observar una referencia generada automáticamente por el proceso de transformación seguida de una lista de restricciones. La referencia está constituida por el elemento `ID`, mientras que la lista de restricciones incluye características de la clasificación, como las propiedades `id` (`SatelliteImageClassification_id`) o `name` (`SatelliteImageClassification_name`)

y las relaciones con instancias de otros clases, como `Performance_indicator` (`SatelliteImageClassification_hasPerforman- ceIndicator`) o `Technician` (`SatelliteImageClassification_hasTechnician`). Además, la restricción de cardinalidad asegura que cada clasificación tenga exactamente un identificador, un nombre, etc. Básicamente, la definición del resto de clases de la ontología sería similar a ésta.

```
...
<owl:Class rdf:ID="SatelliteImageClassification">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#SatelliteImageClassification_id"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#SatelliteImageClassification_name"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#SatelliteImageClassification_hasPerformanceIndicator"/>
      <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#SatelliteImageClassification_hasTechnician"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
...
```

Tabla 4.3: Fragmento de la ontología *EIM* en formato *OWL/XML*.

Las siguientes secciones analizan con detalle las ontologías de datos, aunque dado el volumen de los metadatos que recogen los *EIMs*, para facilitar su estudio, se han dividido en dos: (i) metadatos de mapas ecológicos (cartografía) y sectorizaciones por un lado —a partir de un mapa o un conjunto de mapas cartográficos, mediante un proceso de sectorización, se obtiene un mapa ecológico—, y (ii) metadatos de satélite y clasificaciones por otro —a partir de una imagen o un conjunto de imágenes de satélite, mediante un proceso de clasificación, se obtiene una imagen clasificada—, si bien el *Anexo B* de este documento incluye un modelo conceptual con su diseño completo. Como adelanto, las *Figuras 4.6* y *4.7* muestran parcialmente los diagramas de las ontologías *EIM* (metadatos) y *EID* (meta-metadatos) —esta última representa la información manejada por el servicio de mediación—, respectivamente. También, en [ACG, 2009] se pueden encontrar ejemplos de instancias de ellas.

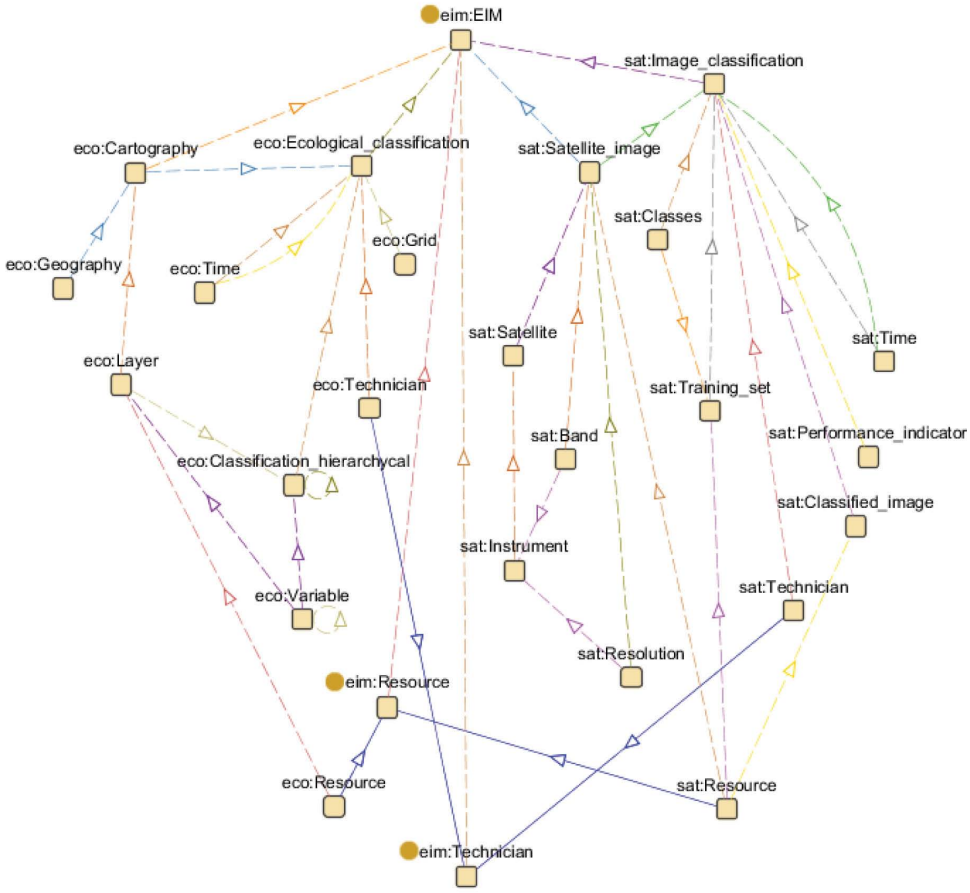


Figura 4.6: Diagrama de la ontología *EIM* (metadatos).

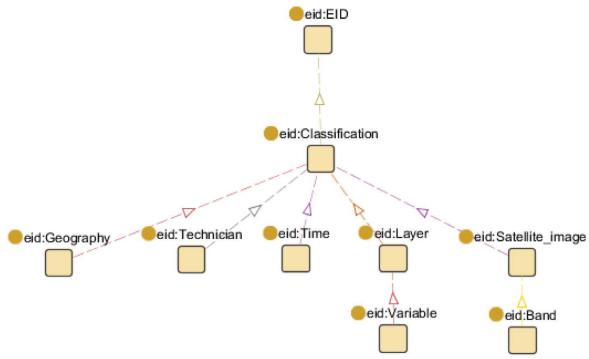


Figura 4.7: Diagrama de la ontología *EID* (meta-metadatos).

4.2.2.1. Modelado de metadatos ecológicos y sectorizaciones

Para el estudio realizado en los proyectos de investigación, se ha seleccionado una región del sudeste de España, concretamente el área que comprende la extensión de las provincias de Granada y Almería. Dos números que permiten hacerse una idea de la magnitud de los datos recopilados son la utilización de 5 mapas cartográficos (un mapa climático, un mapa litológico, un mapa geomorfológico, un mapa edáfico y un mapa de vegetación y usos del suelo) que recogen un total de 150 variables ecológicas (temperatura, precipitación, altitud, pendiente, orientación...) en su conjunto. La representación de sus metadatos se puede observar en el modelo conceptual de la *Figura 4.8*. La información cartográfica (clase `Cartography`) comprende información geográfica (`Geography`) —que incluye los datos de identificación de la zona de estudio, su área (`area`), perímetro (`perimeter`), georeferenciación (`utm_x` y `utm_y`), etc.— y los mapas ecológicos (`Layer`). Cada mapa debe recoger, al menos, la información de una variable ecológica (`Variable`) —como su clasificación (`category`), tipo (`type`), valores máximo y mínimo (`min_value` y `max_value`), etc.— y se almacena en un archivo (`Resource`).

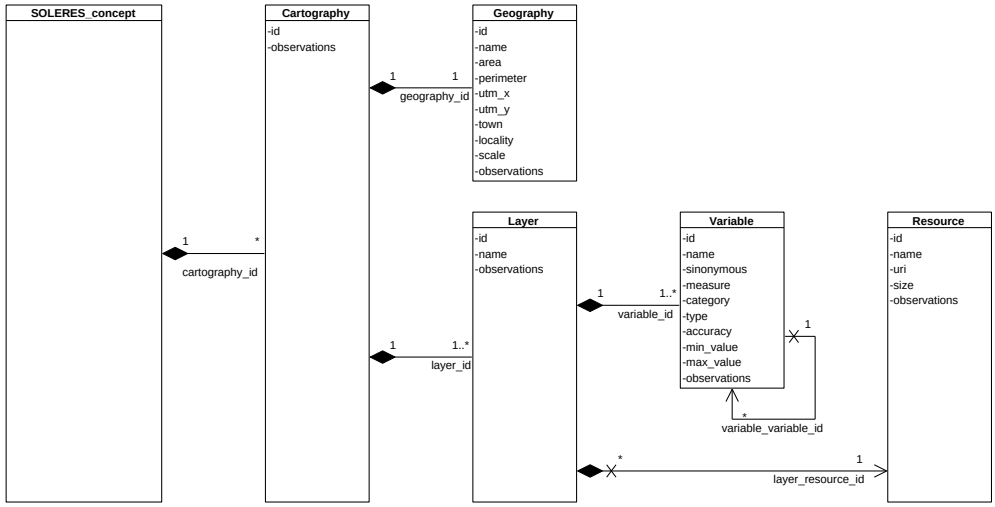


Figura 4.8: Modelo conceptual de los metadatos de los mapas cartográficos.

El modelado de la información medioambiental se encuentra vinculado al proceso de clasificación ecológica, también denominada sectorización ecológica. Éste implica la representación de las relaciones ecológicas más significativas que tienen lugar entre los elementos de un territorio. Partiendo de un mapa territorial, se representa un área dividida en subsistemas o unidades medioambientales internamente homogéneas, en los que existen asociaciones entre las variables biológicas y abióticas. La integración de estas variables puede ser trazada a diferentes escalas espaciales, lo que implica su clasificación en diferentes niveles de heterogeneidad o jerarquías. Siguiendo con el estudio, se ha realizado una sectorización con tres niveles jerárquicos: en el primer nivel, a partir de

los sectores de los mapas climático y litológico, se ha generado un nuevo mapa con los sectores climáticos divididos a su vez en subsectores litológicos; en el segundo nivel, a partir de este nuevo mapa y de los sectores del mapa geomorfológico, se ha creado un segundo mapa con los subsectores litológicos divididos nuevamente en subsectores geomorfológicos; y en el tercer y último nivel, a partir de este segundo mapa y de los sectores de los mapas edáfico y de vegetación y usos del suelo, se ha obtenido un tercer mapa, denominado mapa ecológico. El esquema de esta sectorización aparece representado la *Figura 4.9*.

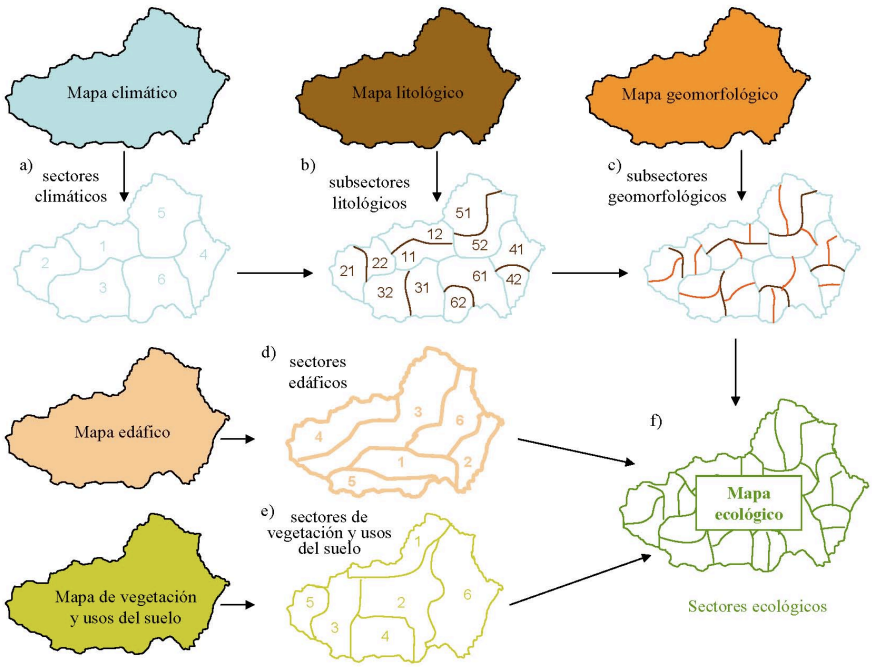


Figura 4.9: Esquema de una sectorización ecológica jerárquica de tres niveles.

Los conceptos asociados al modelado del proceso de clasificación ecológica (`Ecological_classification`) se muestran en la *Figura 4.10*. La clasificación ecológica de un territorio se lleva a cabo siguiendo un procedimiento de descripción jerárquica integrada (`Classification_hierarchycal`), que puede crear nuevos mapas ecológicos (`Layer`) y nuevas variables (`Variable`) a partir de los existentes (como se puede observar en las relaciones entre estas clases). Este procedimiento de división del territorio se basa en la generación de cuadrículas (`Grid`), donde se representa la dirección y la dimensión de los datos en su proyección espacial. Cada clasificación puede ser realizada por uno o más técnicos (`Technician`) en un momento determinado del tiempo (`Time`).

Dada su extensión, en [ACG, 2009] se encuentra la representación completa de la ontología *EIM* en notación *OWL/XML*.

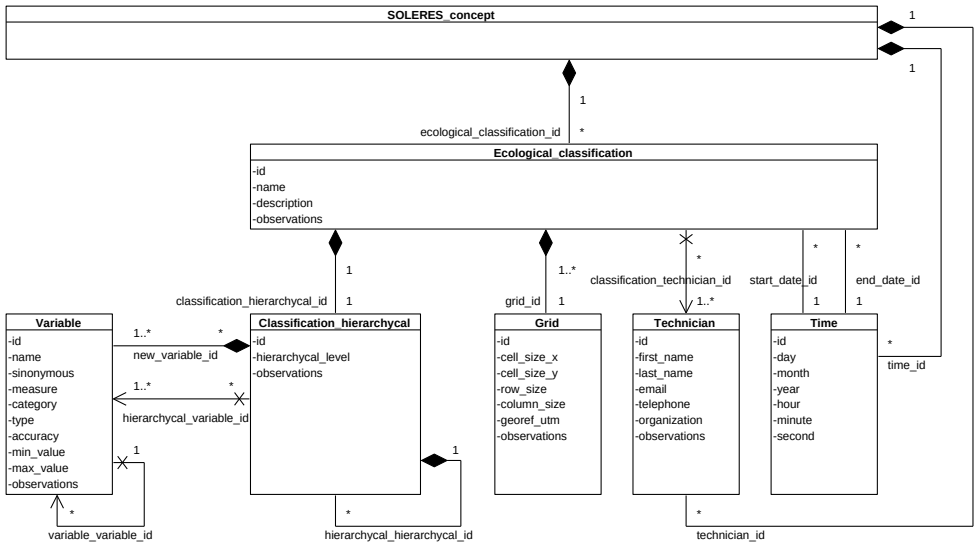


Figura 4.10: Modelo conceptual de los metadatos de las sectorizaciones ecológicas.

4.2.2.2. Modelado de metadatos de satélite y clasificaciones

Con la información de satélite se ha realizado un proceso análogo al efectuado con la información ecológica, ahora a partir de una serie de imágenes de satélite obtenidas de la región seleccionada. La representación de sus metadatos aparece en el modelo conceptual de la *Figura 4.11*, donde cada imagen (`Satellite_image`) se adquiere a través de un satélite de teledetección (`Satellite`) del que básicamente se recogen su nombre (`name`) y el nombre de la agencia a la que pertenece (`agency_name`). Los satélites utilizan un conjunto de instrumentos o sensores (`Instrument`), que actúan en ciertas bandas (`Band`) del espectro para adquirir las imágenes con una determinada resolución (`Resolution`). Esta resolución describe la información espectral (`spectral`), radiométrica (`radiometric`), espacial (`spatial`) y temporal (`temporal`) de la imagen (la *Tabla 4.4* muestra el ejemplo de las especificaciones de una imagen *Landsat*). La información de cada imagen de satélite se puede almacenar en uno o más archivos (`Resource`) con su formato específico.

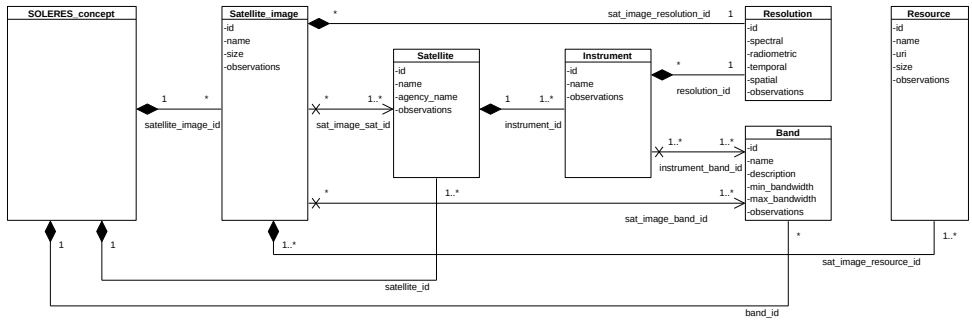


Figura 4.11: Modelo conceptual de los metadatos de las imágenes de satélite.

Elemento	**Comentarios**
Nombre	*LANDSAT*
Observaciones	Pertenece al programa *Landsat* de la *NASA*
Instrumento	Sensor *Thematic Mapper*^{TM}
Resolución	Radiométrica: 8 bits Temporal: 15 días Espacial: banda #6, 120 × 120 m; resto de bandas, 30 × 30 m Espectral: 7 bandas
Bandas	Banda #1 (Azul): de 0,45 a 0,52 μm Banda #2 (Verde): de 0,52 a 0,60 μm Banda #3 (Rojo): de 0,63 a 0,69 μm Banda #4 (Infrarrojo cercano): de 0,76 a 0,90 μm Banda #5 (Infrarrojo medio): de 1,55 a 1,75 μm Banda #6 (Infrarrojo termal): de 10,40 a 12,50 μm Banda #7 (Infrarrojo medio): de 2,08 a 2,35 μm

Tabla 4.4: Hoja de especificaciones de una imagen *Landsat*.

En este caso, el modelado de la información de satélite también se encuentra vinculado a un proceso de clasificación. El objetivo de la clasificación de imágenes de satélite es el mismo que el de la sectorización de mapas cartográficos, sólo que el proceso ahora consiste en dividir los píxeles de la imagen en clases discretas (*spectral classes*). La imagen clasificada resultante es esencialmente un mapa temático de la imagen original. Este proceso de clasificación aparece representado gráficamente en la *Figura 4.12*.

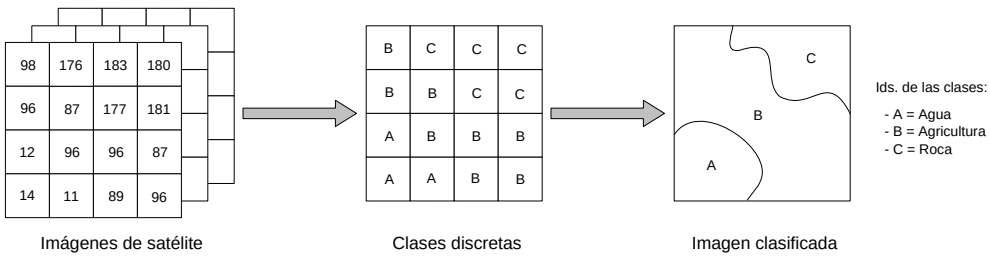


Figura 4.12: Proceso de clasificación de las imágenes de satélite.

Los conceptos asociados con la información de una clasificación se representan agrupados en torno a la clase `Image_classification` de la *Figura 4.13*. Como sucede en el caso de la sectorización ecológica, la clasificación puede ser realizada por uno o más técnicos (`Technician`) en un momento de terminado del tiempo (`Time`) y utiliza un conjunto de entrenamiento (`Training_set`) y un método de evaluación, en este caso "*indicador de rendimiento*" (`Performance_indicator`) —aunque existen otros métodos de evaluación, como *elipses*, *histogramas*, *divergencia transformada*, *divergencia estadística* o *distancia de Jeffries-Martusita*—. Su resultado es una imagen clasificada (`Classified_image`) con un conjunto de clases (`Classes`). Tanto el conjunto de entrenamiento como la imagen clasificada también se almacenan en diferentes tipos de archivos (`Resource`).

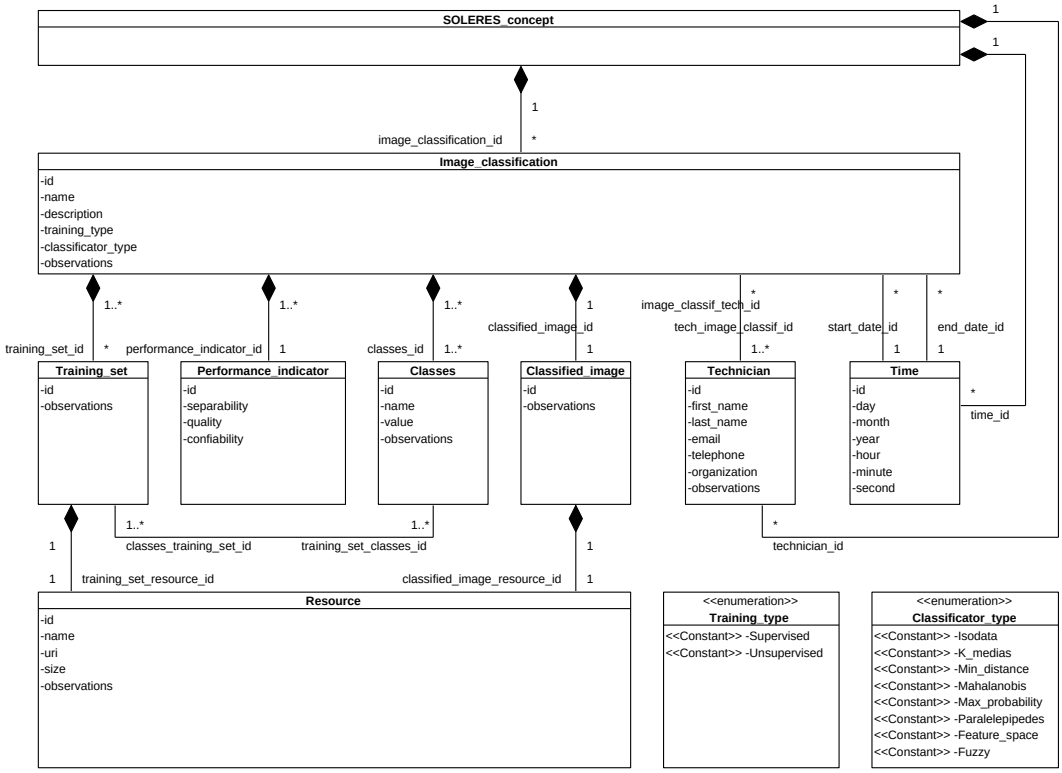


Figura 4.13: Modelo conceptual de los metadatos de las clasificaciones de las imágenes de satélite.

Los procedimientos comunes de clasificación (`Training_type`) se agrupan según el método utilizado en supervisados (`Supervised`) y no supervisados (`Unsupervised`). Existen diferentes tipos de algoritmos de clasificación supervisados basados en la estadística. Algunos de los más populares son *paralelepípedos* (`Paralelepipedes`), *distancia mínima* (`Min_distance`), *difusos* (`Fuzzy`) o *distancia de Mahalanobis* (`Mahalanobis`), entre otros. En una clasificación no supervisada, el analista no es necesario que conozca la zona de estudio; sólo tiene que especificar el número de clases y los algoritmos realizan las agrupaciones, únicamente basándose en la información cuantitativa de los datos. Algunos de estos algoritmos son *ISODATA*, *K-means*, *líder*, *modelo neural no supervisado* o *MaxiMin*. Esta información se identifica en la *Figura* con `Classificator_type`.

4.2.2.3. Modelado de meta-metadatos del servicio de mediación

Para que los agentes de mediación del sistema puedan llevar a cabo su labor en los procesos de búsqueda y recuperación de la información, utilizan un conjunto de metadatos obtenidos a partir de los metadatos más relevantes recogidos por los agentes de procesamiento en los *EIMs*, además de incorporar nuevos. Estos "meta-metadatos" son

los denominados *EIDs*. Cada agente de mediación cuenta con un repositorio global que almacena los *EIDs* generados y exportados por cada uno de los agentes de procesamiento asociados a él, que localmente también almacenan una copia. En el modelo conceptual representado en la *Figura 4.14* se puede observar cómo se ha reunido la información más destacada de ambos tipos de clasificaciones, la ecológica y la de imágenes de satélite, procedente de los *EIMs*, en la clase `Classification` y aparecen propiedades nuevas como el número de niveles jerárquicos (`number_of_levels`) —en el caso de una clasificación ecológica— o el número de imágenes de satélite (`number_of_satellite_images`) —en el caso de una clasificación de imágenes—, información que no aparece recogida explícitamente en los *EIMs*. Lo mismo sucede con el resto de clases, `Geography`, `Technician`, `Time`, `Layer`/`Variable` o `Satellite_image`/`Band`, que recogen la mínima información relevante para el proceso. También, las clases `Layer` y `Satellite_image` incluyen nuevas propiedades: el número de variables (`number_of_variables`) y el número de bandas (`number_of_bands`), respectivamente.

En [ACG, 2009] también se encuentra la representación completa de la ontología *EID* en notación *OWL/XML*.

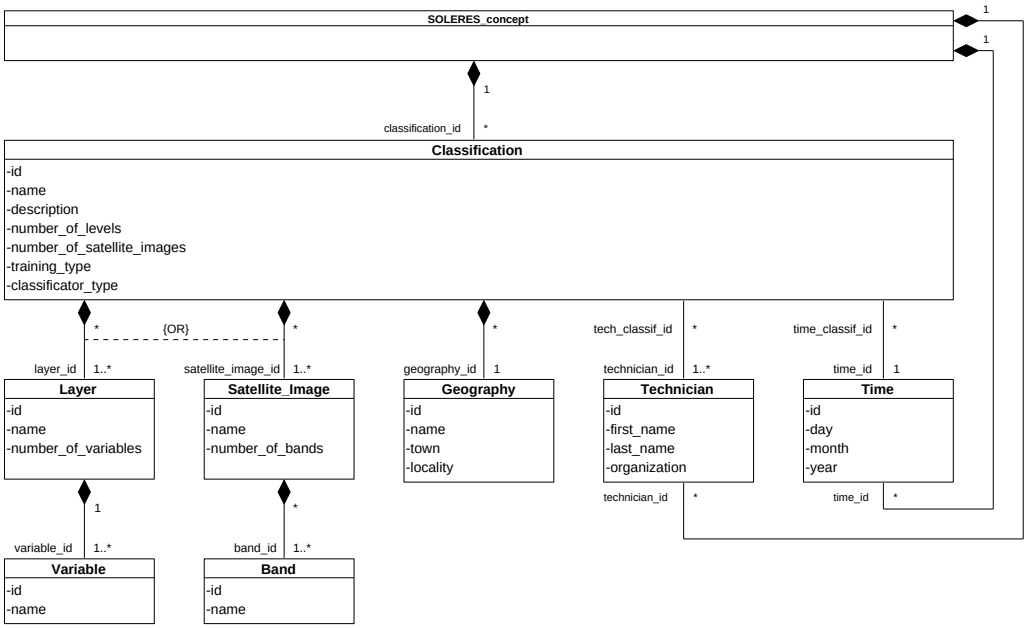


Figura 4.14: Modelado de los meta-metadatos del servicio de mediación.

4.3. Implementación de *OntoTrader* en *SOLERES-KRS*

En la *Sección 4.1* ya se anticipa que *SOLERES-KRS* va a estar formado por un conjunto distribuido de *EPUs* que trabajan de forma independiente en la gestión del conocimiento

del sistema, los *Mapas de Información Medioambiental* (*EIMs*) y sus correspondientes *metaDatos de Información Medioambiental* (*EIDs*). La cooperación entre ellas se ha logrado mediante el desarrollo de un servicio de mediación basado en la especificación *OntoTrader* e implementado con los agentes de mediación ("traders"). Se puede decir que estos agentes actúan de intermediarios (aunque de forma indirecta) entre los agentes de interfaz de *SOLERES-HCI* y los agentes de procesamiento o *EPUs* ("exportadores") para satisfacer las demandas de información de los usuarios. Para que los agentes de mediación puedan llevar a cabo su labor en la búsqueda de información, cada *EPU* exporta sus *EIDs* al agente de mediación con el que se encuentra asociada, de manera que éste cuenta con un repositorio global en el que almacena los *EIDs* de todas las *EPUs* asociadas a él. Los agentes de consulta poseen el rol de "importadores" y su función es la de obtener a través del agente de mediación la información demandada por los usuarios, bien con los meta-metadatos que almacena éste o bien con los metadatos que almacenan los agentes de procesamiento filtrados por el agente de mediación. El rol de "administrador" del servicio de mediación lo posee el agente de gestión, cuya función es la de definir, administrar y verificar que se cumplen las directivas del servicio; este agente, además, hace de intermediario directo con *SOLERES-HCI* y es el encargado de controlar el resto del entorno del subsistema planificando las tareas que se realizan.

Para definir los protocolos de comportamiento e interacción con los agentes de mediación a través de sus interfaces y, por extensión, con cualquier otro agente del sistema, se han implementado las tres ontologías de servicio/proceso [Asensio et al., 2012] especificadas en la *Sección 2.4*. Cuando un agente necesita de otro para realizar una acción específica, construye un mensaje en el que describe tal acción mediante una de estas ontologías y lo envía al segundo agente. Este último, una vez que ha recibido el mensaje, extrae el contenido de la ontología, realiza la acción adecuada y utiliza la misma ontología para indicarle el resultado de la acción al primero. Este proceso se repite con las diferentes acciones en las que intervienen más de un agente. La *Tabla 4.5* recoge la definición de la ontología *Lookup* en notación *OWL/XML*. En ella se aprecian tres clases principales —`Concept`, `Action` y `Predicate`— de las que heredan el resto; por ejemplo, `QueryForm` es una subclase de `Concept`, `Query` de `Action` o `QueryError` de `Predicate`. Para cada una de las propiedades de los conceptos se define el tipo (`type`), el dominio (`domain`) y el rango (`range`); así, la propiedad `queryFormURI`, cuyo dominio es el concepto `QueryForm`, tiene como tipo `DatatypeProperty` y como rango el tipo de dato `string`.

4.4. Despliegue de *SKRS* en un entorno real

Haciendo un breve repaso del *framework* de *TKRS* [Asensio et al., 2011b], una vez definidos el modelo de la arquitectura del sistema y el modelo de un repositorio de implementaciones, mediante el modelo de configuración del sistema y una serie de transformaciones *M2M* y *M2T* es posible llegar al despliegue del mismo en un entorno dado. Esto es precisamente lo que se plantea a continuación. El modelo de la arquitectura de *SOLERES-KRS* y el modelo del repositorio con la implementación de *TKRS* basada en agentes software se han definido en las *Secciones 4.3.1* y *3.3*, respectivamente.

```
1  <?xml version="1.0"?>
2
3  <!DOCTYPE rdf:RDF [
    <!ENTITY lookup "http://www.ual.es/acg/ont/TKRS/LookupOntology.owl#">
    <!ENTITY owl "http://www.w3.org/2002/07/owl#">
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
   ]>
4
5  <rdf:RDF xml:base = "&lookup;" xmlns = "&lookup;" xmlns:lookup = "&lookup;"
    xmlns:owl = "&owl;" xmlns:rdf = "&rdf;" xmlns:rdfs = "&rdfs;" xmlns:xsd = "&xsd;">
6    <owl:Ontology rdf:about="&lookup;"><rdfs:label>Lookup Ontology</rdfs:label></owl:Ontology>
7    <owl:Class rdf:ID="Concept"/>
8    <owl:Class rdf:ID="Action"/>
9    <owl:Class rdf:ID="Predicate"/>
10   <owl:Class rdf:ID="QueryForm"><rdfs:subClassOf rdf:resource="#Concept"/></owl:Class>
11     <owl:FunctionalProperty rdf:ID="queryFormURI">
12       <rdf:type rdf:resource="&owl;DatatypeProperty"/>
13       <rdfs:domain rdf:resource="#QueryForm"/>
14       <rdfs:range rdf:resource="&xsd;string"/>
15     </owl:FunctionalProperty>
16   <owl:Class rdf:ID="OfferSeq"><rdfs:subClassOf rdf:resource="#Concept"/></owl:Class>
17     <owl:FunctionalProperty rdf:ID="offerSeqURI">
18       <rdf:type rdf:resource="&owl;DatatypeProperty"/>
19       <rdfs:domain rdf:resource="#OfferSeq"/>
20       <rdfs:range rdf:resource="&xsd;string"/>
21     </owl:FunctionalProperty>
22   <owl:Class rdf:about="#Query"><rdfs:subClassOf rdf:resource="#Action"/></owl:Class>
23   <owl:Class rdf:ID="QueryError"><rdfs:subClassOf rdf:resource="#Predicate"/></owl:Class>
24   <owl:Class rdf:ID="EmptyOfferSeq"><rdfs:subClassOf rdf:resource="#Predicate"/></owl:Class>
25   <owl:Class rdf:ID="NotEmptyOfferSeq"><rdfs:subClassOf rdf:resource="#Predicate"/></owl:Class>
26   ...
27 </rdf:RDF>
```

Tabla 4.5: Definición de la ontología *Lookup* en *OWL/XML*.

Para poder referenciar sus elementos desde el modelo de configuración, previamente se ha ejecutado la correspondiente transformación *M2M* a cada uno de ellos, obteniéndose la nueva representación que aparece en las *Tablas 4.6 y 4.7* en base a la gramática desarrollada en la *Sección 3.4*.

En la *Tabla 4.6*, donde aparece el modelo de la arquitectura, se puede observar cómo la entidad `TKRS` va seguida del nombre del sistema, `SOLERES_KRS`, y contiene tres entidades `Node` correspondientes a los tres nodos del mismo (líneas #3 a #28, #29 a #48 y #49 a #61, respectivamente). Cada una de ellas va seguida del nombre del nodo (por ejemplo, el primero, "`Node_1`") y encierra entre "{}" los atributos (`ip`, `port`, etc.) seguidos de su valor (en el caso de la dirección ip del "`Node_1`", "`192.168.1.11`"). A continuación, aparecen los módulos que incluye (`ServiceModule`, `ManagementModule`, etc.) de nuevo seguidos de su nombre ("`ServiceModule_1_1`", "`ManagementModule_1_1`", etc.); también entre "{}" definen sus atributos correspondientes (aquellos módulos que posean) y, mediante la relación `hasNode`, indica el nodo al que pertenece. Finalmente, cada nodo hace referencia al sistema en el que opera mediante la relación `hasTKRS`. De forma similar, en la *Tabla 4.7* se encuentra el modelo del repositorio, donde para la platafor-

```
Package SOLERES
TKRS SOLERES_KRS {
  Node Node_1 {
    ip "192.168.1.11"
    port "1099"
    dbport "3306"
    ServiceModule ServiceModule_1_1 { hasNode Node_1 }
    ManagementModule ManagementModule_1_1 { hasNode Node_1 }
    QueryModule QueryModule_1_1 {
      usesLookupInterface TradingModule_1_1
      hasNode Node_1
    }
    TradingModule TradingModule_1_1 {
      usesLookupInterface true
      usesRegisterInterface true
      usesAdminInterface false
      usesLinkInterface true
      usesProxyInterface false
      isFederatedWith "Node_2.TradingModule_2_1"
      hasNode Node_1
    }
    ProcessingModule ProcessingModule_1_1 {
      ambient Ambient_1
      usesRegisterInterface "Node_2.TradingModule_2_1"
      hasNode Node_1
    }
 hasTKRS SOLERES_KRS
  }
  Node Node_2 {
    ip "192.168.1.12"
    port "1099"
    dbport "3306"
    ServiceModule ServiceModule_2_1 { hasNode Node_2 }
    ManagementModule ManagementModule_2_1 { hasNode Node_2 }
    QueryModule QueryModule_2_1 {
      usesLookupInterface TradingModule_2_1
      hasNode Node_2
    }
    TradingModule TradingModule_2_1 {
      usesLookupInterface true
      usesRegisterInterface true
      usesAdminInterface false
      usesLinkInterface true
      usesProxyInterface false
      hasNode Node_2
    }
    hasTKRS SOLERES_KRS
  }
  Node Node_3 {
    ip "192.168.1.13"
    port "1099"
    dbport "3306"
    ServiceModule ServiceModule_3_1 { hasNode Node_3 }
    ManagementModule ManagementModule_3_1 { hasNode Node_3 }
    QueryModule QueryModule_3_1
    {
      usesLookupInterface "Node_2.TradingModule_2_1"
      hasNode Node_3
    }
    hasTKRS SOLERES_KRS
  }
}
```

Tabla 4.6: Modelo de la arquitectura de *SOLERES-KRS* expresado según la gramática definida.

ma "`Java_JADE`" se especifican cinco `SimpleModules` (líneas #5 a #24), cada uno de ellos seguido de su nombre (por ejemplo, el primero, "`ServiceModuleImpl`") y definen el atributo `uri` ("`http://www.ual.es/acg/rep/TKRS/ServiceModule.class`"), así como la plataforma a la que pertenecen ("`Java_JADE`") a través de la relación `hasPlatform`.

```
1  Package UAL_Repository
2
3  ImplementationRepository {
4    Platform Java_JADE {
5      SimpleModule ServiceModuleImpl {
6        uri "http://www.ual.es/acg/rep/TKRS/ServiceModule.class"
7        hasPlatform Java_JADE
8      }
9      SimpleModule ManagementModuleImpl {
10       uri "http://www.ual.es/acg/rep/TKRS/ManagementModule.class"
11       hasPlatform Java_JADE
12     }
13     SimpleModule QueryModuleImpl {
14       uri "http://www.ual.es/acg/rep/TKRS/QueryModule.class"
15       hasPlatform Java_JADE
16     }
17     SimpleModule TradingModuleImpl {
18       uri "http://www.ual.es/acg/rep/TKRS/TradingModule.class"
19       hasPlatform Java_JADE
20     }
21     SimpleModule ProcessingModuleImpl {
22       uri "http://www.ual.es/acg/rep/TKRS/ProcessingModule.class"
23       hasPlatform Java_JADE
24     }
25   }
26 }
```

Tabla 4.7: Modelo del repositorio de *TKRS* expresado según la gramática definida.

Ahora sí, se puede definir el modelo de configuración de *SOLERES-KRS* (*Tabla 4.8*), donde mediante un conjunto de sentencias (`Statement`), se asocian los módulos que forman parte de cada nodo del sistema con su correspondiente implementación en el repositorio (según la arquitectura *PIM/PSM* definida). Por ejemplo, la primera de ellas (líneas #4 a #7) asocia el módulo "`SOLERES.KRS.Node_1.ServiceModule_1_1`" de la arquitectura (a través de la relación `hasTKRSModule`) con la implementación "`ACG_Repository.Java_JADE.ServiceModuleImpl`" (a través de la relación `hasImplementationRepositoryModule`) y así sucesivamente.

Finalmente, a partir de la definición de este modelo de configuración y de la transformación *M2T* presentada también en la *Sección 3.4*, se obtiene el código para el despliegue del sistema en un entorno real. Esta transformación genera varios archivos: por un lado, un archivo *script* (ver *Tabla 4.9*) con comandos del sistema operativo como "`mkdir`", "`cd`" o "`wget`" para crear una estructura de carpetas que albergue los archivos de implementación del sistema y donde se compila ("`javac TKRS.java`"), por otro lado, un conjunto de archivos *Java* (colocados en la carpeta correspondiente de la estructura), uno por cada nodo del sistema; en ellos se realiza tanto la configuración como la inicia-

```
Package SOLERES_Configuration

Configuration {
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_1.ServiceModule_1_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ServiceModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_1.ManagementModule_1_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ManagementModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_1.QueryModule_1_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.QueryModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_1.TradingModule_1_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.TradingModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_1.ProcessingModule_1_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ProcessingModuleImpl"
  }

  Statement {
    hasTKRSModule "SOLERES.KRS.Node_2.ServiceModule_2_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ServiceModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_2.ManagementModule_2_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ManagementModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_2.QueryModule_2_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.QueryModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_2.TradingModule_2_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.TradingModuleImpl"
  }

  Statement {
    hasTKRSModule "SOLERES.KRS.Node_3.ServiceModule_3_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ServiceModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_3.ManagementModule_3_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.ManagementModuleImpl"
  }
  Statement {
    hasTKRSModule "SOLERES.KRS.Node_3.QueryModule_3_1"
    hasImplementationRepositoryModule "ACG_Repository.Java_JADE.QueryModuleImpl"
  }
}
```

Tabla 4.8: Modelo de configuración de *SOLERES-KRS* expresado según la gramática definida.

```
 1  #!/bin/bash
 2  clear
 3  mkdir /SOLERES_KRS/Node_1/modules
 4  cd /SOLERES_KRS/Node_1/modules
 5  wget http://www.ual.es/acg/rep/TKRS/ServiceModule.class
 6  wget http://www.ual.es/acg/rep/TKRS/ManagementModule.class
 7  wget http://www.ual.es/acg/rep/TKRS/TradingModule.class
 8  wget http://www.ual.es/acg/rep/TKRS/QueryModule.class
 9  wget http://www.ual.es/acg/rep/TKRS/ProcessingModule.class
10  cd /SOLERES_KRS/Node_1
11  javac TKRS.java
12  mkdir /SOLERES_KRS/Node_2/modules
13  cd /SOLERES_KRS/Node_2/modules
14  wget http://www.ual.es/acg/rep/TKRS/ServiceModule.class
15  wget http://www.ual.es/acg/rep/TKRS/ManagementModule.class
16  wget http://www.ual.es/acg/rep/TKRS/TradingModule.class
17  wget http://www.ual.es/acg/rep/TKRS/QueryModule.class
18  cd /SOLERES_KRS/Node_2
19  javac TKRS.java
20  mkdir /SOLERES_KRS/Node_3/modules
21  cd /SOLERES_KRS/Node_3/modules
22  wget http://www.ual.es/acg/rep/TKRS/ServiceModule.class
23  wget http://www.ual.es/acg/rep/TKRS/ManagementModule.class
24  wget http://www.ual.es/acg/rep/TKRS/QueryModule.class
25  cd /SOLERES_KRS/Node_3
26  javac TKRS.java
27  cd /
```

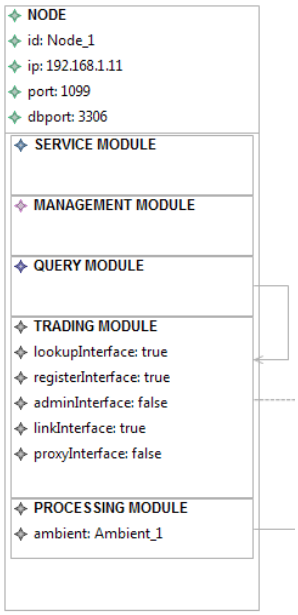
Tabla 4.9: Archivo *script* para el despliegue de *SOLERES-KRS*.

lización (en el constructor de la clase) de todos sus parámetros (atributos de la clase) a partir de los valores especificados en el diseño de la arquitectura.

Las *Tablas 4.10, 4.11 y 4.12* muestran el contenido de estos archivos *Java*, que se corresponden con la implementación de los correspondientes nodos definidos en la *Figura 4.4* (extraídos de ella y colocados junto a estas *Tablas* para clarificar). Por ejemplo, en la *Tabla 4.10* se puede apreciar la especificación de la clase `InformationSystem` —que forma parte del paquete `SOLERES_KRS.Node_1` (línea #1)—, donde se realiza la configuración del *Nodo 1* de *SOLERES-KRS*. En la línea #3 se importan las clases que implementan los diferentes módulos de *TKRS*. Los atributos se declaran en las líneas #7-#14 y entre ellos se encuentran tanto las propiedades del nodo (`ip`, `port`, etc.), como los módulos de *TKRS* que incluye (en este caso, todos); en el constructor de la clase (líneas #26-#25), se inicializan todos los atributos anteriores con los valores indicados mediante la herramienta gráfica, etc. Esto mismo se repite para los *Nodos 2* y *3* de la arquitectura del sistema.

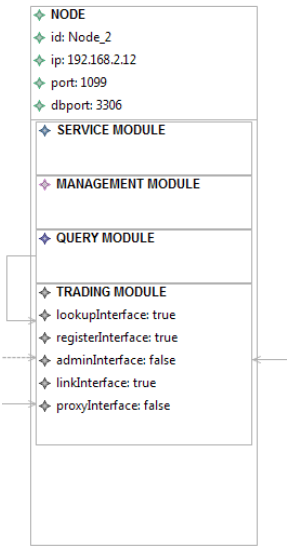
4.5. Evaluación y validación de la propuesta

Con el objetivo de evaluar el sistema, se ha desarrollado un entorno específico de experimentación, donde se pueden llevar a cabo diferentes tipos de pruebas, tanto de las tareas de gestión (inserción, modificación y eliminación), como de consulta de información, atendiendo a los modelos de reflexión, delegación y federación analizados en la



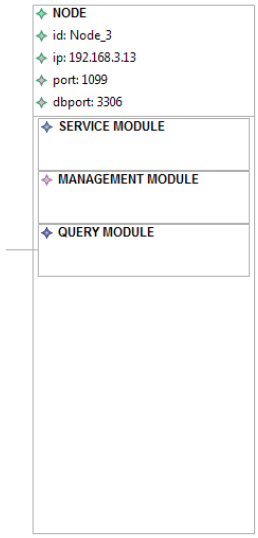

```
1  package SOLERES_KRS.Node_1;
2
3  import SOLERES_KRS.Node_1.modules.*;
4
5  public class InformationSystem {
6    private String ip = null;
7    private int port = -1;
8    private int dbport = -1;
9    private ServiceModule serviceModule = null;
10   private ManagementModule managementModule = null;
11   private QueryModule queryModule = null;
12   private TradingModule tradingModule = null;
13   private ProcessingModule processingModule = null;
14
15   public InformationSystem() {
16     this.ip = "192.168.1.11";
17     this.port = 1099;
18     this.dbport = 3306;
19     this.serviceModule = new ServiceModule();
20     this.managementModule = new ManagementModule();
21     this.queryModule = new QueryModule();
22     this.tradingModule = new TradingModule();
23     this.processingModule = new ProcessingModule();
24     // CODE
25   }
26   // CODE
27 }
```

Tabla 4.10: Archivo *Java* para la configuración del *Nodo 1* de *SOLERES-KRS*.




```
1  package SOLERES_KRS.Node_2;
2
3  import SOLERES_KRS.Node_2.modules.*;
4
5  public class InformationSystem {
6    private String ip = null;
7    private int port = -1;
8    private int dbport = -1;
9    private ServiceModule serviceModule = null;
10   private ManagementModule managementModule = null;
11   private QueryModule queryModule = null;
12   private TradingModule tradingModule = null;
13
14   public InformationSystem() {
15     this.ip = "192.168.1.12";
16     this.port = 1099;
17     this.dbport = 3306;
18     this.serviceModule = new ServiceModule();
19     this.managementModule = new ManagementModule();
20     this.queryModule = new QueryModule();
21     this.tradingModule = new TradingModule();
22     // CODE
23   }
24   // CODE
25 }
```

Tabla 4.11: Archivo *Java* para la configuración del *Nodo 2* de *SOLERES-KRS*.




```
1  package SOLERES_KRS.Node_3;
2
3  import SOLERES_KRS.Node_3.modules.*;
4
5  public class InformationSystem {
6    private String ip = null;
7    private int port = -1;
8    private int dbport = -1;
9    private ServiceModule serviceModule = null;
10   private ManagementModule managementModule = null;
11   private QueryModule queryModule = null;
12
13   public InformationSystem() {
14     this.ip = "192.168.1.13";
15     this.port = 1099;
16     this.dbport = 3306;
17     this.serviceModule = new ServiceModule();
18     this.managementModule = new ManagementModule();
19     this.queryModule = new QueryModule();
20     // CODE
21   }
22   // CODE
23 }
```

Tabla 4.12: Archivo *Java* para la configuración del *Nodo 3* de *SOLERES-KRS*.

Sección 3.2.2. En las siguientes *Secciones* se realiza una explicación detallada de estas tareas acompañada de ejemplos y se exponen los criterios y conclusiones para la validación de la propuesta.

4.5.1. Escenario de gestión

El escenario de gestión tiene lugar siempre que un usuario del sistema realiza una inserción, modificación o eliminación de un *EIM* en una *EPU*. En este escenario intervienen un agente de interfaz, un agente de gestión, un agente de procesamiento y un agente de mediación. Las comunicaciones que se establecen entre estos tres últimos agentes se realizan haciendo uso de la ontología *Register*. A continuación se van a detallar, de forma genérica para estas tres acciones, los pasos concretos que se establecen en el proceso y que también aparecen reflejados en el diagrama de secuencia de la *Figura 4.15*:

— Secuencia #1. Para comenzar, un usuario inicia el proceso de inserción, modificación o eliminación de un *EIM* a través de la interfaz gráfica del sistema, preparando y seleccionando toda la información necesaria. El agente de interfaz interviene activamente adaptando la interfaz a las necesidades y preferencias del usuario, y asesorándole en su labor.

— Secuencia #2–#4. Con toda la información disponible, el agente de interfaz construye un mensaje y lo traslada al agente de gestión correspondiente mediante un mensaje que indica la acción a realizar (inserción, modificación o eliminación) y dónde debe realizarse, es decir, la *EPU* o agente de procesamiento concreto.

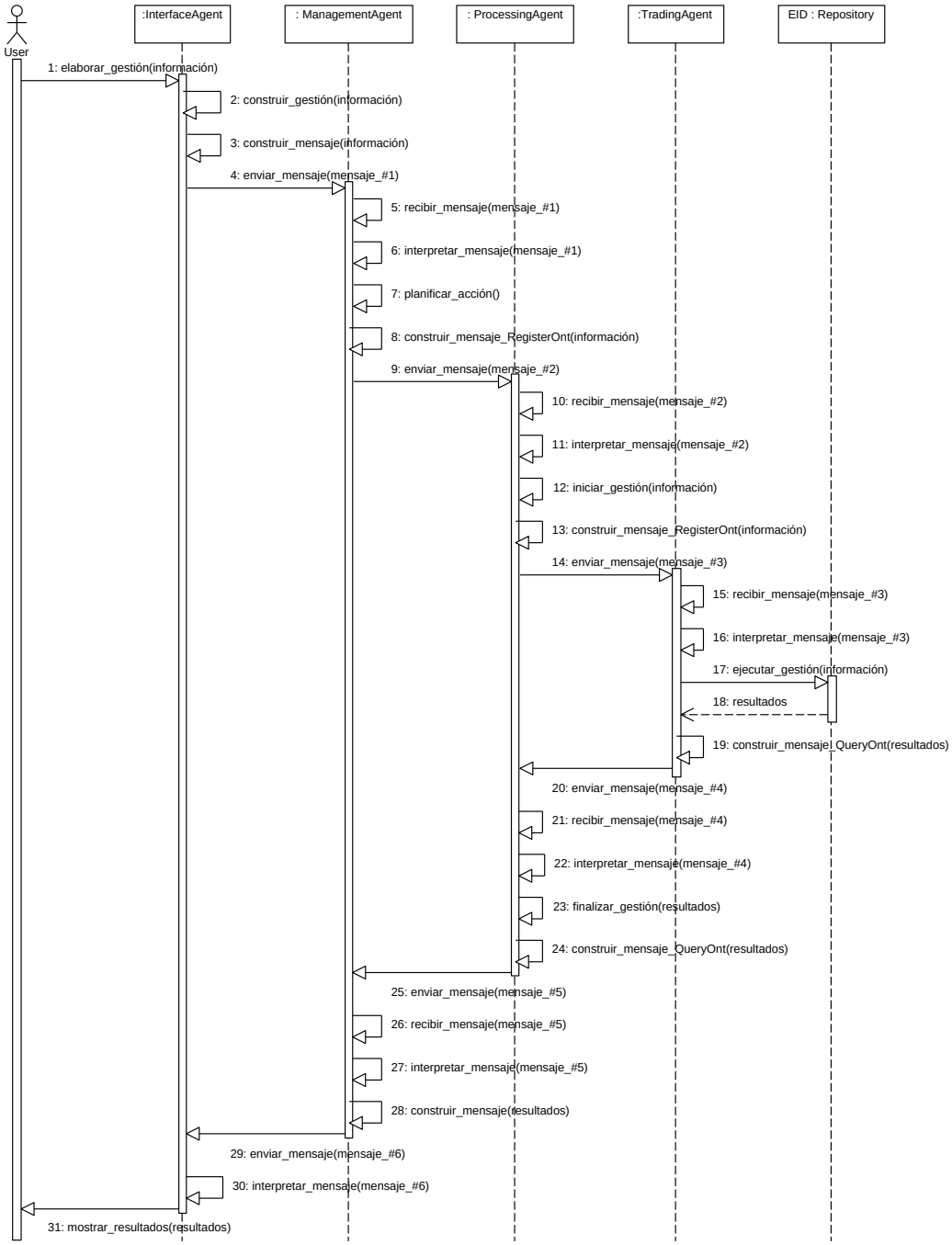


Figura 4.15: Diagrama de secuencia del escenario de gestión.

— Secuencia #5–#9. El agente de gestión recibe el mensaje procedente del agente de interfaz, extrae su contenido, lo interpreta y planifica la acción. Entonces, genera un nuevo mensaje utilizando la ontología *Register* para indicarle al agente de procesamiento implicado la tarea que tiene que realizar, incluyendo la información que necesita, y se lo envía.

— Secuencia #10–#14. Ahora, el agente de procesamiento destinatario del mensaje lo recibe y extrae su contenido. Comprueba la tarea que debe realizar y la pone en marcha, siempre realizando las operaciones necesarias sobre su repositorio de *EIMs* en primer lugar y, seguidamente, trasladando los cambios a su repositorio de *EIDs*. Si se produjera cualquier error en el transcurso de la tarea, ésta quedaría anulada y se continuaría directamente por el paso #23. Los cambios tienen que ser trasladados también al agente de mediación con el que se encuentra asociado el agente de procesamiento para mantener la consistencia de la información del sistema. Por eso, genera un mensaje utilizando la ontología *Register* con los cambios efectuados en su repositorio de *EIDs* y lo envía.

— Secuencia #15–#20. El agente de mediación extrae el contenido del mensaje tras recibirlo y ejecuta la acción correspondiente sobre su interfaz *Register*. Una vez finalizada, empaqueta el resultado de la misma en un nuevo mensaje construido en base a la ontología *Register* que envía de vuelta al agente de procesamiento.

— Secuencia #21–#25. El agente de procesamiento recibe el mensaje, finaliza la acción y envía el resultado global de la acción en un nuevo mensaje, también generado utilizando la ontología *Register*, al agente de gestión que se la encomendó.

— Secuencia #26–#28. El agente de gestión traslada este mensaje con el resultado al agente de interfaz que se comunicó inicialmente con él.

— Secuencia #29–#31. Y, finalmente, el agente de interfaz muestra el resultado al usuario.

Las *Tablas 4.13 y 4.14* muestran a modo de ejemplo un fragmento del contenido de los mensajes de solicitud/respuesta intercambiados entre el agente de gestión y el agente de procesamiento, definidos mediante la ontología *Register*, en el caso particular de la inserción de un nuevo *EIM*. En ambos mensajes aparece reflejada la acción implicada, `Export`. En el primero se hace uso del concepto `Offer`, donde se especifica la localización del *EIM* con el atributo `uri`. En el segundo mensaje, se indica el resultado satisfactorio de la acción mediante el predicado `ExportedOffer` y se emplea un concepto adicional, `OfferId`, donde se especifica el identificador del *EIM* insertado.

4.5.2. Escenarios de consulta

En el proceso de búsqueda y recuperación de información, la configuración del modelo de tres niveles $< \mathcal{O}, \mathcal{M}, \mathcal{I} >$ de *TKRS* queda estructurada de la siguiente forma con la implementación basada en agentes de *SOLERES*: en el *Nivel 1* (*N1*) ($\mathcal{O}$) los agentes de interfaz (*SOLERES-HCI*) preparan las consultas realizadas por los usuarios antes de

```
1 ...
2 <Offer rdf:about=\"&register;OfferInstance\">
3   <offerURI rdf:datatype=\"&xsd;string\">http://www.ual.es/acg/tmp/SKRS/offer001.owl</offerURI>
4 </Offer>
5 <Export rdf:about=\"&register;ExportInstance\"/>
6 ...
```

Tabla 4.13: Fragmento de un mensaje de solicitud utilizando la ontología *Register*.

```
1 ...
2 <OfferId rdf:about=\"&lookup;OfferIdInstance\">
3   <offerIdId rdf:datatype=\"&xsd;long\">1</offerIdId>
4 </OfferId>
5 <ExportedOffer rdf:about=\"&register;ExportedOfferInstance\"/>
6 <Export rdf:about=\"&register;ExportInstance\"/>
7 ...
```

Tabla 4.14: Fragmento de un mensaje de respuesta utilizando la ontología *Register*.

enviarlas a los agentes de gestión (*SOLERES-KRS*); en el *Nivel 2* (*N2*) ($\mathcal{M}$) intervienen los agentes de consulta y los agentes de mediación para localizar y/o recuperar la información demandada; y, por último, en el *Nivel 3* (*N3*) ($\mathcal{I}$) operan los agentes de procesamiento para recuperar la información no disponible en el nivel anterior.

A modo de recordatorio, aunque se pueden plantear múltiples escenarios a la hora de procesar una consulta realizada por un usuario, todos ellos resultan como una combinación de tres escenarios principales, que tienen que ver con los tres modelos de mediación analizados en el *Capítulo 2*:

— Escenario de **reflexión**. La consulta del usuario se puede resolver directamente por un agente de mediación sin la intervención de ningún agente de procesamiento.

— Escenario de **delegación**. La consulta del usuario no se puede resolver directamente por un agente de mediación y requiere la intervención de uno o más agentes de procesamiento.

— Escenario de **federación**. La consulta del usuario se puede resolver por un agente de mediación federado con otro u otros agentes de mediación, con o sin la intervención de uno o más agentes de procesamiento.

La *Figura 4.16* resume los tres escenarios básicos definidos y los agentes que intervienen en cada uno de ellos. Las líneas de trazo discontinuo indican el orden de las comunicaciones que establecen los diferentes agentes.

En general, la secuencia de acciones que tienen lugar en los tres escenarios es muy similar. Todo comienza cuando un usuario realiza una consulta a través de la interfaz. Un agente de interfaz es el encargado de traducirla al lenguaje de consulta *SPARQL*

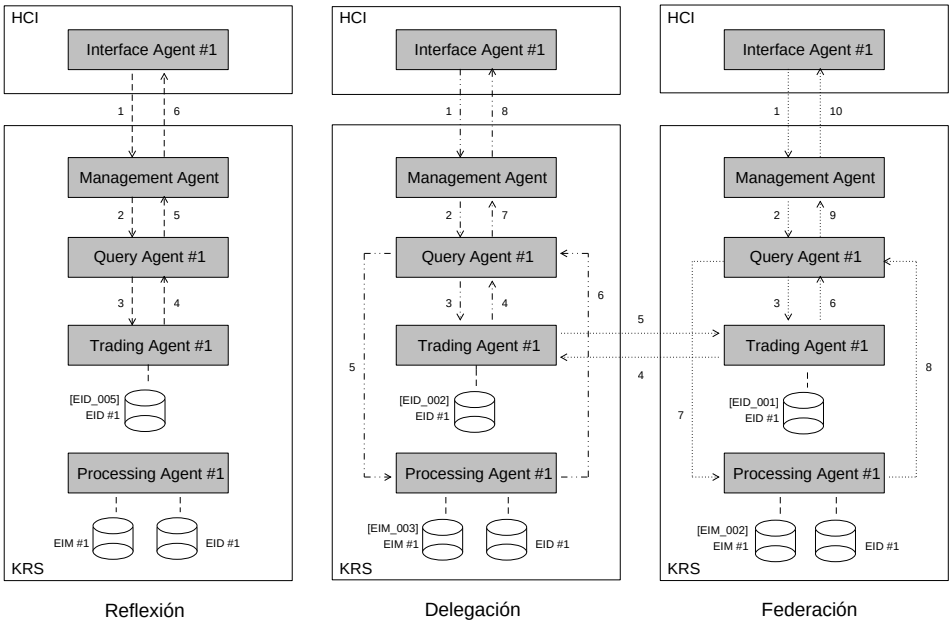


Figura 4.16: Escenarios de consulta principales: reflexión, delegación y federación.

(*SPARQL Protocol and RDF Query Language*) utilizado en el sistema y trasladarla a un agente de gestión. Este agente lo único que hace es planificar la acción y transferir la consulta a un agente de consulta, que es el encargado de decidir qué modelo seguir (reflexión, delegación o federación) basándose en la información demandada. Mientras las solicitudes que se envían entre los agentes se construyen con mensajes del tipo "`QUERY-REF`" en los que se embebe la consulta en *SPARQL*, los resultados devueltos por los agentes se construyen usando mensajes "`INFORM-REF`", como se puede apreciar en las siguientes *Secciones*, donde se detalla cada uno de los escenarios planteados.

4.5.2.1. Escenario de reflexión

La *Figura 4.17* muestra el diagrama de secuencia del escenario de **reflexión**. Este escenario tiene lugar cuando algún usuario del sistema realiza una consulta del tipo "*Necesitamos obtener el nombre de las variables utilizadas en las clasificaciones realizadas durante el año 2008*", y cuyo resultado se puede obtener directamente de los *EIDs* que almacena en su repositorio un agente de mediación. Como se puede observar en el diagrama, una vez formulada la consulta por el usuario a través de la interfaz gráfica, un agente de interfaz la codifica en el lenguaje de consulta *SPARQL* y genera un mensaje `QUERY-REF`, el mensaje #1/secuencia #4, que envía a un agente de gestión. Este mensaje se encuentra redactado en texto plano (*Tabla 4.15*).

Un mensaje siempre tiene cuatro cláusulas básicas:

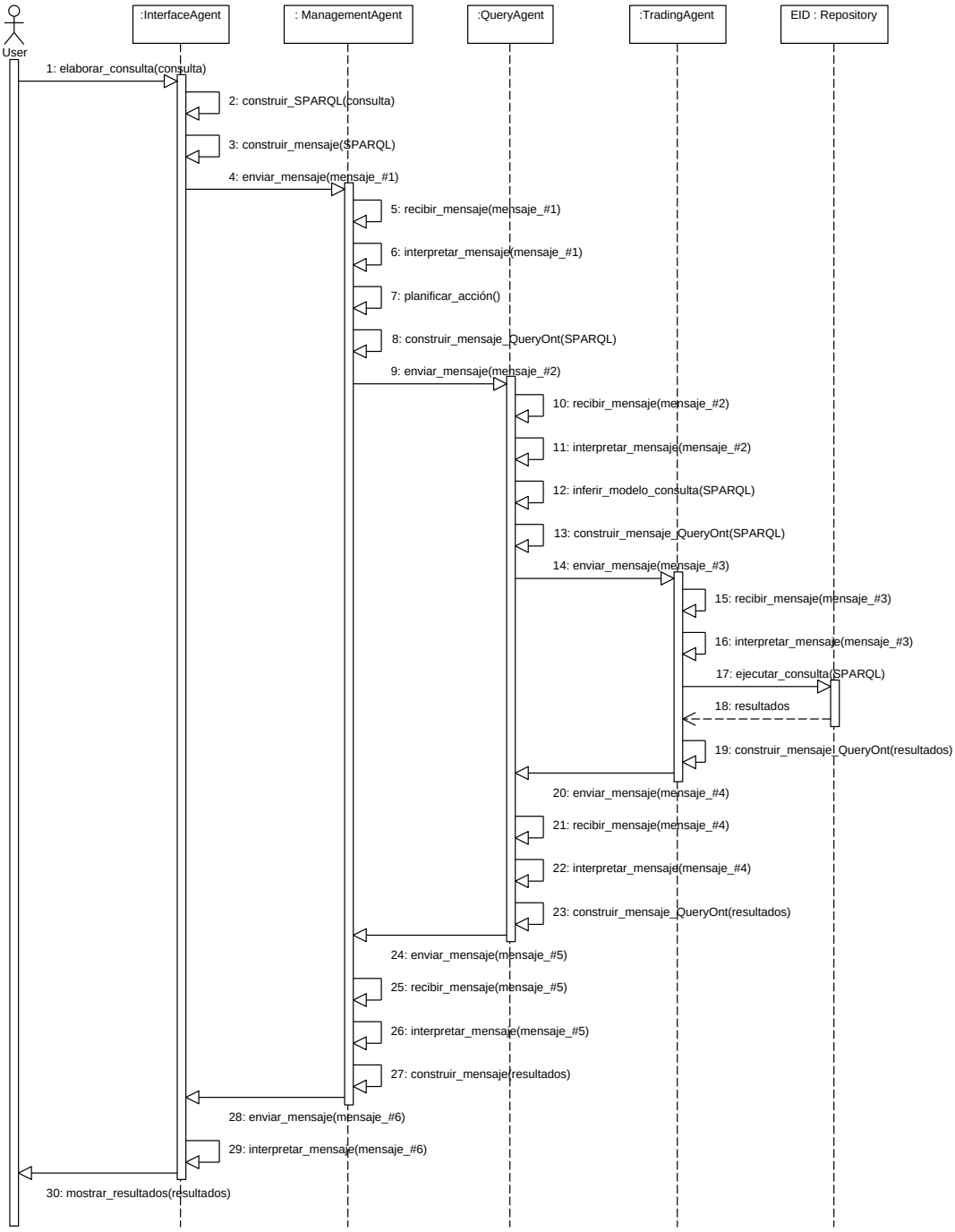


Figura 4.17: Diagrama de secuencia del escenario de reflexión.

```
1  (QUERY-REF
2    :sender  ( agent-identifier :name InterfaceAgent@SOLERES-KRS
      :addresses (sequence http://skrs-02.home.es:7778/acc ))
3    :receiver  (set ( agent-identifier :name ManagementAgent@SOLERES-KRS ) )
4    :content  "PREFIX eid: <http://www.ual.es/acg/ont/SKRS/EIDOntology.owl#>
5     PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
6     SELECT DISTINCT ?EID__VariableName
7     WHERE
8     {
9      ?EID__EID eid:EID_eimId ?EID__EID_eimID .
10     ?EID__EID eid:EID_hasClassification ?EID__Classification .
11     ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
12     ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
13     ?EID__Variable eid:Variable_name ?EID__VariableName .
14     ?EID__Classification eid:Classification_hasTime ?EID__Time .
15     ?EID__Time eid:Time_year ?EID__TimeYear .
16     FILTER (?EID__TimeYear = 2008) .
17    }
18    ORDER BY ASC(?EID__VariableName)"
19   :language  SPARQL )
```

Tabla 4.15: Escenario de reflexión: mensaje #1 enviado por el agente de interfaz al agente de gestión.

— `:sender` (línea #2), donde se indica el nombre del agente que envía el mensaje (en este ejemplo, el agente de interfaz);

— `:receiver` (línea #3), donde se indica el nombre del agente destinatario del mensaje (el agente de gestión);

— `:content` (líneas #4 a #17), que incluye el contenido del mensaje (la consulta en *SPARQL*);

— `:language`, donde se especifica el tipo de lenguaje utilizado para escribir el contenido del mensaje (en este caso, *SPARQL*).

Una vez que el agente de gestión interpreta que se trata de una consulta de datos, reenvía dicha petición a un agente de consulta. Para ello, elabora el mensaje #2/secuencia #9 (*Tabla 4.16*) (se han omitido ciertos fragmentos del contenido del mensaje para reducir su longitud). El contenido del mensaje (líneas #4–#17) aparece redactado en formato *RDF* utilizando la ontología *Lookup* y se hace uso del par "concepto/acción" para formular la consulta. Concretamente, se utilizan el concepto `QueryForm` (líneas #8–#15), donde se especifica la *URI* del archivo que contiene la consulta (en *SPARQL*) y la acción a realizar, `Query` (línea #16). Como en el mensaje anterior, la cláusula `:language` indica la ontología utilizada (línea #18). Por último, una nueva cláusula, `:ontology` (línea #19), especifica la *URI* en la que se encuentra dicha ontología.

Al recibir este mensaje, el agente de consulta analiza la sentencia de la consulta y la contrasta con el esquema de metadatos disponible en el repositorio del agente de mediación, infiriendo si puede ser resuelta o no directamente por este último (modelo de reflexión). En el caso concreto del ejemplo, todos los datos solicitados se encuentran

disponibles en el agente de mediación, por lo que genera un nuevo mensaje `QUERY-REF` similar al anterior utilizando la ontología *Lookup*, el mensaje #3/secuencia #14, y lo envía al agente de mediación con el que se encuentra asociado. Este último, lanza la consulta en *SPARQL* contra su repositorio (secuencia #17), obteniendo los resultados. Estos resultados son los que envía de vuelta en un mensaje `INFORM-REF` (mensaje #4/secuencia #20) (*Tabla 4.17*), en este caso, al agente de consulta.

```
1  (QUERY-REF
2    :sender  ( agent-identifier :name ManagementAgent@SOLERES-KRS
     :addresses (sequence http://skrs-02.home.es:7778/acc ))
3    :receiver  (set ( agent-identifier :name QueryAgent@SOLERES-KRS )
     ( agent-identifier :name InterfaceAgent@SOLERES-KRS ) )
4    :content  "...
5     <rdf:RDF
      xml:base = "&lookup;"
      xmlns = "&lookup;"
      xmlns:lookup = "&lookup;"
      xmlns:owl = "&owl;"
      xmlns:rdf = "&rdf;"
      xmlns:rdfs = "&rdfs;"
      xmlns:xsd = "&xsd;">
6      <owl:Ontology rdf:about=\"&lookup;\"/>
7      ...
8      <QueryForm rdf:about=\"&lookup;QueryFormInstance\">
9        ...
10       <queryFormURI rdf:datatype=\"&xsd;string\">
11         http://www.ual.es/acg/tmp/SKRS/queryForm001.sparql

12           <!-- queryForm001.sparql
              PREFIX eid: <http://www.ual.es/acg/ont/SKRS/EIDOntology.owl#>
              PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
              SELECT DISTINCT ?EID__VariableName
              WHERE
              {
               ?EID__EID eid:EID_eimId ?EID__EID_eimID .
               ?EID__EID eid:EID_hasClassification ?EID__Classification .
               ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
               ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
               ?EID__Variable eid:Variable_name ?EID__VariableName .
               ?EID__Classification eid:Classification_hasTime ?EID__Time .
               ?EID__Time eid:Time_year ?EID__TimeYear .
               FILTER (?EID__TimeYear = 2008) .
              }
              ORDER BY ASC(?EID__VariableName)
           -->

13       </queryFormURI>
14       ...
15     </QueryForm>
16     <Query rdf:about=\"&lookup;QueryInstance\"/>
17    </rdf:RDF>"
18   :language  "Query Ontology"
19   :ontology  http://www.ual.es/acg/ont/TKRS/LookupOntology.owl )
```

Tabla 4.16: Escenario de reflexión: mensaje #2 enviado por el agente de gestión al agente de consulta.

```
(INFORM-REF
 :sender  ( agent-identifier :name TradingAgent@SOLERES-KRS
  :addresses (sequence http://skrs-02.home.es:7778/acc ))
 :receiver  (set ( agent-identifier :name QueryAgent@SOLERES-KRS ) )
 :content  "...
  <rdf:RDF
   xml:base = "&lookup;"
   xmlns = "&lookup;"
   xmlns:lookup = "&lookup;"
   xmlns:owl = "&owl;"
   xmlns:rdf = "&rdf;"
   xmlns:rdfs = "&rdfs;"
   xmlns:xsd = "&xsd;">
    <owl:Ontology rdf:about=\"&lookup;\"/>
    ...
    <OfferSeq rdf:about=\"&lookup;OfferSeqInstance\">
      ...
      <offerSeqURI rdf:datatype=\"&xsd;string\">
        http://www.ual.es/acg/tmp/SKRS/offerSeq001.xml

          <!-- offerSeq001.xml

           <?xml version="1.0"?>
           <sparql xmlns="http://www.w3.org/2005/sparql-results#">
             <head>
               <variable name="EID__VariableName"/>
             </head>
             <results>
               <result>
                 <binding name="EID__VariableName">
                   <literal datatype="http://www.w3.org/2001/XMLSchema#string">C1</literal>
                 </binding>
               </result>
               <result>
                 <binding name="EID__VariableName">
                   <literal datatype="http://www.w3.org/2001/XMLSchema#string">C1L1</literal>
                 </binding>
               </result>
               ...
             </results>
           </sparql>

          -->

      </offerSeqURI>
      ...
    </OfferSeq>
    <NotEmptyOfferSeq rdf:about=\"&lookup;NotEmptyOfferSeqInstance\"/>
    <Query rdf:about=\"&lookup;QueryInstance\"/>
  </rdf:RDF>"
 :language  "Query Ontology"
 :ontology  http://www.ual.es/acg/ont/TKRS/LookupOntology.owl )
```

Tabla 4.17: Escenario de reflexión: mensaje #4 enviado por el agente de mediación al agente de consulta.

Para construir este mensaje se utiliza la serie "concepto/acción/predicado" de la ontología *Lookup*. El concepto `OfferSeq` (líneas #8–#15) muestra los resultados encontrados tras procesar la consulta, la acción continúa siendo `Query` (línea #17) y el predicado `NotEmptyOfferSeq` (línea #16) muestra que la consulta devuelve algún resultado. Se puede observar que el concepto `OfferSeq` contiene la *URI* de un documento en formato *XML* estándar (línea #11), cuyo contenido se muestra comentado en la línea #12. Si la consulta no devolviese ningún resultado o se hubiese producido algún error, no se utilizaría el concepto `OfferSeq` y, en su lugar, se utilizaría el predicado `EmptyOfferSeq` o algún predicado de la ontología que indicase el error concreto, respectivamente.

Entonces, cuando el agente de consulta recibe los resultados procedentes del agente de mediación, los transfiere directamente al agente de gestión, construyendo un nuevo mensaje `INFORM-REF` y modificando las cabeceras `:sender` y `:receiver` (mensaje #5/secuencia #24). El agente de gestión realiza la misma operación nuevamente, construyendo el mensaje de respuesta dirigido al agente de interfaz (mensaje #6/secuencia #28), que lo decodifica y muestra los resultados en la interfaz de usuario, finalizando así la secuencia completa de este escenario.

4.5.2.2. Escenario de delegación

La *Figura 4.18* muestra el diagrama de secuencia para el escenario de **delegación**. En este escenario, la consulta del usuario no se puede resolver directamente por un agente de mediación y requiere la intervención de uno o más agentes de procesamiento para completarla. Se da cuando se realiza una consulta del tipo "*Necesitamos obtener el valor mínimo de la variable $\mathcal{X}$ utilizado sobre la capa de información $\mathcal{Y}$*". En este caso, el resultado no se puede obtener directamente de los meta-metadatos que almacena en su repositorio el agente de mediación y es necesario consultar los metadatos de uno o más agentes de procesamiento.

Se puede observar que la secuencia entre el agente de interfaz, el agente de gestión y el agente de consulta es la misma que para el escenario de reflexión. La consulta anterior genera la sentencia en *SPARQL* de la *Tabla 4.18*, que se transmite en los mensajes `QUERY-REF` (para simplificar, sólo se muestra la consulta). En este ejemplo, se han seleccionado la variable "E6" y la capa "Edaphic sectors".

Al recibir el mensaje, el agente de consulta infiere que el tipo de consulta es mediante delegación, por lo que fracciona la consulta original (secuencia #13) preparando una doble consulta. La primera de ellas es la que aparece en la *Tabla 4.19* y sigue su curso hacia el agente de mediación (secuencia #8). El objetivo de esta consulta consiste en localizar en el repositorio de *EIDs* del agente de mediación los *EIMs* almacenados en los repositorios de los agentes de procesamiento en los que se hace referencia a la variable indicada. Después de consultar su repositorio *EID* asociado, el agente de mediación devuelve al agente de consulta el mensaje que aparece en la *Tabla 4.20* (mensaje #4/secuencia #21).

Este mensaje indica que el agente de mediación ha localizado la variable "E6" en el *EIM* con identificador "EIM_0000000003", codificada como "VAR_0000000202". Una vez recibido por el agente de consulta, éste genera un nuevo mensaje utilizando la ontología *Lookup* con la segunda consulta, en la que aplica un filtro a la consulta original con

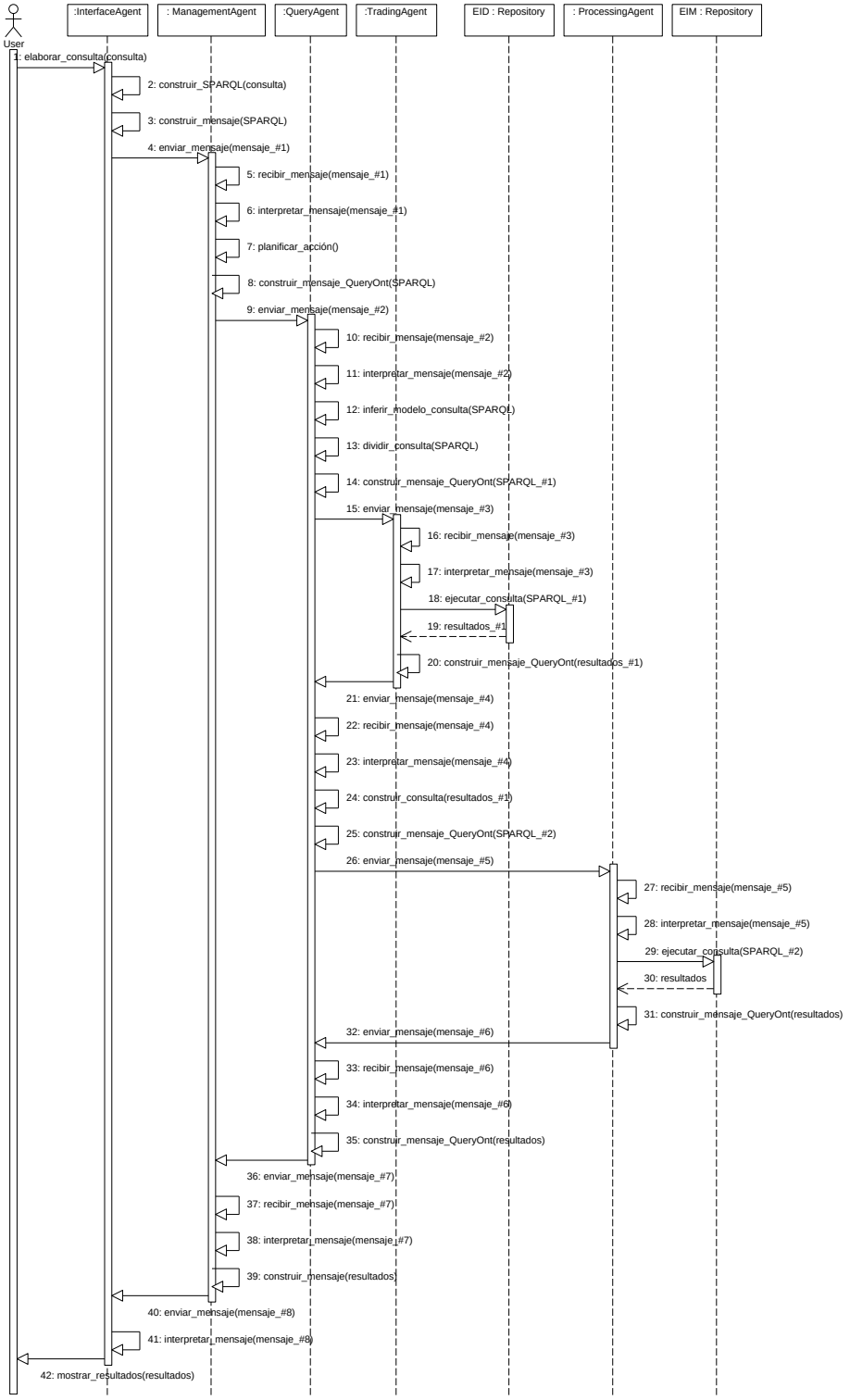


Figura 4.18: Diagrama de secuencia del escenario de delegación.

los datos aportados por la consulta anterior; específicamente, los resultados se filtran para aquellos documentos que contienen la variable llamada "VAR_0000000202". Este mensaje se envía al agente o agentes de procesamiento correspondientes, para que puedan completar la consulta. El nuevo mensaje es el que aparece en la *Tabla 4.21*.

Finalmente, la secuencia concluye con la devolución de los resultados por parte de los agentes hasta que los recibe el agente de interfaz, que se encarga de mostrarlos al usuario a través de la interfaz gráfica.

```
PREFIX eid: <http://www.ual.es/acg/ont/SKRS/EIDOntology.owl#>
PREFIX eim: <http://www.ual.es/acg/ont/SKRS/EIMOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?EIM__VarMin
WHERE
{
 ?EID__EID eid:EID_eimId ?EID__EID_eimID .
 ?EID__EID eid:EID_hasClassification ?EID__Classification .
 ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
 ?EID__Layer eid:Layer_name ?EID__LayerName .
 ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
 ?EID__Variable eid:Variable_id ?EID__VariableId .
 ?EID__Variable eid:Variable_name ?EID__VariableName .
 ?EIM__Var eim:Variable_id ?EID__VariableId .
 ?EIM__Var eim:Variable_name ?EIM__VariableName .
 ?EIM__Var eim:Variable_minimumValue ?EIM__VarMin .
 FILTER (regex(?EID__LayerName, "Edaphic sectors")) .
 FILTER (regex(?EID__VariableName, "E6")) .
}
```

Tabla 4.18: Consulta en *SPARQL* utilizada en el escenario de delegación.

```
PREFIX eid: <http://www.ual.es/acg/ont/SKRS/EIDOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?EID__EID_eimID ?EID__VariableId
WHERE
{
 ?EID__EID eid:EID_eimId ?EID__EID_eimID .
 ?EID__EID eid:EID_hasClassification ?EID__Classification .
 ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
 ?EID__Layer eid:Layer_name ?EID__LayerName .
 ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
 ?EID__Variable eid:Variable_id ?EID__VariableId .
 ?EID__Variable eid:Variable_name ?EID__VariableName .
 FILTER (regex(?EID__LayerName, \"Edaphic sectors\")) .
 FILTER (regex(?EID__VariableName, \"E6\")) .
}
ORDER BY ASC(?EID__EID_eimID)
```

Tabla 4.19: Consulta en *SPARQL* enviada al agente de mediación en el escenario de delegación.

```
(INFORM-REF
 :sender  ( agent-identifier :name TradingAgent@SOLERES-KRS
  :addresses (sequence http://tkrs.ual.es:7778/acc ))
 :receiver  (set ( agent-identifier :name QueryAgent@SOLERES-KRS ) )
 :content  "...
  <rdf:RDF xml:base = "&lookup;" xmlns = "&lookup;" xmlns:lookup = "&lookup;" xmlns:owl = "&owl;"
   xmlns:rdf = "&rdf;" xmlns:rdfs = "&rdfs;" xmlns:xsd = "&xsd;">
    <owl:Ontology rdf:about=\"&lookup;\"/>
    ...
    <OfferSeq rdf:about=\"&lookup;OfferSeqInstance\">
      ...
      <offerSeqURI rdf:datatype=\"&xsd;string\">
        http://www.ual.es/acg/tmp/SKRS/offerSeq002.xml
          <!-- offerSeq002.xml
           <?xml version=\"1.0\"?>
           <sparql xmlns=\"http://www.w3.org/2005/sparql-results#\">
             <head><variable name=\"EID__EID_eimID\"/>
             <variable name=\"EID__VariableId\"/></head>
             <results><result>
               <binding name=\"EID__EID_eimID\"><literal ...>EIM_0000000003</literal></binding>
               <binding name=\"EID__VariableId\"><literal ...>VAR_0000000202</literal></binding>
             </result></results>
           </sparql>
          -->
      </offerSeqURI>
      ...
    </OfferSeq>
    <Query rdf:about=\"&lookup;QueryInstance\"/>
    <NotEmptyOfferSeq rdf:about=\"&lookup;NotEmptyOfferSeqInstance\"/>
  </rdf:RDF>"
 :language  "Query Ontology"
 :ontology  http://www.ual.es/acg/ont/TKRS/LookupOntology.owl )
```

Tabla 4.20: Escenario de delegación: mensaje #4 enviado por el agente de mediación al agente de consulta.

4.5.2.3. Escenario de federación

Como se ha mencionado, el escenario de **federación** surge cuando el sistema ha sido configurado para que un agente de mediación pueda propagar la consulta a un segundo agente federado si el primero no puede satisfacerla. En este modelo, se pueden dar diferentes combinaciones. Una situación es aquélla en la que el segundo *trader* puede satisfacer la consulta directamente, lo que es un caso de "federación/reflexión", es decir, un modelo de federación en el primer agente de mediación y, a continuación, reflexión en el segundo. Otra situación se presenta cuando el segundo agente permite redirigir la consulta a una fuente de datos (agente de procesamiento); en este caso, se da un caso de "federación/delegación", es decir, federación en el primer agente de mediación y delegación a un agente de procesamiento. También se podría dar el caso de "múltiple federación", donde el sistema se ha configurado para que las consultas se propaguen entre agentes de mediación federados si no las pueden satisfacer.

La *Figura 4.19* muestra el diagrama de secuencia que sigue este escenario. Para simplificarlo, el diagrama presenta un caso de "federación/reflexión", ya que el funcionamiento es similar en el resto de combinaciones. Un posible ejemplo de consulta en

este escenario podría ser "*Necesitamos obtener el identificador de las clasificaciones de la ciudad $\mathcal{X}$ en las que existe información sobre las variables $\mathcal{Y}$ y $\mathcal{Z}$*". Su resultado se puede obtener directamente de los meta-metadatos que almacenan en sus repositorios los agentes de mediación. Para este ejemplo, se asume que la variable $\mathcal{Y}$ es un metadato del repositorio de un agente de mediación y la variable $\mathcal{Z}$ se encuentra en el repositorio de otro agente. Ahora, la consulta en *SPARQL* generada es la que aparece en la *Tabla 4.22*, mientras que la secuencia de mensajes `QUERY-REF` y sus correspondientes `INFORM-REF` intercambiados por los agentes del sistema es similar a las analizadas en los escenarios de reflexión y delegación.

```
1  (QUERY-REF
2   :sender  ( agent-identifier :name QueryAgent@SOLERES-KRS
     :addresses (sequence http://tkrs.ual.es:7778/acc ))
3   :receiver  (set ( agent-identifier :name ProcessingAgent@SOLERES-KRS ) )
4   :content  "...
5    <rdf:RDF
      xml:base = "&lookup;"
      xmlns = "&lookup;"
      xmlns:lookup = "&lookup;"
      xmlns:owl = "&owl;"
      xmlns:rdf = "&rdf;"
      xmlns:rdfs = "&rdfs;"
      xmlns:xsd = "&xsd;">
6      <owl:Ontology rdf:about=\"&lookup;\"/>
7      ...
8      <QueryForm rdf:about=\"&lookup;QueryFormInstance\">
9        ...
10       <queryFormURI rdf:datatype=\"&xsd;string\">
11         http://www.ual.es/acg/tmp/SKRS/queryForm004.sparql

12           <!-- queryForm004.sparql
              PREFIX eim: <http://www.ual.es/acg/ont/SKRS/EIMOntology.owl#>
              PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
              SELECT DISTINCT ?EIM__VarMin
              WHERE
              {
               ?EIM__Var eim:Variable_id ?EIM__VarId .
               ?EIM__Var eim:Variable_name ?EIM__VariableName .
               ?EIM__Var eim:Variable_minimumValue ?EIM__VarMin .
               FILTER (?EIM__VarId = "VAR_0000000202") .
              }
             -->

13       </queryFormURI>
14       ...
15     </QueryForm>
16     <Query rdf:about=\"&lookup;QueryInstance\"/>
17   </rdf:RDF>"
18  :language  "Query Ontology"
19  :ontology  http://www.ual.es/acg/ont/TKRS/LookupOntology.owl )
```

Tabla 4.21: Escenario de delegación: mensaje #5 enviado por el agente de consulta al agente de procesamiento.

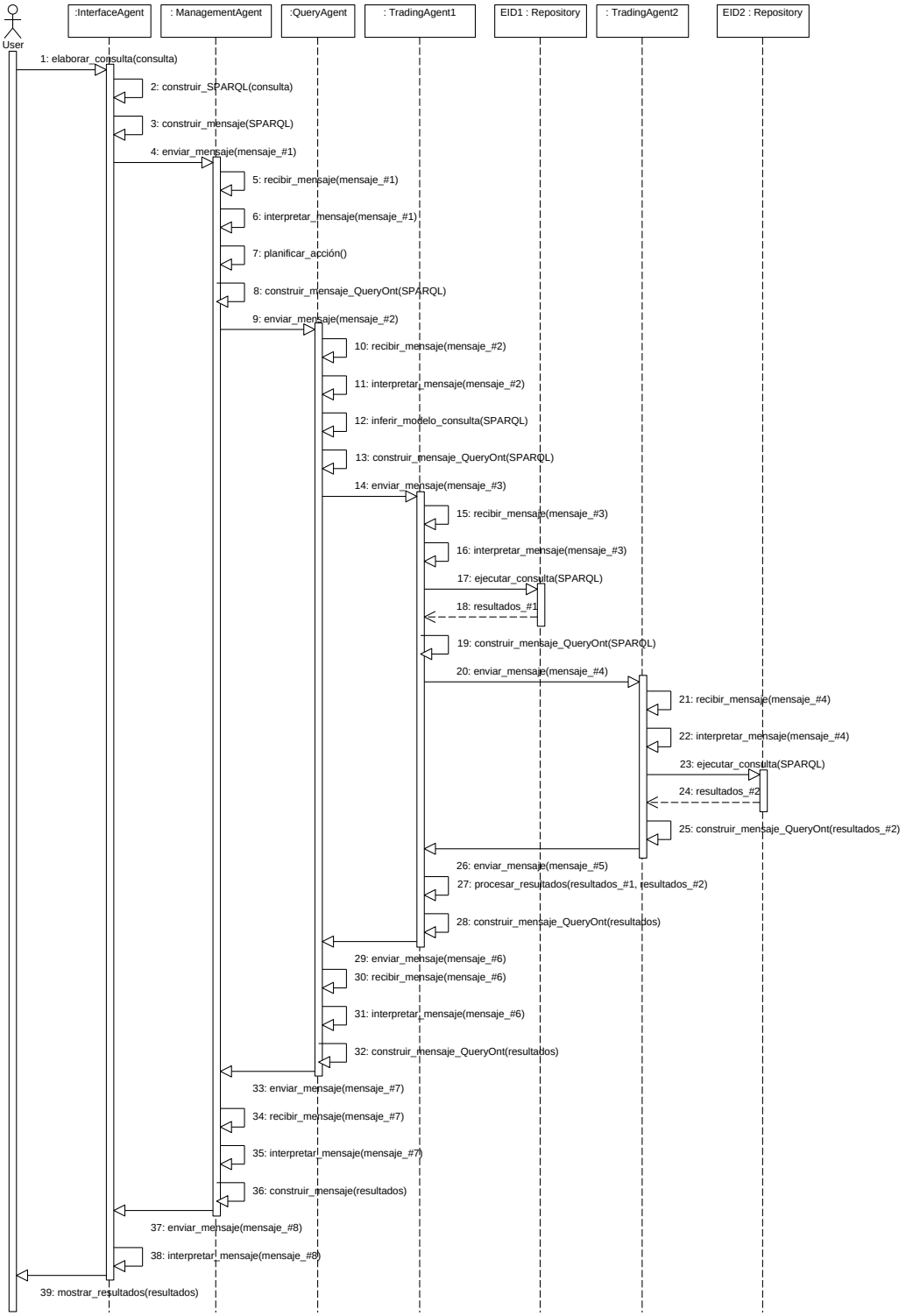


Figura 4.19: Diagrama de secuencia del escenario de federación.

```
PREFIX eid: <http://www.ual.es/acg/ont/SKRS/EIDOntology.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?EID__ClassificationId
WHERE
{
 ?EID__EID eid:EID_eimId ?EID__EID_eimID .
 ?EID__EID eid:EID_hasClassification ?EID__Classification .
 ?EID__Classification eid:Classification_id ?EID__ClassificationId .
 ?EID__Classification eid:Classification_hasGeography ?EID__Geography .
 ?EID__Geography eid:Geography_town ?EID__GeographyTown .
 ?EID__Classification eid:Classification_hasLayer ?EID__Layer .
 ?EID__Layer eid:Layer_hasVariable ?EID__Variable .
 ?EID__Variable eid:Variable_name ?EID__VariableName .
 ?EID__Variable eid:Variable_id ?EID__VariableId .
 FILTER (regex(?EID__GeographyTown, "Granada and Almeria")) .
 FILTER (regex(?EID__VariableName, "^C") || regex(?EID__VariableName, "^E")) .
}
ORDER BY ASC(?EID__ClassificationId)"
```

Tabla 4.22: Consulta en *SPARQL* utilizada en el escenario de federación.

4.5.3. Validación del sistema

Como punto de partida para la validación del sistema, se establecen determinadas "restricciones" que se pueden evaluar en diferentes áreas del *framework* de *TKRS*. La *Figura 4.20* muestra un esquema del desarrollo de *TKRS*. Como se puede observar, las *características* principales son el uso del **servicio de mediación**, de los **agentes software** y de las **ontologías**, cada una de las cuales presenta una serie de restricciones que hacen posible que el sistema implementado posea el funcionamiento deseado (validación). También se han impuesto otra serie de restricciones a los **datos de entrada** y **de salida**, como se verá a continuación. En total, se analiza un conjunto de cinco características.

Por un lado, en el caso de la función de mediación, el servicio se ha diseñado e implementado siguiendo la **especificación *RM-ODP*** (*primera característica*), lo que ha permitido desarrollar un sistema de mediación con las funciones consideradas; el estándar de mediación *ODP* aparece descrito en la *Sección 1.5*. Por otro lado, los diferentes objetos de sistema (incluyendo el mediador) se han implementado como agentes software utilizando la **plataforma *JADE*** (*segunda característica*), lo que permite validar su comportamiento, así como su correcto funcionamiento; además, *JADE* es un *framework* que cumple con la especificación *FIPA* (*Foundation for Intelligent Physical Agents*), una organización de estándares de la *IEEE Computer Society* que promueve la tecnología basada en agentes y la interoperabilidad de sus estándares con otras tecnologías.

La *tercera característica* clave en el desarrollo de *TKRS* es el uso de ontologías. Su diseño se presupone correcto por dos razones: la primera se refiere a que las ontologías de servicio/proceso se han desarrollado siguiendo la especificación de *JADE* para el intercambio de mensajes entre los agentes y bajo la especificación de las interfaces del *trader ODP*; la segunda razón se fundamenta en que las ontologías de datos se han diseñado a partir de cuestionarios rellenados por los expertos que han realizado las clasificaciones de datos bajo el marco del proyecto de investigación [ACG, 2010]. En

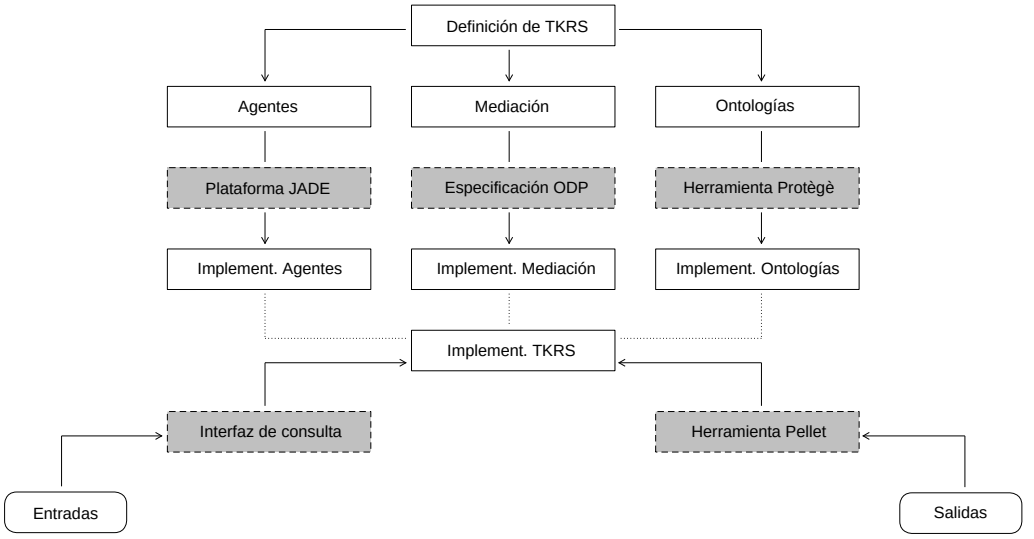


Figura 4.20: Restricciones que permiten verificar la implementación de *TKRS*.

base a estas premisas, todas las ontologías se han implementado y verificado mediante la **herramienta Protègè**, una herramienta para el diseño de ontologías estándar.

Como se ha mencionado, en la implementación de *TKRS*, también se han impuesto una serie de restricciones tanto a los datos de entrada como de salida. Las restricciones de entrada (*cuarta característica*) permiten comprobar que las consultas que se ejecutan en el sistema están correctamente construidas, se pueden traducir a *SPARQL* y ejecutar en el sistema. Estas restricciones se han establecido en una **interfaz** que guía al usuario en la construcción de la consulta. La interfaz se encuentra disponible en el portal web desarrollado como entorno de pruebas para la validación y evaluación de la propuesta [ACG, 2012][1], concretamente en la sección "*Complex Query*", mostrada en la *Figura 4.21*. Esta sección se encuentra dividida en dos partes: la parte izquierda, donde se puede seleccionar tanto un repositorio de *EIMs* como de *EIDs* del entorno o especificar la *URL* donde se encuentre, sobre el que se va a realizar la consulta, y la parte derecha, donde aparece una tabla con tres pestañas, "*Query*", "*Result*" y "*SPARQL*". La consulta se construye en la primera de ellas seleccionando las propiedades y estableciendo los filtros. Las propiedades (que aparecen en la columna izquierda) se pueden incorporar tanto a la cláusula `SELECT` como `WHERE` de la consulta realizando la selección correspondiente en la columna central. El proceso para añadir nuevas expresiones a la cláusula `WHERE` es el siguiente: (1) en primer lugar, se debe seleccionar una propiedad; (2) tras esto, en la columna central aparecerán una serie de operadores (como se puede observar en la *Figura*) para establecer el filtro y, después de seleccionar uno de ellos, (3) se debe especificar el valor de la propiedad. En la columna derecha aparece la representación en forma de árbol de la consulta. Una vez ejecutada, el resultado se obtiene en la pestaña

[1]Entorno de pruebas de *SOLERES*: `http://tkrs.ual.es/SKRS/`

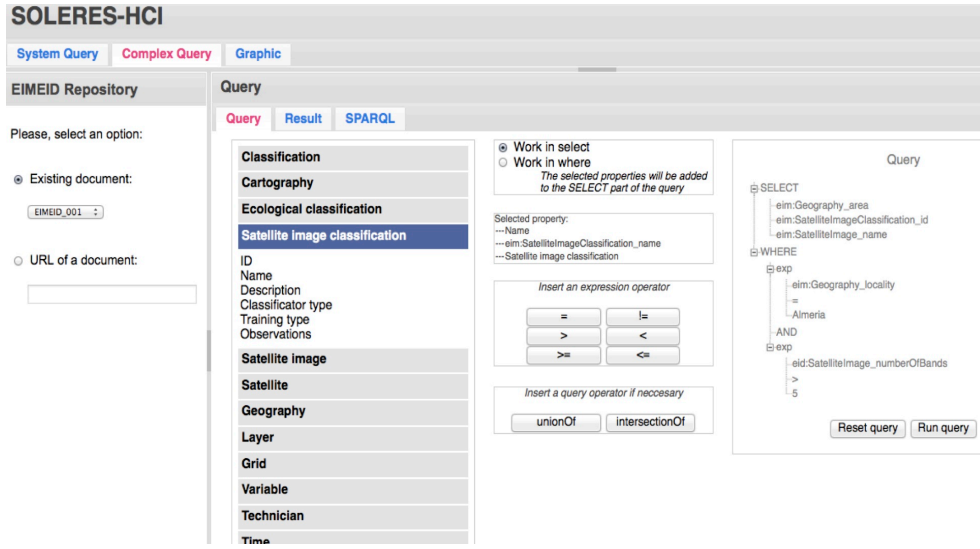


Figura 4.21: Parte de la sección "Complex Query" del entorno de pruebas.

"*Result*" expresado en formato *XML*, mientras que en la pestaña "*SPARQL*" se puede observar la consulta expresada en este lenguaje.

El entorno posee además otras dos secciones: "*System Query*" y "*Graphic*". En la sección "*System Query*" (también dividida en dos partes y cuyo diseño se muestra en la *Figura 4.22*), hay disponibles una serie de capacidades o consultas predefinidas ("*Capabilities*") sobre la información del sistema, donde se pueden ejecutar y probar los diferentes escenarios planteados. Al seleccionar una de ellas en la parte izquierda, se despliega un formulario para rellenar los parámetros necesarios antes de su ejecución; como se puede observar, en este caso se ha seleccionado una consulta para "obtener el nombre de las variables de una determinada capa utilizadas en las clasificaciones de una ciudad concreta", siendo preciso especificar tanto la capa como la ciudad. Después de ejecutar la consulta, aparece cierta información en las pestañas de la tabla situada en la parte derecha, "*Result*", "*SPARQL*" y "*Trace*": en la primera de ellas, se obtiene el resultado en formato *XML*; en la segunda, aparece la consulta expresada en *SPARQL*; y en la tercera, se muestra una traza de todos los mensajes intercambiados por los agentes, que permite comprobar y validar su funcionamiento y correcta implementación. Los mensajes pueden ser desplegados para su inspección. Así por ejemplo, en la *Figura 4.23* se puede observar que el usuario ha desplegado el mensaje `QUERY-REF` (generado por el agente `InterfaceAgent` y enviado al `IMIAgent`) para inspeccionar su resultado. Por último, la sección "*Graphic*", permite visualizar gráficamente en forma de árbol los repositorios de información utilizados.

Tanto en la sección "*Complex Query*", como en la sección "*System Query*", una vez que el sistema devuelve los resultados de las consultas (en las respectivas pestañas "*Result*"), se puede verificar que son correctos mediante una **herramienta externa**

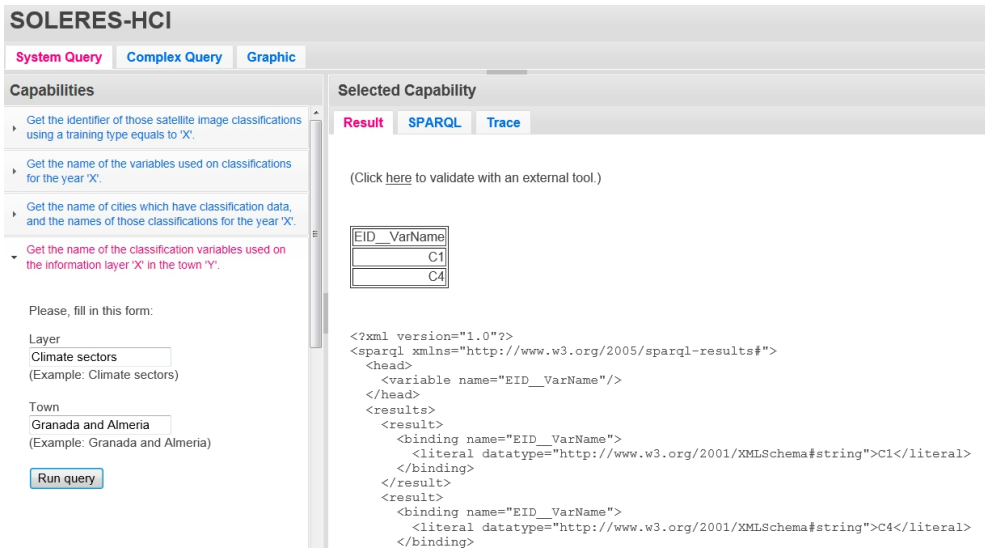


Figura 4.22: Parte de la sección "System Query" del entorno de pruebas.

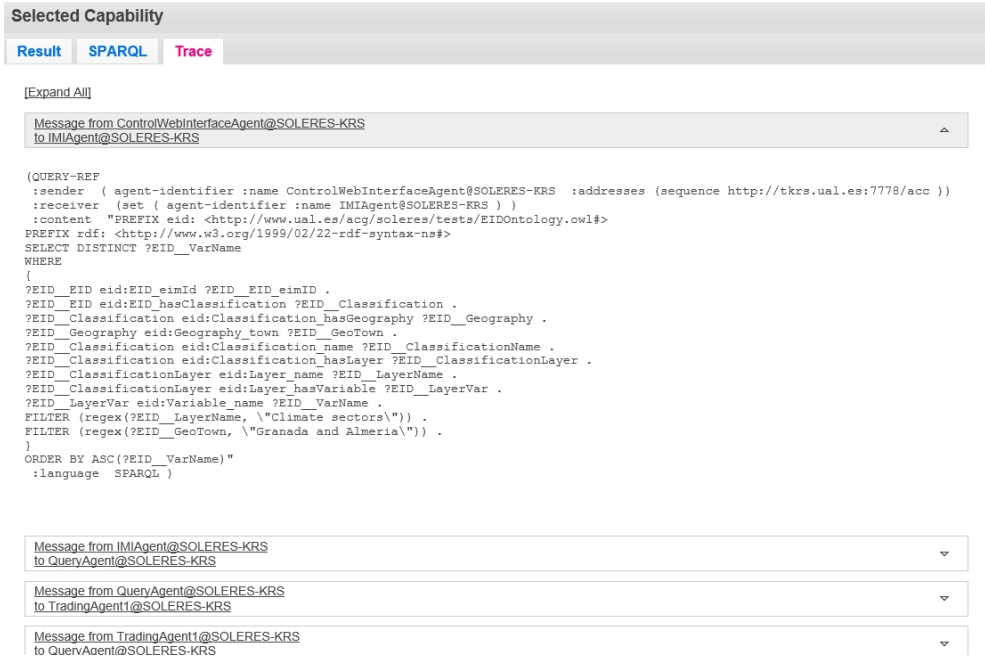


Figura 4.23: Parte de la traza de los mensajes intercambiados por los agentes.

[SWRG, 2004] (*quinta característica*), enlazada desde este entorno de pruebas, a la que se envían las consultas con las referencias a las ontologías de datos utilizadas y se pueden obtener los datos que debe devolver el sistema; esta herramienta también comprueba la validez de la ontología, advirtiendo si existe algún tipo de error en ella.

Con el análisis de estas cinco características quedan validados tanto la implementación de *TKRS* como el modelo de recuperación de datos *Query-Searching/Recovering-Response* (*QS/RR*) propuestos.

4.6. Trabajos relacionados

Como era de esperar, el avance de las tecnologías ha influido positivamente en el diseño y desarrollo de los *WIS*. Tecnologías como los servicios de mediación, los agentes software y las ontologías se han incorporado a ellos brindando un amplio abanico de posibilidades. Dado que el caso de estudio analizado en este *Capítulo* tiene como dominio el medio ambiente, se ha realizado una revisión un tanto exhaustiva de los sistemas que, en este ámbito, han sido diseñados en base a ellas [Asensio et al., 2007]. Todos se pueden catalogar en una de estas tres categorías: gestión de información medioambiental, asistencia en la toma de decisiones para la resolución de problemas medioambientales y simulación. En la *Tabla 4.23* aparece una comparativa de los sistemas estudiados en relación al sistema *SOLERES*. Se han tenido en cuenta características como: (a) la utilización de mecanismos de mediación, (b) el uso de modelado y diseño de ontologías, (c) la incorporación de algún tipo de agente de interfaz o similar, y (d) la tecnología empleada para su implementación. Como se puede apreciar, ninguno de los sistemas, salvo uno, utiliza un servicio de mediación como tal (implementado parcialmente), ni tampoco todos emplean las ontologías ni algún tipo de agente de interfaz que medie por el usuario, a diferencia del sistema *SOLERES*, que sí cumple estas tres características.

Sistema	**Mediación**	**Ontologías**	**Ag. usuario**	**Tecnologías**	**Dominio**
InfoSleuth	No	Sí	Sí	*XML/RDF*, *KQML*, *OKBC*	Recursos del agua
EDEN-IW	No	Sí	Sí	*JADE*, *DAML+OIL*	Recursos del agua
NZDIS	No	No	Sí	*CORBA/OQL*, *MOF*	Datos medioambientales
FSEP	No	No	No	*JACK*	Meteorología
MAGIC	No	No	Sí	*FIPA-ACL*, *CORBA*	Tratamiento del agua
DIAMON	No	No	Sí	*Java/C++*, *FIPA-ACL*	Tratamiento del agua
BUSTER	Sí	Sí	No	*OIL*, *FIPA-OS*	Información geográfica
SOLERES	Sí	Sí	Sí	*JADE*, *OWL*, *SPARQL*, *UML*	Ecología

Tabla 4.23: Comparativa de las arquitecturas de *EMIS*.

De forma independiente, existen abundantes ejemplos de la aplicación de las ontologías en sistemas en los que es fundamental atender a la semántica de la información. Por ejemplo, en [Ceccaroni et al., 2004], los autores presentan un sistema de apoyo a la toma de decisiones medioambientales denominado *OntoWEDSS*, donde se hace uso de

una ontología para modelar el proceso de tratamiento de aguas residuales, proporcionando un vocabulario común y una conceptualización explícita. Se puede encontrar otro ejemplo en [Di Lecce et al., 2004], donde un sistema de monitorización de la calidad del aire utiliza una ontología para definir mensajes y acciones de comunicación de forma concisa y sin ambigüedades. En [Brilhante, 2004], los autores presentan *Ecolingua*, una ontología de la familia *EngMath* para la representación de datos ecológicos cuantitativos. Todos estos ejemplos muestran el uso de las ontologías para construir modelos que describen las entidades en un dominio dado y caracterizar las relaciones y restricciones asociadas a ellas.

En [Zhan et al., 2007], los autores presentan una ontología para representar datos geográficos y funciones relacionadas con ellos. Para satisfacer la necesidad de un *GIS* interoperable, en [An and Zhao, 2007] los autores proponen el diseño de un modelo de *Geo Ontología* para integrar información geográfica. También, se ha explorado la aplicación de las ontologías en el campo de la recuperación de información geográfica [Song et al., 2007]. Aparece un uso distinto en [Huang, 2008], donde una extensión de *OWL* cuenta con nuevas primitivas para modelar localizaciones y relaciones espaciales con una ontología geográfica. También, han aparecido extensiones de ontologías existentes en este dominio de conocimiento: en [Shen et al., 2008], los autores proponen una ontología geográfica basada en *GML* (*Geography Markup Language*) [OGC, 2007a] y se extiende el perfil *OWL-S* a perfiles geográficos. Otro caso es una extensión de las ontologías *SWEET* (*Semantic Web for Earth and Environmental Terminology*) de la *NASA*, que incluye parte del dominio de la hidrogeología [Tripathi and Babaie, 2008].

Como se ha indicado al comienzo de este *Capítulo*, para el modelado de las diferentes ontologías se ha utilizado un enfoque orientado a objetos, seleccionando dos representaciones: una visual con *UML* y otra textual mediante *OWL*. En la literatura, existen numerosos estudios de modelado medioambiental que también hacen uso de esta aproximación (algunos de ellos aparecen resumidos en la *Tabla 4.24*).

Algunos investigadores sugieren el uso de *UML* como lenguaje para la representación gráfica del modelado ontológico [Cranefield and Purvis, 1999] [Falkovych et al., 2003] [Kogut et al., 2002], señalándolo como un recurso para el intercambio de información entre los especialistas fácil de leer e interpretar. En algunos casos, los modelos *UML* necesitan ser transformados en representaciones ontológicas usando herramientas específicas de transformación de modelos. En [Gasevic et al., 2004], el autor define transformaciones *XSLT* para construir una ontología en formato *XMI* a partir de un modelo lógico en *UML*. Otro ejemplo de transformación de modelos es *OUP* (*Ontology UML Profile*) [Djuric et al., 2005], que permite la utilización de los estándares *ODM/MDA*; también emplea *XSLT* para transformar ontologías *OUP* en *OWL*. En [Gronmo et al., 2002], los autores generan modelos *GML* a partir de modelos *UML* conceptuales que cumplen con la norma *ISO 19118*. *ONTOMET* [Bermudez, 2004] es un *framework* basado en *UML* para comunidades de información geoespacial que proporciona un entorno flexible y permite la interoperación de especificaciones de metadatos formales, vocabularios del dominio y extensiones. *GSIP* (*GEOINT Structure Implementation Profile*) y *GEOUML* son dos enfoques de modelado *UML* basados en el estándar *ISO/TC211*: *GSIP* [OGC, 2007b] utiliza un modelo de datos *UML* basado en los modelos de datos *ISO/TC211* para características, geometría espacial, topología, tiempo y cobertura;

Nombre	Referencia	Aprox. OO.	Tecnologías	Organización	Dominio
Gasevic *et al.*	[Gasevic et al., 2004]	*UML*	*XSLT*, transf. modelos	Univ. Belgrado, Serbia y Montenegro	Ontología general
Pan *et al.*	[Pan et al., 2006]	*UML*	Transf. modelos	*IBM China Research Lab*, Beijing, China	Ontología general
OUP	[Djuric et al., 2005]	*UML* y *OWL*	*ODM*, *MDA*, *XSLT*	Univ. Belgrado, Serbia y Montenegro	Ontología general
Gronmo *et al.*	[Gronmo et al., 2002]	*UML*	*GML*, *ISO 19188*	*SINTEF Telecom & Informatics*, Noruega	Info. geográfica
ONTOMET	[Bermudez, 2004]	*UML* y *OWL*	*ISO 19115*	Univ. Drexel, Filadelfia, PA, USA	Info. hidrológica y geoespacial
GSIP	[OGC, 2007b]	*UML* y *OWL*	*ISO/TC211*	*National Geospatial-Intelligence Agency*	Info. geoespacio-temporal
GEOUML	[Belussi et al., 2004]	*UML*	*ISO/TC211*, *ISO 19100**	Dept. Informática, Univ. Verona, Italia	GIS
Geographical-Space	[Wang et al., 2007]	*UML*	Razonamiento semántico	Univ. Wuhan, China	Info. geoespacial
GeoOWL	[Lieberman et al., 2007]	*OWL*	*ISO/TC211*	*W3C*	Info. geográfica
HYMAPS-OWL	[ABED, 2010]	*OWL*	Sistemas de apoyo a la decisión	Univ. Purdue, USA	Info. ecológica e hidrológica

Tabla 4.24: Enfoques *UML* y *OWL* para modelado medioambiental.

GEOUML [Belussi et al., 2004] es un marco *UML* para el modelado conceptual de *GIS* basado en algunos estándares *ISO 19100**.

Por otro lado, *OWL* [W3C, 2004a] es un lenguaje estándar del *W3C* que sigue una filosofía orientada a objetos para definir ontologías web, aunque hoy en día, está ampliamente extendido a cualquier dominio. Para su funcionamiento, *OWL* se apoya en otros lenguajes como *XML*, *RDF*, y *RDF-S* (*RDF-Schema*). También existen en la literatura abundantes enfoques basados en *OWL* para definir los modelos ontológicos en los *EMIS*. En particular, es preciso destacar *HYMAPS-OWL* [ABED, 2010] y *ETHAN* [Project Spire, 2003] ya que están aplicados a dominios ecológicos (más relacionados con el dominio del sistema *SOLERES*). El primero de ellos consta de tres aplicaciones diferentes para modelado hidrológico; facilita capas con información espacial en un mapa, que incluyen hidrología del suelo, carreteras, ferrocarriles, límites, corrientes y lagos, dando cobertura a todos los Estados Unidos, excepto Alaska. Las ontologías *ETHAN* se centran en taxonomías biológicas y características asociadas de historia natural.

4.7. Resumen y conclusiones

A lo largo de este *Capítulo* se ha detallado el diseño y la implementación de *SOLERES-KRS*, el subsistema responsable de la gestión del conocimiento en *SOLERES*, un sistema enmarcado en el ámbito de la ecología, concretamente en el estudio de mapas cartográficos, imágenes de satélite y sus clasificaciones. Para su construcción se han seguido las directrices marcadas en el *Capítulo 3* para la construcción de *Sistemas de Representación del Conocimiento basados en Mediación* (*TKRS*) y seleccionado la tecnología de agentes software. Dada la arquitectura de tres niveles de información del sistema, se han

modelado dos ontologías de datos, la ontología *EIM* para representar los metadatos de la información medioambiental, y la ontología *EID* para representar los meta-metadatos, utilizados por los agentes de mediación, que implementan el modelo *OntoTrader*. La comunicación y coordinación de los diferentes agentes también se ha realizado mediante las tres ontologías de servicio/proceso detalladas en el *Capítulo 2*.

El sistema se ha diseñado mediante el editor de modelos *TKRS* desarrollado en *GMF* y se ha realizado una implementación basada en *JADE* del *Sistema Multi-Agente*, donde tanto las ontologías que utilizan los agentes para comunicarse e interactuar, como las ontologías que representan el conocimiento del sistema se han expresado en *OWL/XML*, aunque se han diseñado previamente utilizando *UML*. Como se puede observar, se hace uso de un amplio abanico de herramientas y tecnologías emergentes que, hoy en día, la mayoría de los *WIS* emplean para su desarrollo. *SOLERES-KRS* representa un claro ejemplo en el que convergen varias de ellas, quedando demostrada su validez con los argumentos planteados y las pruebas de evaluación a las que ha sido sometido.

Capítulo 5

Resultados y conclusiones finales

Capítulo 5

Resultados y conclusiones finales

Contenidos

Actualmente, la proliferación de los *Sistemas de Información* en cualquier ámbito es un hecho fácilmente constatable. La abundancia de información y la necesidad de consultarla y analizarla del mejor modo para facilitar una adecuada toma de decisiones está más que presente. El empleo de métodos y técnicas estandarizados en el diseño de estos sistemas es algo extendido; sin embargo aún queda recorrido en la mejora de la interacción con otros sistemas, así como en los mecanismos de explotación de la información. En un intento de cubrir estas necesidades se ha presentado un modelo de servicio de mediación ontológico con una doble perspectiva funcional: por un lado favorece la interoperabilidad de los propios componentes del sistema, haciéndose extensible a la interacción con otros objetos o componentes externos y sistemas, y por otro facilita los procesos de búsqueda y recuperación de la información tomando parte en ellos. Es más, se ha presentado una propuesta de diseño de un *Sistema de Representación del Conocimiento* que gira en torno a este modelo de servicio de mediación y se ha proporcionado un marco para facilitar, además del diseño, la implementación del mismo partiendo de su arquitectura, de un repositorio de implementaciones y de un modelo de configuración, básicamente. También se ha desarrollado un escenario de pruebas concreto en el ámbito de los *Sistemas de Información para la Gestión Medioambiental* (*Environmental Management Information Systems*, *EMIS*). Por lo tanto, el trabajo aquí presentado ofrece su contribución tanto en el campo de los *servicios de mediación* como en el campo de los *Sistemas de Información*.

El *Capítulo* se organiza en cuatro secciones. Comienza presentando y justificando las aportaciones de esta investigación en la *Sección 5.1*. Seguidamente, en la *Sección 5.2*, se exponen las limitaciones de la propuesta. La *Sección 5.3* describe el trabajo futuro y las nuevas líneas de investigación que nacen a raíz de ésta. Para finalizar, en la *Sección 5.4*, se indican las publicaciones realizadas durante el desarrollo del trabajo.

5.1. Aportaciones

Las principales aportaciones de este trabajo son las siguientes:

— Se ha realizado un estudio de la función de mediación estándar de *ODP* a partir del cual se ha elaborado una lista de propiedades o requisitos que debe cumplir un servicio de mediación ontológico para entornos abiertos, necesarios para el posterior diseño del modelo de mediación *OntoTrader* (*Capítulo 2*).

— Se han analizado los diferentes escenarios o modelos operativos de mediación —reflexión, delegación y federación— que se pueden dar en el modelo cliente/servidor de tres niveles en el que se basa el mecanismo *Query-Searching/Recovering-Response* (*QS/RR*) seguido por el servicio de mediación *OntoTrader* en el proceso de consulta (*Capítulo 2*).

— Se ha definido el modelo *OntoTrader*, que sigue el diseño de un servicio de mediación *stand-alone* implementando las interfaces *Lookup*, *Register* y *Admin* y hacien-

do uso de un repositorio de documentos con un conjunto de metadatos obtenidos de la información del dominio, en lugar de las especificaciones de los objetos o componentes y de los servicios que ofrecen como ocurre en el servicio de mediación tradicional. Estos metadatos facilitan tanto la integración de diferentes fuentes de información, componentes o sistemas, como el proceso de búsqueda y recuperación de la información, estableciendo filtros en base a ellos para acotar la búsqueda [Iribarne et al., 2010a] (*Capítulo 2*).

— Se ha modelado una ontología de servicio/proceso —en términos de conceptos, acciones y predicados— por cada una de las interfaces implementadas para definir el comportamiento y los protocolos de interacción que deben seguir los objetos o componentes que interactúen con el servicio de mediación [Asensio et al., 2012] (*Capítulo 2*).

— Se ha propuesto la integración del modelo de mediación *OntoTrader* y el uso de ontologías para la representación del conocimiento en el diseño de un *Sistema de Representación del Conocimiento basado en Mediación* (*TKRS*), definiendo un marco de trabajo dentro de la *Ingeniería Dirigida por Modelos*, concretamente siguiendo un enfoque *PIM/PSM*, que permite la definición de la arquitectura independientemente de su implementación [Asensio et al., 2011b] (*Capítulo 3*).

— Se ha modelado la arquitectura distribuida de *TKRS* formada por nodos compuestos, a su vez, de módulos con diferentes funcionalidades, atendiendo a una arquitectura de datos con tres niveles —datos, metadatos y meta-metadatos— [Asensio et al., 2011c] [Asensio et al., 2011b] (*Capítulo 3*).

— Se ha analizado la influencia del modelo *OntoTrader* en los procesos de gestión y consulta de información (concretamente, en los escenarios de reflexión, delegación y federación) de *TKRS* (*Capítulo 3*).

— Se ha implementado un editor gráfico utilizando *Eclipse GMF* para facilitar el diseño de modelos *TKRS* [Asensio et al., 2011c] [Asensio et al., 2011b] (*Cap. 3*).

— Se ha modelado un repositorio de implementaciones de *TKRS* que permite recoger las diferentes implementaciones de los componentes del sistema realizadas con distintas plataformas de desarrollo [Asensio et al., 2011b] (*Capítulo 3*).

— Se ha modelado la configuración de *TKRS*, con el objetivo de establecer una relación entre cada elemento de la arquitectura y su correspondiente implementación en el repositorio, posibilitando el despliegue del sistema en un entorno real [Asensio et al., 2011b] (*Capítulo 3*).

— Se ha desarrollado un *Lenguaje Específico del Dominio* (*Domain Specific Language*, *DSL*) para facilitar la creación de los modelos de configuración de *TKRS* [Asensio et al., 2011b] (*Capítulo 3*).

— Se ha analizado un caso de estudio, *SOLERES-KRS*, el subsistema responsable de la gestión del conocimiento en *SOLERES*, un sistema enmarcado en el ámbito

de la ecología, concretamente en el estudio de cartografía, imágenes de satélite y sus clasificaciones. Para su diseño e implementación se ha seleccionado la tecnología de agentes software y *Sistemas Multi-Agente* (*Multi-Agent Systems*, *MAS*) [Criado et al., 2010b], se han modelado 2 ontologías de datos [Padilla et al., 2008] [Asensio et al., 2008b] [Asensio et al., 2011a] [Iribarne et al., 2011b] y se ha desarrollado una transformación *M2T* que permite el despliegue del sistema en un entorno dado a partir de su modelo de configuración (*Capítulo 4*).

— Finalmente, se ha desarrollado un entorno específico de experimentación y pruebas que ha permitido evaluar y validar la propuesta [ACG, 2012] (*Capítulo 4*).

5.2. Limitaciones del marco desarrollado

Como es natural, la contribución del trabajo a los campos de los servicios de mediación y de los *Sistemas de Representación del Conocimiento* presenta algunas limitaciones:

— El modelo de mediación *OntoTrader* implementa tan sólo las políticas de consulta *def_search_cardPolicy*, *max_search_cardPolicy* y *exact_type_matchPolicy*, definidas en la función de mediación de *ODP*. Sería conveniente implementar el resto, así como una política de "almacenamiento-y-reenvío" mediante la cual se almacene la petición en el caso de no poder ser satisfecha con la información disponible en ese momento y se posponga la respuesta hasta disponer de los metadatos necesarios para satisfacerla.

— También, tendría especial interés la implementación de un modelo de almacenamiento por extracción además del utilizado por demanda, en el que el propio servicio de mediación se ocupe de localizar nuevos metadatos e incorporarlos a su repositorio.

— Aunque el modelo *OntoTrader* posibilita la federación de mediadores, al seguir un diseño *stand-alone* que no implementa las interfaces *Link* y *Proxy* no es posible modificar en tiempo de ejecución las federaciones que se establecen al definir la arquitectura de *TKRS*, así como sus características.

— Como ya se mencionaba en el *Capítulo 1*, se utilizan ontologías de datos para la representación del conocimiento del sistema, pero no se abordan aspectos relacionados con el razonamiento e inferencia de conocimiento a partir del existente por motivos de extensión del trabajo.

— La propuesta necesita un editor gráfico que facilite el diseño de los modelos de repositorios de implementaciones similar al desarrollado para el diseño de la arquitectura de *TKRS*, si bien actualmente se utiliza el tradicional *Eclipse Reflective Ecore Model Editor*.

— En la línea de la limitación anterior, también es necesaria una aplicación que integre los editores gráficos de modelos y automatice las transformaciones de modelos, proporcionando una mejor experiencia a todos aquellos interesados en el uso de este marco.

5.3. Trabajo futuro y líneas de investigación abiertas

Teniendo en cuenta las limitaciones señaladas en la *Sección 5.2* se proponen las siguientes líneas de trabajo para continuar esta investigación:

— En primer lugar, añadir al modelo *OntoTrader*, al menos, las políticas de consulta definidas en la función de mediación tradicional de *ODP* y estudiar la inclusión de la política de "almacenamiento-y-reenvío" y del modelo de almacenamiento de metadatos por extracción.

— En segundo lugar, sería muy conveniente modificar el diseño *stand-alone* del servicio de mediación *OntoTrader* para que implemente las interfaces *Link* y *Proxy*, junto con las correspondientes ontologías de servicio/proceso, y convertirse así en un "*full-service trader*".

— En tercer lugar, analizar y explotar las posibilidades que ofrecen las ontologías de datos en cuanto a razonamiento e inferencia de conocimiento a partir del existente en el sistema.

— En cuarto lugar, utilizar el servicio de mediación para establecer de forma automática la conexión entre los elementos de la arquitectura y su correspondiente implementación, que ahora mismo tiene que realizar un técnico manualmente mediante el *DSL* definido. Esto implicaría que el propio servicio de mediación localizase de forma autónoma los componentes más adecuados para la implementación del *TKRS*, realizándose automáticamente el proceso a partir de la definición de la arquitectura del sistema con el editor gráfico.

— En quinto y último lugar, estaría el desarrollo del editor gráfico para el diseño de los modelos de repositorios de implementaciones y de la aplicación que integraría todas las herramientas utilizadas en el marco de trabajo para el diseño e implementación de sistemas *TKRS*.

5.4. Publicaciones derivadas de esta investigación

A continuación aparecen las publicaciones que se han realizado a lo largo del desarrollo de este trabajo ordenadas cronológicamente:

— Asensio, J. A., Padilla, N., and Iribarne, L. (2007). *Agentes Software y su Utilización en Sistemas de Información Medioambiental*. Técnicas Avanzadas de la Informática. Fundamentos y Aplicaciones en el Medio Ambiente, pp. 79-106. Aconcagua Libros. ISBN: 978-84-96178-15-1.

— Iribarne, L., Ayala, R., Padilla, N., and Asensio, J. A. (2007). *Modelo de Sistemas: Casos de Estudio de Modelado UML en Entornos Medioambientales*. Técnicas Avanzadas de la Informática. Fundamentos y Aplicaciones en el Medio Ambiente, pp. 107-133. Aconcagua Libros. ISBN: 978-84-96178-15-1.

— Asensio, J. A., Iribarne, L., Padilla, N., and Ayala, R. (2008). *Implementing Trading Agents for Adaptable and Evolutive UI-COTS Components Architectures*. In Proceedings of the International Conference on e-Business (ICE-B), pp. 259-262. Porto, Portugal. INSTICC Press.

— Padilla, N., Iribarne, L., Asensio, J. A., Muñoz, F. J., and Ayala, R. (2008). *Modelling an Environmental Knowledge-Representation System*. In Emerging Technologies and Information Systems for the Knowledge Society, vol. 5288 of Lecture Notes in Computer Science (LNCS), pp. 70-78. Springer. DOI: 10.1007/978-3-540-87781-3_8.

— Iribarne, L., Asensio, J. A., Padilla, N., and Ayala, R. (2008). *SOLERES-HCI: Modelling a Human-Computer Interaction Framework for Open EMS*. In The Open Knowledge Society, vol. 19 of Communications in Computer and Information Science (CCIS), pp. 320-327. Springer. DOI: 10.1007/978-3-540-87783-7_41.

— Asensio, J. A., Iribarne, L., Padilla, N., Muñoz, F. J., Ayala, R., Cruz, M., and Menenti, M. (2008). *A MDE-Based Satellite Ontology for Environmental Management Systems*. In Information Technology, Information Systems, and Knowledge Management. Springer. En prensa. ISBN: 978-0-387-88750-0.

— Almendros, J., Iribarne, L., Asensio, J. A., Padilla, N., and Vicente-Chicote, C. (2009). *An Eclipse GMF Tool for Modelling User Interaction*. In Visioning and Engineering the Knowledge Society. A Web Science Perspective, vol. 5736 of Lecture Notes in Computer Science (LNCS), pp. 405-416. Springer. DOI: 10.1007/978-3-642-04754-1_42.

— Iribarne, L., Padilla, N., Criado, J., Asensio, J. A., and Ayala, R. (2010). *A Model Transformation Approach for Automatic Composition of COTS User Interfaces in Web-Based Information Systems*. Journal of Inf. Systems Management (JISM), vol. 27 (3), pp. 207-216. Taylor & Francis. DOI: 10.1080/10580530.2010.493816.

— Iribarne, L., Padilla, N., Asensio, J. A., Muñoz, F. J., and Criado, J. (2010). *Involving Web-Trading Agents & MAS - An Implementation for Searching and Recovering Environmental Information*. In Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART), vol. 2, pp. 268-273. Valencia, Spain. INSTICC Press.

— Criado, J., Padilla, N., Iribarne, L., Asensio, J. A., and Muñoz, F. J. (2010). *Ontological Trading in a Multi-Agent System*. In Trends in Practical Applications of Agents and Multiagent Systems, Special Sessions and Workshops of the 8th International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS), Advances in Intelligent and Soft Computing (AISC), vol. 71, pp. 321-329. Salamanca, Spain. Springer. DOI: 10.1007/978-3-642-12433-4_38.

— Criado, J., Padilla, N., Iribarne, L., and Asensio, J. A. (2010). *User Interface Composition with COTS-UI and Trading Approaches: Application for Web-Based*

Environmental Information Systems. In Knowledge Management, Information Systems, E-Learning, and Sustainability Research, Part I, vol. 111 of Communications in Computer and Information Science (CCIS), pp. 259-266. Springer. DOI: 10.1007/978-3-642-16318-0_29.

— Iribarne, L., Padilla, N., Asensio, J. A., Criado, J., Ayala, R., Almendros, J., and Menenti, M. (2011). *Open-Environmental Ontology Modeling*. IEEE Transactions on Systems, Man, and Cybernetics, Part A, vol. 41 (4), pp. 730-745. IEEE. DOI: 10.1109/TSMCA.2011.2132706.

— Iribarne, L., Criado, J., Padilla, N., and Asensio, J. A. (2011). *Using COTS-Widgets Architectures for Describing User Interfaces of Web-Based Information Systems*. International Journal of Knowledge Society Research (IJKSR), vol. 2 (3), pp. 61-72. IGI Global. DOI: 10.4018/jksr.2011070106.

— Asensio, J. A., Criado, J., Iribarne, L., and Padilla, N. (2011). *Domain-Specific Ontologies Trading for Retrieval and Integration of Information in Web-Based Information Systems*. Semantic Web Personalization and Context Awareness: Management of Personal Identities and Social Networking, pp. 69-80. IGI Global. DOI: 10.4018/978-1-61520-921-7.ch007.

— Asensio, J. A., Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2011). *A Trading-Based Knowledge Representation Metamodel for Management Information System Development*. In Jornadas Sistedes, Proceedings of the 16th Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 701-706. A Coruña, Spain.

— Asensio, J. A., Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2011). *A Model-Driven Approach for Deploying Trading-Based Knowledge Representation Systems*. In On the Move to Meaningful Internet Systems: OTM 2011 Workshops, vol. 7046 of Lecture Notes in Computer Science (LNCS), pp. 180-189. Springer. DOI: 10.1007/978-3-642-25126-9_28.

— Asensio, J. A., Padilla, N., and Iribarne, L. (2012). *An Ontology-Driven Case Study for the Knowledge Representation of Management Information Systems*. In Information Systems, E-learning, and Knowledge Management Research, vol. 278 of Communications in Computer and Information Science (CCIS), pp. 426-432. Springer. DOI: 10.1007/978-3-642-35879-1_51.

Anexo A

Especificación de la arquitectura de *TKRS*

Anexo A

Especificación de la arquitectura de *TKRS*

Contenidos

Este *Apéndice* describe los diferentes elementos de la arquitectura del *Sistema de Representación del Conocimiento basado en Mediación* propuesto: *TKRS*, *Node* y *Module* (*ManagementModule*, *ServiceModule*, *QueryModule*, *TradingModule* y *ProcessingModule*). También, realiza una descripción de la unidad de agrupación *Ambient*.

A.1. TKRS

TKRS representa el Sistema de Representación del Conocimiento basado en Mediación (*Trading-based Knowledge Representation System*).

```
class TKRS {
   String name;

   String getName ();
   void setName (in String name);
   void start ();
   void stop ();
   void add_node (in Node node);
   void remove_node (in String name);
   Node describe_node (in String name);
   NodeSeq list_nodes ();
   void add_ambient (in Ambient ambient);
   void remove_ambient (in String name);
   Ambient describe_ambient (in String name);
   AmbientSeq list_ambients ();
};
```

A.1.1. Operación GetName

Signatura

```
   String getName ();
```

Función

La operación `getName` devuelve el nombre del sistema.

A.1.2. Operación SetName

Signatura

```
   void setName (in String name);
```

Función

La operación `setName` establece el nombre del sistema. El parámetro `name` identifica el sistema.

A.1.3. Operación Start

Signatura

```
void start ();
```

Función

La operación `start` inicia el sistema, iniciando todos sus nodos.

A.1.4. Operación Stop

Signatura

```
void stop ();
```

Función

La operación `stop` detiene el sistema, deteniendo todos sus nodos.

A.1.5. Operación Add_Node

Signatura

```
void add_node (in Node node);
```

Función

La operación `add_node` añade un nodo al sistema. El parámetro `node` recoge la información del nuevo nodo.

A.1.6. Operación Remove_Node

Signatura

```
void remove_node (in String name);
```

Función

La operación `remove_node` elimina un nodo del sistema. El parámetro `name` identifica el nodo a eliminar.

A.1.7. Operación Describe_Node

Signatura

```
Node describe_node (in String name);
```

Función

La operación `describe_node` devuelve la información de un nodo del sistema. El parámetro `name` identifica el nodo cuya información se demanda.

A.1.8. Operación List_Nodes

Signatura

```
NodeSeq list_nodes ();
```

Función

La operación `list_nodes` devuelve una lista con la información de todos los nodos del sistema.

A.1.9. Operación Add_Ambient

Signatura

```
void add_ambient (in Ambient ambient);
```

Función

La operación `add_ambient` añade un ambiente al sistema. El parámetro `ambient` recoge la información del nuevo ambiente.

A.1.10. Operación Remove_Ambient

Signatura

```
void remove_ambient (in String name);
```

Función

La operación `remove_ambient` elimina un ambiente del sistema. El parámetro `name` identifica el ambiente a eliminar.

A.1.11. Operación Describe_Ambient

Signatura

```
Ambient describe_ambient (in String name);
```

Función

La operación `describe_ambient` devuelve la información de un ambiente del sistema. El parámetro `name` identifica el ambiente cuya información se solicita.

A.1.12. Operación List_Ambients

Signatura

```
AmbientSeq list_ambients ();
```

Función

La operación `list_ambients` devuelve una lista con la información de todos los ambientes del sistema.

A.2. Node

Node representa cada uno de los nodos que constituyen el Sistema de Representación del Conocimiento.

```
class Node {
   String name;
   String ip;
   String port;
   String db_port;

   String getName ();
   void setName (in String name);
   String getIp ();
   void setIp (in String ip);
   String getPort ();
   void setPort (in String port);
   String getDb_port ();
   void setDb_port (in String db_port);
   void start ();
   void stop ();
   void add_management_module (ManagementModule module);
   void remove_management_module (String name);
   ManagementModule describe_management_module (String name);
   void add_service_module (ServiceModule module);
   void remove_service_module (String name);
   ServiceModule describe_service_module (String name);
   void add_query_module (QueryModule module);
   void remove_query_module (String name);
   QueryModule describe_query_module (String name);
   QueryModuleSeq list_query_modules ();
   void add_processing_module (ProcessingModule module);
```

```
   void remove_processing_module (String name);
   ProcessingModule describe_processing_module (String name);
   ProcessingModuleSeq list_processing_modules ();
   void add_trading_module (TradingModule module);
   void remove_trading_module (String name);
   TradingModule describe_trading_module (String name);
   TradingModuleSeq list_trading_modules ();
}
```

A.2.1. Operación GetName

Signatura

```
String getName ();
```

Función
La operación `getName` devuelve el nombre del nodo.

A.2.2. Operación SetName

Signatura

```
void setName (in String name);
```

Función

La operación `setName` establece el nombre del nodo. El parámetro `name` identifica el nodo.

A.2.3. Operación GetIp

Signatura

```
String getIp ();
```

Función

La operación `getIp` devuelve la dirección ip del nodo.

A.2.4. Operación SetIp

Signatura

```
void setIp (in String ip);
```

Función

La operación `setIp` establece la dirección ip del nodo. El parámetro `ip` representa la dirección ip.

A.2.5. Operación GetPort

Signatura

```
String getPort ();
```

Función

La operación `getPort` devuelve el puerto de comunicación del nodo.

A.2.6. Operación SetPort

Signatura

```
void setPort (in String port);
```

Función

La operación `setPort` establece el puerto de comunicación del nodo. El parámetro `port` representa el puerto de comunicación.

A.2.7. Operación GetDb_Port

Signatura

```
String getDb_port ();
```

Función

La operación `getDb_port` devuelve el puerto de comunicación de la base de datos con información del nodo.

A.2.8. Operación SetDb_Port

Signatura

```
void setDb_port (in String db_port);
```

Función

La operación `setDb_port` establece el puerto de comunicación de la base de datos con información del nodo. El parámetro `db_port` representa el puerto de comunicación de la base de datos.

A.2.9. Operación Start

Signatura

```
void start ();
```

Función

La operación `start` inicia el nodo, iniciando todos sus módulos.

A.2.10. Operación Stop

Signatura

```
void stop ();
```

Función

La operación `stop` detiene el nodo, deteniendo todos sus módulos.

A.2.11. Operación Add_Service_Module

Signatura

```
void add_service_module (ServiceModule module);
```

Función

La operación `add_service_module` añade un nuevo *ServiceModule* al nodo. El parámetro `module` especifica el nuevo *ServiceModule*.

A.2.12. Operación Remove_Service_Module

Signatura

```
void remove_service_module (String name);
```

Función

La operación `remove_service_module` elimina el *ServiceModule* del nodo. El parámetro `name` identifica el *ServiceModule* a eliminar.

A.2.13. Operación Describe_Service_Module

Signatura

```
ServiceModule describe_service_module (String name);
```

Función

La operación `describe_service_module` devuelve la información del *ServiceModule* del nodo. El parámetro `name` identifica el *ServiceModule* cuya información se solicita.

A.2.14. Operación Add_Management_Module

Signatura

```
void add_management_module (ManagementModule module);
```

Función

La operación `add_management_module` añade un nuevo *ManagementModule* al nodo. El parámetro `module` especifica el nuevo *ManagementModule*.

A.2.15. Operación Remove_Management_Module

Signatura

```
void remove_management_module (String name);
```

Función

La operación `remove_management_module` elimina el *ManagementModule* del nodo. El parámetro `name` identifica el *ManagementModule* a eliminar.

A.2.16. Operación Describe_Management_Module

Signatura

```
ManagementModule describe_management_module (String name);
```

Función

La operación `describe_management_module` devuelve la información del *Management-Module* del nodo. El parámetro `name` identifica el *ManagementModule* cuya información se solicita.

A.2.17. Operación Add_Query_Module

Signatura

```
void add_query_module (QueryModule module);
```

Función

La operación `add_query_module` añade un nuevo *QueryModule* al nodo. El parámetro `module` especifica el nuevo *QueryModule*.

A.2.18. Operación Remove_Query_Module

Signatura

```
void remove_query_module (String name);
```

Función

La operación `remove_query_module` elimina un *QueryModule* del nodo. El parámetro `name` identifica el *QueryModule* a eliminar.

A.2.19. Operación Describe_Query_Module

Signatura

```
QueryModule describe_query_module (String name);
```

Función

La operación `describe_query_module` devuelve la información de un *QueryModule* del nodo. El parámetro `name` identifica el *QueryModule* cuya información se solicita.

A.2.20. Operación List_Query_Modules

Signatura

```
QueryModuleSeq list_query_modules ();
```

Función

La operación `list_query_modules` devuelve una lista con todos los *QueryModules* del nodo.

A.2.21. Operación Add_Trading_Module

Signatura

```
void add_trading_module (TradingModule module);
```

Función

La operación `add_trading_module` añade un nuevo *TradingModule* al nodo. El parámetro `module` especifica el nuevo *TradingModule*.

A.2.22. Operación Remove_Trading_Module

Signatura

```
void remove_trading_module (String name);
```

Función

La operación `remove_trading_module` elimina un *TradingModule* del nodo. El parámetro `name` identifica el *TradingModule* a eliminar.

A.2.23. Operación Describe_Trading_Module

Signatura

```
TradingModule describe_trading_module (String name);
```

Función

La operación `describe_trading_module` devuelve la información de un *TradingModule* del nodo. El parámetro `name` identifica el *TradingModule* cuya información se demanda.

A.2.24. Operación List_Trading_Modules

Signatura

```
TradingModuleSeq list_trading_modules ();
```

Función

La operación `list_trading_modules` devuelve una lista con todos los *TradingModules* del nodo.

A.2.25. Operación Add_Processing_Module

Signatura

```
void add_processing_module (ProcessingModule module);
```

Función

La operación `add_processing_module` añade un nuevo *ProcessingModule* al nodo. El parámetro `module` especifica el nuevo *ProcessingModule*.

A.2.26. Operación Remove_Processing_Module

Signatura

```
void remove_processing_module (String name);
```

Función

La operación `remove_processing_module` elimina un *ProcessingModule* del nodo. El parámetro `name` identifica el *ProcessingModule* a eliminar.

A.2.27. Operación Describe_Processing_Module

Signatura

```
ProcessingModule describe_processing_module (String name);
```

Función

La operación `describe_processing_module` devuelve la información de un *ProcessingModule* del nodo. El parámetro `name` identifica el *ProcessingModule* cuya información se solicita.

A.2.28. Operación List_Processing_Modules

Signatura

```
ProcessingModuleSeq list_processing_modules ();
```

Función

La operación `list_processing_modules` devuelve una lista con todos los *ProcessingModules* del nodo.

A.3. Module

Module representa un conjunto de funcionalidades ofrecido por un nodo del sistema. Existen cinco tipos diferentes de módulos, que heredan los atributos y operaciones de Module: *ServiceModule*, *ManagementModule*, *QueryModule*, *TradingModule* y *ProcessingModule*.

```
class Module {
   String name;

   String getName ();
   void setName (in String name);
   void start ();
   void stop ();
}
```

A.3.1. Operación GetName

Signatura

```
   String getName ();
```

Función

La operación `getName` devuelve el nombre del módulo.

A.3.2. Operación SetName

Signatura

```
   void setName (in String name);
```

Función

La operación `getName` establece el nombre del módulo. El parámetro `name` identifica el módulo.

A.3.3. Operación Start

Signatura

```
   void start ();
```

Función

La operación `start` inicia el módulo.

A.3.4. Operación Stop

Signatura

```
void stop ();
```

Función

La operación `stop` detiene el módulo.

A.4. ServiceModule

ServiceModule proporciona una serie de servicios a los demás módulos de un nodo, incluyendo registro de módulos y componentes, verificación de estado, etc. Su funcionalidad e implementación depende de los servicios que se deseen ofrecer, así como de la plataforma de desarrollo utilizada. ServiceModule hereda los atributos y operaciones de Module e introduce nuevas.

A.5. ManagementModule

ManagementModule actúa como nexo entre la interfaz de usuario y el resto de módulos de un nodo, permitiendo la configuración de éstos y siendo responsable de la gestión de las demandas realizadas por los usuarios. ManagementModule hereda los atributos y operaciones de Module.

```
class ManagementModule {
   void add_metadata (
      in String processing_module_name,
      in Metadata metadata);
   void modify_metadata (
      in String processing_module_name,
      in String metadata_id,
      in Metadata metadata);
   void remove_metadata (
      in String processing_module_name,
      in String metadata_id);
   Metadata describe_metadata (
      in String processing_module_name,
      in String metadata_id);
   MetadataSeq list_metadata (
      in String processing_module_name);
   QueryResult query_information (
      in Query query);
   TradingParameters get_trading_parameters (
      in String trading_module_name);
   void set_trading_parameters (
      in String trading_module_name,
      in TradingParameters trading_parameters);
}
```

A.5.1. Operación Add_Metadata

Signatura

```
void add_metadata (
   in String processing_module_name,
   in Metadata metadata);
```

Función

La operación `add_metadata` añade un nuevo registro de metadatos en el *ProcessingModule* indicado. El parámetro `processing_module_name` identifica el *ProcessingModule*. El parámetro `metadata` especifica el nuevo registro de metadatos.

A.5.2. Operación Modify_Metadata

Signatura

```
void modify_metadata (
   in String processing_module_name,
   in String metadata_id,
   in Metadata metadata);
```

Función

La operación `modify_metadata` modifica un registro de metadatos específico en el *ProcessingModule* indicado. El parámetro `processing_module_name` identifica el *ProcessingModule*. El parámetro `metadata_id` identifica el registro de metadatos. El parámetro `metadata` especifica el nuevo registro de metadatos.

A.5.3. Operación Remove_Metadata

Signatura

```
void remove_metadata (
   in String processing_module_name,
   in String metadata_id);
```

Función

La operación `remove_metadata` elimina un registro de metadatos específico en el *ProcessingModule* indicado. El parámetro `processing_module_name` identifica el *ProcessingModule*. El parámetro `metadata_id` identifica el registro de metadatos a eliminar.

A.5.4. Operación Describe_Metadata

Signatura

```
Metadata describe_metadata (
   in String processing_module_name,
   in String metadata_id);
```

Función

La operación `describe_metadata` devuelve la información de un registro de metadatos del *ProcessingModule* indicado. El parámetro `processing_module_name` identifica el *ProcessingModule*. El parámetro `metadata_id` identifica el registro de metadatos cuya información se demanda.

A.5.5. Operación List_Metadata

Signatura

```
MetadataSeq list_metadata (in String processing_module_name);
```

Función

La operación `list_metadata` devuelve una lista con todos los registros de metadatos del *ProcessingModule* indicado. El parámetro `processing_module_name` identifica el *ProcessingModule*.

A.5.6. Operación Query_Information

Signatura

```
QueryResult query_information (in Query query);
```

Función

La operación `query_information` devuelve el resultado de una consulta de información realizada al sistema. El parámetro `query` especifica la consulta de información.

A.5.7. Operación Get_Trading_Parameters

Signatura

```
TradingParameters get_trading_parameters (in String trading_module_name);
```

Función

La operación `get_trading_parameters` devuelve los parámetros de configuración del *TradingModule* indicado. El parámetro `trading_module_name` identifica el *TradingModule*.

A.5.8. Operación Set_Trading_Parameters

Signatura

```
void set_trading_parameters (
   in String trading_module_name,
   in TradingParameters trading_parameters);
```

Función

La operación `set_trading_parameters` establece los parámetros de configuración del *TradingModule* indicado. El parámetro `trading_parameters` representa los parámetros de configuración del *TradingModule*.

A.6. QueryModule

QueryModule se ocupa exclusivamente de las consultas de información demandadas por los usuarios. QueryModule hereda los atributos y operaciones de Module e introduce nuevas.

```
class QueryModule {
   QueryResult query_information (in Query query);
   void add_trading_module (in String name);
   void remove_trading_module ();
   String describe_trading_module ();
}
```

A.6.1. Operación Query_Information

Signatura

```
QueryResult query_information (in Query query);
```

Función

La operación `query_information` devuelve el resultado de una consulta de información realizada al sistema. Esto implica la descomposición de la consulta en subconsultas trasladadas a *TradingModules* y *ProcessingModules*, la unificación de los resultados parciales y su posterior devolución. El parámetro `query` especifica la consulta de información.

A.6.2. Operación Add_Trading_Module

Signatura

```
void add_trading_module (in String name);
```

Función

La operación `add_trading_module` asocia un *TradingModule* del sistema al *QueryModule* para las consultas de información. El parámetro `name` especifica el *TradingModule*.

A.6.3. Operación Remove_Trading_Module

Signatura

```
void remove_trading_module ();
```

Función

La operación `remove_trading_module` elimina la asociación del *QueryModule* a un *TradingModule* del sistema.

A.6.4. Operación Describe_Trading_Module

Signatura

```
String describe_trading_module ();
```

Función

La operación `describe_trading_module` devuelve el *TradingModule* del sistema al que se encuentra asociado el *QueryModule* para las consultas de información.

A.7. TradingModule

TradingModule permite la búsqueda y localización de la información en el sistema, estableciendo un filtro basado en los parámetros de las consultas. *TKRS* debe poseer, al menos, un *TradingModule* en alguno de sus nodos. TradingModule hereda los atributos y operaciones de Module e introduce nuevos.

```
class TradingModule {
   boolean lookup_interface;
   boolean register_interface;
   boolean admin_interface;
   boolean link_interface;
   boolean proxy_interface;

   boolean getLookup_interface ();
   void setLookup_interface (in boolean lookup_interface);
   boolean getRegister_interface ();
   void setRegister_interface (in boolean register_interface);
   boolean getAdmin_interface ();
   void setAdmin_interface (in boolean admin_interface);
   boolean getLink_interface ();
   void setLink_interface (in boolean link_interface);
   boolean getProxy_interface ();
   void setProxy_interface (in boolean proxy_interface);
   void add_meta_metadata (in MetaMetadata meta_metadata);
   void modify_meta_metadata (
      in String meta_metadata_id,
      in MetaMetadata meta_metadata);
   void remove_meta_metadata (in String meta_metadata_id);
   MetaMetadata describe_meta_metadata (in String meta_metadata_id);
   MetaMetadataSeq list_meta_metadata ();
```

```
    QueryResult query_meta_metadata (in Query query);
    void add_federation (in String name);
    void remove_federation (in String name);
    TradingModule describe_federation (in String name);
    TradingModuleSeq list_federations ();
    TradingParameters get_parameters ();
    void set_parameters (in TradingParameters parameters);
}
```

A.7.1. Operación GetLookup_Interface

Signatura

```
boolean getLookup_interface ();
```

Función

La operación `getLookup_interface` devuelve si el *TradingModule* tiene activada la interfaz *Lookup*.

A.7.2. Operación SetLookup_Interface

Signatura

```
void setLookup_interface (in boolean lookup_interface);
```

Función

La operación `setLookup_interface` activa o desactiva la interfaz *Lookup* en el *TradingModule*. El parámetro `lookup_interface` establece la activación o desactivación de la interfaz *Lookup*.

A.7.3. Operación GetRegister_Interface

Signatura

```
boolean getRegister_interface ();
```

Función

La operación `getLookup_interface` devuelve si el *TradingModule* tiene activada la interfaz *Register*.

A.7.4. Operación SetRegister_Interface

Signatura

```
void setRegister_interface (in boolean register_interface);
```

Función

La operación `setRegister_interface` activa o desactiva la interfaz *Register* en el *TradingModule*. El parámetro `register_interface` establece la activación o desactivación de la interfaz *Register*.

A.7.5. Operación GetAdmin_Interface

Signatura

```
boolean getAdmin_interface ();
```

Función

La operación `getLookup_interface` devuelve si el *TradingModule* tiene activada la interfaz *Admin*.

A.7.6. Operación SetAdmin_Interface

Signatura

```
void setAdmin_interface (in boolean admin_interface);
```

Función

La operación `setAdmin_interface` activa o desactiva la interfaz *Admin* en el *TradingModule*. El parámetro `admin_interface` establece la activación o desactivación de la interfaz *Admin*.

A.7.7. Operación GetLink_Interface

Signatura

```
boolean getLink_interface ();
```

Función

La operación `getLookup_interface` devuelve si el *TradingModule* tiene activada la interfaz *Link*.

A.7.8. Operación SetLink_Interface

Signatura

```
void setLink_interface (in boolean link_interface);
```

Función

La operación `setLink_interface` activa o desactiva la interfaz *Link* en el *TradingModule*. El parámetro `link_interface` establece la activación o desactivación de la interfaz *Link*.

A.7.9. Operación GetProxy_Interface

Signatura

```
boolean getProxy_interface ();
```

Función

La operación `getLookup_interface` devuelve si el *TradingModule* tiene activada la interfaz *Proxy*.

A.7.10. Operación SetProxy_Interface

Signatura

```
void setProxy_interface (in boolean proxy_interface);
```

Función

La operación `setProxy_interface` activa o desactiva la interfaz *Proxy* en el *TradingModule*. El parámetro `proxy_interface` establece la activación o desactivación de la interfaz *Proxy*.

A.7.11. Operación Add_Meta_Metadata

Signatura

```
void add_meta_metadata (in MetaMetadata meta_metadata);
```

Función

La operación `add_meta_metadata` añade un nuevo registro de meta-metadatos en el repositorio del *TradingModule*. El parámetro `metadata` especifica el nuevo registro de meta-metadatos.

A.7.12. Operación Modify_Meta_Metadata

Signatura

```
void modify_meta_metadata (
   in String meta_metadata_id,
   in MetaMetadata meta_metadata);
```

Función

La operación `modify_meta_metadata` modifica un registro de meta-metadatos específico en el repositorio del *TradingModule*. El parámetro `meta_metadata_id` identifica el registro de meta-metadatos. El parámetro `metadata` especifica el nuevo registro de meta-metadatos.

A.7.13. Operación Remove_Meta_Metadata

Signatura

```
void remove_meta_metadata (in String meta_metadata_id);
```

Función

La operación `remove_metadata` elimina un registro de meta-metadatos específico en el repositorio del *TradingModule*. El parámetro `meta_metadata_id` identifica el registro de meta-metadatos a eliminar.

A.7.14. Operación Describe_Meta_Metadata

Signatura

```
MetaMetadata describe_meta_metadata (in String meta_metadata_id);
```

Función

La operación `describe_meta_metadata` devuelve la información de un registro de meta-metadatos del repositorio del *TradingModule*. El parámetro `meta_metadata_id` identifica el registro de meta-metadatos cuya información se demanda.

A.7.15. Operación List_Meta_Metadata

Signatura

```
MetaMetadataSeq list_meta_metadata ();
```

Función

La operación `list_meta_metadata` devuelve una lista con todos los registros de meta-metadatos del repositorio del *TradingModule*.

A.7.16. Operación Query_Meta_Metadata

Signatura

```
QueryResult query_meta_metadata (in Query query);
```

Función

La operación `query_meta_metadata` devuelve el resultado de una consulta de meta-metadatos. El parámetro `query` especifica la consulta de meta-metadatos.

A.7.17. Operación Add_Federation

Signatura

```
void add_federation (in String name);
```

Función

La operación `add_federation` asocia otro *TradingModule* del sistema al *TradingModule* para utilizarlo en las consultas de meta-metadatos, pudiendo delegar en él. El parámetro `name` especifica el *TradingModule* asociado al primero.

A.7.18. Operación Remove_Federation

Signatura

```
void remove_federation (in String name);
```

Función

La operación `remove_federation` elimina una asociación del *TradingModule* a otro *TradingModule* específico del sistema. El parámetro `name` especifica el *TradingModule* asociado al primero.

A.7.19. Operación Describe_Federation

Signatura

```
TradingModule describe_federation (in String name);
```

Función

La operación `describe_federation` devuelve la información de otro *TradingModule* específico del sistema al que se encuentra asociado el *TradingModule* para utilizarlo en las consultas de meta-metadatos. El parámetro `name` especifica el *TradingModule* asociado al primero.

A.7.20. Operación List_Federations

Signatura

```
TradingModuleSeq list_federations ();
```

Función

La operación `list_federations` devuelve la información de todos los *TradingModules* del sistema a los que se encuentra asociado el *TradingModule* para utilizarlos en las consultas de meta-metadatos.

A.7.21. Operación Get_Parameters

Signatura

```
TradingParameters get_parameters ();
```

Función

La operación `get_parameters` devuelve los parámetros de configuración del *TradingModule*.

A.7.22. Operación Set_Parameters

Signatura

```
void set_parameters (in TradingParameters parameters);
```

Función

La operación `set_parameters` establece los parámetros de configuración del *TradingModule*. El parámetro `port` representa los parámetros de configuración del *TradingModule*.

A.8. ProcessingModule

ProcessingModule es el responsable de la gestión de las fuentes de conocimiento (inserción, modificación y eliminación de información). *TKRS* debe poseer, al menos, un *ProcessingModule* en alguno de sus nodos. ProcessingModule hereda los atributos y operaciones de Module e introduce nuevas.

```
class ManagementModule {
   void add_metadata (in Metadata metadata);
   void modify_metadata (in String metadata_id, in Metadata metadata);
   void remove_metadata (in String metadata_id);
   Metadata describe_metadata (in String metadata_id);
   MetadataSeq list_metadata ();
   void add_trading_module (in String name);
   void remove_trading_module ();
   String describe_trading_module ();
   QueryResult query_metadata (in Query query);
}
```

A.8.1. Operación Add_Metadata

Signatura

```
void add_metadata (in Metadata metadata);
```

Función

La operación `add_metadata` añade un nuevo registro de metadatos en el repositorio del *ProcessingModule* a la vez que genera un nuevo registro de meta-metadatos que, también, almacena localmente y lo añade en el *TradingModule* al que se encuentra asociado. El parámetro `metadata` especifica el nuevo registro de metadatos.

A.8.2. Operación Modify_Metadata

Signatura

```
void modify_metadata (in String metadata_id, in Metadata metadata);
```

Función

La operación `modify_metadata` modifica un registro de metadatos específico en el repositorio del *ProcessingModule* a la vez que modifica su registro de meta-metadatos localmente y en el *TradingModule* al que se encuentra asociado. El parámetro `metadata_id` identifica el registro de metadatos. El parámetro `metadata` especifica el nuevo registro de metadatos.

A.8.3. Operación Remove_Metadata

Signatura

```
void remove_metadata (in String metadata_id);
```

Función

La operación `remove_metadata` elimina un registro de metadatos específico en el repositorio del *ProcessingModule* a la vez que elimina su registro de meta-metadatos localmente y en el *TradingModule* al que se encuentra asociado. El parámetro `metadata_id` identifica el registro de metadatos a eliminar.

A.8.4. Operación Describe_Metadata

Signatura

```
Metadata describe_metadata (in String metadata_id);
```

Función

La operación `describe_metadata` devuelve la información de un registro de metadatos del repositorio del *ProcessingModule*. El parámetro `metadata_id` identifica el registro de metadatos cuya información se demanda.

A.8.5. Operación List_Metadata

Signatura

```
MetadataSeq list_metadata ();
```

Función

La operación `list_metadata` devuelve una lista con todos los registros de metadatos del repositorio del *ProcessingModule*.

A.8.6. Operación Add_Trading_Module

Signatura

```
void add_trading_module (in String name);
```

Función

La operación `add_trading_module` asocia un *TradingModule* del sistema al *ProcessingModule* para utilizarlo en la gestión de los metadatos. Esto implica la generación e inserción de todos los meta-metadatos en el *TradingModule* asociado, que hacen referencia a los metadatos del repositorio del *ProcessingModule*. El parámetro `name` especifica el *TradingModule*.

A.8.7. Operación Remove_Trading_Module

Signatura

```
void remove_trading_module ();
```

Función

La operación `remove_trading_module` elimina la asociación del *ProcessingModule* a un *TradingModule* del sistema. Esto implica la eliminación de los meta-metadatos del *TradingModule* que hagan referencia a los metadatos del repositorio del *ProcessingModule*.

A.8.8. Operación Describe_Trading_Module

Signatura

```
String describe_trading_module ();
```

Función

La operación `describe_trading_module` devuelve el *TradingModule* del sistema al que se encuentra asociado el *ProcessingModule* para utilizarlo en la gestión de los metadatos.

A.8.9. Operación Query_Metadata

Signatura

```
QueryResult query_metadata (in Query query);
```

Función

La operación `query_metadata` devuelve el resultado de una consulta de metadatos. El parámetro `query` especifica la consulta de metadatos.

A.9. Ambient

Ambient representa una agrupación de *ProcessingModules* atendiendo a cualquier criterio: localización, información que gestionan, etc.

```
class Ambient {
   String name;

   String getName ();
   void setName (in String name);
   void add_processing_module (in String name);
   void remove_processing_module (in String name);
   ProcessingModule describe_processing_module (in String name);
   ProcessingModuleSeq list_processing_modules ();
}
```

A.9.1. Operación GetName

Signatura

```
String getName ();
```

Función

La operación `getName` devuelve el nombre del ambiente.

A.9.2. Operación SetName

Signatura

```
void setName (in String name);
```

Función

La operación `setName` establece el nombre del ambiente. El parámetro `name` identifica el ambiente.

A.9.3. Operación Add_Processing_Module

Signatura

```
void add_processing_module (in String name);
```

Función

La operación `add_processing_module` añade un *ProcessingModule* del sistema al ambiente para realizar una agrupación. El parámetro `name` especifica el *ProcessingModule*.

A.9.4. Operación Remove_Processing_Module

Signatura

```
void remove_processing_module (in String name);
```

Función

La operación `remove_processing_module` elimina un *ProcessingModule* específico del sistema agrupado en el ambiente. El parámetro `name` especifica el *ProcessingModule*.

A.9.5. Operación Describe_Processing_Module

Signatura

```
ProcessingModule describe_processing_module (in String name);
```

Función

La operación `describe_processing_module` devuelve la información del *ProcessingModule* específico del sistema agrupado en el ambiente. El parámetro `name` especifica el *ProcessingModule* cuya información se demanda.

A.9.6. Operación List_Processing_Modules

Signatura

```
ProcessingModuleSeq list_processing_modules ();
```

Función

La operación `list_processing_modules` devuelve la información de todos los *ProcessingModules* del sistema agrupados en el ambiente.

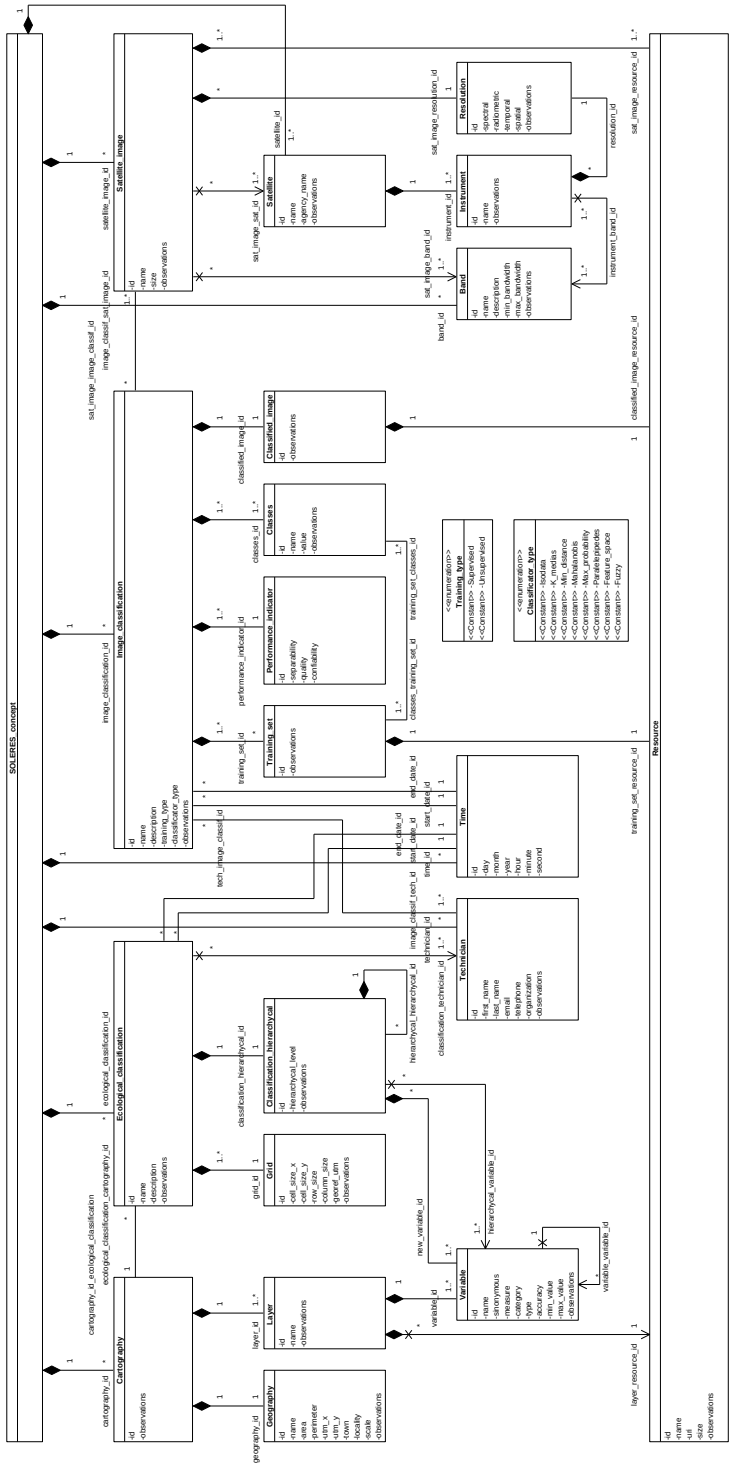
Anexo B

Modelo conceptual de la ontología *EIM* de *SOLERES-KRS*

Anexo B

Modelo conceptual de la ontología *EIM* de *SOLERES-KRS*

Este *Apéndice* muestra el modelo conceptual de los *Mapas de Información Medioambiental* (*Environmetal Information Map*, *EIM*) utilizados en el sistema *SOLERES-KRS*.



Acrónimos

Acrónimos

ACL	Agent Communication Language
AC-MDSD	Architecture-Centric Model-Driven Software Development
ATL	ATLAS Transformation Language
CASE	Computer Aided Software Engineering
CIM	Computation Independent Model
CORBA	Common Object Request Broker Architecture
COSM	Common Open Service Market
COTS	Commercial Off-The-Shelf
CSCW	Computer-Supported Cooperative Work
DAML	DARPA Agent Markup Language
DBMS	Database Management System
DPC	Dynamic Process Collaboration
DSDM	Desarrollo de Software Dirigido por Modelos
DSL	Domain Specific Language
DSS	Decision Support System
EID	Environmental Information metaData
EIM	Environmental Information Map
EMF	Eclipse Modeling Framework
EMIS	Environmental Management Information System
EPU	Environmental Process Unit
ESS	Executive Support System
FIPA	Foundation for Intelligent Physical Agents

GIS	Geographic Information System
GMF	Graphical Modeling Framework
GML	Geography Markup Language
HCI	Human-Computer Interaction
IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IS	Information System
ISO	International Organization for Standardization
JADE	Java Agent Development framEwork
JET	Java Emmiter Templates
KQML	Knowledge Query and Manipulation Language
KRR	Knowledge Representation and Reasoning
KRS	Knowledge Representation System
M2M	Model-To-Model
M2T	Model-To-Text
MAS	Multi-Agent System
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
MIS	Management Information System
MOF	Meta-Object Facility
OCL	Object Constraint Language
ODA	Ontology-Driven Architecture
ODE	Ontology-Driven Engineering
ODM	Ontology Definition Metamodel
ODP	Open Distributed Processing
OIL	Ontology Interchange Language
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
OUP	Ontology UML Profile

OWL	Ontology Web Language
PIM	Platform Independent Model
PSM	Platform Specific Model
QS/RR	Query-Searching/Recovering-Response
QVT	Query/View/Transformation
RDF	Resource Description Framework
RM-ODP	Reference Model of Open Distributed Processing
SGML	Standard Generalized Markup Language
SIDL	Service Interface Description Language
SPARQL	SPARQL Protocol And RDF Query Language
TAC	Trading Agent Competition
TKRS	Trading-based Knowledge Representation System
TPS	Transaction Processing System
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WIMP	Windows, Icons, Menus and Pointers
WIS	Web-based Information System
WMIS	Web-based Management Information System
WSDL	Web Services Definition Language
XML	eXtensible Markup Language

Bibliografía

Bibliografía

[ABED, 2010] ABED (2010). *A web GIS for hydrolic model operation*. Agricultural & Biological Engineering Department, Purdue University. https://engineering.purdue.edu/mapserve/LTHIA7/index.html.

[ACG, 2009] ACG (2009). *Examples of EIM and EID instances in SOLERES*. Applied Computing Group, University of Almería, Spain. http://acg.ual.es/soleres/tests/index.htm.

[ACG, 2010] ACG (2010). *SOLERES: Un sistema de información espacio-temporal para la gestión medioambiental basado en redes neuronales, agentes y componentes software*. Applied Computing Group, University of Almería, Spain. http://acg.ual.es/soleres/.

[ACG, 2012] ACG (2012). *SOLERES test environment*. Applied Computing Group, University of Almería, Spain. http://tkrs.ual.es/SKRS/.

[AgentLink, 1998] AgentLink (1998). *AgentLink, European co-ordination action for agent based computing*. AgentLink. http://www.agentlink.org.

[Almendros-Jiménez et al., 2009] Almendros-Jiménez, J. M., Iribarne, L., Asensio, J. A., Padilla, N., and Vicente-Chicote, C. (2009). An Eclipse GMF tool for modelling user interaction. In *Visioning and Engineering the Knowledge Society. A Web Science Perspective*, volume 5736 of *Lecture Notes in Computer Science (LNCS)*, pages 405–416. Springer. DOI: 10.1007/978-3-642-04754-1_42.

[Alonso, 2002] Alonso, E. (2002). AI and agents: State of the art. *Artificial Intelligence Magazine*, 23(3):25–30.

[An and Zhao, 2007] An, Y. and Zhao, B. (2007). Geo ontology design and comparison in geographic information integration. In *Fourth Int. Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 4, pages 608–612. IEEE Computer Society.

[Apache Foundation, 2006] Apache Foundation (2006). *The Apache Velocity Project*. The Apache Software Foundation. http://velocity.apache.org/.

[Asensio et al., 2011a] Asensio, J. A., Criado, J., Iribarne, L., and Padilla, N. (2011a). Domain-specific ontologies trading for retrieval and integration of information in web-based information systems. Semantic Web Personalization and Context Awareness:

Management of Personal Identities and Social Networking, pages 69–80. IGI Global. DOI: 10.4018/978-1-61520-921-7.ch007.

[Asensio et al., 2007] Asensio, J. A., Iribarne, L., and Padilla, N. (2007). Agentes software y su utilización en sistemas de información medioambiental. Técnicas Avanzadas de la Informática. Fundamentos y Aplicaciones en el Medio Ambiente, pages 79–106. Aconcagua Libros. ISBN: 978-84-96178-15-1.

[Asensio et al., 2008a] Asensio, J. A., Iribarne, L., Padilla, N., and Ayala, R. (2008a). Implementing trading agents for adaptable and evolutive UI-COTS components architectures. In *Proceedings of the International Conference on e-Business (ICE-B)*, pages 259–262, Porto, Portugal. INSTICC Press.

[Asensio et al., 2008b] Asensio, J. A., Iribarne, L., Padilla, N., Muñoz, F. J., Ayala, R., Cruz, M., and Menenti, M. (2008b). A MDE-based satellite ontology for environmental management systems. In *Information Technology, Information Systems, and Knowledge Management*. Springer. En prensa. ISBN: 978-0-387-88750-0.

[Asensio et al., 2011b] Asensio, J. A., Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2011b). A model-driven approach for deploying trading-based knowledge representation systems. In *On the Move to Meaningful Internet Systems: OTM 2011 Workshops*, volume 7046 of *Lecture Notes in Computer Science*, pages 180–189. Springer. DOI: 10.1007/978-3-642-25126-9_28.

[Asensio et al., 2011c] Asensio, J. A., Iribarne, L., Padilla, N., and Vicente-Chicote, C. (2011c). A trading-based knowledge representation metamodel for management information system development. In *Jornadas Sistedes, Proc. of the 16th Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, pages 701–706, A Coruña, Spain.

[Asensio et al., 2012] Asensio, J. A., Padilla, N., and Iribarne, L. (2012). An ontology-driven case study for the knowledge representation of management information systems. In *Information Systems, E-learning, and Knowledge Management Research*, volume 278 of *Communications in Computer and Information Science (CCIS)*, pages 426–432. Springer. DOI: 10.1007/978-3-642-35879-1_51.

[ATLAS Group, 2004] ATLAS Group (2004). *ATL Transformation Language (ATL)*. ATLAS Group. http://www.eclipse.org/atl/.

[Bearman, 1997] Bearman, M. (1997). *Tutorial on ODP trading function*. Faculty of Information Sciences Engineering. University of Canberra.

[Beitz and Bearman, 1994] Beitz, A. and Bearman, M. (1994). An ODP trading service for DCE. In *Proceedings of the First International Workshop on Services in Distributed and Networked Environments*, pages 42–49. IEEE.

[Belussi et al., 2004] Belussi, A., Negri, M., and Pelagatti, G. (2004). GeoUML: A geographic conceptual model defined through specialization of ISO TC211 standards. In *Proceedings of the 10th EC GI & GIS Workshop*, pages 1–10, Warsaw, Poland.

[Bermudez, 2004] Bermudez, L. E. (2004). Ontomet: Ontology metadata framework. Philadelphia, PA. Drexel University.

[Bézivin, 2005] Bézivin, J. (2005). On the unification power of models. *Software and System Modeling*, 4(2):171–188.

[Bicocchi et al., 2010] Bicocchi, N., Baumgarten, M., Brgulja, N., Kusber, R., Mamei, M., Mulvenna, M. D., and Zambonelli, F. (2010). Self-organized data ecologies for pervasive situation-aware services: The knowledge networks approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 40(4):789–802.

[Brachman and Schmolze, 1985] Brachman, R. J. and Schmolze, J. G. (1985). An overview of the kl-one knowledge representation system. *Cognitive Science*, 9(2):171–216.

[Branco et al., 2010] Branco, M., Zaluska, E., de Roure, D., Lassnig, M., and Garonne, V. (2010). Managing very large distributed data sets on a data grid. *Concurrency and Computation: Practice and Experience*, 22(11):1338–1364.

[Brilhante, 2004] Brilhante, V. B. (2004). An ontology for quantities in ecology. In *Advances in Artificial Intelligence - 17th Brazilian Symposium on Artificial Intelligence (SBIA), Proceedings*, Lecture Notes in Computer Science, pages 144–153. Springer.

[Ceccaroni et al., 2004] Ceccaroni, L., Cortés, U., and Sànchez-Marrè, M. (2004). OntoWEDSS: Augmenting environmental decision-support systems with ontologies. *Environmental Modelling and Software*, 19(9):785–797.

[Chaudhuri, 2012] Chaudhuri, S. (2012). How different is big data? In *IEEE 28th International Conference on Data Engineering (ICDE)*, page 5. IEEE Computer Society.

[Collins et al., 2009] Collins, J., Ketter, W., and Gini, M. L. (2009). Flexible decision control in an autonomous trading agent. *Electronic Commerce Research and Applications*, 8(2):91–105.

[Cranefield and Purvis, 1999] Cranefield, S. and Purvis, M. (1999). UML as an ontology modelling language. In *Proc. of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 46–53.

[Craske et al., 1999] Craske, G., Tari, Z., and Kumar, K. R. (1999). DOK-Trader: A CORBA persistent trader with query routing facilities. In *Distributed Objects and Applications (DOA)*, pages 230–240.

[Criado et al., 2010a] Criado, J., Padilla, N., Iribarne, L., and Asensio, J. A. (2010a). User interface composition with COTS-UI and trading approaches: Application for Web-based Environmental Information Systems. In *Knowledge Management, Information Systems, E-Learning, and Sustainability Research, Part I*, volume 111 of *Communications in Computer and Information Science (CCIS)*, pages 259–266. Springer. DOI: 10.1007/978-3-642-16318-0_29.

[Criado et al., 2010b] Criado, J., Padilla, N., Iribarne, L., Asensio, J. A., and Muñoz, F. (2010b). Ontological trading in a multi-agent system. In *Trends in Practical Applications of Agents and Multiagent Systems - 8th International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS), Special Sessions and Workshops*, volume 71 of *Advances in Intelligent and Soft Computing (AISC)*, pages 321–329, Salamanca, Spain. Springer. DOI: 10.1007/978-3-642-12433-4_38.

[Czarnecki and Eisenecker, 2000] Czarnecki, K. and Eisenecker, U. W. (2000). *Generative programming - methods, tools and applications*. Addison-Wesley.

[Czarnecki and Helsen, 2006] Czarnecki, K. and Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–646.

[DARPA, 1993] DARPA (1993). *Knowledge Query and Manipulation Language (KQML)*. Defense Advanced Research Projects Agency. http://www.csee.umbc.edu/kqml/.

[Di Lecce et al., 2004] Di Lecce, V., Pasquale, C., and Piuri, V. (2004). A basic ontology for multi agent system communication in an environmental monitoring system. In *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA)*, pages 45 – 50, Boston, MA. Institute of Electrical and Electronics Engineers.

[Djuric et al., 2005] Djuric, D., Gasevic, D., Devedzic, V., and Damjanovic, V. (2005). A UML profile for OWL ontologies. In *Model Driven Architecture, European MDA Workshops: Foundations and Applications (MDAFA)*, volume 3599 of *Lecture Notes in Computer Science*, pages 204–219. Springer.

[EMF, 2013] EMF (2013). *Eclipse Modeling Framework*. http://www.eclipse.org/modeling/emf/.

[Falkovych et al., 2003] Falkovych, K., Sabou, M., and Stuckenschmidt, H. (2003). UML for the semantic web: Transformation-based approaches. In *Knowledge Transformation for the Semantic Web*, pages 92–106.

[FIPA, 2002a] FIPA (2002a). *FIPA Abstract Architecture Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00001/.

[FIPA, 2002b] FIPA (2002b). *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00061/.

[Franklin and Graesser, 1997] Franklin, S. and Graesser, A. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, UK. Springer-Verlag.

[Fulbright and Stephens, 1994] Fulbright, R. and Stephens, L. M. (1994). Classification of multiagent systems. Technical report, M.H.Seymour, LU TP.

[Gama and May, 2011] Gama, J. and May, M. (2011). Ubiquitous knowledge discovery. *Intell. Data Anal.*, 15(1):1.

[Gasevic et al., 2004] Gasevic, D., Djuric, D., Devedzic, V., and Damjanovic, V. (2004). Converting UML to OWL ontologies. In *Proceedings of the 13th International Conference on World Wide Web - WWW (Alternate Track Papers & Posters)*, pages 488–489. ACM.

[GMF, 2013] GMF (2013). *Graphical Modeling Framework*. http://eclipse.org/gmf-tooling/.

[Goh and Foo, 2008] Goh, D. and Foo, S. (2008). *Social information retrieval systems: Emerging technologies and applications for searching the web effectively*. IGI Global, Hershey, PA.

[Gómez-Pérez and Corcho, 2002] Gómez-Pérez, A. and Corcho, Ó. (2002). Ontology specification languages for the semantic web. *IEEE Intelligent Systems*, 17(1):54–60.

[Greenfield and Short, 2003] Greenfield, J. and Short, K. (2003). Software factories: Assembling applications with patterns, models, frameworks and tools. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 16–27. ACM.

[Gronmo et al., 2002] Gronmo, R., Solheim, I., and Skogan, D. (2002). Experiences of UML-to-GML encoding. In *Proceedings of the 5th AGILE Conference*, pages 1–8.

[Gruber, 1995] Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928.

[Gutierrez-Pulido et al., 2006] Gutierrez-Pulido, J. R., Ruiz, M. A. G., Herrera, R., Cabello, E., Legrand, S., and Elliman, D. (2006). Ontology languages for the semantic web: A never completely updated review. *Knowl.-Based Syst.*, 19(7):489–497.

[Hayes-Roth, 1995] Hayes-Roth, B. (1995). An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365.

[Hirschheim et al., 1995] Hirschheim, R., Klein, H. K., and Lyytinen, K. (1995). *Information systems development and data modeling: Conceptual and philosophical foundations*. Cambridge University Press, New York, NY, USA.

[Honavar, 1999] Honavar, V. (1999). Intelligent agents and Multi-Agent Systems. In *IEEE Conference on Evolutionary Computation (CEC)*. Invited Tutorials.

[Huang, 2008] Huang, M. (2008). A new method to formal description of spatial ontology. *Information Technology and Environmental System Sciences*, (3):417–421.

[Iribarne et al., 2008] Iribarne, L., Asensio, J. A., Padilla, N., and Ayala, R. (2008). SOLERES-HCI: Modelling a Human-Computer Interaction framework for open EMS. In *The Open Knowledge Society*, volume 19 of *Communications in Computer and Inf. Science (CCIS)*, pages 320–327. Springer. DOI: 10.1007/978-3-540-87783-7_41.

[Iribarne et al., 2007] Iribarne, L., Ayala, R., Padilla, N., and Asensio, J. A. (2007). Modelo de sistemas: Casos de estudio de modelado UML en entornos medioambientales. Técnicas Avanzadas de la Informática. Fundamentos y Aplicaciones en el Medio Ambiente, pages 107–133. Aconcagua Libros. ISBN: 978-84-96178-15-1.

[Iribarne et al., 2011a] Iribarne, L., Criado, J., Padilla, N., and Asensio, J. A. (2011a). Using COTS-widgets architectures for describing user interfaces of Web-based Information Systems. *International Journal of Knowledge Society Research (IJKSR)*, 2(3):61–72. DOI: 10.4018/jksr.2011070106.

[Iribarne et al., 2011b] Iribarne, L., Padilla, N., Asensio, J. A., Criado, J., Ayala, R., Almendros-Jiménez, J. M., and Menenti, M. (2011b). Open-environmental ontology modeling. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 41(4):730–745. DOI: 10.1109/TSMCA.2011.2132706.

[Iribarne et al., 2010a] Iribarne, L., Padilla, N., Asensio, J. A., Muñoz, F., and Criado, J. (2010a). Involving web-trading agents & MAS – an implementation for searching and recovering environmental information. In *Int. Conference on Agents and Artificial Intelligence (ICAART)*, volume 2, pages 268–273, Valencia, Spain. INSTICC Press.

[Iribarne et al., 2010b] Iribarne, L., Padilla, N., Criado, J., Asensio, J. A., and Ayala, R. (2010b). A model transformation approach for automatic composition of COTS user interfaces in web-based information systems. *Journal of Information Systems Management (JISM)*, 27(3):207–216. DOI: 10.1080/10580530.2010.493816.

[Iribarne et al., 2004] Iribarne, L., Troya, J. M., and Vallecillo, A. (2004). A trading service for COTS components. *The Computer Journal*, 47(3):342–357.

[ISO, 2004] ISO (2004). *Environmental Management Systems - Specification with guidance for use*. International Organization for Standardization.

[ISO et al., 1996] ISO, IEC, and ITU-T (1996). *Reference Model of Open Distributed Processing (RM-ODP)*. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC) and Telecommunication Standardization Sector (ITU-T). http://www.rm-odp.net.

[ITU-T and ISO/IEC, 1998] ITU-T and ISO/IEC (1998). *Information Technology – Open Distributed Processing – ODP Trading Function – Part 1: Specification*. ITU-T Rec. X.950 | ISO/IEC 13235-1.

[Jennings et al., 1998] Jennings, N. R., Sycara, K. P., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

[Kalibatiene and Vasilecas, 2011] Kalibatiene, D. and Vasilecas, O. (2011). Survey on ontology languages. In *Perspectives in Business Informatics Research - 10th International Conference (BIR) Proceedings*, volume 90 of *Lecture Notes in Business Information Processing*, pages 124–141. Springer.

[Kogut et al., 2002] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., and Smith, J. (2002). UML for ontology development. *Knowledge Engineering Review*, 17(1):61–64.

[Kurtev, 2007] Kurtev, I. (2007). State of the art of QVT: A model transformation language standard. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE), Third International Symposium*, volume 5088 of *Lecture Notes in Computer Science*, pages 377–393. Springer.

[Kutvonen, 1996] Kutvonen, L. (1996). Overview of the DRYAD trading system implementation. In *The IFIP/IEEE International Conference on Distributed Platforms: Client/Server and Beyond: DCE, CORBA, ODP and Advanced Distributed Applications - ICDP*, pages 314–326. Chapman & Hall.

[Lamparter and Schnizler, 2006] Lamparter, S. and Schnizler, B. (2006). Trading services in ontology-driven markets. In *ACM Symposium on Applied Computing (SAC)*, pages 1679–1683.

[Lieberman et al., 2007] Lieberman, J., Singh, R., and Goad, C. (2007). *W3C Geospatial Ontologies*. W3C Incubator Group Report. http://www.w3.org/2005/Incubator/geo/XGR-geo-ont-20071023/.

[Lukasheh et al., 2001] Lukasheh, A. F., Droste, R. L., and Warith, M. A. (2001). Review of Expert System (ES), Geographic Information System (GIS), Decision Support System (DSS), and their applications in landfill design and management. *Waste Management & Research*, 19(2):177–185.

[Madden, 2012] Madden, S. (2012). From databases to big data. *IEEE Internet Computing*, 16(3):4–6.

[Maes, 1994] Maes, P. (1994). Modeling adaptive autonomous agents. *Artificial Life*, 1(1-2):135–162.

[Maes, 1995] Maes, P. (1995). Artificial life meets entertainment: Lifelike autonomous agents. *Commun. ACM*, 38(11):108–114.

[Maes, 1997] Maes, P. (1997). Agents that reduce work and information overload. *Software Agents*. AAAI Press/MIT Press.

[Mangina, 2002] Mangina, E. (2002). *Review of software products for Multi-Agent Systems*. Applied Intelligence (UK). Ltd for AgentLink : http://www.AgentLink.org.

[Mathe et al., 2008] Mathe, J. L., Werner, J., Lee, Y., Malin, B., and Ledeczi, A. (2008). Model-based design of clinical information systems. *Methods of Information in Medicine*, 47(5):399–408.

[McGuinness et al., 2002] McGuinness, D. L., Fikes, R., Hendler, J. A., and Stein, L. A. (2002). DAML+OIL: An ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5):72–80.

[Merz et al., 1994] Merz, M., Müller-Jones, K., and Lamersdorf, W. (1994). Service trading and mediation in distributed computing systems. In *Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 450–457.

[Müller et al., 1995] Müller, K., Merz, M., and Lamersdorf, W. (1995). The TRADEr: Integrating trading into DCE. In *The 3rd IFIP Conference on Open Distributed Processing*, pages 476–487. Chapman & Hall.

[Nwana, 1996] Nwana, H. S. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244.

[OGC, 2007a] OGC (2007a). *Geography Markup Language (GML).* Open Geospatial Consortium. http://www.opengeospatial.org/standards/gml/.

[OGC, 2007b] OGC (2007b). *GEOINT structure implementation profile schema processing.* Open Geospatial Consortium.

[Okawa et al., 2008] Okawa, T., Kaminishi, T., Hirabayashi, S., Koizumi, H., and Sawamoto, J. (2008). An information system development method based on the link of business process modeling with executable UML modeling and its evaluation by prototyping. In *AINA Workshops*, pages 1057–1064.

[OMG, 1990] OMG (1990). *Object Management Architecture (OMA).* Object Management Group. http://www.omg.org/oma/.

[OMG, 1997] OMG (1997). *Unified Modeling Language (UML).* Object Management Group. http://www.uml.org/.

[OMG, 1998] OMG (1998). *MASIF-RTF Results.* Object Management Group.

[OMG, 2002] OMG (2002). *Meta-Object Facility (MOF).* Object Management Group. http://www.omg.org/mof/.

[OMG, 2003] OMG (2003). *Model Driven Architecture (MDA).* Object Management Group. http://www.omg.org/mda/.

[OMG, 2006] OMG (2006). *Object Constraint Language (OCL).* Object Management Group. http://www.omg.org/spec/OCL/.

[OMG, 2008] OMG (2008). *Meta Object Facility (MOF) 2.0 Query / View / Transformation (QVT).* Object Management Group. http://www.omg.org/spec/QVT/.

[OOC, 2001] OOC (2001). *ORBacus trader. ORBacus for C++ and Java.*

[Padilla et al., 2008] Padilla, N., Iribarne, L., Asensio, J. A., Muñoz, F. J., and Ayala, R. (2008). Modelling an environmental knowledge-representation system. In *Emerging Technologies and Information Systems for the Knowledge Society*, volume 5288 of *Lecture Notes in Computer Science (LNCS)*, pages 70–78. Springer. DOI: 10.1007/978-3-540-87781-3_8.

[Pan et al., 2006] Pan, Y., Xie, G. T., Ma, L., Yang, Y., Qiu, Z., and Lee, J. (2006). Model-Driven Ontology Engineering. In *J. Data Semantics VII*, volume 4244 of *Lecture Notes in Computer Science*, pages 57–78. Springer.

[Popma, 2003] Popma, R. (2003). *Java Emitter Templates (JET) tutorial*. Azzurri Ltd. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html.

[PrismTech, 2001] PrismTech (2001). *OpenFusion trading service whitepaper*. PrismTech. http://www.prismtech.com/openfusion/resources/white-papers/.

[Project Spire, 2003] Project Spire (2003). *ETHAN*. http://www.csee.umbc.edu/csee/research/spire/.

[Ramos et al., 2005] Ramos, A. C., Gensel, J., Villanova-Oliver, M., and Martin, H. (2005). Adapted information retrieval in web information systems using PUMAS. In *AOIS*, pages 243–258.

[REDIAM, 2007] REDIAM (2007). *Red de Información Ambiental de Andalucía*. Junta de Andalucía. http://www.rediam.es/.

[Schadt et al., 2010] Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L., and Nolan, G. P. (2010). Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657.

[Schmidt, 2001] Schmidt, D. C. (2001). *The ADAPTATIVE communication environment (TAO): AceORB*. University of California. http://www.theaceorb.com/.

[Shen et al., 2008] Shen, J., Krishna, A., Yuan, S., Cai, K., and Qin, Y. (2008). A pragmatic gis-oriented ontology for location based services. In *Australian Software Engineering Conference (ASWEC)*, pages 562–569. IEEE Computer Society.

[SIDo Group, 2007] SIDo Group (2007). *ATL use case - ODM implementation (bridging UML and OWL)*. L3I lab in La Rochelle. http://www.eclipse.org/atl/usecases/ODMImplementation/.

[Smith et al., 1994] Smith, D. C., Cypher, A., and Spohrer, J. C. (1994). Kidsim: Programming agents without a programming language. *Commun. ACM*, 37(7):54–67.

[Song et al., 2007] Song, J., Zhu, Y., and Wang, J. (2007). A study of semantic retrieval system based on geo-ontology with spatio-temporal characteristic. In *Proceedings of International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, volume I-II, pages 1029–1034.

[Sowa, 2000] Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks Cole Publishing.

[Stahl and Völter, 2006] Stahl, T. and Völter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, Chichester, UK.

[SWRG, 2004] SWRG (2004). *Pellet OWL reasoner*. Semantic Web Research Group, Mind Lab, University of Maryland Institute for Advanced Computer Studies. http://www.mindswap.org/2003/pellet/.

[Taniar and Rahayu, 2004] Taniar, D. and Rahayu, J. W. (2004). *Web Information Systems*. Print. IGI Global, Hershey.

[Tripathi and Babaie, 2008] Tripathi, A. and Babaie, H. A. (2008). Developing a modular hydrogeology ontology by extending the SWEET upper-level ontologies. *Computers & Geosciences*, 34(9):1022–1033.

[Tsai et al., 2007] Tsai, W.-T., Huang, Q., Xu, J., Chen, Y., and Paul, R. A. (2007). Ontology-based dynamic process collaboration in service-oriented architecture. In *Service-Oriented Comp. and App. (SOCA)*, pages 39–46.

[van Heijst et al., 1997] van Heijst, G., Schreiber, A. T., and Wielinga, B. J. (1997). Using explicit ontologies in KBS development. *International Journal of Human-Computer Studies*, 46(2):183–292.

[Vasudevan and Bannon, 1999] Vasudevan, V. and Bannon, T. (1999). WebTrader: Discovery and programmed access to web-based services. In *Poster at the 8th International WWW Conference (WWW8)*, Toronto, Canada. http://www.objs.com/agility/tech-reports/9812-web-trader-paper/WebTraderPaper.html.

[W3C, 1999] W3C (1999). *Resource Description Framework (RDF)*. World Wide Web Consortium. http://www.w3.org/RDF/.

[W3C, 2004a] W3C (2004a). *OWL Web Ontology Language reference*. World Wide Web Consortium. http://www.w3.org/OWL/.

[W3C, 2004b] W3C (2004b). *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium. http://www.w3.org/TR/rdf-schema/.

[W3C, 2006] W3C (2006). *Ontology Driven Architectures and potential uses of the Semantic Web in Systems and Software Engineering*. World Wide Web Consortium. http://www.w3.org/2001/sw/BestPractices/SE/ODA/.

[W3C, 2008] W3C (2008). *SPARQL Query Language for RDF*. World Wide Web Consortium. http://www.w3.org/TR/rdf-sparql-query/.

[Wang et al., 2007] Wang, Y., Dai, J., Sheng, J., Zhou, K., and Gong, J. (2007). Geo-ontology design and its logic reasoning. In *SPIE Proceedings - Geospatial Information Science*, volume 6753, pages 675309–1 – 675309–11.

[Wiebe and Chan, 2012] Wiebe, A. J. and Chan, C. W. (2012). Ontology Driven Software Engineering. In *25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4. IEEE.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10:115–152.

[Xiao-feng et al., 2006] Xiao-feng, M., Bao-wen, X., Qing, L., Ge, Y., Jun-yi, S., Zhengding, L., and Yan-xiang, H. (2006). A survey of web information technology and application. *Wuhan University Journal of Natural Sciences*, 11:1–5.

[Xpand, 2007] Xpand (2007). *Eclipse Xpand*. http://www.eclipse.org/modeling/m2t/xpand/.

[Zhan et al., 2007] Zhan, Q., Li, D., and Shao, Z. (2007). An architecture for ontology-based geographic information semantic grid service. In *Geoinformatics 2007: Geospatial Information Technology and Applications, SPIE Proceedings*, volume 6754, pages 67541U – 67541U–9.

[Ziming and Liyi, 2007] Ziming, Z. and Liyi, Z. (2007). An integrated approach for developing e-commerce system. In *International Conference on Wireless Communications, Networking and Mobile Computing (WiCom)*, pages 3596–3599.

Este documento ha sido generado con LaTeX.

Todas las *Figuras* y *Tablas* contenidas en
el presente documento son originales.

Un Sistema de Representación del Conocimiento
Basado en Mediación Ontológica

José Andrés Asensio Cortés
Departamento de Informática
Grupo de Investigación de Informática Aplicada (TIC-211)
Universidad de Almería
Almería, a 21 de Junio de 2013

http://www.ual.es/personal/jacortes/
http://acg.ual.es/

José Andrés Asensio

Un Sistema de Representación del Conocimiento Basado en Mediación Ontológica

Uno de los mayores retos que plantea la sociedad de la información en la que nos encontramos inmersos es hacer frente a la ingente cantidad de información manejada en casi cualquier ámbito. Los Sistemas de Información tratan de dar solución a este problema, pero aún presentan ciertas dificultades en los procesos de búsqueda y recuperación de la información, debido fundamentalmente al gran volumen que gestionan, a la heterogeneidad de dicha información y a su dispersión en múltiples fuentes.

Partiendo de este contexto, el objetivo principal del presente trabajo de investigación consiste en el planteamiento de un marco de trabajo que facilite el diseño y desarrollo de Sistemas de Representación del Conocimiento (TKRS) basados en un Modelo de Mediación Ontológico (OntoTrader), siguiendo las directrices de la Ingeniería Dirigida por Modelos (MDE) y de la Ingeniería Dirigida por Ontologías (ODE), para conseguir la separación de la arquitectura del sistema de su implementación.

La propuesta viene acompañada de un entorno de pruebas junto con una serie de escenarios para verificar los procesos de gestión y consulta de información según unos modelos operativos identificados, y del caso de estudio de SOLERES-KRS, un Sistema de Representación del Conocimiento diseñado y desarrollado bajo este marco mediante la tecnología de Agentes Software y Sistemas Multi-Agente.

http://www.ual.es/personal/jacortes/