UNIVERSIDAD
POLITECNICA
DE VALENCIA

DEPARTAMENTO DE COMUNICACIONES

ESPECIFICACIÓN OWL DE UNA ONTOLOGÍA PARA TELEEDUCACIÓN EN LA WEB SEMÁNTICA

TESIS DOCTORAL PROPUESTA POR:
ROBERTO ROMERO LLOP

DIRIGIDA POR EL DOCTOR:
CARLOS ENRIQUE PALAU SALVADOR

VALENCIA, 2007

Resumen

Debido al gran desarrollo de la World Wide Web, tanto en cantidad de contenidos y nodos como en velocidades de acceso, aparecen por parte de la comunidad científico-técnica propuestas de utilización de la misma con objetivos más ambiciosos que la mera descarga de ficheros para ser presentados al usuario.

Con esa finalidad se desarrolla la Web Semántica, sistema que pretende introducir información entendible por Agentes Inteligentes, permitiendo de este modo que estos Agentes puedan aumentar sus bases de conocimiento y realizar inferencias que faciliten procesos realizados actualmente de forma manual por los usuarios. De esta forma surgen los lenguajes ontológicos para la web, y en concreto el lenguaje recomendado por la World Wide Web Consortium (W3C) denominado Web Ontology Language (OWL), así como razonadores relacionados como FACT++, Racer y Pellet.

A més, con el objetivo de aprovechar el potencial de la web, se han ido generando gran cantidad de contenidos educativos, que debido a los altos costes de producción generan una necesidad de potenciar la reutilización de dichos contenidos. Aparece en este caso el concepto de objeto educativo, que es susceptible de ser reaprovechado para otras experiencias de aprendizaje, con alguna modificación o sin modificación alguna, generando la denominada interoperabilidad de objetos educativos.

El presente trabajo pretende potenciar esta interoperabilidad de objetos educativos. Para ello se especifica una ontología completa para teleeducación, basada en la lógica descriptiva y desarrollada en el lenguaje OWL, para que pueda ser utilizada por medio de la Web Semántica. Se estudian, desarrollan e implementan dentro de esta ontología conceptos relacionados con la interacción de los distintos agentes que intervienen en una experiencia de aprendizaje a través de la web.

La ontología presentada va además acompañada de una especificación de arquitectura de pares o Peer to Peer (P2P) basada en las arquitecturas de tablas de búsqueda distribuidas (DHTs), que denominaremos DHT Semántica. La DHT Semántica está diseñada para permitir la explotación por parte de Agentes Inteligentes de la ontología especificada, con una alta tolerancia a fallos en nodos de la arquitectura. Estos Agentes asisten en la búsqueda de objetos educativos más allá de la búsqueda por palabras claves. Finalmente, tanto la ontología como la arquitectura se validan utilizando un conjunto de experiencias educativas on-line reales.

Resum

Degut al gran desenvolupament de la World Wide Web, tant en quantitat de continguts i nodes como en velocitats d'accés, apareixen per part de la comunitat científic tècnica propostes d'utilització de la mateixa amb objectius més ambiciosos que la simple descàrrega de fitxers per a ser presentats a l'usuari.

Amb esta finalitat es desenvolupa la Web Semàntica, sistema que pretén introduir informació comprensible per Agents Intel·ligents, permetent d'esta manera que estos Agents puguen augmentar les seues bases de coneixement i realitzar inferències que faciliten processos realitzats actualment de forma manual per els usuaris. D'esta manera apareixen els llenguatges ontològics per a la web, i en concret el llenguatge recomanat per la World Wide Web Consortium (W3C) anomenat Web Ontology Language (OWL), així com raonadors relacionats com FACT++, Racer i Pellet.

A més, amb l'objectiu d'aprofitar el potencial de la web, s'han anat generant gran quantitat de continguts educatius, que degut als alts costos de producció generen una necessitat de potenciar la reutilització dels anomenats continguts. Apareix en eixe cas el concepte d'objecte educatiu, que pot tornar a ser aprofitat per a altres experiències d'aprenentatge, amb alguna modificació o sense modificació alguna, generant la denominada interoperabilitat d'objectes educatius.

El present treball pretén potenciar esta interoperabilitat dels objectes educatius. Per a aconseguir-ho s'especifica una ontologia completa per a teleeducació, basada en la lògica descriptiva y desenvolupada en el llenguatge OWL, per tal que puga ser utilitzada mitjançant la Web Semàntica. S'estudien, desenrotllen i implementen dins d'esta ontologia conceptes relacionats amb la interacció dels diferents agents que intervenen en una experiència d'aprenentatge mitjançant la web.

L'ontologia presentada va també acompanyada d'una especificació d'arquitectura de parells o Peer to Peer (P2P) basada en les arquitectures de taules de cerca distribuïdes (DHTs),que denominarem DHT Semàntica. La DHT Semàntica està dissenyada per a permetre l'explotació per part d'Agents Intel·ligents de l'ontologia especificada, amb una alta tolerància a errades en nodes de l'arquitectura. Estos Agents ajuden en la cerca d'objectes educatius més enllà de la cerca per paraules clau. Finalment, tant l'ontologia com l'arquitectura es validen utilitzant un conjunt d'experiències educatives on-line reials.

Abstract

Due to the development of the World Wide Web, in the number of contents, nodes and transmission speeds, the scientific community is making proposals for different uses. These uses are nowadays far more ambitious than downloading files to be rendered to the user.

The Semantic Web is developed with this aim. The Semantic Web includes information which can be understood by Intelligent Agents (IA). In this process, the IAs can expand their Knowledge Bases (KB) and perform inferences that can ease some human tasks. The information in the Semantic Web uses different ontology languages for the web, as the World Wide Web Consortium (W3C) recommendation named Web Ontology Language (OWL). The inferences can be done by different Inference Engines (IEs) as FACT++, Racer and Pellet.

In addition to this, many educative contents have appeared in the web, with high production cost. For this reason reusing contents is nowadays a need, with little or no changes. These units of contents to be independently shared are know as Learning Objects (LOs). These LOs have to interoperate among them in order to build e-learning experiences.

The document presented fosters this LOs interopeability. To achieve this objective, a complete ontology is specified, based on Description Logic (DL) and developed in OWL, in order to be used in the Semantic Web. In this ontology, concepts related to the different relationships among the stakeholders of an e-learning experience are studied, developed and implemented.

A new architecture called Semantic DHT is defined in order to exploit the ontology. The Semantic DHT is a Peer to Peer (P2P) architecture based on Distributed HastTables (DHTs), and allows the Intelligent Agents access to the ontological information, with a high node fail tolerance. These Agents facilitate the ontological searches done by the users. Finally, the ontology and the architecture are validated against real e-learning experiences.

Agradecimientos

Isaac Newton escribió “Si he visto más allá es gracias a que he subido a hombros de gigantes”. No me gustaría empezar este documento sin agradecer a todos los gigantes que me han ayudado y apoyado a la hora de poder ir más allá, y que sin ellos este trabajo no hubiera sido posible.

En primer lugar a mi director de tesis D. Carlos Enrique Palau Salvador, por su dedicación a este trabajo y por saber indicarme líneas y caminos a seguir que han sido cruciales para el desarrollo satisfactorio de las investigaciones y resultados. También agradezco a él y al grupo de investigación al que pertenece por haberme introducido en el campo de la teleformación desde mis últimos años de la carrera.

A D. Patricio Montesinos Sanchis y Dª. Mónica López Sieben, director y subdirectora del Centro de Formación Permanente de la Universidad Politécnica de Valencia, centro al que pertenezco desde hace diez años. Les agradezco el haberme facilitado y apoyado en todo momento en la realización de la tesis doctoral, y por considerar que el desarrollo y la formación de las personas es un aspecto clave en un puesto de trabajo. Agradezco también al grupo de sistemas de información del centro, y en especial a Víctor y Rafel, que han realizado críticas constructivas a lo largo del todo el proceso. Espero poder ayudar a Víctor ahora que le llega su oportunidad.

Deseo agradecer a gigantes de la comunidad científico-técnica que sólo conozco por su trabajo, como Tim Berners-Lee e Ian Horrocks, que han servido de inspiración y guía para esta tesis.

Finalmente a mi familia, por haber estado en todo momento animándome a continuar este camino y saber disculpar la falta de la merecida atención a ellos debido a las parcelas de tiempo ocupadas en la investigación y desarrollo de este trabajo. En especial a mi mujer Eliana, que más ha sufrido mis noches y fines de semana de dedicación. Han sido gigantes en la determinación de que tenía que continuar con este objetivo cuando yo mismo flaqueaba y han sabido ver siempre la luz al final del túnel.

A Eliana

UNIVERSIDAD
POLITECNICA
DE VALENCIA

DEPARTAMENTO DE COMUNICACIONES

ESPECIFICACIÓN OWL DE UNA ONTOLOGÍA PARA TELEEDUCACIÓN EN LA WEB SEMÁNTICA

TESIS DOCTORAL PROPUESTA POR:
ROBERTO ROMERO LLOP

DIRIGIDA POR EL DOCTOR:
CARLOS ENRIQUE PALAU SALVADOR

VALENCIA, 2007

Capítulo 1: Introducción

1.1 Introducción

Hoy en día los sistemas Telemáticos, y más concretamente, la World Wide Web se está introduciendo en todos los aspectos de la vida de las sociedades avanzadas. Desde consultar cuentas bancarias, a reservar vuelos directamente a las compañías, obtener las últimas noticias, incluso tener una conversación síncrona o asíncrona con los amigos. Otro aspecto más en el que está introduciéndose es en el educativo, y en concreto en las Universidades, donde el impacto será aún mayor cuando las generaciones que fueron al colegio con Internet en casa lleguen a la Universidad.

Las tendencias actuales en Teleeducación se centran en la reutilización de los contenidos (utilizando los llamados Objetos Educativos), ya que tras un periodo eufórico en que todos los contenidos on-line se generaban desde cero, se ha llegado a un punto de madurez donde se ha advertido el gran coste de recursos que conlleva la generación de un material on-line de

calidad desde cero frente a la reutilización de contenidos de calidad generados anteriormente y cuya adaptación a las nuevas necesidades sea factible.

Además, para conseguir dicha re-usabilidad se han de generar mecanismos que permitan buscar los Objetos Educativos que sirvan para nuestros propósitos. La ardua tarea de búsqueda de los Objetos Educativos debería poder desarrollarse mayoritariamente por parte de sistemas no humanos, es decir, Agentes Inteligentes que puedan entender la meta-información sobre los Objetos Educativos y presentar uno o un conjunto de ellos que cumpla las expectativas planteadas.

Entendemos como Agente Inteligente [RUS96] a toda entidad que percibe y actúa sobre un entorno. El entorno que trataremos en esta Tesis será Internet, y pretenderemos que los Agentes actúen sobre el entorno desarrollando un cierto grado de inteligencia. Todo agente dispone de unos perceptores, que le permiten recoger información sobre el entorno, y de unos actuadores por medio de los cuales produce modificaciones en el entorno que le permiten conseguir o acercarse a un objetivo.

Un Agente Inteligente desarrolla, en mayor o menor medida, las siguientes características más destacables para conseguir sus objetivos [JUL00]:

- Continuidad temporal: un Agente se desarrolla normalmente por medio de procesos que tienen una duración ilimitada. Esta característica permitirá que los Agentes estén continuamente relacionándose con el entorno.
- Autonomía: que indica que no necesita órdenes directas de los usuarios, sino que muchas veces es el entorno y su propia experiencia lo que le provoca la realización de una acción. Cuanto más autónomo sea un Agente para cumplir sus objetivos, menos esfuerzo demandará de parte del usuario, que podrá centrarse en otras tareas.
- Sociabilidad: un Agente se puede comunicar con otros agentes heterogéneos de su entorno, generando en cierto modo un sistema multiagente. Esta característica se desarrolla de una forma más eficaz en arquitecturas con una gran escalabilidad y puede conllevar también la relación entre Agentes de las mismas características, lo que puede generar una red entre iguales o Peer to Peer (P2P).
- Racionalidad: que indica que un agente es capaz de realizar las acciones correctas basándose en los datos que obtiene u obtuvo del

entorno. Para conseguirlo se basa en otras características como la adaptabilidad y autonomía.

La definición de Agente Inteligente no indica el grado de inteligencia que debe desarrollar el mismo, encontrando distintos niveles de elaboración a la hora de desarrollar la mencionada inteligencia. Si simplemente buscan palabras en los documentos, en lugar de aplicar sistemas de agentes será mejor utilizar motores de búsqueda ya existentes, como Google o incluso Gnutella, que nos permitiera alcanzar gran cantidad de elementos que debería revisar una persona para indicar cuál es el que queremos de verdad. Estos sistemas de indexamiento de palabras clave con una cierta jerarquía en la que los términos más generales tienen como hijos a los términos más específicos se conoce como sistemas Tesauros [GAR04].

En los sistemas de búsqueda actuales, se ha generalizado la tendencia a definir metadatos sobre los objetos educativos, siguiendo una sintaxis estándar para colocar los datos y recursos. El lenguaje más utilizado en el entorno de Internet para la comunicación de los metadatos es el Extensible Markup Language XML [RFC3023], lenguaje fruto de una simplificación del lenguaje Standard Generalized Markup Language SGML [ISO8879]. Sobre XML se empieza a trabajar con lenguajes que nos permiten además relacionar datos y recursos, siendo el más aceptado el lenguaje Resource Description Framework (RDF). RDF sin embargo no tienen la riqueza semántica necesaria para poder definir una lógica y ontología completa, por lo que las ontologías implementadas en RDF necesitan de especificaciones para programadores en cada caso, que sientan las bases de una lógica y ontología inherente pero no entendible directamente por Agentes.

Sin embargo, ahora se está empezando a probar seriamente la introducción de lenguajes ontológicos, que no sólo describen recursos colocándoles datos y relacionándolos, sino que también describen de forma estándar las relaciones entre los distintos recursos a nivel de clases, subclases y además a nivel lógico, utilizando la Lógica Descriptiva.

En estos últimos años se está afianzando el lenguaje de descripción de Ontologías por la Web (OWL, Web Ontology Language), procedente del trabajo colaborativo entre investigadores americanos y los europeos, que ya está utilizándose para la descripción general de Servicios Web (OWL-S) [ANK04], así como en campos como la medicina con bastante éxito [STE04]. OWL es un lenguaje ontológico construido sobre RDF/XML y que fusiona las características de DAML y de OIL, y que nos permite

trabajar con Lógica Descriptiva (en su versión OWL DL) y que se está erigiendo como estándar para el desarrollo de razonadores de esta lógica. Si XML nos permite definir clases y los datos de objetos de esas clases y RDF nos permite incluir sintácticamente relaciones entre los recursos, OWL nos permite incluir, de forma estándar, las relaciones de una ontología a nivel de especializaciones (clases y subclases) y ciertas relaciones lógicas entre ellas, como definir una clase como la intersección de otras clases.

Por otra parte, otra tendencia detectada en las herramientas de Teleeducación actuales es su enfoque hacia las interacciones que realiza el estudiante con los otros actores de su experiencia educativa (foros, blogs, correos electrónicos, calendarios, chats, herramientas colaborativas, preguntas y exámenes). Esta tendencia contrasta con el interés que existía en el pasado que se centraba en la generación de contenidos estáticos, y dejaba a un lado la parte colaborativa, tratándola como un servicio colateral totalmente separado de los contenidos.

Es decir, las nuevas tendencias de Teleeducación dan más importancia a todo lo que diferencia un verdadero sistema teleeducativo de un simple libro o un CD con contenidos multimedia estáticos. Estos enfoques pretenden dos objetivos determinados: por un lado incentivar el interés del alumno por medio de su integración activa en el proceso educativo y por otro lado potenciar el enfoque constructivista de la educación, llegando incluso a impulsar el denominado aprendizaje indirecto [FOWL05], es decir, el aprendizaje por el contexto dejado por otras experiencias educativas.

1.2 Motivación de la Tesis

Podemos decir que la teleeducación ha pasado por varias fases: una inicial de experimentación, donde aparecen gran cantidad de soluciones para los mismos problemas, fomentada por grandes inversiones a fondo perdido (proyectos Europeos y proyectos americanos de investigación) y que nos deja una gran cantidad de soluciones, la mayoría válidas pero incompatibles entre sí y con escasa aceptación en el mercado.

Después, en una segunda fase, basándose en la experiencia y viendo lo complicado que es generar contenidos de calidad, aparece la necesidad de exportación/importación (necesidad creada muchas veces por los mismos inversores, como el Ministerio de Defensa de los Estados Unidos) de recursos educativos (interoperabilidad), con lo que aparecen varios estándares con el objetivo de recoger casi todas las posibilidades y permitir un intercambio eficiente de contenidos. Para facilitar dicho intercambio

aparece la necesidad de marcar los objetos educativos por medio de metadatos que los describan y permitan escogerlos e instalarlos en los nuevos sistemas. Los propios metadatos facilitan a su vez la instalación en los nuevos sistemas, ya que normalmente incluyen información lógica sobre el objeto educativo (estructura, navegabilidad, evaluaciones, etc.). De nuevo para esta segunda fase se parte primero de lenguajes de descripción de tipos de datos enviados y estructura de esos datos, como el XML (que ya tiene herramientas maduras para creación y modificación de documentos, absorción de información y búsqueda). Estos lenguajes dejan la lógica y las relaciones entre los recursos menos definidas, de forma que en cada estándar se define una ontología particular que han de conocer los programadores.

Pero ahora se plantea la necesidad de agentes de búsqueda más inteligentes y por tanto la necesidad de pasar a una tercera fase, con el uso de lenguajes ontológicos que permitan que sean los agentes inteligentes los que entiendan la lógica y sepan ampliar su base de conocimiento por medio de documentos RDFS (sobre el que se van describiendo ya más estándares y hay algo de madurez en herramientas para su tratamiento), OWL o en un futuro SWRL.

Por tanto, lo que nos motiva a la realización de dicha Tesis es contribuir al desarrollo de una tercera fase, por medio de la especificación de arquitecturas y agentes de búsqueda para objetos educativos descritos por medio de la lógica descriptiva. Además desarrollaremos el enfoque constructivista de los objetos educativos, es decir, la descripción de las distintas interacciones entre los participantes (alumnos, profesores y tutores) del proceso teleeducativo.

1.3 Objetivos

La presente Tesis pretende estudiar y aplicar las dos tendencias comentadas anteriormente:

Los objetivos generales de esta Tesis son:

- **Estudiar y aplicar las nuevas tendencias en interoperabilidad de recursos educativos**, por medio de los lenguajes ontológicos. Estos lenguajes están influyendo a toda la Web en el desarrollo de la Web Semántica, pero nosotros nos centraremos en el ámbito de los objetos educativos.

- **Incluir en la descripción de los objetos educativos las interacciones de los usuarios**, pretendiendo ir más allá de la facilitación de la interacción y permitiendo el almacenamiento de dicha interacción para su uso posterior, por medio de búsquedas inteligentes o como parte que ha enriquecido los contenidos con información contextual.

- **Contribuir al desarrollo y mejora de los sistemas de Teleeducación**, por medio de sistemas que permitan búsquedas más inteligentes que simples tesauros y añadiendo información constructivista.

- **Impulsar el uso de la Web Semántica en Teleeducación**. Este objetivo se debe a mi interés sobre Internet y las nuevas posibilidades que va ofreciendo, desde los laboratorios de simulación por medio de applets de Java [ROM 99], [RAG002], hasta los estándares de transmisión de datos por medio de la Web como XML [MAN02]. En estos momentos es necesario que la teleeducación dé un paso más y añada lógica a la descripción de sus recursos, desarrollándose dentro de la Web Semántica.

- **Estudiar las posibilidades de agentes inteligentes en la Teleeducación**. Estos agentes son los que revolucionarán la futura web semántica.

Siendo los objetivos específicos los siguientes:

- **Mostrar el estado del arte en la Teleeducación basado en la Web Semántica**, así como presentar fuentes de información que permitan una actualización continua en las últimas tendencias.

- **Estudiar las ontologías educativas existentes más importantes**. Para hacer esto revisaremos las bases lógicas en las que se basan.

- **Describir las interacciones más importantes que se producen en un proceso educativo por Internet**. Esto nos permitirá abstraer las características principales de las mismas para el desarrollo de la ontología.

- **Definir, desarrollar y evaluar una ontología para educación que permita la reutilización y compartición de recursos educativos,** orientándose a su vez hacia unas técnicas instruccionales constructivistas.

- **Expresar dicha ontología en OWL**, de forma que pueda ser aprovechada por los Agentes de propósito general que se están utilizando para este lenguaje ontológico.

- **Proponer y definir una arquitectura para la web semántica en teleeducación**, que permita el uso y explotación de la ontología definida.

- **Definir, desarrollar y evaluar un prototipo de agente de búsqueda en la web semántica**, basado en la arquitectura propuesta, que nos facilite la interoperativilidad entre objetos educativos.

1.4 Principales aportaciones

Los objetivos planteados en esta Tesis generan las siguientes aportaciones al campo de estudio tratado, que hasta donde llega mi conocimiento no han sido desarrollados:

- **Primeras integraciones de las interacciones de los actores de una experiencia educativa en la parte de interoperabilidad de objetos educativos**. Esto supone un salto respecto a las teorías actuales de interoperabilidad a nivel de contenidos estáticos, que abarcan a lo sumo en algunos casos a las evaluaciones. Aunque en algunos casos, como en las evaluaciones, existen ejemplos en XML y XML/RDF, nuestras propuestas se realizarán sobre OWL DL.

- **Propuesta, implementación y evaluación de una ontología de educación dedicada a las interacciones comentadas en el punto anterior**, llegando a su implementación y uso en casos y comunidades de aprendizaje reales.

- **Propuesta, implementación y uso de un prototipo de exportación de interacciones a un lenguaje ontológico de lógica descriptiva (OWL DL).** Esto nos permitirá definir sistemas que no empiezan desde cero, sino que son capaces de extraer la información

necesaria de experiencias educativas ya existentes, y por tanto facilitar la actualización de un sistema a otro.

- **Desarrollo de técnicas de extracción de la información importada a la lógica descriptiva**. En estos casos se experimentará con protocolos que nos permitan atacar a distintos razonadores, como el caso de DIG (Description Logic Implementation Group Interface).

- **Propuesta de arquitectura de explotación de la interoperabilidad conseguida por medio de la ontología propuesta**, ampliando propuestas actuales basadas sólo en los contenidos estáticos. Además, dicha arquitectura se basará en sistemas descentralizados P2P.

- **Demostración por medio de prototipos del uso de las herramientas actuales así como indicaciones respecto a sus rendimientos**. Esto permite hacerse una idea de las dificultades y ventajas del uso de estas tecnologías, así como obtener medidas de rendimiento que nos permitirán compararlas con otras técnicas más básicas.

1.5 Retos encontrados

Roal Amudsen nunca hubiera conquistado el polo sur si se hubiera retirado ante los retos que planteaba su expedición, que nadie antes había realizado y para la que no se contaba con equipamiento óptimo y probado. Hoy en día visitar el polo sur es, aunque difícil, bastante más sencillo que en 1911.

Una de las limitaciones inherentes a la novedad de los temas tratados en la Tesis es la falta de herramientas probadas y optimizadas para la validación del prototipo, así como la inexistencia de metodologías definidas de trabajo para interconectar dichas herramientas. Además, una vez planteada la ontología, no se han encontrado arquitecturas probadas y escalables para su uso.

En este caso particular se ha detectado una falta de herramientas optimizadas, fáciles de usar y compatibles para el trabajo con el lenguaje ontológico elegido (OWL), así como de motores de inferencias optimizados y preparados para la interconectividad. Este problema se ha resuelto por medio del uso de distintas herramientas, cada una para una determinada

parte del estudio, interconectadas de forma manual o por medio de estándares de comunicación (DIG, o el propio OWL).

Esta falta de herramientas maduras ha sido un reto que se ha tenido que vencer por medio de implementaciones propias, el uso de varias herramientas en desarrollo y la realización de comparativas de las mismas.

La inexistencia de arquitecturas optimizadas para el uso de ontologías en la web semántica nos ha conducido, a su vez, a la definición de una arquitectura que supla la carencia en el campo de estudio.

A su vez, el trabajo con la parte de conocimiento no explícita (es decir, el contexto del curso en lugar de los contenidos estáticos generados por el autor) nos ha planteado dificultades para su esquematización, debido a su inherente falta de estructura fija, así como la necesidad del uso de datos reales para su experimentación. Para resolver esta parte se ha planteado una estructura abierta de ontología de interacciones, con pocas reglas que permitan su máxima interoperabilidad.

La necesidad de datos reales para el estudio se ha resuelto gracias a la experiencia del Centro de Formación Permanente de la Universidad Politécnica de Valencia, que ha dejado a la disposición de este estudio toda la información contextual real de varias comunidades virtuales llevadas a cabo en años anteriores.

1.6 Alcance de la Tesis

Como en todo desarrollo que pretenda ser realista, es necesario delimitar el alcance de esta Tesis y centrarse en la finalidad para la cual ha sido concebida, de modo que podamos conseguir los objetivos propuestos. En un campo tan amplio como la propia web, desarrollo de ontologías y los agentes inteligentes, debemos enfocar nuestro trabajo en áreas concretas y determinadas, especialmente en aquellas donde se tratan los aspectos realmente innovadores.

Formularemos una ontología completa sobre las principales áreas de una experiencia educativa por internet, y para realizar una prueba de concepto trataremos con la parte que no se contempla en la mayoría de las ontologías similares. Esta parte será la relacionada con los foros, elementos novedosos en una ontología y sobre los que existen escasos estudios al respecto. Asimismo los foros constituyen una de las partes más complejas de

modelizar, implementar y evaluar su rendimiento en búsquedas, siendo las otras partes de la ontología más sencillas a la hora de realizar estas tareas.

Especificaremos además una arquitectura de explotación completa, indicando todos los elementos que intervendrán en dicha arquitectura para facilitar la publicación de información y el tratamiento de la misma por parte de los agentes inteligentes. Asimismo, centraremos los ejemplos de implementación en la parte realmente innovadora de extracción de información de las comunidades virtuales actuales y la consulta de dicha información por medio de un prototipo de agente de búsqueda.

Respecto a las técnicas de extracción de la información de las interacciones se definirá un caso práctico sólo a modo de ejemplo. Dicho sistema de extracción permitirá interactuar con el sistema de información del Centro de Formación Permanente de la Universidad Politécnica de Valencia, si bien casi todas las herramientas básicas de interacción de los distintos sistemas de gestión de aprendizaje son bastante similares. La elección de este ejemplo es debida a que disponemos de los permisos necesarios para extraer datos reales de experiencias educativas que han tenido lugar en el pasado

1.7 Organización de la memoria

En este primer capítulo se han introducido los principales factores que motivan la realización de esta Tesis, así como los objetivos que se pretenden alcanzar. El resto de la memoria se desarrolla por medio de capítulos tal y como se indica a continuación.

En el capítulo 2 se realiza un estudio del estado del arte de la web actual, hablando de los pasos desde el pasado de la web estática, al presente de la web dinámica y web 2.0 y el futuro con la web semántica. Dedicaremos gran parte del capítulo a la web semántica y la estructura de capas, ya que es uno de los pilares sobre los que se construye esta Tesis.

A continuación, en el mismo capítulo, se realiza un estudio del estado del arte en ontologías de teleeducación existentes, comenzando con una descripción de las principales teorías de aprendizaje, describiendo luego las principales iniciativas en teleeducación y las principales ontologías en este campo. Se completa después un estudio más minucioso sobre los foros, dejando patente la falta de ontologías que cubran este campo.

En el capítulo 3 se trata la lógica, los lenguajes lógicos y su relación con la web semántica. En una primera parte se sientan las bases lógicas de forma

incremental, comenzando por la lógica proposicional, luego la lógica de primer orden (FOL) y terminando en la lógica objetivo de esta Tesis, la lógica descriptiva (DL). En esta parte se trata el concepto de suposición de mundo abierto (OWA) y el concepto de razonamiento, presentando además métodos para realizar dichos razonamientos. Estos conceptos son necesarios para la implementación de agentes inteligentes que se basen en la web semántica.

En la segunda parte del capítulo se estudian los principales lenguajes ontológicos, y llegaremos a las tendencias actuales de utilización del lenguaje web para ontologías (OWL), siendo este otro de los pilares de esta Tesis. Para desarrollar los conceptos de OWL hablaremos de los distintos niveles de OWL y sus implicaciones respecto a la riqueza descriptiva del lenguaje y la carga computacional de los razonamientos. También introducimos los distintos constructores y axiomas utilizados en OWL.

Ya en el capítulo 4 comenzamos a presentar las primeras aportaciones de esta Tesis. Tras describir las interacciones de tipo evaluación en la teleeducación, se estudian la implementación y metadatos utilizados por el estándar más utilizado en evaluaciones, IMS QTI, comparándolo con nuestra aplicación basada en XML para la realización de exámenes, que incluye una especificación de los metadatos (TeML). En este punto se compararán algunos aspectos de las dos especificaciones. La parte de nuestra ontología final relacionada con las evaluaciones se basa en estas dos especificaciones, que se complementan entre sí.

En el capítulo 5 se define una arquitectura implementable en la web actual sobre la que podrían correr nuestras aplicaciones utilizando la ontología implementada en esta Tesis. Esta arquitectura es un desarrollo completamente nuevo y que no ha sido tratado de esta forma hasta donde llega mi conocimiento, y se basa en los conceptos de red Peer to Peer (P2P), las tablas de hashtables dinámicas (DHT) y la web semántica. Esta arquitectura se ha bautizado con el nombre de DHT Semántica, y permite no sólo el funcionamiento de nuestros agentes y nuestra ontología, sino cualquier sistema basado en agentes inteligentes que funcionen con OWL, que necesiten una gran escalabilidad.

A lo largo del capítulo tratamos la descripción de funcionamiento y elementos de la DHT Semántica, así como los conceptos de escalabilidad y seguridad, conceptos críticos en toda aplicación web de cierta envergadura.

Después de definir una arquitectura sobre la que se desarrollará el sistema, pasamos en el capítulo 6 a especificar formalmente una ontología especialmente diseñada para la interoperabilidad. En este apartado definimos por separado las sub-ontologías para contenidos estáticos, evaluaciones e interacciones entre usuarios, como son los foros. Después se fusionan las subontologías en una ontología global, explicando los puntos de interconexión entre las subontologías.

Cabe destacar que, aunque existen especificaciones para la interoperabilidad, no existen especificaciones completas que incluyan los contenidos estáticos y las interacciones, no existiendo tampoco actualmente ninguna especificación basada en OWL y preparada para ser atacada por agentes inteligentes genéricos.

En el capítulo 7 pasaremos a la fase de validación de la ontología y obtención de resultados. El objetivo de este capítulo es realizar un conjunto de prototipos que nos permitan realizar una prueba de concepto sobre la ontología definida. La primera parte de la implementación consiste en la generación en OWL de la ontología definida en el capítulo 6. Para realizar esta parte se evalúan las herramientas de generación de ontologías en OWL principales. Una vez generada nuestra otología en OWL se comprueba la corrección lógica de la misma por medio de la clasificación y el chequeo de consistencia.

Más adelante, dentro del mismo capítulo, se describe el proceso de importación de datos del sistema de información del que obtenemos los datos reales de foros, a nuestra ontología descrita. Cualquier importación desde otro sistema de información seguiría un proceso similar, utilizando probablemente las mismas herramientas. Se realizan evaluaciones de rendimiento en la importación que nos permiten evaluar la escalabilidad en la importación y la complejidad computacional.

Como parte final del capítulo 7 se realiza una prueba del uso por medio de un modelo de agente inteligente, evaluando primero los principales razonadores o motores de inferencia actuales y modelizando después la carga de conocimientos en el agente. También se realizan pruebas de búsquedas por palabras clave o búsquedas por medio de inferencias. Por medio de esta prueba de concepto se valida que la ontología definida en esta Tesis es implementable y utilizable por medio de agentes inteligentes.

Por último tenemos un anexo con el glosario de términos utilizados, para facilitar la lectura y comprensión de la memoria, y un anexo con el conjunto de referencias utilizadas en la memoria.

Capítulo 2: Estudio del Estado del Arte

2.1 Introducción

En este capítulo se estudian las tendencias actuales en el desarrollo de la web actual, tanto a nivel general como respecto al campo de la teleeducación.

A lo largo del capítulo nos situaremos en los últimos desarrollos en los campos anteriormente mencionados, lo que nos permitirá construir a partir de ese punto para obtener resultados realmente innovadores, mostrando que hasta la fecha y hasta donde llega nuestro conocimiento no han sido tratados.

También nos permitirá este capítulo entender mejor el enfoque para los capítulos posteriores, donde se habla de tendencias construccionistas y sobre todo de desarrollos para el nuevo concepto emergente de web semántica.

2.2 Estado del arte de la Web actual. Web Semántica

2.2.1 Introducción

Los servicios y utilidades ofertados en la web están aumentando y esto está conllevando asimismo que la cantidad de información disponible a los usuarios también se incremente.

Un ejemplo de la importancia de la web como fuente de información sería la Tesis que nos ocupa. Las visitas que se han realizado a diferentes bibliotecas para recabar información en el proceso de elaboración de esta Tesis han sido más bien escasas. En cambio, las consultas que se han realizado a la web han sido numerosas, ya que además de permitirnos obtener en tiempo real una información variada, nos ha facilitado el acceso a fuentes de información actualizadas. Así hemos podido consultar artículos antes de que se publicaran en sus correspondientes revistas.

Asimismo, el uso de la web como fuente de información ha permitido dirigir el sentido de las búsquedas en función de la información que se iba obteniendo. En este sentido se han realizado ampliaciones, se ha profundizado en nuevas líneas de investigación que han ido surgiendo a cada paso; todo ello de una forma rápida y funcional

Otra de las ventajas de la utilización de la web es que ha permitido el acceso a recursos de información variados, que no se pueden encontrar en formato físico y que en ocasiones han hecho posible incluso un intercambio de opiniones y una realimentación que ha enriquecido el resultado final del trabajo. Nos referimos a la participación en discusiones por medio de foros, al uso de listas de interés relacionadas con los temas desarrollados en la Tesis [WWW73], [WWW74].

Además, ha sido posible consultar de forma inmediata definiciones e información básica cada vez que ha ido surgiendo un concepto nuevo a medida que avanzaba la investigación. Estas fuentes de información consultadas merecen un cierto grado de confianza, ya que en el caso de Wikipedia [WWW67] se indican las definiciones y artículos que han sido revisadas por diversos actores.

Sin embargo no todo han sido ventajas en el uso de la web para la elaboración de esta Tesis. En este sentido, la gran cantidad de información a la que se ha podido acceder se ha convertido, en ocasiones, en un inconveniente, ya que es necesario seleccionar los recursos que nos interesan y que nos van a ser útiles, y saber diferenciarlos de aquellas fuentes que no son fiables ni nos aportan datos nuevos. Aunque los ejemplos son numerosos, citaremos tres de ellos por resultar especialmente significativos.

Gracias al mundialmente conocido buscador Google [WWW66], que dispone de más de dos billones de recursos indexados, no hemos tenido que rastrear toda la web en busca de información, pero sí nos hemos visto obligados a cribar toda la que íbamos obteniendo. Esto es así porque este buscador únicamente permite realizar consultas simples y basadas en el contenido de texto, de forma que por cada entrada se han obtenido cantidades ingentes pero inoperantes de recursos relacionados.

Si bien es cierto que este buscador sitúa en las primeras posiciones los recursos más referenciados, ello no ha evitado que se haya tenido que revisar muchos de los recursos de forma manual en un proceso minucioso. Algunos recursos se caracterizaban por tener contenidos de gran calidad, y/o habían sido desarrollados por personalidades de reconocido prestigio en la temática; pero en otros casos no era así, o bien los recursos contenían información repetida o se centraban de un modo secundario en la temática, de modo que ha habido que desecharlos

Otro ejemplo de sobreinformación se ha producido con el uso de Citeseer [WWW9], que aunque se basa en la metainformación para realizar búsquedas, tiene un alcance limitado.

Por su parte, los foros y listas de distribución, han permitido encontrar respuesta a algunas de las dudas e inquietudes que se han planteado en el proceso de elaboración de esta Tesis, y conocer las experiencias similares a las que se han enfrentado otros investigadores. Sin embargo, se han recibido más de 1.000 correos al mes en algunas de las listas de interés, lo que nos ha obligado nuevamente a realizar un elaborado proceso de selección, ya que no podíamos, por razones obvias de limitación temporal, acceder a todos estos contenidos.

Luego, en este océano infinito de información en crecimiento vertiginoso que es la Web, necesitamos disponer de las herramientas adecuadas para

acceder a aquellos recursos e informaciones que realmente nos interesan de una forma eficiente. Estas herramientas, para que realmente sean útiles, deberían ser capaces de desarrollar cierta inteligencia que nos permitiera acotar las búsquedas que queremos realizar. Para ello, una gran parte de la información que podemos encontrar en la web debería ser entendible por las máquinas de forma que éstas pudieran evaluar la calidad y adecuación de los contenidos a las necesidades del usuario.

En los siguientes apartados se muestra el grado de desarrollo y de preparación que ha alcanzado la web actual, para que en ella puedan realizar las máquinas búsquedas inteligentes. Se parte de la web sintáctica para comentar la web más dinámica llamada web 2.0 y por último la web semántica. El apartado de la web semántica se desarrollará a fondo explicando las capas en las que se basa, ya que va a ser uno de los puntos fundamentales de esta Tesis.

2.2.2 Web Sintáctica

La Web, como la conocemos actualmente se puede considerar como una Web sintáctica: un lugar donde los ordenadores se encargan únicamente de la tarea de transmisión y presentación de los recursos. Por otro lado, son los usuarios los encargados de realizar las distintas interpretaciones de los recursos presentados y de relacionar los recursos entre sí de forma lógica. Estas tareas que han de realizar los usuarios son costosas sobre todo con la gran cantidad de información disponible actualmente en la web [GOB03]. Otros autores denominan a este estadio de la web como Web 1.0 [ORE05].

Basándonos en una web con estas características, se puede exigir poco a los sistemas de búsqueda automáticos. A lo sumo se pueden realizar sistemas que busquen palabras o conjunto de palabras, ordenados en función de la importancia que dan otras personas a ese recurso por medio de las veces que se encuentra dicho recurso como hipervínculo en otros recursos. Google trabaja de esta forma, por medio de indexaciones y asignaciones de peso de importancia según las veces que sea referenciado un recurso en otros recursos. Pongamos por ejemplo la búsqueda de recursos que contengan el texto “Semantic Web”, y probablemente obtendremos la página oficial de dicha iniciativa en W3C como una de los primeros resultados de la búsqueda.

Sin embargo, estos sistemas de búsqueda están incapacitados para la búsqueda de recursos con unas características lógicas más complejas. Por ejemplo, sería imposible solicitar al sistema el conjunto de “artículos que

traten como tema central la Web Semántica y que estén escritos por autores prestigiosos en ese campo". La razón estriba en que la información que tienen y pueden entender las máquinas en la web sintáctica es información de cómo presentar el recurso, e hipervínculos a recursos relacionados. El contenido semántico (si es un artículo, si trata principalmente de la Web Semántica, si el autor está dentro de los prestigiosos) sólo es entendible por los humanos.

Un concepto distinto al de la web sintáctica y que no podemos confundir es el de web estática. Una web estática es una web de un servidor que presenta contenidos prefijados de antemano y que no varían en función de las peticiones de los usuarios y de la información que se disponga en cada momento. En estos casos lo único que define de forma completa el contenido a presentar es la dirección URL que solicita el usuario.

En oposición a la web estática se encuentra el concepto de web dinámica, en la que los recursos y contenidos que presenta el servidor a cada usuario son variables, ya no dependiendo solamente de la URL que solicita el usuario, sino de otros parámetros. Uno de estos nuevos parámetros que varían los contenidos presentados es la información que dispone el servidor en el momento en que se solicita el recurso, como ocurre cuando se consulta información meteorológica de un lugar. Otro posible parámetro puede ser el usuario que accede al recurso, en caso que se haya identificado, teniendo como ejemplo en este caso cualquier aplicación web de lectura de correo electrónico. Otro posible parámetro a tener en cuenta es información extra enviada por el solicitante del recurso incluida en la propia URL (método HTTP/GET) o de forma separada (método HTTP/POST).Esta información extra la suele introducir el usuario por medio de formularios que aparecen en la web. Normalmente las webs estáticas basan su información en un conjunto de ficheros situados en directorios y las webs dinámicas procesan además información proveniente de bases de datos, ya sean monolíticas o distribuidas.

Las webs dinámicas y estáticas pueden aparecer dentro de la web sintáctica, ya que el concepto de dinámico o estático sólo se refiere a la forma de generar los recursos, no a la utilidad de los mismos.

2.2.3 Web 2.0

Web 2.0 [ORE05] es un término que se extendió a finales del 2004, y que en principio intenta agrupar un conjunto de nuevos servicios y tecnologías que están emergiendo en la Web. Sus bases son:

- La web es la única plataforma. Toda acción sobre los sistemas se realiza vía web, por medio de navegadores estándar, sin instalar nada. Este punto rompe con la tendencia clásica de generar programas que se tengan que instalar en los ordenadores de los usuarios, ya que pasamos a ofrecer servicios por medio de la web a cualquier usuario que tenga un navegador.

  Como principal ventaja de esta decisión tenemos que siempre disponemos del servicio actualizado, y que los programadores no han de preparar un conjunto de versiones instalables distintas para los distintos sistemas operativos. Además, a nivel comercial se deja de ofrecer un programa que se ha de instalar y mantener en cada usuario y en su lugar se ofrece un servicio que se puede dar de alta o de baja de forma inmediata para cada usuario.

  Como principal crítica tenemos que no podemos aprovechar todo el potencial que tiene programar para un sistema operativo determinado, y que las acciones que podemos realizar sobre el ordenador del usuario son más limitadas. Estos tipos de críticas son similares a las que se realizaron cuando se dejó de programar en código ensamblador para todas las aplicaciones. Hay una gran cantidad de aplicaciones que no demandan todo el potencial de las máquinas de los usuarios y su desarrollo más óptimo es el que se desarrolla por medio de servicios web.

- Uso de tecnologías Rich Internet Aplication (RIA), que se basan en simular la mayoría de funcionalidades de aplicaciones locales que ejecutaríamos normalmente desde nuestro ordenador. Suelen potenciar la ejecución en el cliente dentro de un contexto cerrado y limitado denominado *sandbox*. De esta forma las aplicaciones sufren una restricción de acceso a los recursos locales, Como ejemplo tenemos la programación en Ajax (Javascript + CSS , donde el cliente se comunica con el servidor por medio de XML y recibe XML que interpreta) o los desarrollos en modelo/vista como los realizados con JavaServer Faces (JSF). Estas tecnologías suelen ser ineficaces cuando el usuario tiene un nivel de seguridad alto en su navegador, de forma que no permite ejecutar comandos en Java, Javascript o ActiveX. La mayoría de servicios extra ofrecidos por Google se han implementado siguiendo la filosofía de web 2,0, aunque el buscador no está implementado con esta tecnología.

- Focalización en la socialización en la web. Con esta frase nos referimos a aplicaciones en las que los usuarios participan en la construcción de los contenidos, por medio de sistemas de publicación como pueden ser los blogs. Este apartado también se refiere a aplicaciones en las que los usuarios pueden colaborar entre sí, por medio de aplicaciones colaborativas.

  Dentro de las aplicaciones colaborativas se encuentra aplicaciones de escritura colaborativa como wiki [WWW51]. También incluye sistemas (como por ejemplo, de sindicación como RSS) que permiten la colaboración de varias webs, de forma que unas se alimentan de datos de las otras.

  El principio por el que se impulsan las aplicaciones en las que colabora el usuario se basa en que estas opciones más participativas implican más a los usuarios y hacen que aumente el interés de los mismos en la actividad que se está llevando a cabo por web.

Como podemos apreciar, la definición del término Web 2.0 en sí es ambigua y no define estándares ni pautas fijas para concluir si una web cumple Web 2.0 o no, por lo que algunos autores [WWW70], aunque piensan que las tecnologías que se utilizan son muy útiles, critican el término como un término más bien comercial al que se adhieren muchos con simplemente instalar un blog o por programar con Ajax. Estos mismos autores argumentan que no deja de ser más que un avance dentro de la web sintáctica, ya que la información, excepto la información sobre cómo presentar los recursos, es sólo entendible por los usuarios.

2.2.4 Web Semántica

Ya desde los principios de la World Wide Web se pensaba en esto. Debería haber alguna información entendible por las máquinas que permitiera a éstas realizar análisis más profundos y ayudarnos a las personas en nuestras tareas.

Para esto, ya en el año 1998, [TBL1998] Tim Berners-Lee, uno de los padres de la Web actual, escribió su visión sobre lo que sería la Web Semántica. Una nueva forma de contenido en la Web que tenga significado para las máquinas, que disparará una revolución de nuevas posibilidades [TBL2001].

Podemos definir la Web Semántica como una meta-web que se construye encima de la actual WWW [MAE2001]. Se fundamenta en la idea de tener datos en la Web definidos y vinculados de forma que puedan ser usados por máquinas no sólo a la hora de pintarlo sobre pantalla, sino para tareas de automatización, integración, interpretación y reutilización de datos a través de varias aplicaciones [TBL2001].

El objetivo de la Web Semántica es proveer un marco común que permita que los datos sean compartidos y reutilizados por distintas aplicaciones, compañías y comunidades. Es un esfuerzo colaborativo liderado por W3C con la participación de un gran número de investigadores y socios industriales [WWW1]. En siguientes apartado se describe más a fondo la Web Semántica.

2.2.5 Estructura de capas

Tim Berners-Lee ha sido el propulsor principal de la Web Semántica como se entiende en el World Wide Web Consortia (W3C). Las premisas son:

- **Seguir utilizando la Web actual**. Utilizar los sistemas que ya existen en la Web. Si la Web actual funciona con lenguajes de marcas (Markup Lenguaje), URIs y los sistemas funcionan ignorando las marcas que no entienden, la Web semántica ha de basarse en esos tipos de lenguajes. De esta forma se puede compatibilizar la Web actual con la semántica, añadiendo a la primera anotaciones semánticas.

  Esto nos permite seguir con la gran inercia que tiene la Web como la conocemos ahora y aprovecharnos de ella.

- **Estructura de capas.** La mejor forma de reutilizar es apoyarse en lenguajes conocidos y construir encima de ellos. Esto también conlleva desventajas, ya que en algunos casos se sacrificará legibilidad por parte de los humanos y optimización en la compresión, a cambio de utilizar lenguajes existentes. También algunos autores critican la ambigua relación y separación entre las distintas capas [IAN2003].

  La estructura propuesta por [TBL2000], es la siguiente (Figura 2.1):

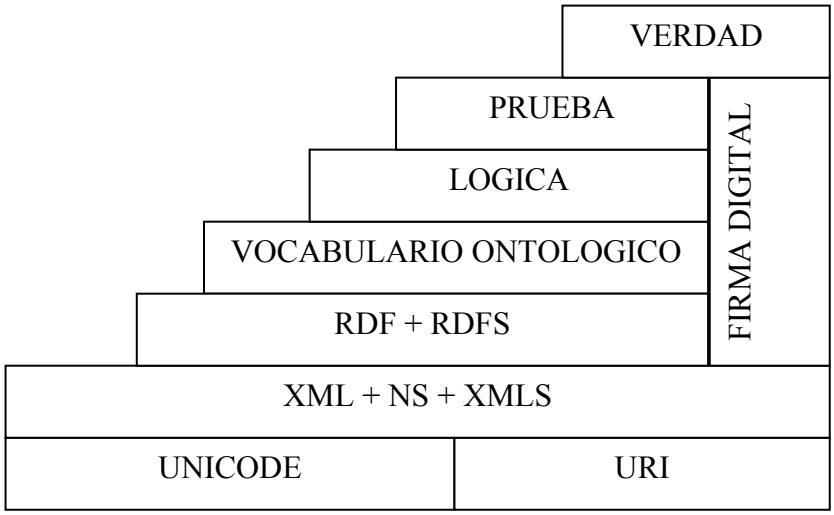


Figura 2.1. estructura de capas de la Web Semántica

A continuación explicaremos las distintas capas, y veremos que la realidad desarrollada varía, sobre todo en las últimas capas, respecto del modelo propuesto.

2.2.5.1 Unicode + URI Símbolos e Identificadores de Recursos

Unicode es el estándar de símbolos utilizado, que permite representar textos en todos los idiomas conocidos, y sin necesidad de inventar reglas auxiliares para caracteres no americanos.

Podemos definir el standard Unicode como el estándar universal de codificación de caracteres usado para la representación de texto en computadores. Dicho estándar es independiente del computador utilizado y la plataforma utilizada, y además permite la representación de caracteres de todas las lenguas escritas del mundo, facilitando la programación multiidioma y multiplataforma. Codifica a su vez de forma estándar acentos, diéresis y algunos símbolos matemáticos,

Unicode se ha definido de modo que es totalmente compatible con el juego universal de caracteres definido en ISO/IEC 10646, y crece a partir del ASCII, ampliándolo a alfabetos distintos al americano. Unicode define tres

formatos de codificación a bits, conocidos como Unicode Transformation Format, UTF-8, UTF-16 o UTF 32, dependiendo de los bits utilizados por carácter, utilizando UTF 32 tamaño fijo para los caracteres y siendo UTF-8 el que permite longitud variable en los caracteres y mayor compresión para almacenamiento de caracteres ASCII.

Una vez definidos los símbolos sobre los que se construyen las siguientes capas, se define una capa que nos permite referirnos a un espacio de objetos o conceptos sobre los que pretenderemos ofrecer información, de forma que los agentes inteligentes sepan a lo que nos referimos. En este caso se recurre a los identificadores que ya existen en la Web, es decir: Uniform Resource Identifiers (URIs).

Las URIs conforman un espacio de identificadores, de forma que una URI [RFC3986] es una secuencia de caracteres que identifica un recurso abstracto o físico. Dichas secuencias de caracteres pueden abreviarse por medio de formas relativas, siendo en todo caso su identificación inequívoca.

El espacio de identificadores representa con la misma sintaxis a todo tipo de recurso (lo que le da el carácter de uniforme), y está abierto a la aparición de cualquier otro nuevo tipo de recurso.

Las URIs identifican los recursos. Si además de identificarlos, esa información nos permite localizarlos, hablamos de Uniform Resource Locators (URLs) [RFC2718].

La principal ventaja de las URIs es que se diseñaron para la web, por lo que no tienen por qué ser centralizadas, aunque algunas (por ejemplo las basadas en http) se basen en sistemas jerárquicos centralizados.

Sin embargo esta especificación, al carecer de sistemas de control, es incapaz de asegurar la unicidad de los identificadores. No se puede asegurar ni rechazar a priori que un par de URIs distintas se refieran al mismo recurso.

2.2.5.2 XML + NS + XML-s. Capa de Intercambio de datos

Extensible Mark-up Language (XML) es el lenguaje que se ha convertido en el lenguaje para el intercambio de datos estructurados en la Web. Se basa en un sistema SGML (ISO 8879), siendo un sistema anidado de marcas. Actualmente la gran mayoría de las Bases de Datos son capaces de aceptar y producir datos estructurados en forma de XML.

La propiedad principal de este lenguaje es la capacidad de transmitir datos estructurados, además de definir la estructura que han de cumplir los datos. Esta definición de estructuras se realiza normalmente, por medio de Document Type Definitions (DTDs) o XMLSchemas (XMLSs).

DTD es el documento que define la gramática de un objeto de datos definido en un documento XML, y nos permite validar un documento XML. DTD utiliza una sintaxis distinta a la de los documentos XML. Por otro lado XMLSchema (XMLS) es un documento que también puede definir la gramática de un objeto de datos definido en un documento XML, con la diferencia que su sintaxis coincide con la de XML.

Se puede afirmar que XMLS es un documento XML, que tiene un DTD que define su gramática, y cada documento XMLS a su vez define la gramática de otros documentos XML. Esto nos permite que, suponiendo que los procesos disponen información a priori de la gramática única de los documentos XMLS, utilicemos el mismo parser para decodificar los datos de un documento y la definición de su gramática.

XML permite que los procesos puedan recibir datos estructurados por medio de XML, y puedan validar su corrección por medio de su correspondiente DTD o XMLs para ver si los datos se han generado y recibido de forma correcta, así como generar una estructura de datos interna igual a la recibida.

Al no haber quedado a priori de acuerdo los programadores sobre cada estructura de datos en concreto, el lenguaje permite una flexibilidad necesaria para la Web, lo cual permite definir nuevas estructuras de datos de forma rápida y descentralizada. Una vez definida una nueva estructura, lo único que hemos de hacer es publicarla, ya que todo documento XML identifica unívocamente la localización del documento DTD o XMLs para su validación.

Los espacios de nombres, Name Spaces (NS) se definen como una URI que representa a un conjunto de nombres, dentro de la estructura de datos en la que están definidos. A nivel práctico sirven para definir sufijos de URIs, de forma que un conjunto de términos comunes y definidos por la misma entidad compartan el mismo espacio de nombres. Por ejemplo:

```
<x xmlns:edi="http://ecommerce/schem">
  <edi:bill>1232</edi:bill>
</x>
```
[Ej. 2.1]

En el ejemplo 2.1 vemos que evitamos la repetición de la URI completa <http://ecommerce/schem:bill> y para facilitar su lectura y compactar la información utilizamos en su lugar <edi:bill>.

Los espacios de nombres son de gran importancia para los lenguajes RDF, RDFS y OWL, ya que cada uno realiza una definición formal de palabras reservadas en su sintaxis, creando un espacio de nombres, que a su vez utiliza los espacios de nombres de los lenguajes de las capas inferiores. Estos espacios de nombres se obvian a lo largo de toda la Tesis para facilitar la comprensión de los ejemplos.

2.2.5.3 RDF + RDF+s. Capa de Aserciones.

Resource Description Framework (RDF) es una recomendación W3C [WWW2], que permite realizar aserciones. RDF ofrece un modelo para representar recursos (que pueden ser valores literales como un valor entero), y relacionarlos entre sí por medio de propiedades. RDF pretende describir recursos, que se representan por medio de URIs, por medio de propiedades (también representadas por URIs). Por tanto definimos una propiedad como un aspecto específico, característica, atributo o relación utilizado para describir a un recurso. Cada propiedad se corresponderá a un significado específico, y se definirán los valores permitidos en cada caso y los recursos a los que puede describir.

Entendemos las aserciones (también llamadas sentencias, declaraciones o enunciados) como el conjunto de un recurso específico, una propiedad y un valor de dicha propiedad. Al recurso que describimos lo denominamos como sujeto, a la propiedad como predicado y al valor de la propiedad como objeto. Por tanto, entenderemos como aserciones en este caso el conjunto <sujeto, predicado, objeto>, que puede ser presentado como un gráfico.

[Ej. 2.2]

```
<Pedro, trabajaCon, Juan>
```

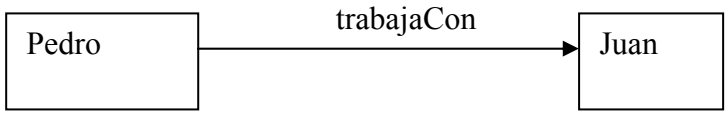
o

```
trabajaCon(Pedro,Juan)
```

o

```
Pedro.trabajaCon(Juan)
```

o



En el ejemplo 2.2 podemos ver tres formas textuales de representar la misma aserción. Además se presenta una forma gráfica de representación de la misma aserción.

Las aserciones describen los recursos (URIs), y se pueden hacer las aserciones que queramos, para dar más información respecto a los sujetos, objetos e incluso propiedades (predicados). De esta manera se consigue describir cualquier recurso, por lo que estamos ofreciendo información respecto al mismo, y así podemos realizar un sistema que proporcione información de forma incremental, aserción a aserción. El recurso que describamos en una aserción puede ser en principio propiedad de otra aserción distinta.

Su sintaxis está basada en XML (su capa inferior) y se compone de unas marcas, describiéndose los recursos mediante los tags <*Description*>, donde el atributo *about* describe al sujeto y cada elemento dentro de la descripción se consideran propiedades de ese sujeto.

[Ej. 2.3]

```
<Description
about="http://www.dcom.upv.es/personal/Pedro">
  <trabajaCon
resource="http://www.dcom.upv.es/personal/Juan">
</Description>
```

Como podemos ver en el ejemplo 2.3, utilizamos la marca <*Description*> para enmarcar la aserción, en la que realizamos la misma descripción que en el ejemplo 2.2.

RDF es un lenguaje maduro con una gran aceptación dentro de las principales entidades relacionadas con la web, y se han desarrollado

multitud de herramientas que permiten realizar consultas (queries) sobre la información en RDF o grafos, mediante estándares de acceso a datos RDF como RDQ, RDQL, SeRQL, SPARQL o N3QL entre otros.

RDF es, además, un lenguaje extremadamente versátil, demasiado para ser un lenguaje ontológico. De hecho, no se reserva palabras clave para distinguir clases (entendidas como una agrupación de objetos), y no distingue entre propiedades y elementos (todo son recursos).

Para resolver esto aparece el RDFSchema (RDFS), que sí que incluye términos como *Class, Property, type, subClassOf, range, domain*, de forma que están incluidos en la semántica del lenguaje y por tanto podremos realizar aserciones cuyo significado entiendan los distintos procesos. A continuación se presentan algunos ejemplos explicatorios.

[Ej 2.4]

```
<Description
about="http://www.dcom.upv.es/general/SeresHumanos">
<rdf:type rdf:Class>
</Description>
```

o

```
<rdf:Class rdf:resource=
"http://www.dcom.upv.es/general/SeresHumanos">
```

En el ejemplo 2.4 vemos dos formas en las que podemos describir a los seres humanos indicando que son una clase. La diferencia es que con RDF se necesita una convención entre el emisor y el receptor de la información para que sepa la semántica de la propiedad *Class*, lo que no es necesario en RDFS por estar incluido ya en el lenguaje.

[Ej. 2.5]

```
<Description
about="http://www.dcom.upv.es/personal/Pedro">
  <rdf:type rdf:resource=
"http://www.dcom.upv.es/general/SeresHumanos">
</Description>
```

En el ejemplo 2.5 vemos la descripción del recurso Pedro, indicando que Pedro es un (objeto perteneciente a la clase) ser humano.

[Ej. 2.6]

```
<rdfs:Property rdf:ID="trabajaCon">
    <rdfs:domain rdf:resource="#SeresHumanos"/>
```

```
    <rdfs:range rdf:resource="#SeresHumanos"/>
  </rdfs:Property>
```

En este caso vemos que, en el ejemplo 2.6 se indica que *trabajaCon* es una propiedad, que relaciona un Ser Humano (sujeto descrito por la marca *rdfs:domain*) con otro Ser Humano (objeto, descrito por la marca *rdfs:range*).

Aunque RDFS permite desarrollar una cierta ontología, no hay restricciones de cardinalidad o existencia. Esta falta de concreción nos limita la semántica del lenguaje, impidiendo, por ejemplo, indicar por medio del lenguaje que todas las personas tienen una madre biológica, por ejemplo.

2.2.5.4 Capa Ontológica, capa Lógica, capa de Prueba

En este punto trataremos de forma conjunta las tres capas, ya que la capa ontológica es la que enriquece la información por medio de una primera ontología en la que se pueden describir relaciones de pertenencia a clases e instancias. Este tipo de aserciones son las que hemos comentado ya resuelve RDFS.

La capa lógica es la que trata de las relaciones lógicas entre los objetos de estudio, generando una ontología más detallada y completa. Un lenguaje ontológico suele dar solución a estas dos capas, como es el caso de OWL.

Por último la capa de Prueba es donde se realizan las comprobaciones de si una suposición es cierta, o se devuelven ejemplos de objetos que cumplan unas restricciones definidas. Es donde trabajan los razonadores como FaCT o Racer.

Estas tres capas enumeradas se describirán con mucho más detalle en los apartados de Lógica y Lenguajes ontológicos, donde hablaremos de lógicas, OWL y razonadores.

2.2.5.5 Capa de Verdad y Firma Digital

La capa de verdad es la que permite decidir qué grado de fiabilidad le concedemos a nuestro razonamiento, basándonos en la confianza en nuestras fuentes de información. Para obtener una buena deducción no es suficiente el uso de un razonador que devuelva soluciones correctas, sino que además tenemos que asegurarnos que confiamos en las premisas.

En la Web la forma de confiar en una información es:

- Averiguar de dónde viene realmente la información, y que dicha información no ha sido modificada desde que se generó.

- Decidir si esa información de ese origen en concreto es fiable para nosotros, normalmente en base a nuestra experiencia o consultado a un tercero sobre la fiabilidad de la fuente.

Aquí es donde aparece la firma digital, ya que permite a cualquier nivel (en principio el estándar es a nivel de XML Digital Signature [WWW68]) decidir si esa información es realmente de quien dice que es, y evitar falsificaciones.

Esto nos permitirá saber que la información viene de donde dice que viene, o si tiene una firma digital que le ha entregado alguna entidad en la que confiemos, como Verisign. También existen en la actualidad sistemas distribuidos de confianza, como el que utiliza PGP [RFC2440].

2.3 Estado del arte en Ontologías para Teleeducación

2.3.1 Introducción

En este apartado trataremos la Teleeducación, entendida como la formación a distancia utilizando la web. La Teleeducación enmarca la utilización de nuevas tecnologías hacia el desarrollo de metodologías alternativas para el aprendizaje.

Para poder abarcar uno de los objetivos de esta Tesis, que es contribuir al desarrollo y mejora de los sistemas de Teleeducación, necesitamos presentar un estado del arte que nos permita confirmar que esta Tesis completa partes aún no alcanzadas en el campo de investigación y que van en línea con las últimas tendencias del mismo.

Además veremos en este apartado las principales teorías sobre el aprendizaje que definen las distintas formas de que se produzca aprendizaje en el sujeto que recibe la formación, ya que cada enfoque recomienda unas herramientas distintas para que se produzca el aprendizaje. Este punto nos permite abrir más las miras de la Teleeducación basándonos en las raíces de los sistemas y métodos de aprendizaje en sí mismos.

Después presentaremos el significado que tiene la formación por la web, las ventajas, y alguno de los inconvenientes que suele acarrear. Continuaremos

describiendo someramente las distintas recomendaciones, especificaciones, modelos de referencia y estándares relacionados con la Teleeducación, tratando sobre todo su relación con las ontologías y los metadatos. Este apartado, en conjunto con los apartados de aportaciones de la Tesis, nos permitirá ver que se ha cubierto áreas todavía no tratadas por otros investigadores de la materia.

A continuación describiremos la ontología relacionada con la educación más aceptada, el estándar IEEE-LOM-1484.12.1, y hablaremos de las posibilidades de la web semántica en la Teleeducación. Es en sí la propuesta del uso de la Web Semántica para mejorar los sistemas de Teleeducación uno de los objetivos principales de la Tesis, y que justifica la posterior definición de una ontología que nos permita este propósito.

Por último describiremos con detalle los foros en la teleeducación y las ontologías utilizadas hasta la fecha para los mismos. En este apartado veremos y justificaremos que para la prueba de concepto de la ontología, que llevaremos a cabo en capítulos posteriores, se desarrollen ejemplos para los foros, ya que ésta es una de la partes de la ontología más novedosa en la actualidad, pese a que los sistemas de Teleeducación cada vez enfatizan más la importancia de las herramientas que fomentan la participación e interacción de los agentes de un proceso de aprendizaje.

2.3.2 Teorías de aprendizaje

La Teleeducación se aplica con fines instruccionales, es decir, se aplica sobre temas que suponemos se pueden enseñar a los alumnos. Se distinguen tres orientaciones del diseño instruccional: Conductismo, Cognoscitivismo y Constructivismo [NEW96].

2.3.2.1 Conductismo (Behavioral)

Se basa en los cambios observables en la conducta del sujeto, y se enfoca en la repetición de patrones de conducta hasta que estos se realizan de forma automática.

Esta teoría aparece a principios del siglo XX con B.F. Skinner, con sus ideas del condicionamiento operante. Postula que el aprendizaje ocurre cuando hay cambios en el comportamiento como respuesta a unos estímulos externos. Se suelen basar en esta teoría los simuladores, que reflejan el entorno de forma que aprendamos a reaccionar bajo unas condiciones determinadas.

En este caso el profesor diseña completamente el entorno educativo y el alumno es un agente básicamente pasivo. Se diseñan unos objetivos del aprendizaje a modo de comportamientos deseados del alumno, y usa consecuencias para reforzar el comportamiento.

La evaluación se suele hacer con tests individuales al final de cada lección, que se puede evaluar de forma totalmente objetiva (por ejemplo, en un test, contando las respuestas acertadas).

2.3.2.2 Cognoscitivismo (Information Processing)

Se basa en los procesos que tienen lugar debajo de los cambios de conducta. Estos cambios son observables y pueden usarse como indicadores para entender lo que está pasando en la mente del que aprende.

Esta teoría surge a mediados del siglo XX con George Miller, con ideas como que la mente humana funciona como un computador, cogiendo información, procesándola, guardándola y relocalizándola de nuevo si la necesita. Esta teoría afirma que el aprendizaje ocurre como un cambio en el conocimiento interno que tiene cada persona, en su memoria.

El alumno realiza tres procesos: atención (fijarse en los ejemplos), codificación-internalización (almacenar el conocimiento extraído en nuestra memoria) y obtener de nuevo ese conocimiento (retrieval) para utilizarlo. Los ejemplos más típicos son los que a partir de ejemplos complejos enseñan la conceptualización de un hecho y luego piden al alumno que explique qué conceptualización han obtenido.

El profesor en este caso de dedica a organizar la nueva información y a buscar enlaces con el conocimiento que ya tienen los alumnos, intentando buscar técnicas para apoyar la atención, codificación y uso del conocimiento.

Para evaluar que el proceso de aprendizaje es correcto, se le suele pedir al alumno que esquematice los conceptos que haya abstraído y que los relacione entre sí (por medio de mapas mentales, por ejemplo).

2.3.2.3 Constructivismo

Se sustenta en la premisa de que cada persona construye su propia perspectiva del mundo que le rodea a través de sus propias experiencias y esquemas mentales desarrollados.

El Constructivismo se reafirma como teoría a finales del siglo XX, y representa un conjunto de ideas como aprendizaje situacional y aprendizaje colaborativo.

En este caso el aprendizaje se hace cuando los alumnos construyen nuevas ideas o conceptos basados en su conocimiento o experiencia anterior (cada aprendizaje depende del entorno social en el que esté el alumno), resolviendo problemas realistas, normalmente en colaboración con otros alumnos.

Las palabras clave en este caso son "experiencia" y "colaborar", y el profesor ha de plantear problemas realistas, complejos y en los que el alumno se identifique. Además ha de impulsar actividades colaborativas entre los alumnos para que nazca el conocimiento entre ellos.

La evaluación de esta última teoría se suele realizar de forma continua, observado el comportamiento de un alumno en un grupo y evaluando los distintos proyectos realizados por el grupo.

Como podemos ver esta teoría es la que da más importancia a la actuación del alumno, y es la responsable de las teorías de centrarnos en el aprendizaje de cada alumno más que en la enseñanza impartida por el profesor, como se hizo para el diseño de los créditos ECTS (European Credit Transfer System), y de iniciativas para fomentar el aprendizaje colaborativo entre individuos (P2PLearning).

2.3.3 Teleeducación

La web es un canal idóneo para el aprendizaje por sus características:

- Interconecta la mayoría de las fuentes de conocimiento instruccional (Universidades, Centros de Investigación, etc.) de los países desarrollados.

- Interconecta, a su vez, a gran cantidad e potenciales alumnos, que si son ingenieros, suelen estar muy familiarizados con el uso de internet.

- El uso de la Web está creciendo conforme el ancho de banda va creciendo y los precios van bajando. En Estados Unidos, por ejemplo, el ratio de estudiantes por ordenador conectado a Internet ha mejorado de 20 estudiantes en 1998 a 5.6 en 2002 [ANS03].

Por estas razones, la creación de contenidos para la Web ha ido creciendo considerablemente, especialmente dentro de las Universidades, siempre tan ávidas de investigar nuevas posibilidades. Sin embargo la mayoría de cursos se han empezado desde cero, y no han podido ser reutilizados para generar otros cursos relacionados.

El coste de crear contenidos es muy alto [SOE96], entre otras cosas porque la formación vía web deja poco espacio para la improvisación por parte del profesor, y esto obliga a generar contenidos muy bien definidos y estructurados. Este alto coste nos hace que con sólo una edición del curso no se amorticen normalmente los gastos de producción, y que la reutilización sea uno de las principales metas en la actualidad.

2.3.3.1 Reutilización

Lo que entendemos por reutilización en este caso es la posibilidad de estructurar un curso con (una o más) piezas de otras fuentes, es decir, contenidos de otros cursos que ya se hayan producido. Estas piezas suelen llamarse como Objetos de Contenido Compartibles (Sharable Content Objects, SCOs) u Objetos de Aprendizaje Compartibles (Sharable Learning Objects, SLOs).

Las características necesarias para facilitar que un SCO sea reusable son [DOD01]:

- **Independencia** entre SCOs, para poder coger los que queramos sin tener que llevar arrastrado al resto. Este punto no es sencillo, ya que decidir a qué nivel un curso es independiente del resto de piezas (nivel de lección, de asignatura, de curso completo…) puede llegar a ser muy complicado [TELE01].

- **Calidad reconocida**, de modo que podamos coger un SCO sabiendo que es bueno sin necesidad de estudiarlo y evaluarlo, porque ya lo han hecho otros. Un punto importante respecto a la calidad es que un SCO pueda mejorar y por tanto tener distintas versiones.

- **Durabilidad**, de forma que cuando hagamos un SCO podamos estar seguros de que encajará en futuros cursos, aunque haya cambios tecnológicos.

- **Interoperabilidad**, de forma que los contenidos puedan ser operados por distintas plataformas de presentación de contenidos, por lo que hemos de separar el proceso de generación de SCOs con el proceso de uso de esos SCOs.

- Permitir su **uso por agentes**. Como los SCOs se pretende que sean muchos y se presenten de forma distribuida (en la web), necesitaremos sistemas que nos permitan localizar los SCOs que más se adapten a nuestras necesidades, por lo que los agentes han de entender las características de los SCOs. En este punto es donde entra la web semántica y los lenguajes ontológicos.

Como podemos comprobar, el desarrollo de una ontología nos puede facilitar las características arriba mencionadas. Una ontología potencia la independencia, ya que la ontología estructurará los cursos y permitirá una fácil separación lógica. También permitirá la interoperabilidad, ya que la ontología es independiente de la plataforma y puede servir de lenguaje común que permita conversiones entre unas plataformas y otras, así como permitir el uso por parte de agentes. Las características de durabilidad y calidad reconocida pueden facilitarse a su vez con fechas de creación y caducidad, así como por atributos que reconozcan a los autores de los cursos.

2.3.4 Iniciativas de Teleeducación

2.3.4.1 Tipos de iniciativas en Teleeducación

En este apartado se clasificarán las distintas iniciativas. Dicha clasificación no es totalmente estanca, sino que puede que alguna iniciativa esté ubicada entre dos clasificaciones [MAK02] (ver Figura 2.2):

- **Especificaciones Técnicas**. Basándose en la Investigación y Desarrollo en eLearning y las necesidades de los usuarios, distintas organizaciones generan especificaciones, denominadas "libros blancos", detallando un modelo de eLearning. Dicho modelo es abstracto. Una de las organizaciones más importantes que genera especificaciones en eLearning es IMS Global Learning Consortium.

- **Estándares**. Algunas especificaciones, cuando han sido comprobados y acreditados, pasan a ser estándares. Si quien acredita es un organismo, que para eLearning tenemos IEEE LTSC, ISO/EC-JTC1/SC36 y CEN/ISSS, se denomina estándar de jure, y si lo que

les acredita es el mercado, al ser el más aceptado por éste, se denomina estándar de facto.

- **Modelos de Referencia**. Son el resultado de concretar más una especificación, de forma que se llega a implementar un modelo más completo (que puede basarse en Especificaciones Técnicas y Estándares, según que partes del modelo, y además recibe feed-back de los que implementan soluciones comerciales, por si es necesario concretar algún punto), incluso proveer una implementación de referencia y herramientas para comprobar si una implementación comercial cumple dico modelo de referencia. Como ejemplo tenemos ADL-SCORM.

- **Implementaciones comerciales**. Son el eslabón final de cadena y representa los productos comerciales, que suelen seguir algún modelo de referencia o al menos alguna especificación, lo que se puede entender como un cierto sello de calidad. No se serán objeto de esta Tesis las implementaciones comerciales, ya las entidades que realizan implementaciones comerciales suelen participar en la definición de los modelos de referencia y especificaciones técnicas.

Como vemos en la figura 2.2 las distintas iniciativas se realimentan entre sí, de forma que puede que una parte de un modelo de referencia se base en un estándar y luego lo mejore, modificando a su vez el estándar en el que se basa. A su vez, una especificación técnica puede ser parte de un modelo de referencia y a su vez un modelo de referencia modificar una especificación técnica que el mercado acepta como mejora de la misma.

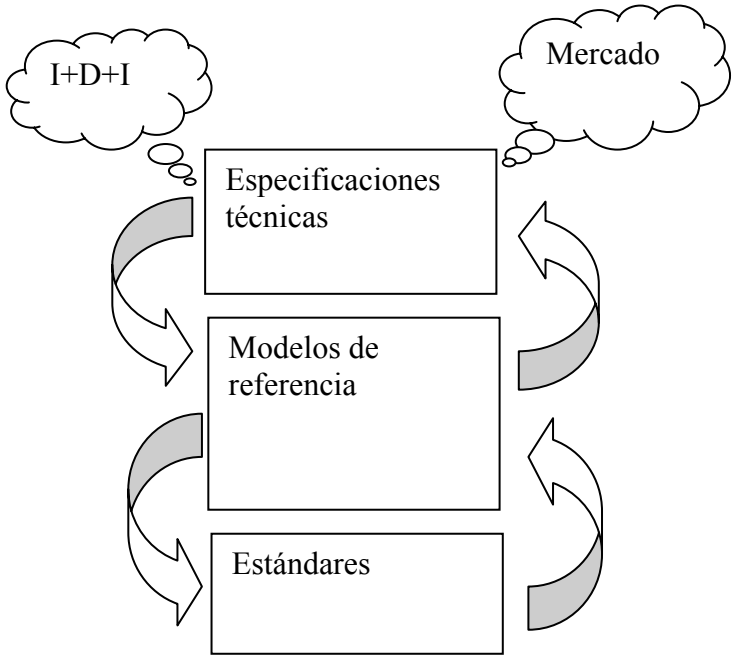


Fig. 2.2. Relación entre iniciativas de eLearning

A continuación revisaremos las iniciativas de teleeducación más destacables en la actualidad. Las más destacables son las provenientes de las especificaciones técnicas de IMS, el modelo de ADL denominado SCORM, y los estándares de IEEE LTSA, realimentándose entre sí. En el observatorio de LTSO [ANI04],[WWW69] podemos encontrar un seguimiento actualizado sobre esta temática.

A partir de la revisión de estas iniciativas se abstraerán las partes más comunes y más interesantes que puedan servir para la definición de nuestra ontología, y se observarán las posibles carencias que pueda suplir la misma.

2.3.4.2 IMS

IMS Global Learning Consortium [WWW15], es un consorcio de empresas e instituciones interesadas en la teleeducación (sus miembros son trabajadores de SUN, WebCT, MIT, Blackboard y varias Universidades).

Su función es publicar y actualizar especificaciones técnicas, entendidas como una serie de servicios que se utilizan para construir un sistema de eLearning. Las especificaciones están en continuo desarrollo, y suele reutilizar a su vez estándares cuando una especificación ha sido aprobada como tal.

Las especificaciones de servicios [IMS03b] se distinguen entre los servicios de la capa común (Common Services Layer, CSL) y los de capa de aplicación (ASL). CSL son los servicios que no son específicamente de eLearning, pero donde se apoyan éstos, como el servicio de identificación. ASL son los específicos de eLearning (evaluaciones, calendario, secuencialidad, gestión de contenidos,…). Destaca sobre todo la especificación de evaluaciones (IMS QTI) [QTI06].

Respecto a sus especificaciones de estructuras de datos (Metadata Specification) se basan en XML [IMS03a], y es en él donde se ha basado IEEE-1484.1.2. (IEEE LOM), que veremos más adelante. Además, IMS define sus entidades como Componentes, que se pueden entender como parte de la ontología implícita en IMS (Competencia, Contenido, Catálogo de Cursos, Objetivo, objeto LOM,…).

Respecto a su apoyo a sistemas colaborativos, incluye un módulo colaborativo que no está muy desarrollado. Respecto a la libertad del alumno a elegir distintos caminos, soporta un servicio de secuenciación (Sequencing), que dice reglas para que el alumno pueda elegir entre varias posibilidades.

Respecto a la búsqueda de contenidos, soporta un Discovery Service, para buscar por toda la web, y en las primeras versiones habla del uso opcional de UDDI (uno de los protocolos sustituidos por OWL-S).

IMS proporciona un Glosario [IMS03c], bastante interesante para estar familiarizado con los acrónimos y entidades más importantes en eLearning.

2.3.4.3 OKI

Open Knowledge Initiative (OKI) [WWW16] es una iniciativa del MIT que define un conjunto de APIs en Java, a modo de modelo de referencia para que se implementen dichas APIs. Hay algunos servicios que son iguales a IMS, por lo que la API de OKI de esos servicios se puede considerar como una implementación de referencia del servicio.

2.3.4.4 IEEE LTSA

IEEE Learning Technology Systems Architecture (IEEE LTSA) es una especificación de una arquitectura de forma muy abstracta (ver Figura 2.3).

En dicha arquitectura hay unos Procesos (los que procesan información), unos almacenes (de contenidos y seguimiento de alumnos), y unos flujos de datos entre los dos grupos anteriores.

Esta especificación de arquitectura se engloba dentro de IEEE 1484 Learning Technology Standards Committee (LTSC), que incluye estándares destacables para los metadatos educativos [LTSC02], para la definición de tests y contestaciones de los mismos [IEEE05], y para la definición del API de comunicación entre el usuario y el Learning Management System [IEEE05].

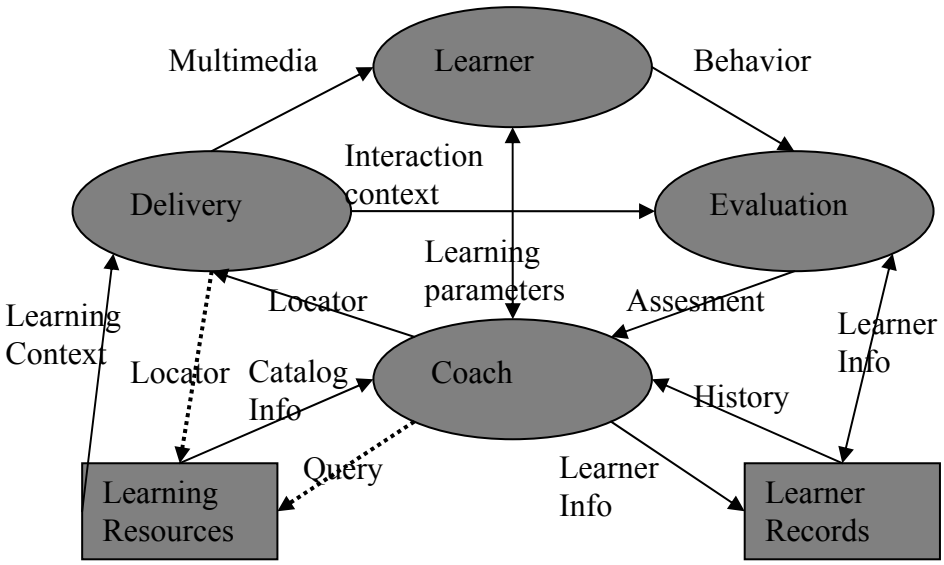


Figura 2.3. Arquitectura IEEE LTSA

2.3.4.5 ADL-SCORM

Advanced Distributed Learning es un consorcio de entidades liderada por el Departamento de Defensa de los Estados Unidos (DoD). A petición del DoD han creado un Modelo de Referencia denominado *Sharable Content Object Referente Model* (SCORM).

SCORM pretende ser el modelo a seguir por la gran mayoría de sistemas de eLearning, ya que su mayor objetivo es la reutilización, y está apoyado por el DoD. Se basa en modelos de referencia de IMS, Aviation Industry Computer-Based-Training Committee (AICC), Alliance of Remote Instructional Authoring & Distribution Networks for Europe [WWW21] (ARIADNE, proyecto europeo financiado por el V programa marco) y en Estándares de IEEE Learning Technology Standards Comité (IEEE LTSC).

SCORM se basa en una arquitectura que separa por completo la producción de contenidos (SCOs) y la distribución de contenidos a los alumnos, que se hará a través de un Learning Management System (LMS). Habla de Experiencia de Aprendizaje (Learning Experience, LE) a lo que entenderíamos como un curso, ya que es un agregado de SCOs empaquetado para ser utilizado.

El LMS debe poder trabajar con los contenidos de cualquier SCO que cumpla el modelo de referencia:

- Servir e inicializar SCOs a los clientes.
- Permitir la navegación a través de los SCOs.
- Tener funciones de comunidad web (foros, chats, etc.).
- Seguir y guardar los datos de estado de un alumno en un momento determinado.
- Comunicarse con el SCO por medio de una API estándar.

Estas características son las que permiten que un SCO pueda interoperar entre distintos LMSs, ya que todos los LMSs que cumplan con el modelo de referencia, sabrán manipular e interactuar con el SCO.

Para realizar una división que permita trabajar sólo con una parte del modelo, el modelo SCORM se basa en tres submodelos:

- **Content Aggregation Model** (Modelo de agregado de contenidos), que describe qué es una LE y cómo empaquetarla, poniéndole los metadatos necesarios para que el LMS pueda explotarlo.

- **Run-Time Environment Model** (Modelo de entorno de ejecución), que nos dice cómo se han de comunicar el LMS y el SCO mientras el alumno está recibiendo la LE.

- **Sequencing and Navigation Model** (Modelo de secuenciación y navegación), que describe cómo puede moverse un alumno por el curso (con más o menos libertad) y define unos objetivos (goals) que se pueden utilizar como reglas para decidir el camino que ha de seguir el alumno. Está basado en el IMS Sequencing Model.

Como resumen vemos que el uso de las ontologías se hace a través de los meta-datos que describen el SCO, que se basa principalmente en IEEE LOM 1484.12.1

Como utiliza IMS para la secuenciación, permite un cierto constructivismo. A nivel de detalles en temas colaborativos, deja sin especificar el tema pasándole toda la responsabilidad a quien implemente el LMS correspondiente, sin generar un modelo de referencia al respecto.

De todas formas aún se encuentra en proceso de desarrollo. La última versión estable (SCORM 2004 2nd Edition) salió el 22 de Julio de 2004, apareciendo ya en septiembre de 2004 una adenda con modificaciones. Actualmente (Mayo 2006) se está trabajando sobre una tercera edición de SCORM 2004, que se encuentra aún en estado de public draft

2.3.4.6 SIF

El Schools Interoperatibility Framework (SIF) es una especificación para la iniciativa K-12 (que pretende introducir el uso de nuevas tecnologías de la información en la educación en países de habla inglesa), para promover la interoperabilidad entre aplicaciones que cumplan la especificación SIF.

Esta estructura pretende que cada aplicación tenga al menos un agente, y que los distintos agentes del sistema se comuniquen pasando por un punto centralizado (con todas las consecuencias que tiene centralizar comunicaciones), de forma que cada agente no tenga que conocer a todos los demás.

2.3.4.7 OpenUSS

OpenUSS [WWW18] es un proyecto que ha desarrollado una especificación y está buscando realizar implementaciones de referencia. Dicha especificación se basa en Java 2 Enterprise Edition y tiene encima unos

componentes principales (Foundation Components) que se encargan de representar asignaturas, alumnos, profesores, etc. y por encima de esta se colocan los servicios horizontales a otros sistemas, como los chats, contenidos para las clases, etc.

2.3.4.8 SUN Microsystems E-Learning Framework

Es una implementación de referencia (define las APIs) y se basa en 4 capas:

- **Presentación**, que es la que lleva el contacto directo con el usuario. Lleva la lógica de la navegación y de la presentación. Se basa en portales.

- Capa de **servicios comunes**, donde se incluyen también los servicios de comunicaciones (e-mail, foros, web cast, white-boards y chats)

- Capa de **e-Learning**, que lleva la parte de exámenes, simulaciones y trabajo en grupo, entrega de contenidos, y herramientas de autor para generar contenidos.

- Capa de **recursos**, que tiene los repositorios de contenidos y los meta-data para poder obtener esos contenidos. Permite el uso de meta-datos de IMS y de Dublín Core Metadata. También tiene repositorios de evaluaciones y de información de los alumnos y otros roles.

Esta separación en capas se representa por medio de un conjunto de APIs que se relacionan entre sí y con la capa de presentación por encima del resto, siguiendo el paradigma modelo-vista. Además dispone de una implementación de referencia que permite ver sus funcionalidades.

2.3.4.9 Otras Iniciativas

Existen otras iniciativas entre las que se puede destacar el EU UNIVERSAL Project (proyecto financiado en el V programa marco), que buscaba construir redes de búsqueda y entrega de materiales por medio de un broker educacional. Se puede ofrecer contenidos y pedir contenidos, y se basa en SOAP/WSDL y JXTA/EDUTELLA [WWW19].

El Gobierno del Reino Unido, dentro de su política de introducción de nuevas tecnologías en todos sus servicios, ha sacado UL Electronic Governement Interoperatibility Framework (eGIF), que se basa en gran medida en el modelo de IMS [WWW20].

2.3.5 Ontologías en Teleeducación

Como ontologías en educación nos referimos a las ontologías que se aplican a los componentes de un curso on-line. Existen otras ontologías en educación como la de Knowledge On Demand (KOD), que utiliza una ontología en XML pero para describir las distintas áreas de conocimiento [SAM02].

Una vez enmarcado el ámbito, encontramos tres ontologías destacables: Dublin Core Group, ARIADNE, e IEEE LOM.

La primera es una de las más antiguas y un esfuerzo centralizado de crear unos metadatos globales. Dublín Core Group [WWW21] se creó en 1999 en Ohio, y define una serie de conceptos globales como *creator*, *date*, que siguen utilizándose en otros lenguajes (como en las anotaciones de OWL). Se basa en 15 elementos principales: título, creador, tema, descripción, publicador, colaborador, fecha, tipo, formato, identificador, fuente, idioma, relación, cobertura y derechos. Su sencillez y generalidad hace que se reutilicen en otras muchas ontologías y lenguajes.

ARIADNE (Alliance of Remote Instructional Authoring and Distribution Networks for Europe) es una asociación internacional apoyada institucionalmente por los programa Marco Europeos y su fin es fomentar el intercambio de experiencias en el área de la educación abierta y a distancia. Para realizar esto, centra su labor en la elaboración de esquemas para los metadatos educativos (llamado en este caso ARIADNE Metadata Set, AMS), bastante parecido a la versión de IEEE que veremos a continuación (incluso existe una tabla de equivalencias entre ambos).

La última, más aceptada como hemos visto en las iniciativas, es el IEEE LTSC LOM (IEEE-LOM-1484.12) [WWW23]. Este modelo de objetos de aprendizaje (Learning Object Model, LOM) divide la información en 9 subgrupos, y además existe una especificación para trascribirlo en ISO/IEC 11404 (IEEE-LOM-1484.12.2), XML (IEEE-LOM-1484.12.3) y RDF (IEEE-LOM-1484.12.4) (Binding). El sistema es bastante completo (Se basó en la recomendación de IMS, que a su vez luego acepto IEEE LOM como su base de definición).

IEEE-LOM utiliza varias estructuras comunes para algunas de sus propiedades:

- Para la descripción de roles, como los autores utiliza tripla de las propiedades *Role*, *Entity* y *Date*. La propiedad *Role* indica el tipo de rol, mientras que *Entity* identifica a la persona y *Date* la fecha de ese rol.

- Para la utilización de clasificaciones ya existentes, utiliza la pareja de propiedades *Catalog* y *Entry*. La identificación de la URI de la clasificación se encuentra en la propiedad *Catalog*, mientras que el valor dentro de la clasificación se introduce dentro de la propiedad *Entry*.

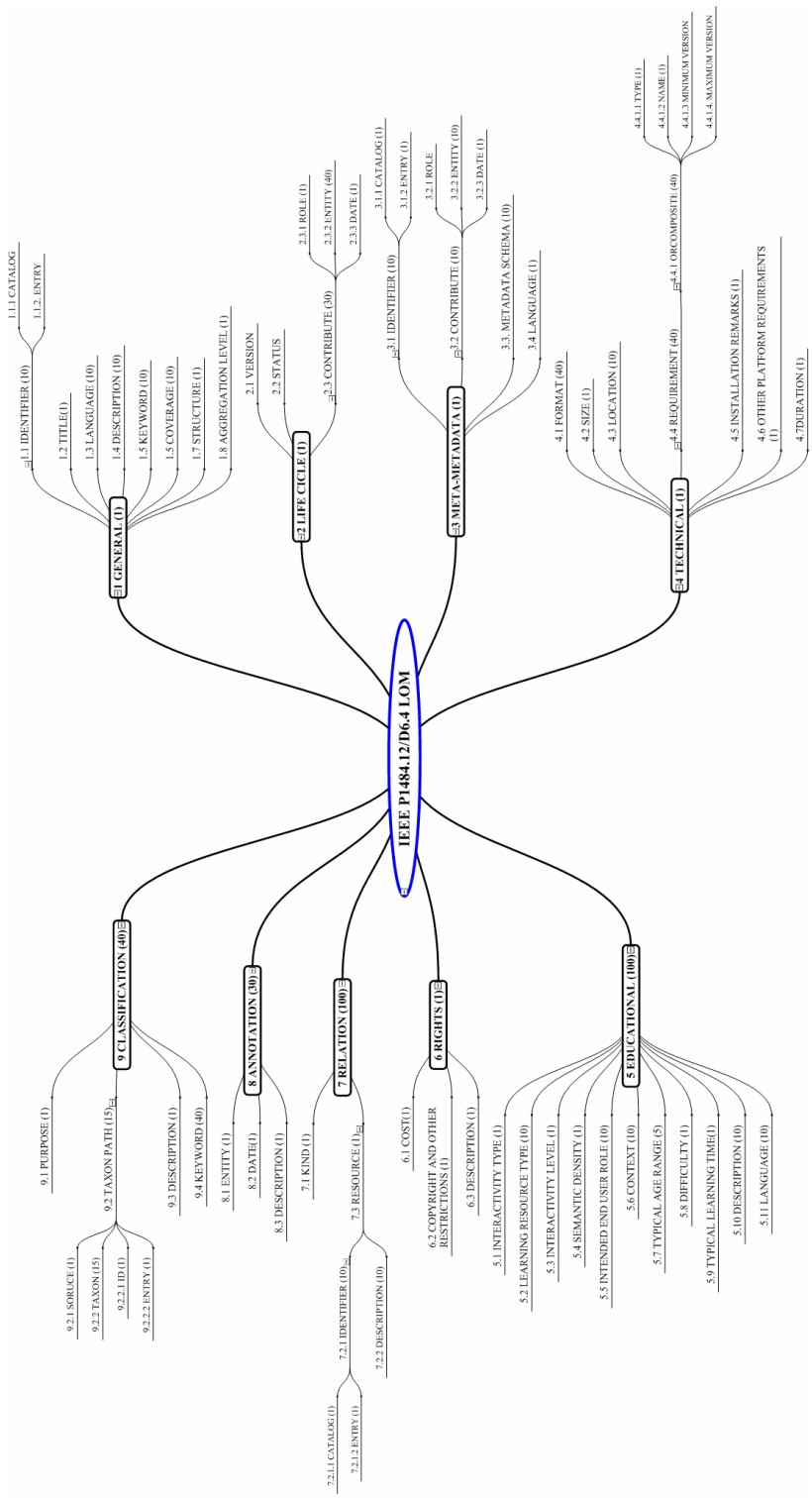


Figura 2.4. Taxonomía de IEEE LOM

Como podemos ver en la figura 2.4, según el modelo un objeto de aprendizaje tiene las siguientes propiedades:

- Los **datos generales**, título, lenguaje, palabras clave, estructura y nivel de granularidad. Este conjunto de datos nos permitiría en su caso búsquedas por texto, sobre todo para el título y las palabras clave.

- **Ciclo de vida**, donde controla versiones, estado y autores. Esta información nos permite distinguir entre distintas versiones, así como poder solicitar la última versión que esté en estado terminada y revisada.

- **Meta-metadatos**, son los datos a los metadatos que estamos describiendo. En este punto se añaden también los autores de dichos metadatos.

- **Datos Técnicos** de formato, tamaño, requerimientos (y cómo instalarlos), incluyendo información sobre la duración. Los requerimientos técnicos pueden ser múltiples y además poder elegir entre varias posibilidades válidas por medio de la propiedad *orcomposite*.

- **Datos educacionales**, tipo de interactividad, tipo de objeto de aprendizaje (pregunta, una clase, una figura, etc), densidad semántica (relación entre el tamaño y la cantidad de información para el aprendizaje presentada), perfil del alumno, contexto, dificultad.

- **Derechos de propiedad**. Copyright, donde se describen los distintos derechos de uso así como los precios relacionados para los mismos. Este punto es uno de los menos desarrollados por IEEE LOM.

- **Datos de relaciones**, habla de los Learning Objects con los que se relaciona (usa metadatos de Dublin Core, *ispartof*, …). Este apartado nos permitirá que cuando exploremos un elemento sepamos, y por tanto podamos explorar también, los objetos de aprendizaje relacionados.

- **Anotaciones**, son comentarios entre los creadores de contenidos. Son campos bastante abiertos para anotaciones dirigidas a seres humanos y no entendibles por los posibles agentes inteligentes.

- **Clasificación**, sobre qué temática se habla, si es una disciplina, una idea, una competencia,…. Permite definir varias clasificaciones, cada una con su propiedad *Purpose* que indica respecto a qué se clasifica, como por ejemplo al tipo de pedagogía utilizado, incluyendo también una información detallada dentro de la propiedad *TaxonPath*, donde se indica un localizador a la clasificación particular en la propiedad *Source* y un valor dentro de esa clasificación en la subpropiedad *Path- Entry (ver figura 2.4).*

Como podemos apreciar, IEEE LOM cubre muchos aspectos de descripción de un objeto de aprendizaje, entendiendo el mismo como un contenido fijo creado por algún autor o grupo de autores. Destaca su gran versatilidad al permitir distintas clasificaciones con su tupla de propiedades *Catalog* y *Entry*.

2.3.6 Estado del arte en las interacciones de tipo Foro

En este apartado veremos una de las interacciones usada no solamente en el ámbito de la formación on-line, sino en otros ámbitos de la web: los foros. Este apartado aparece para demostrar con más detalle el desarrollo de las ontologías y clasificaciones en los foros, donde veremos la falta de desarrollo y estandarización que actualmente existe.

Los foros son una herramienta que permite gestionar comentarios y relacionarlos entre sí, pudiendo visionar dichos comentarios desde la World Wide Web. Se desarrollaron a partir de las listas de correo (mailing lists) donde se empezó a contestar a correos de la lista de forma que se generaba un cierto hilo conversacional.

Normalmente un foro es uno o varios hilos (threads o topics) sobre los que los usuarios realizan comentarios o contestan a comentarios de otros usuarios. En los siguientes apartados veremos las funcionalidades que suelen llevar aparejadas.

Los foros son una herramienta de interacción que se sitúa entre la correspondencia de e-mail y las conversaciones síncronas por medio de chats. Permite el almacenamiento de la historia de la conversación, así como la existencia de ciertos moderadores de los foros.

Los distinguimos de los blogs (weblogs) ya que normalmente los blogs son un conjunto de artículos alrededor de una temática que mandan usuarios y que son comentados (por un sistema similar al foro) posteriormente por otros usuarios y el autor. Aunque algunos foros permiten que se te envíe por e-mail las interacciones de otros usuarios, se distingue de las listas de correo porque la acción más común es la visita del foro más que la espera pasiva a recibir mensajes. De hecho, debido a que el tiempo para leer e-mails es limitado, las listas de e-mail se vuelven inapropiadas en temáticas en las que se reciben gran número de interacciones.

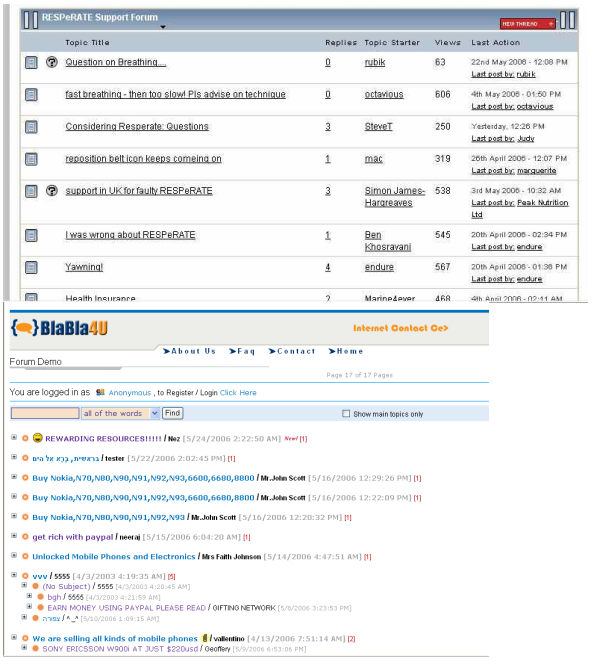


Fig. 2.5. Ejemplos de estructuras visuales de árboles con threads

El nacimiento de los foros como los conocemos actualmente se da en 1996 con Ultimate Bulletin Board [WWW41], con el nacimiento de la estructura visual de árbol con threads (ver Fig. 2.5) (de hecho, muchos foros aún son considerados aún como Bulletin Board Systems o BBS). Aún así hubo otras herramientas de conversación como los newsgroups y de intercambio de información como los BBSs (que funcionaban por modem antes de Internet), pero consideraremos los foros como los foros que actualmente se muestran por la web.

Hoy en día la mayoría de plataformas de teleformación ofrecen herramientas de tipo foro, aunque la utilización de foros sobrepasa el ámbito del

aprendizaje y podemos encontrar aplicaciones dedicadas exclusivamente a la gestión de foros, desde aplicaciones Open Source como phpBB [WWW42] (basada en php y preferentemente base de datos MySQL) hasta aplicaciones de pago como BlaBla4U [WWW42] (basado en .asp y Microsoft SQLServer) o FuseTalk [WWW43] (programado en ColdFusion y con una versión para ASP.NET). Podemos encontrar en [WWW44] una interesante comparativa sobre aplicaciones de foros con fecha de noviembre de 2005.

2.3.6.1 Motivación de los Foros

Los foros nos permiten interactuar por medio de comentarios enlazados con los distintos grupos de interés de la comunidad (alumnos y profesores en un curso, miembros en un foro temático, etc). Estas interacciones nos permiten:

- Generar una base de conocimiento, ya que los distintos comentarios y conversaciones quedan almacenados en el sistema, pudiendo buscar en el mismo en la mayoría de los casos. Es el caso de la comunidad de desarrolladores en Java de Sun (JavaDeveloper) [WWW46], donde se pueden realizar búsquedas por los distintos foros. Muchas veces se genera una base de conocimientos más elaborada, donde se revisan las cuestiones más importantes planteadas y resueltas en los foros y se incluyen a modo de preguntas frecuentes (faqs).

- Fomentar la comunicación entre el individuo y el grupo, generando un sentimiento de pertenencia y reconocimiento. Por ejemplo, los alumnos de los cursos on-line pueden contactar y hacer preguntas a la comunidad de aprendizaje que consideran interesantes para los demás, incluso contestar a preguntas de otros alumnos, realizando un cierto reconocimiento dentro del grupo. Esto hace que se faciliten la segunda y tercera interacción básica según Moore [MOO89]: aprendiz-experto (tutor) y aprendiz-aprendiz.

- Para los cursos on-line los foros permiten al tutor añadir información extra o actualizaciones cuando el curso ya ha arrancado. Por ejemplo, el cambio de una ley a mitad de un curso sobre fiscalidad puede ser comentado por el tutor cuando algunos alumnos ya hayan pasado la unidad que ha quedado obsoleta.

- Sirven para sondear a la comunidad, sea de un curso on-line o no. Por ejemplo si una unidad no está clara aparecerán comentarios al

respecto en el foro, o si algunos alumnos no están satisfechos probablemente lo indiquen en el mismo. Esto puede servir durante la impartición del curso para ir modificando alguna de las acciones o poniendo más cuidado en la tutorización si es necesario.

- Puede generar una comunidad de aprendizaje a su alrededor. Por ejemplo, un foro sobre programación de php puede concentrar tanto expertos como novicios en la temática de forma que pueden interactuar de forma algo más informal y permitir un tipo de formación muy dirigida, incluso produciendo conocimiento como se indica en la teoría de la espiral de Nonaka [NON95] (Ver figura 2.6), donde se va generando conocimiento pasando entre conocimientos tácitos (los que no se han expresado formalmente) y explícitos de forma matricial. En este caso corresponde a un proceso de socialización-externalización. De hecho, en algunos casos los propios usuarios de un curso on-line solicitan el mantenimiento del foro durante mucho más tiempo que el curso.

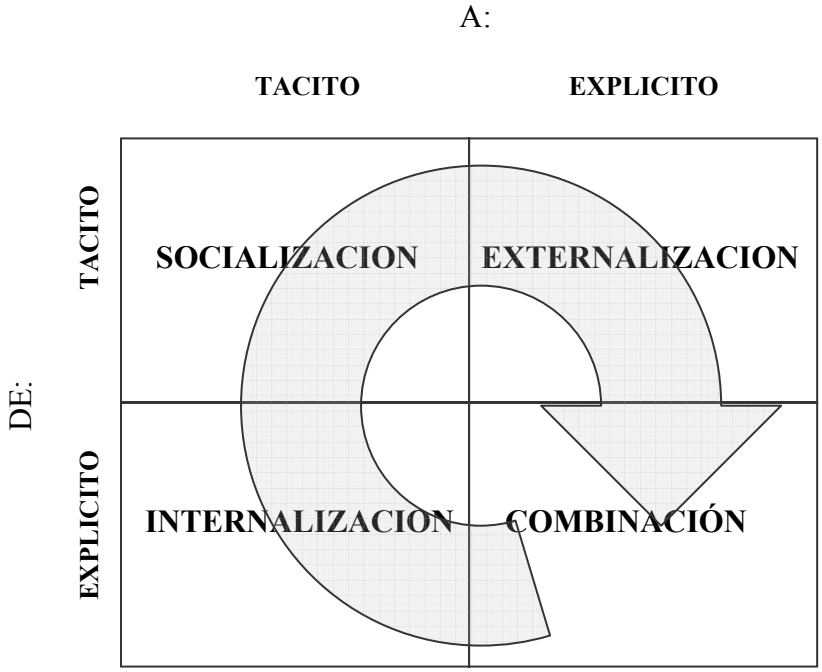


Fig. 2.6. Espiral de Nonaka

2.3.6.2 Limitaciones de los Foros

Las propias características de los foros acarrean ciertas limitaciones en algunos casos:

- Introducción en el foro de comentarios poco interesantes o no relacionados con el área de interés del mismo. Esto genera cierta información no útil en el foro que dificulta la información interesante. Un caso típico son los spams realizados a foros (comentarios publicitarios no relacionados con la temática colocados por personas o agentes de spam que escanean la web en busca de foros). Otro caso típico es utilizar el foro para comunicaciones entre usuarios que no son interesantes para la comunidad en sí, sino sólo para los dos individuos que siguen la conversación.

  Para evitar esta información no útil se suele recurrir a la figura del moderador del foro (que pude decidir qué introducir y qué no en el foro, o borrar comentarios, o indicar a usuarios que sigan su comunicación en privado por medio de chat o e-mail).

- Introducción en el foro de contenidos ofensivos u opiniones a modo de calumnias. Estos comentarios se suelen realizar al amparo del anonimato de algunos foros y genera la falsa impresión que todo el foro comparte esas ideas o calumnias. Algunos de estos casos son investigados judicialmente como es el caso de e-valencia.org en mayo de 2005.

  Estos comentarios se suelen evitar con el uso de un moderador y no permitiendo el completo anonimato en los foros. De todas formas hoy en día se realiza el seguimiento judicial en España ya que la LSSI [LSSI02] obliga a los sistemas a almacenar las IPs y fechas de conexión al sistema durante al menos un año.

  Este punto y el anterior en principio se basan en el seguimiento de [RFC1855], conocida también como nettiquete y que regula el buen comportamiento en las comunicaciones uno a uno y uno a un grupo.

- Tiempos entre comentarios y contestaciones. Al ser un tipo de conversación asíncrona, la contestación a un comentario normalmente no es instantánea. En estos casos el posible actor que replique ha de entrar en el foro, mirar el comentario y estar

interesado en contestar a la misma. Luego el sujeto interesado ha de acceder de nuevo a leer la respuesta.

Para acortar estos tiempos, algunos foros envían avisos por medio de e-mails a los usuarios cuando se contesta a un comentario suyo o cuando se realiza una interacción en un thread o comentario en el que nos hemos registrado como interesados. Los usuarios de los foros han de acostumbrarse también a realizar comentarios que sean interesantes para el grupo para obtener respuestas.

- Dificultades para fomentar la participación. Sobre todo en las comunidades sobre cursos on-line, a veces es difícil arrancar conversaciones hiladas entre los alumnos, que se limitan a leer los comentarios ya existentes (lurking), leer los contenidos del curso y aprobar el examen en su caso. De hecho hay cursos en los que las únicas interacciones en los foros son de los tutores para informar a la comunidad sobre algún tema.

  Para evitar la pasividad en los foros los tutores han de animar a los alumnos a participar, o incluso recurrir a otra figura como el facilitador que realiza estas tareas de animación y seguimiento general [MON02].

2.3.6.3 Caracterización de los foros

En este caso veremos dos clasificaciones de los foros: según las características que implemente y según las interacciones que contenga.

Según características implementadas tenemos:

- **Foros con / sin moderador**. Debido al esfuerzo que significa moderar todo un foro, en algunos casos puede haber un moderador centralizado o con posibilidad de poder delegar la moderación para algunas partes del mismo.

- **Foros independientes / embebidos** con otros recursos. Los foros relacionados con otros recursos suelen ser por ejemplo los relacionados con un curso on-line, o incluso podemos considerar que algunos blogs tienen un foro relacionado con los artículos publicados. Los blogs, por sí mismos también pueden dar lugar a conversaciones (artículos que comentan otros artículos) entre blogs distintos (un blog A comenta el texto de un blog B), siguiendo la

especificación TrackBack [TRO04]. Comentaremos esta especificación más adelante.

- **Generales / Personalizados** para los usuarios. La personalización puede indicarte los comentarios aún no leídos, avisarnos por e-mail cuando haya contestaciones a nuestros comentarios o incluso recordar mis preferencias de navegación y lay-outs.

- **Grandes y generales / pequeños y enfocados** a una temática. Normalmente los grandes y generales pretenden una socialización más allá de una disciplina determinada (como ocurre en Japón con 2Channel [WWW48]). En Europa y Estados Unidos se extienden más los pequeños y enfocados como pueden ser foros de programación en php [WWW47].

- **Sólo textuales / foros que permiten más tipos de recursos**, como imágenes, links, archivos adjuntos. Los textuales normalmente son más ligeros y permiten un mayor número de interacciones, mientras que los otros permiten una mejor expresión de los comentarios.

Por otro lado, basándonos en las interacciones que contiene [DRI04], podemos realizar la siguiente clasificación:

- **Nivel de interacción de los foros**: midiendo cantidad y frecuencia de nuevos comentarios y respuestas a los mismos (dos indicadores separados). Esta característica se puede dividir por perfiles en los cursos on-line (tutores, facilitadores, aprendices).

- **Nivel de Centrado en la temática** del foro: midiendo los mensajes relacionados con la temática frente al total de mensajes.

- **Tiempo medio de respuestas**: midiendo horas/días entre que se realiza un comentario que necesita respuesta y la respuesta si aparece.

- **Dedicación media a revisar otros comentarios**: tiempo medio de los usuarios (en caso de ser identificado) visitando el foro y no interactuando en el mismo. Se pude medir también por medio de medir la media de comentarios visitados por cada usuario.

- En los cursos on-line se puede medir las **relación entre interacciones alumno-alumno, alumno-tutor y tutor-alumno**. Sobre todo es importante la interacción entre alumnos, que indica la creación de una comunidad de aprendizaje.

2.3.6.4 Metadatos y Ontologías en los Foros

Una vez descritos los foros así como los distintos tipos y características de los mismos, pasamos a enumerar los trabajos relativos a la generación de ontologías y metadatos de los foros. Así como para los contenidos estáticos y los cuestionarios hay gran cantidad de propuestas para la interoperabilidad, no encontramos apenas ninguna información de los foros, debido a dos causas:

1. Se prioriza la interoperabilidad en el siguiente orden:

   a) de los contenidos estáticos del curso (páginas html, algún recurso en javascript, imágenes, videos, etc), como ocurre en SCORM Content Agregation Model (CAM) [ADL04a].

   b) de la estructura del curso y navegación (unidades y forma de navegar), como ocurre en SCORM S&N [ADL04c].

   c) de la interacción del alumno con los contenidos (pruebas y exámenes). En este caso SCORM lo recoge en su libro sobre Run Time Environment [ADL04b].

   Como no se han resuelto aún los puntos anteriormente citados, la interoperabilidad de los resultados de interacciones entre los miembros de la comunidad de aprendizaje (como los foros) no se han estudiado aún. De hecho, por ejemplo el referente actual SCORM indica que no soporta ningún tipo de información relativa a las interacciones colaborativas de los cursos [ADL04c] ni a nada que no sea el aprendizaje individualizado alumno-sistema [REB04].

2. Se considera poco útil la exportación (interoperabilidad) de las interacciones de un curso, por no haber encontrado aún beneficios que lo justifiquen.

   Lo único que se ha encontrado que puede ser considerado como posible exportación de foros, es la exportación de los diferentes skins

(apariencias, traducciones a otros idiomas) de los foros así como la particularización de foros generales (por ejemplo, un foro en el que sólo puede abrir threads el administrador)[MODX06]. Esto permite que un sistema que está ya particularizado por medio de un archivo XML que describe las modificaciones (MODX) se pueda actualizar con nuevas versiones del software de foros y al aplicar de nuevo el MODX quede de nuevo particularizado (ver Fig. 2.7).

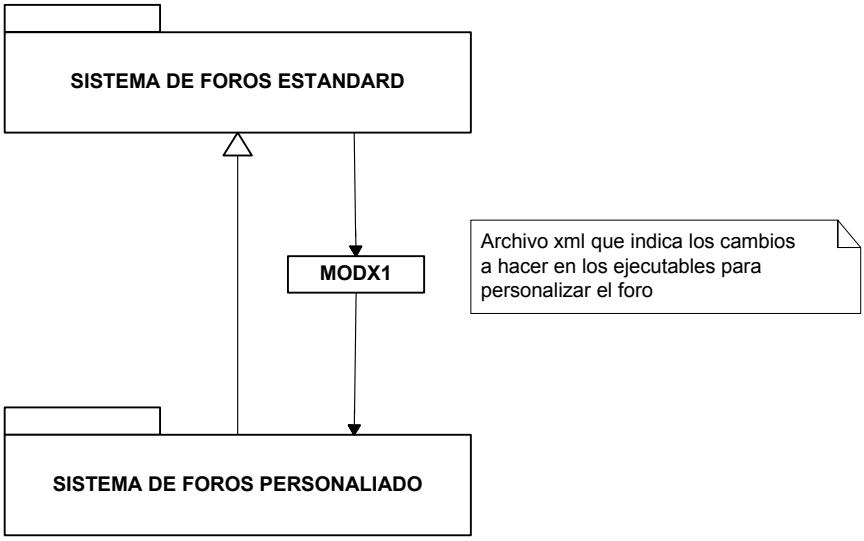


Fig. 2.7. Ejemplo de funcionamiento del MODX de BBphp

Pese a no existir más implementaciones, consideramos que la exportación de metainformación de los foros y de los foros en sí puede generar múltiples beneficios:

- Como hemos comentado anteriormente, un foro bien gestionado genera una base de conocimiento que podríamos aprovechar en otras imparticiones del mismo. De hecho, una forma rudimentaria de hacerlo es generar un apartado de preguntas frecuentes a partir del foro.

  El mantener un foro para sucesivas ediciones del curso nos puede servir para enriquecer al mismo. Eliminar directamente un foro de un curso es como si perdiera la memoria y experiencia obtenida por medio de someterse a una impartición del mismo.

- Nos permitiría ampliar las búsquedas, ya que no sólo buscaríamos por los contenidos de los cursos, sino por los topics de los foros (la

temática si la marcamos) o incluso por los contenidos textuales de los mismos foros.

Por ejemplo, si queremos buscar la solución a un problema concreto, es más fácil que lo encontremos en un foro que encontrarlo explícitamente en los contenidos estáticos de un curso. Mientras en los contenidos de un curso aparece información estructurada sobre un área de conocimiento, en los foros aparece información completamente desestructurada pero muy orientada a los intereses de los miembros de la comunidad.

2.3.7 TrackBack

El ejemplo más similar a una ontología para los foros ha sido el sistema de enlace entre blogs distintos implementado por TrackBack. Trackback apareció como primera versión en 2002 y permite enlazar artículos de distintos blogs a modo de conversaciones (como si fuera un foro distribuido).

TrackBack nos permite avisar a un Blog A de que alguno de sus artículo se comenta en un artículo del Blog B. Si bien la mayoría de los blogs permiten añadir comentarios a un artículo directamente, esto puede que no sea la solución más adecuada porque:

a) produce una centralización de la información. A veces hay blogs que no están interesados en almacenar comentarios de otros usuarios ajenos al blog.

b) resta importancia al segundo artículo, ya que queda como un simple comentario al primero y no como una contribución del segundo autor (en su blog). Además, hace que el segundo artículo esté condenado a desaparecer si el primer blog desaparece.

c) permite comentar de forma estándar artículos de otros blogs. Esto nos permite comentar los mejores artículos, que normalmente se encuentran distribuidos en distintos blogs

Trackback se puede considerar como una forma de averiguar quién referencia nuestros recursos (linkback), y así poder ver más información relacionada con los mismos.

2.3.7.1 Funcionamiento de Trackback

Trackback funciona basándose en la arquitectura Representational State Transfer REST [FIE00]. Se basa en HTTP y XML, de forma que el cliente manda al servidor una petición por POST y el servidor le contesta con un archivo XML.

En el caso de Trackback el funcionamiento es el siguiente:

1. Un usuario genera un artículo (artículo B) comentando un artículo de otro blog (artículo A).

2. El usuario localiza una URL permanente del artículo (permanent link) del artículo al que referencia. Indica a su sistema que quiere referenciar ese artículo.

3. Se realiza un mecanismo automático de descubrimiento de la URL de Trackback del artículo A (trackback ping URL).

4. El blog del artículo B actúa como cliente y manda un HTTP POST a la Trackback ping URL indicando cierta metainformación sobre el artículo B.

5. El cliente HTTP contesta por medio de un archivo XML donde indica si todo ha sido correcto o si ha habido algún error.

La forma de averiguar la trackback ping URL a partir de la URL permanente se hace por medio de información en RDF insertada en la URL permanente, donde indica metainformación sobre el artículo A entra la que se encuentra su ping URL.

Pingback [LAN02] funciona de forma similar, sólo que utilizando XML-RPC y que coloca dentro de la página del blog A una marca <link rel="pingback" href="ServidorPingBack"> (o una propiedad en la cabecera del HTTPResponse).

Podemos ver el funcionamiento en la figura 2.8.

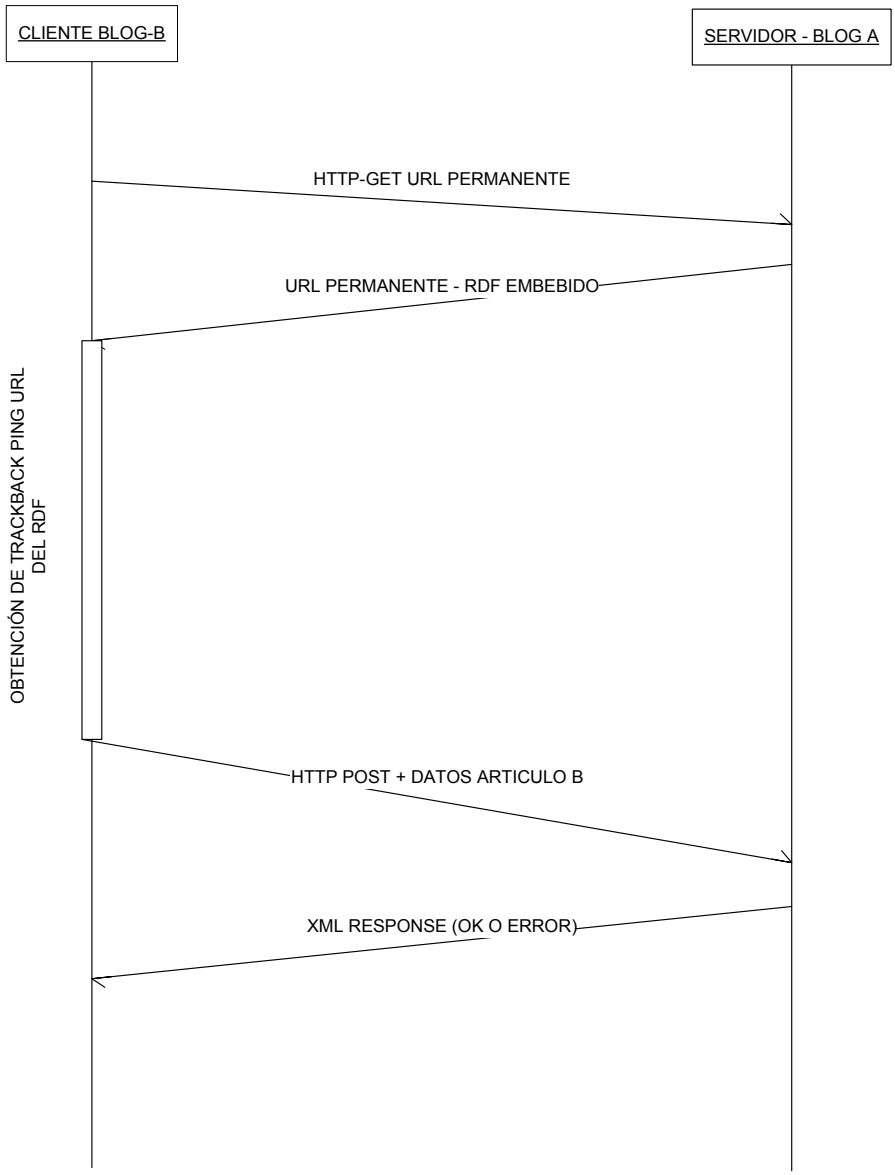


Fig. 2.8 funcionamiento de trackback

2.3.7.2 Metainformación en Trackback

Trackback encapsula la metainformación de los artículos de dos formas distintas:

- Para el artículo de origen (el que se referencia) en el propio artículo se incrusta una descripción en RDF utilizando campos de Dublin Core. En el ejemplo 2.7 podemos ver un ejemplo de RDF incrustado.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

xmlns:trackback="http://madskills.com/public/xml/rss/module/trackback/"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description

rdf:about="http://www.w3.org/QA/2005/08/variability_in_specifications.html"
    trackback:ping="http://www.w3.org/QA/sununga/mt-tb.cgi/7"
    dc:title="Variability in Specifications published as a W3C Working Group
Note"

dc:identifier="http://www.w3.org/QA/2005/08/variability_in_specifications.ht
ml"
    dc:subject="Publications"
    dc:description="Variability in Specifications, a companion to the QA
Framework: Specification Guidelines, has been updated and published as a W3C
Working Group Note. The Note contains advanced specification design
considerations and conformance-related techniques. It describes how design
of a specification&amp;apos;s conformance..."
    dc:creator="karl"
    dc:date="2005-08-30T00:46:17+00:00" />
</rdf:RDF>
```

Ej. 2.7. RDF en http://www.w3.org/QA/2005/08/variability_in_specifications.html

- Para enviarla al servidor en el HTTP-POST. En este caso no se utiliza ningún formato más que el del propio POST, indicando los siguientes campos:

  a) Título (title) del artículo B.

  b) Resumen (excerpt) del artículo B. Normalmente el resumen lo especifica el autor o se coge una cantidad de texto del principio del artículo (como en las sindicaciones RSS y ATOM).

  c) URL permanente del artículo B (url). Es el único campo que es obligatorio. En cierto modo indica al padre (Trackback ping URL con la que conecta) el nombre de un artículo hijo suyo (referencia al anterior).

  d) nombre del blog donde está el artículo B (blog_name).

Como podemos ver la metainformación de trackback es muy limitada y no sigue estándar de web semántica (RDF) en la descripción de una fuente a referenciar, cuando se envía la información del artículo B que referencia al anterior se realiza una comunicación HTTP POST.

TrackBack de todas formas sirve para lo que fue diseñado (informar a un artículo que otro articulo le referencia). Por tanto, hasta donde alcanza nuestro conocimiento, no existe ninguna ontología que trabaje con los foros de forma explícita, lo que confirma que realizar la prueba de concepto con la subontología que generemos de foros será un trabajo único dentro del campo en el que estamos trabajando.

Capítulo 3: Lógica y Web Semántica

3.1 Introducción

En este capítulo introducimos los conceptos y explicaciones principales respecto a las teorías de lógica y raciocinio, lo que nos permitirá:

- Conocer las teorías que dan lugar a la lógica descriptiva que se utiliza como base para el lenguaje ontológico elegido (OWL DL).

- Entender la potencia y las limitaciones de los lenguajes ontológicos, así como los algoritmos de razonamiento en los que se basan los razonadores actuales.

- Enmarcar el lenguaje OWL DL dentro de los lenguajes lógicos existentes en la actualidad para la Web.

- Conocer los detalles sintácticos y semánticos de los principales lenguajes lógicos, y en particular los de la familia OWL (Lite, DL y Full).

Es decir, en este capítulo asentaremos las bases que nos permitirán entender OWL DL y enmarcarlo dentro de las distintas lógicas existentes, donde podremos ver que es una de las lógicas más ricas semánticamente y además que permiten razonamientos en la misma con tiempos de procesamiento finito y razonable.

3.2 Lógica y razonamiento

3.2.1 Introducción

En este apartado describiremos los diferentes tipos de lógica, así como las características de cada una y las posibilidades de razonamiento. Aunque no es objetivo de esta Tesis profundizar en estos temas ni en desarrollar nuevos algoritmos para optimizar las inferencias, sí que es necesario conocer al menos los nombres y las implicaciones que actualmente conlleva utilizar una lógica u otra a la hora de describir nuestras ontologías.

Los sistemas de web semántica actuales se basan en una Base de Conocimiento (Knowledge Base, KB), y algún Motor de Inferencia (Inference Engine, IE):

- La **Base de Conocimiento** será un conjunto de sentencias, que pueden ir creciendo y variando en el tiempo, en un lenguaje formal, de forma que los agentes inteligentes reunirán las sentencias de otros servicios en la web, y construirán su base de conocimiento sobre la que trabajar. En la mayoría de los casos los agentes inteligentes son programados con programación declarativa, en la que el agente obtiene la información que necesita saber y se le plantea que resuelva un problema, sin indicarle el algoritmo determinado para resolver ese problema.

- El **Motor de Inferencia** será lo que permitirá al agente sacar conclusiones de la base de conocimiento y resolver el problema que se le ha planteado. El motor de inferencia permitirá obtener nuevas sentencias (conclusiones) a partir de las sentencias de su base de conocimiento, y determinar, por ejemplo, si una sentencia es deducible a partir de las sentencias de su base de conocimiento.

También permite determinar si una determinada sentencia es imposible a partir de su base de conocimiento, es decir, nos lleva a una contradicción. Otro posible resultado es concluir que una sentencia no es imposible, pero no es deducible de nuestra base de conocimiento.

Recordemos que un lenguaje lógico es un lenguaje formal para representar información de la que se puedan sacar conclusiones, y dispone de una sintaxis (las sentencias válidas en el lenguaje) y una semántica (qué significan esas sentencias).

3.2.2 Tipos de lógica

Lenguaje	Qué hay en mi abstracción del mundo	Qué puedo saber respecto a algo
L. Proposicional	Hechos	Verdadero/falso/desconocido
L. Primer Orden	Hechos, objetos, relaciones	Verdadero/falso/desconocido
L. Temporal	Hechos, objetos, relaciones, tiempos	Verdadero/falso/desconocido
L. Probabilística	Hechos	Grado de certeza € [0,1]
L. Difusa	Grado de verdad	Grado de certeza € [0,1]

Tabla 3.1. Tipos de lógica

Las lógicas nos sirven para abstraer el mundo real y quedarnos sólo con nuestra área de estudio, utilizando unos operadores dependientes de cada lógica que nos permiten obtener conclusiones.

Podemos clasificarlas en función de dos parámetros, como podemos observar en la tabla 3.1:

- Según **los distintos elementos** de mi abstracción del mundo real sobre los que podemos concluir algo. El más sencillo es cuando sólo podemos sacar conclusiones de hechos. Por ejemplo, un hecho es que “Pedro es padre de Luis”, donde no podremos nunca separar “Pedro” de “Luis” o de “ser padre de”. La dificultad aumenta cuando podemos dividir los hechos en sujetos, predicados (objetos) y relaciones entre los mismos. De esta forma los hechos los podemos dividir en triplas de elementos diferenciados, como <Pedro,serPadreDe,Luis> en el ejemplo anterior Podemos ir

aumentando la cantidad de elementos distintos añadiendo, por ejemplo, tiempos. En este caso se añade una variable temporal o de periodo temporal a la tripla, pudiendo realizar sentencias como "Juan vivió en Paris del 2004 al 2006", quedando los elementos diferenciados <Juan,vivirEn,Paris,2004 al 2006>. La lógica probabilística habla de hechos, que pueden ser verdad o mentira con un grado de probabilidad (Juan mide 1.83 m, que puede ser vedad o mentira). La lógica Difusa trata del grado de verdad, ya que se trabaja con percepciones difusas en lugar de datos exactos (por ejemplo, puedo decir que Juan es alto, pero el término de alto depende de qué interlocutor lo afirme).

- Según **las conclusiones que se pueden obtener** respecto a los elementos de la abstracción del mundo. Tenemos dos grupos diferenciados, siendo uno de ellos el grupo en el que un elemento es cierto, falta o desconocido (aunque seguro que sólo puede tener el valor de cierto o de falso). El segundo grupo introduce una gradación en la certeza, de forma que un elemento tiene valor graduado de probabilidad de que sea cierto (o ocurra) o no, que varía del 0 como falso al 1 como completamente cierto. Este segundo grupo es más complicado de tratar. Así, por ejemplo, en el primer grupo puedo llegar a conclusiones del tipo"si Juan es mi padre y Pedro si hermano, entonces Pedro es mi tío", en la que ha de darse las dos premisas como ciertas para poder llegar a la conclusión. Sin embargo, en el segundo grupo podemos realizar aserciones como "si hay una probabilidad de que Juan sea mi padre del 80% y una probabilidad de que Pedro sea su hermano del 70%, entonces la probabilidad de que Pedro sea mi tío es del 56%".

Si en nuestra abstracción de la realidad pueden existir sólo hechos, y si además podemos decir que los hechos son ciertos, falsos, o desconocidos, entonces estamos tratando con Lógica Proposicional. Si lo que indicamos sobre los hechos es que pueden ser ciertos según una graduación, trabajamos con lógica probabilística.

Si el mundo abstraído tiene más cosas que hechos, tiene objetos y relaciones entre ellos, podemos estar utilizando Lógica de Primer orden. Si además se utilizan parámetros temporales nos referimos a Lógica Temporal. Normalmente las lógicas con más variables suelen llamarse lógicas de orden superior.

Existen otros tipos de lógica, como podemos ver en la tabla 3.1, pero sólo profundizaremos en la Lógica Proposicional, en la Lógica de Primer Orden (FOL), y dentro de FOL profundizaremos un subconjunto de ésta, la Lógica Descriptiva (Description Logics, DL). Empezaremos con la lógica proposicional, ya que es la base de las otras lógicas, y en particular de la lógica de primer orden. Puesto que la web semántica se basa en la lógica de primer orden, la desarrollaremos con detalle, y en particular un subconjunto de la misma, la lógica DL. La razón por la que trabajaremos con lógica DL radica en que, como veremos, es un subconjunto de FOL que permite implementar razonadores que realizan inferencias en tiempos razonables, y por eso es el subconjunto más utilizado dentro de la web semántica.

3.2.3 Razonamiento

Se define modelo [FRA01] como un mundo estructurado con respecto al cual podemos evaluar la veracidad o no de algún hecho. Por ejemplo, cuando en lógica proposicional tenemos dos hechos "Me levanto pronto los lunes" y "llego pronto al trabajo los lunes", un modelo posible es que sea cierto que me levante pronto los lunes, y falso que llegue pronto al trabajo los lunes. En principio hay tantos modelos como posibilidades de un sistema sin llegar a contradicciones.

Se habla de que una base de conocimiento KB, supone la frase (o función lógica, como combinación de frases) α

$$KB \models \alpha \qquad \text{[Expr. 3.1]}$$

Si α es verdad en todos los modelos (mundos posibles) de KB.

$$KB \models \alpha \quad \text{sii} \quad M(KB) \subseteq M(\alpha) \qquad \text{[Expr. 3.2]}$$

Siendo M(KB) el conjunto de todos los modelos de KB y M(α) el conjunto de todos los modelos en los que α puede ser cierto. Esto nos indicará que la frase α es derivable de nuestra base de conocimiento, ya que no es posible un modelo de KB que no sea modelo de α.

Se dice que la frase (o función lógica) α se puede derivar (deducir o inferir) de la base KB por medio del procedimiento p

$$KB \vdash_p \alpha \qquad \text{[Expr. 3.3]}$$

Dicho procedimiento puede ser:

1. Seguro (con sentido) si siguiendo ese procedimiento p, llegamos a una conclusión α, dicha conclusión es una suposición de nuestra KB, es decir, que el procedimiento nos da sólo conclusiones que son correctas.

   Siempre que $KB \vdash_p \alpha$, es cierto que $KB \models \alpha$ [Expr. 3.4]

2. Completo, cuando si algo se puede concluir de una KB, se puede inferir por medio del procedimiento p. Es decir, si algo es correcto, podremos sacarlo por medio de p.

   Siempre que $KB \models \alpha$, sabemos que $KB \vdash_p \alpha$ [Expr. 3.5]

Nos interesarán siempre procedimientos que den conclusiones correctas y además que puedan obtener siempre una conclusión si es correcta (seguros y completos).

La lógica proposicional tiene procedimientos seguros y completos, sin embargo hay otras lógicas mucho más expresivas (permiten mundos más complejos y por tanto más cercanos a la realidad) que también los tienen, como por ejemplo la lógica de primer orden (FOL) [FRA01] [ARE01].

Sin embargo, otra característica que nos interesará es que dichos procedimientos sean eficientes computacionalmente hablando, por lo que nos veremos obligados a restringir FOL en forma de Lógica Descriptiva (Description Logics. DL) como veremos más adelante. Por esto mismo, a veces nos tendremos que conformar con procedimientos que sean seguros, pero no completos.

3.2.4 Lógica Proposicional LP

La lógica proposicional es el sistema lógico más simple de los que veremos en esta Tesis, y se basa en proposiciones atómicas, de las que sólo podemos indicar si son verdaderas o falsas, y que pueden constituir proposiciones más complejas asociándose con conectores "y", "o", "si… entonces" o "...es equivalente a...". Su sintaxis y semántica, que veremos a continuación, definen con más detalle esta lógica.

3.2.4.1 Sintaxis en LP

La lógica proposicional basa su sintaxis en un conjunto de proposiciones atómicas $\sum = \{A,B,C\ldots\}$ indivisibles e independientes entre sí, que se

podrán combinar entre sí a modo de fórmulas lógicas. Podemos ver en la Tabla 3.2 un resumen de las mismas.

Símbolo	Denominación
T	Verdadero
F	Falso
$\neg A$	Negación
$A \wedge B$	Conjunción
$A \vee B$	Disyunción
$A \rightarrow B$	Implicación
$A \leftrightarrow B$	Equivalencia

Tabla 3.2. Resumen de la sintaxis

3.2.4.2 Semántica en LP

En esta lógica, llamaremos Interpretación I sobre un conjunto de átomos $\sum$ a una función que nos dará un valor verdadero o falso por cada proposición atómica.

$$I: \sum \rightarrow \{T,F\} \qquad \text{[Expr. 3.6]}$$

La interpretación de x la representaremos de la siguiente forma:

$$I\,(x) \text{ o también } x^I \qquad \text{[Expr. 3.7]}$$

Esto nos permitirá pasar a la semántica de esta lógica (Tabla 3.3). Dicha semántica es aplicable también a combinaciones de proposiciones atómicas.

Síntaxis	Significado	Explicación
$I \models A$	A^I	el átomo A es cierto en la interpretación I
$I \models \neg A$	A no se satisface (es falso) en I	A es falso en la interpretación I
$I \models A \wedge B$	$I \models A$ y además $I \models B$	A y B son ciertos en la interpretación I
$I \models A \vee B$	$I \models A$ o (no exclusiva) $I \models B$	Al menos uno de los dos, A o B es cierto en la interpretación I
$I \models A \rightarrow B$	Si $I \models A$, entonces $I \models B$	No es posible que B sea

		cierto en I y no lo sea A
$I \models A \leftrightarrow B$	$I \models$A, si y sólo si $I \models$B (A ≡ B en esa interpretación)	A y B son los dos verdaderos o los dos falsos a la vez en I
$I \models$ T	"Verdadero" es verdad en cualquier interpretación I	
$I \not\models$F	"Falso" no es verdad (no se da) en cualquier interpretación I	

Tabla 3.3. Semántica

Además, existe una relación entre los distintos operadores, de forma que pueden transformarse entre sí. Las más importantes se ven en la tabla 3.4.

Fórmula	Equivalente
$A \rightarrow B$	$\neg(A \wedge \neg B)$
$\neg(A \vee B)$	$(\neg A \wedge \neg B)$
$\neg(A \wedge B)$	$(\neg A \vee \neg B)$
$A \vee (A \wedge B)$	A
$A \wedge (A \vee B)$	A
$\wedge, \vee, \leftrightarrow$ *son conmutativas y asociativas* $\wedge$ *es distributiva frente a* $\wedge$ $\vee$ *es distributiva frente a* $\wedge$	

Tabla 3.4. Equivalencias

3.2.4.3 Formulación en LP

Se dice que una fórmula lógica es [FRA01]:

- **Posible,** si hay alguna interpretación I que la satisface (hace que sea cierta).

- **Imposible**, si no hay ninguna interpretación que la haga cierta, es decir, que en ninguno de los posibles mundos puede darse.

- **Válida o tautología** si con cualquier interpretación I hace que la fórmula sea cierta.

- **Invalidable** si hay al menos una interpretación que hace que la fórmula no sea cierta.

En nuestro campo de trabajo una fórmula será el conjunto de nuestra base de conocimiento, al que añadiremos algunas subfórmula o frases nuevas para comprobar si se deducen de nuestra base de datos, o se contradicen, o sólo son posibles o invalidables.

Los sistemas actuales que tratan con gran número de variables se basan normalmente en demostrar que una fórmula es posible (buscar una implementación que se cumpla, y no hace falta seguir) o recorrerlas todas y ver que ninguna es posible. Se puede convertir cualquier otra formulación en LP en una formulación de posibilidad De hecho una fórmula x es una tautología si la fórmula $\neg$ x es imposible (cuando no se encuentra ninguna interpretación que lo cumpla, tras pasar por todas), y una fórmula x es imposible si $\neg$ x es una tautología. Una fórmula x es Invalidable si $\neg$ x es posible.

Además, las fórmulas se suelen transformar para normalizarlas, de modo puedan ser tratadas con algún algoritmo que necesita esa normalización. Como formas normales tenemos:

- **Conjunctive Normal Form CNF**, que consiste en dejar todo como proposiciones atómicas (negadas o no) enlazadas entre sí por disyunciones (subfórmulas) y éstas a su vez enlazadas por conjunciones.

$$\wedge_{i=1}^{n} \left(\vee_{j=1}^{m} A_{i,j} \right) \qquad \text{[Expr. 3.8]}$$

  De este modo, si todas las subfórmulas contienen T o proposiciones atómicas complementarias, es una tautología.

- **Disjunctive Normal Form (DNF)**, que consiste en dejar todo como proposiciones atómicas (negadas o no) enlazadas entre sí por conjunciones y éstas a su vez enlazadas por disyunciones.

$$\vee_{i=1}^{n} \left(\wedge_{j=1}^{m} A_{i,j} \right) \qquad \text{[Expr. 3.9]}$$

De este modo, si todas las subfórmulas contienen F o proposiciones atómicas complementarias, es imposible.

- **Negation Normal Form (NNF)**, que consiste en que sólo estén negadas las proposiciones atómicas, no quedando ningún paréntesis negado. Se utilizará para el procedimiento Tableaux.

- **Horn Form (HF)**, que consiste en dejar todas las subfórmulas con como máximo una proposición sin negar (no siempre es posible esta transformación). La parte sin negar es la causa de la implicación y la negada es la consecuencia de dicha implicación. Por ejemplo:

$$(A \vee \neg C) \wedge (\neg C \wedge \neg B \wedge A) \qquad \text{[Expr. 3.10]}$$

Que se puede transformar en implicaciones (que se utilizan en lenguajes basados en reglas):

$$(C \rightarrow A) \quad y \quad (A \rightarrow (C \vee B)) \qquad \text{[Expr. 3.11]}$$

3.2.4.4 Razonamiento en LP

El objetivo del razonamiento en esta y las siguientes lógicas es permitir que nuestros sistemas inteligentes puedan obtener y comprobar distintas conclusiones basándose en su base de conocimiento. Por ejemplo, gracias al razonamiento un sistema, basándose en su conocimiento de que si ha de encender las luces de la calle cuando esté oscuro y en su conocimiento de que cuando es de noche está oscuro, podrá deducir que ha de encender las luces cuando sea de noche.

Decimos que una KB en lógica proposicional es un conjunto de fórmulas lógicas, de forma que una interpretación I satisface la KB (es posible) si satisface todas y cada una de sus fórmulas.

$$I \models \text{KB sii } I \models \text{A, para todo A} \in \text{KB} \qquad \text{[Expr. 3.12]}$$

Luego saber si una fórmula Z se puede inferir a partir de la KB, se basa en obligar a que se cumpla Z (sea verdadera) en todas las interpretaciones que satisfacen la KB.

$$\text{KB} \models \text{Z sii } I \models \text{Z para todos los modelos } I \text{ de la KB} \qquad \text{[Expr. 3.13]}$$

Existen dos procedimientos para resolver las inferencias son el procedimiento de enumeración el de tableaux.

3.2.4.5 Razonamiento por enumeración

Este método consiste en poner en una tabla todas las combinaciones posibles de proposiciones atómicas, ver las que son modelo de KB (las que dan verdadero en todas las fórmulas de la KB) y comprobar que dan verdadero también en la fórmula que queremos inferir.

El método realiza un estudio exhaustivo de todas las posibilidades, es decir, de todos los modelos posibles, que tiene como ventaja que se pueden realizar todos los tipos de comprobaciones a la vez (posibilidad, imposibilidad, validez e invalidez). Como desventaja es que conlleva un gran esfuerzo computacional, ya que dadas N variables hemos de valoras las distintas 2^N posibles combinaciones de verdadero o falso.

Podemos ver un ejemplo en la figura 3.1, donde tenemos una base de conocimiento y queremos averiguar si esa base de esa base de conocimiento KB se puede inferir la sentencia *α,* .

$$KB = (A \vee C), (B \wedge \neg C)$$
$$\alpha = (A \vee B)$$

KB ⊨ α?? (veremos si se puede inferir α de KB)

A	B	C	$(A \vee C)$	$(B \wedge \neg C)$	KB	α	I
F	F	F	F	F	F	F	1
F	F	T	T	F	F	F	2
F	T	F	F	T	F	T	3
F	T	T	T	F	F	T	4
T	F	F	T	F	F	T	5
T	F	T	T	F	F	T	6
T	T	F	T	T	T	T	7
T	T	T	T	F	F	T	8

Modelos posibles: 8 ($I_1 \ldots I_8$)

KB cierto en $\{I_7\}$
α cierto en $\{I_3, I_4, I_5, I_6, I_7, I_8\}$
$M(KB) \subseteq M(\alpha)$ (α cierto en TODOS los modelos posibles de KB)
KB ⊨ α?? SI (se puede inferir α de KB)

Figura 3.1. Ejemplo de deducción por Enumeración

3.2.4.6 Razonamiento por método Tableaux

El método Tableaux es un método más óptimo computacionalmente que el razonamiento por enumeración. Sin embargo, sólo puede resolver cuestiones sobre si un conjunto de fórmulas es posible o imposible, con lo que tendremos que realizar algunas modificaciones para llegar a otras conclusiones.

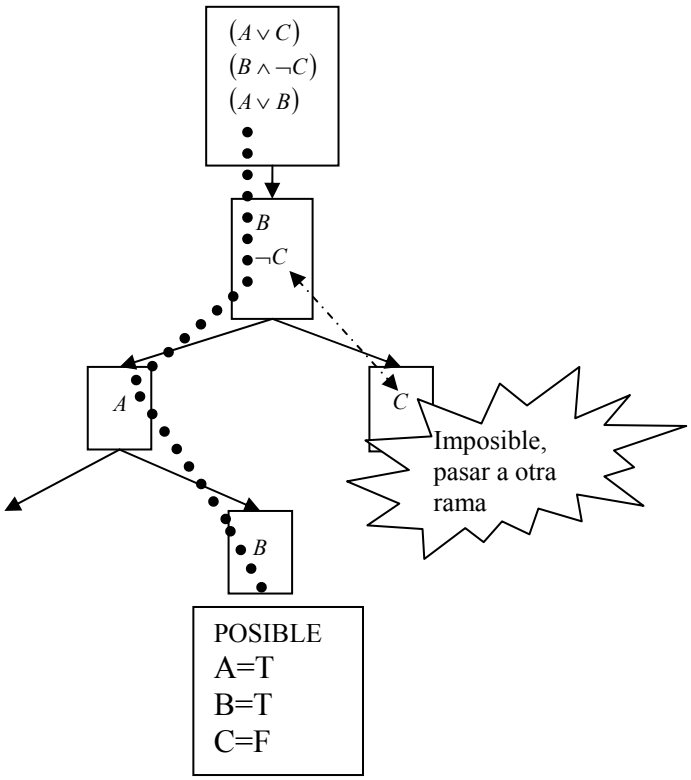
El método Tableaux consiste en reducir el sistema en su forma NNF de conjunciones y disyunciones (será el nodo raíz de nuestro árbol Tableaux) Luego ir descomponiendo las hasta quedarse con todo descompuesto. En la figura 3.2 podemos ver un ejemplo, donde se comprueba la posibilidad de una fórmula, consistente en tres subfórmulas.

Se comienza con todas las fórmulas que son conjunciones, colocando en cada nodo de la rama los valores atómicos de las mismas (por ejemplo, podemos ver en la figura 3.2 que colocamos *B* y ¬*C*). Luego por cada disyunción se dividirá la rama, generando dos nodos paralelos con los valores atómicos de la disyunción (por ejemplo, en la figura 3.2 podemos ver una rama *A* y otra con *C*). En cada formación de un nodo se comprueba que los valores que anoto no contradicen a sus nodos predecesores en la rama hasta la raíz (que no hay en un nodo X y en uno anterior de su rama ¬X). Terminaremos de recorrer una rama cuando hay una contradicción o cuando terminamos con todas las subfórmulas.

Si terminamos de recorrer una rama sin llegar a una contradicción del tipo "X y ¬X", hemos encontrado una Interpretación que lo cumple, por lo que el conjunto de fórmulas es posible. Si recorremos todas las ramas del árbol y todas llegan a contradicción, es que el sistema es imposible.

$(A \vee C), (B \wedge \neg C), (A \vee B)$

Posible??



1. Por cada conjunción se sigue la rama
2. Por cada disyunción se divide la rama
3. Se sigue hasta que una rama es imposible (contradicción) o se han dividido todas las fórmulas a átomos (A, B, C)
4. Si todas las ramas son imposibles, el sistema es imposible

Figura 3.2. Ejemplo de Tableaux

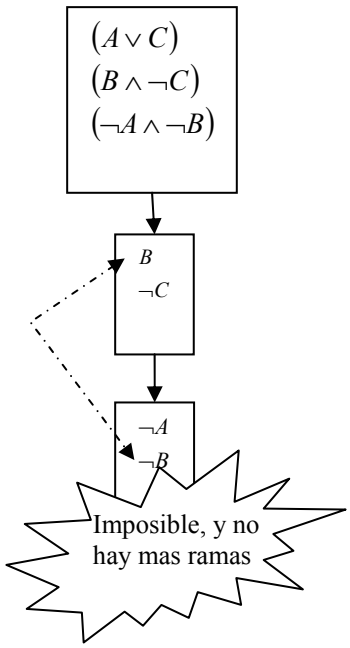
Tableaux es más eficiente (si partimos de la forma NNF) que la Enumeración, ya que no es necesario recorrer todas las posibilidades.

Para saber otras deducciones distintas de las de posibilidad e imposibilidad en un conjunto de fórmulas, tendremos que realizar transformaciones. Por ejemplo, si queremos saber si una fórmula α se puede deducir de una base de conocimiento KB (KB ⊨ α), bastará con ver si es posible que se den KB y ¬α a la vez. Si es imposible, podemos deducir que KB ⊨ α.

$$KB = (A \vee C), (B \wedge \neg C)$$
$$\alpha = (A \vee B)$$
$$\neg\alpha = (\neg A \wedge \neg B)$$

$$(A \vee C), (B \wedge \neg C), (\neg A \wedge \neg B)$$

Posible??



KB ⊨ α?? SI

Figura 3.3. Ejemplo de deducción Tableaux

3.2.5 Lógica de Primer Orden (FOL)

La Lógica de Primer Ordern (First Order Logic o FOL) se puede ver como una ampliación de la lógica proposicional [IAN03], añadiendo sintaxis respecto a los cuantificadores (quantifiers), la semántica con interpretaciones de términos cuantificados y aumenta el significado de las frases de lógica predicativa. La razón por la que se hace dicha extensión es para intentar adentrarnos en la estructura de las sentencias atómicas de la lógica proposicional, que son sólo frases que pueden ser verdaderas o falsas pero sin estructura interna. En FOL [FRA01], las frases atómicas se interpretan como aserciones sobre relación entre objetos (relaciona n objetos, n>=1).

Veamos el ejemplo 3.1: "La pelota es redonda"

```
REDONDA(pelota)
```
[Ej. 3.1]

Donde pelota es un objeto y REDONDA es una propiedad de dicho objeto (símbolo predicativo).

Además, cuando se utilizan cuantificadores se pueden colocar variables en los predicados. Podemos verlo en el ejemplo 3.2: "Hay pelotas que son redondas y verdes"

$$\exists x.[REDONDA(x) \wedge VERDE(x)] \qquad \text{[Ej. 3.2]}$$

3.2.5.1 Sintaxis en FOL

La sintaxis se puede resumir según la tabla 3.5 [FER06]:

- **Términos**, que pueden ser constantes (individuos), variables o funciones de términos.
- **Literales**, que son fórmulas atómicas (Tienen predicados P), y negaciones de las mismas
- **Fórmulas** generales, que añaden conectores lógicos (lógica proposicional) y cuantificadores.

Símbolo	Denominación	descripción
a, b	constante	Un objeto de nuestro mundo
x, y	variables	En el momento de hacer la

		aserción puede ser cualquier constante de nuestro mundo
$f(t_1,\ldots t_n)$		t_i=a ó x ó $F(t_1,\ldots t_n)$ (términos)
$f(a_1,\ldots a_n)$	Términos básicos	Términos que no incluyen variables
$\varphi,\phi \equiv P(t_1,..,t_n)$	fórmulas atómicas	Términos relacionados por un símbolo predicativo P
T	verdadero	
F	Falso	
$\neg\phi$	negación	
$\phi \wedge \varphi$	conjunción	
$\phi \vee \varphi$	disyunción	
$\phi \rightarrow \varphi$	implicación	
$\phi \leftrightarrow \varphi$	equivalencia	
$\exists x.[\varphi]$	Cuantificador existencial	Nos referimos a al menos uno de los objetos que cumple φ
$\forall x.[\varphi]$	Cuantificador universal	Nos referimos a todos los objetos que cumplen φ

Tabla 3.5. Sintaxis FOL

3.2.5.2 Semántica en FOL

En esta lógica [FRA01], llamaremos Interpretación I

$$I = \langle \Delta, \cdot^I \rangle \qquad \text{[Expr. 3.14]}$$

donde Δ es un conjunto no vacío y I es una función que mapea:

a) Los símbolos de función sobre Δ

$$f^I \in [\Delta^n \rightarrow \Delta] \quad \textit{siendo n el orden de } f \qquad \text{[Expr. 3.15]}$$

una función es cierta si en nuestra interpretación existe dicha función y su resultado pertenece a nuestro conjunto Δ.

$$(f(t_1,\ldots,t_n))^I \equiv f^I({t_1}^I,\ldots,{t_n}^I) \in \Delta \qquad \text{[Expr. 3.16]}$$

b) Las constantes sobre elementos de Δ

$$a^I \in \Delta \qquad \text{[Expr. 3.17]}$$

Una constante es cierta en una interpretación si se refiere a un objeto de dicha interpretación.

c) Los símbolos de predicados (predicativos) en relaciones sobre Δ^n

$$P^I \subseteq \Delta^n \quad \textit{siendo n el orden de P} \qquad \text{[Expr. 3.18]}$$

Una fórmula atómica es cierta (se cumple) en una interpretación si los objetos a los que se refiere se relacionan entre sí por medio de P.

$$I \models P(t_1,\ldots,t_n) \qquad sii \qquad \left\langle t_1{}^I,\ldots,t_n{}^I \right\rangle \in P^I \qquad \text{[Expr. 3.19]}$$

d) El conjunto de variables V se asignan según una asignación α (por ejemplo, si mi sistema tiene 2 variables x e y, su asignación podría ser x = pelota, y = sombrero, si pelota y sombrero pertenecen a Δ).

$$\alpha : V \to \Delta \qquad \text{[Expr. 3.20]}$$
$$x^{I,\alpha} = \alpha(x) \quad \textit{el objeto asignado a x por } \alpha$$
$$a^{I,\alpha} = a^I$$
$$\left(f(t_1,\ldots,t_n)\right)^{I,\alpha} \equiv f^I\left(t_1{}^{I,\alpha},\ldots,t_n{}^{I,\alpha}\right)$$

$$I,\alpha \models P(t_1,\ldots,t_n) \qquad sii \qquad \left\langle t_1{}^{I,\alpha},\ldots,t_n{}^{I,\alpha} \right\rangle \in P^I$$

$\alpha[x/d]$ indica que hacemos la asignación α menos para x, que lo sustituimos por d $\in \Delta$.

Se dice que una fórmula ϕ es verdadera en una interpretación I bajo una asignación de variables α si la interpretación I haciendo la sustitución de variables α son algún modelo posible de ϕ.

$$I,\alpha \models \phi \qquad \text{[Expr. 3.21]}$$

La semántica de FOL se puede ver en la tabla 3.6.

Sintaxis	Significado
$I,\alpha \models P(t_1,\ldots,t_n)$	$\langle t_1{}^{I,\alpha},\ldots,t_n{}^{I,\alpha}\rangle \in P^I$
$I,\alpha \models \neg\phi$	$I,\alpha \not\models \phi$, ϕ no se satisface en I,α
$I,\alpha \models \phi \wedge \varphi$	$I,\alpha \models \phi$ y además $I,\alpha \models \varphi$
$I,\alpha \models \phi \vee \varphi$	$I,\alpha \models \phi$ o (no exclusiva) $I,\alpha \models \varphi$
$I,\alpha \models \phi \rightarrow \varphi$	Si $I,\alpha \models \phi$ entonces $I,\alpha \models \varphi$
$I,\alpha \models \phi \leftrightarrow \varphi$	$I,\alpha \models \phi$, si y sólo si $I,\alpha \models \varphi$ ($\phi \equiv \varphi$ en esa interpretación I,α)
$I,\alpha \models \forall x.[\phi]$	Para todo $d \in \Delta : I,\alpha[x/d] \models \phi$
$I,\alpha \models \exists x.[\phi]$	Existe al menos un $d \in \Delta : I,\alpha[x/d] \models \phi$

Tabla 3.6. Semántica FOL

Las fórmulas en FOL se dividen como en la Lógica Proposicional (posible, imposible, válida o inválida), pero esta vez respecto a la un modelo I bajo α, (I,α).

Una fórmula es cerrada (sentencia) si no tiene variables en su interior, no dependiendo de α.

A las relaciones entre operadores de la Lógica Proposicional hay que añadir los de los cuantificadores (Tabla 3.7):

Fórmula	Equivalente
$\forall x.[\forall y.[\phi]]$	$\forall y.[\forall x.[\phi]]$
$\exists x.[\exists y.[\phi]]$	$\exists y.[\exists x.[\phi]]$
$\forall x.[\phi]$	$\neg[\exists x[\neg\phi]]$
$\exists x.[\phi]$	$\neg[\forall x[\neg\phi]]$
$\left\{\begin{array}{l}[\otimes x.[\phi]] \bullet \varphi = [\otimes x.[\phi \bullet \varphi]],\\ \otimes \in \{\exists,\forall\},\\ \bullet \in \{\wedge,\vee\}\end{array}\right\}$ *si* φ *no depende de* x,	
$\exists$ *suele lle* var *dentro* $\wedge$ *ej* $\exists x.[\phi \wedge \varphi]$	
$\forall$ *suele lle* var *dentro* $\rightarrow$ *ej* $\forall x.[\phi \rightarrow \varphi]$	
$\forall x.[\exists y.[\phi]] \neq \exists y.[\forall x.[\phi]]$	
$\exists x.[\forall y.[\phi]] \neq \forall y.[\exists x.[\phi]]$	

Tabla 3.7. Equivalencias de cuantificadores

En FOL podemos incluir el concepto de herencia, por medio de la inclusión (subsumption). Dados P y Q como símbolos predicativos del mismo orden, se dice que P incluye a Q en la Teoría KB:

$$KB \models P \supseteq Q \qquad \text{[Expr. 3.22]}$$

Si para todo objeto o n-tupla de objetos que se relacionan por Q, también se relacionan por P

$$KB \models \forall x_1, \ldots \forall x_n . [Q(x_1, \ldots x_n) \rightarrow P(x_1, \ldots x_n)] \qquad \text{[Expr. 3.23]}$$

Por ejemplo, FAMILIAR incluye a HERMANO, si ambos relacionan a dos seres (todas las relaciones de hermandad dos a dos son a su vez relaciones de familia). Esto se puede ver como una cierta herencia entre los distintos símbolos predicativos.

Si la propiedad es de grado 1 (sólo se refiere a un objeto), se puede entender dicha propiedad como la definición de una clase y la inclusión como una herencia entre clases.

Por ejemplo, decir que Juan es Profesor, PROFESOR (Juan), sitúa a Juan como miembro de la clase profesor. Y si digo que PROFESOR $\subseteq$ TRABAJADOR, puedo concluir que Juan también pertenece a la clase de los trabajadores, TRABAJADOR (Juan).

3.2.5.3 Formulación en FOL

Las fórmulas se pueden expresar siempre en la forma Prenex Normal Form (PNF):

- Eliminar todos los $\rightarrow, \leftrightarrow$.
- Meter dentro de los paréntesis todas las negaciones (Hasta aquí parecido a NNF).
- Sacar al máximo fuera de los paréntesis los cuantificadores.

Existe un procedimiento para resolver el problema de si un sistema es posible (Tableaux para FOL):

- La mecánica es parecida a la utilizada en Tableaux para Lógica Proposicional. Necesita su forma NNF (PNF).

- Se añade que: se atacan primero los $\exists$, y se substituye la variable por una constante totalmente nueva. Cuando hay $\forall$ lo que se hace es sustituir la variable por todas constantes que tenemos, unidas entre si por conjunciones.

Un punto que afecta a los problemas a resolver en FOL es si es resoluble (decidable). Un problema es resoluble si hay algún proceso computacional que resuelve el problema en un número finito de pasos. En el caso de FOL está demostrado [SCHO89] que el problema de averiguar si una fórmula se deduce de una base de conocimiento es irresoluble. Sin embargo, De hay razonadores para FOL como Vampiro[RIA02] o E-SETHEO [MOS97], que normalmente se basan en la búsqueda de incongruencias en una base de conocimiento, utilizando procedimientos no completos, por lo que no podemos asegurar que si el razonador no encuentra incongruencias no las haya.

Para hacer la lógica resoluble hemos de restringir FOL. Una de las posibilidades que simplifica mucho los razonadores es restringir FOL utilizando como mucho dos variables en los símbolos predicativos (L_2).

Existen a su vez sistemas más complejos de resolver que FOL[IAN03], a la vez que más expresivos. Por ejemplo en Lógicas de Mayor Orden (HOL), donde se permite libremente el uso de variables en los símbolos predicativos, y aunque por su complejidad de resolución son intratables para sistemas grandes, existen razonadores, como HOL [GOR93] e Isabelle [PAU94]. De hecho, otra definición para FOL se basa en que el cálculo de lógica predicativa FOL permite variables para referirse a objetos pero no a predicados o funciones.[LUG02].

3.2.6 Lógica Descriptiva DL

3.2.6.1 Introducción

Las Lógicas Descriptivas (DLs) pueden ser vistas como el subconjunto de FOL que permite hacer resoluble sus inferencias [TSA03]. DL fue diseñado como una extensión de frames (sistemas computacionales) y de las redes semánticas. Recibió el nombre de Lógica Descriptiva en los 80, llamándose

anteriormente lenguajes conceptuales (concept lenguajes) y anteriormente sistemas terminológicos (terminological systems).

El primer sistema basado en DL fue KL-ONE[BRA85], viniendo después LOOM (1987), BACK (1988), KRIS (1991), FaCT (1998) y Racer (1999). Hoy en día es una de las bases de la Web Semántica.

los lenguajes DL no son los únicos subconjuntos de FOL resolubles [SER02], ya que hay otro conjunto de lenguajes Horn (basados en reglas, como PROLOG[GUP01]) y F-Logic [KIF95].

Según [BEN02], DL son una familia de formalismos de representación de conocimientos diseñados para representar y razonar sobre ontologías, y otras cosas como esquemas de bases de datos, configuraciones, etc.

Los sistemas DL se basan en una KB que contiene aserciones sobre las clases o dicho de otra forma una taxonomía de clases (terminología, T-Box) y un conjunto de aserciones sobre instancias, o dicho de otra forma, representación de una situación concreta (A-Box). Sobre esta KB se pueden realizar inferencias (Ver Figura 3.4).

Buscando una cierta analogía con las Bases de Datos, T-Box sería la descripción de la Base de Datos (esquemas) y A-Box las tuplas de esa Base de Datos.

T-Box contiene:

- Definiciones de Conceptos (les asigna un nombre)

  $$A \equiv C \qquad \text{[Expr. 3.24]}$$

  siendo en la expresión 3.24 A el nombre del concepto y C su definición (concepto complejo). En el ejemplo 3.3, la clase de los profesores son las personas que imparten al menos una materia:

  $$\text{profesor} \equiv \text{persona} \sqcap \exists \text{IMPARTE.materia} \qquad \text{[Ej. 3.3]}$$

- Axiomas (restringe el modelo)

  $$C_1 \sqsubseteq C_2, \quad C_i \text{ son conceptos complejos} \qquad \text{[Expr. 3.25]}$$

El ejemplo 3.4 representa el axioma "Todo alumno es también una persona". Otra forma de entender el axioma del ejemplo es indicar que la clase *alumno* es subclase de la clase *persona*.

$$\text{alumno} \sqsubseteq \text{persona} \qquad \text{[Ej. 3.4]}$$

A su vez las A-Box contienen:

- Aserciones sobre conceptos.

$$a:C \qquad \text{[Expr. 3.26]}$$

siendo a el nombre de una instancia que pertenecerá al concepto C. Por ejemplo (ejemplo 3.5): Juan es un profesor y además ingeniero (pertenece al conjunto de los profesores y al conjunto de los ingenieros.

$$\text{Juan} : \text{profesor} \sqcap \text{ingeniero} \qquad \text{[Ej. 3.5]}$$

- Aserciones sobre roles.

$$(a,b): R \quad \text{ó} \quad R(a,b) \qquad \text{[Expr. 3.27]}$$

siendo a, b objetos de nuestro sistema y R un rol (propiedad) de nuestro sistema. En el ejemplo 3.6 podemos ver las dos formas de representar una aserción sobre un rol, en la que indicamos que la instancia *Juan* está relacionada por medio de la propiedad *IMPARTE* a la instancia *Telemática*.

$$\begin{array}{c}(\text{Juan, Telemática}): \text{IMPARTE} \\ \text{ó} \\ \text{IMPARTE(Juan,Telemática)}\end{array} \qquad \text{[Ej. 3.6]}$$

En la figura 3.4 podemos ver una representación gráfica de la arquitectura de un sistema de Lógica Descriptiva, en la que hemos utilizado los ejemplos anteriormente comentados. La T-Box contiene información relacionada únicamente con las clases, por lo que la podemos considerar como la parte del conocimiento general. En la A-Box se sitúa la parte de nuestra Base de Conocimiento relacionada con las instancias, por lo que podemos considerar a la A-Box como la parte más concreta de nuestra Base de Conocimiento, la que trata de casos particulares.

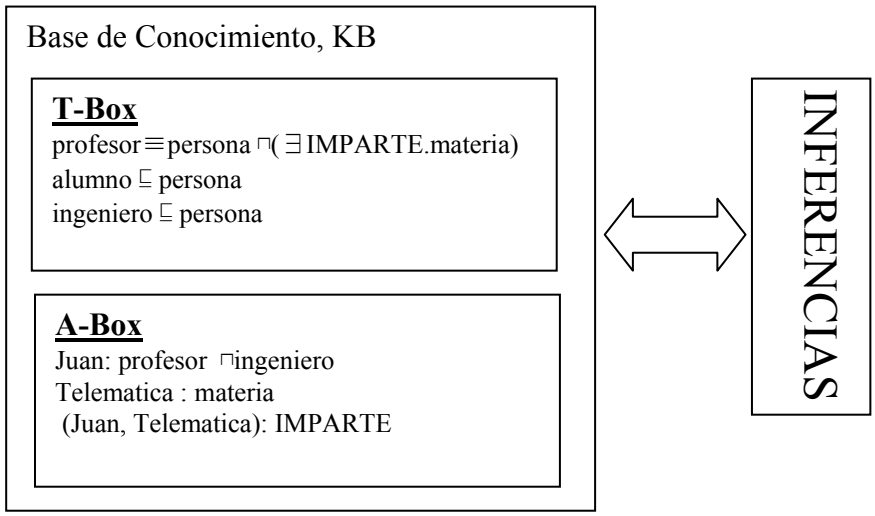


Figura 3.4. arquitectura de un sistema DL

3.2.6.2 Sintaxis y Semántica en DL

Existen varios lenguajes de lógica descriptiva, siendo todos en cualquier caso un subconjunto de la sintaxis de FOL (aunque la notación se simplifica). Los distintos lenguajes de lógica descriptiva los veremos en el apartado siguiente.

Para estudiar la sintaxis veremos como ejemplo el lenguaje denominado $\mathcal{ALC}$, [ALO06]. $\mathcal{ALC}$, es el lenguaje DL más simple y restrictivo, con sólo predicados binarios y que relacionan instancias abstractas (no números, strings u otro tipo de dato definido, ni siquiera por medio de la enumeración de sus elementos). Además, no permite asignar propiedades a los roles (como indicar que un rol es transitivo). Pese a ser $\mathcal{ALC}$, el lenguaje más sencillo, el resto utilizan la misma sintaxis, añadiendo sólo las propiedades de los roles. En la tabla 3.8 se muestra su sintaxis.

Sintaxis	Semántica	Significado
C	$C^I \subseteq \Delta^I$	Concepto simple, primitivo o atómico. Es un subconjunto de Δ^I
R	$R^I \subseteq \Delta^I x \Delta^I$	Rol simple, primitivo o atómico (binario)

$C \sqcup D$	$C^I \cup D^I$	Conjunto de elementos definido por la unión de dos conceptos (unión de conjuntos)
$C \sqcap D$	$C^I \cap D^I$	Conjunto de elementos definido por la intersección de dos conceptos (intersección de conjuntos)
$\neg C$	Δ^I / C	Conjunto de elementos que abarca todo nuestro espacio menos los individuos que agrupa C
$\exists R.C$	$\{x \mid \exists y.[R^I(x,y) \wedge C^I(y)]$	Conjunto de elementos que se relacionan al menos una vez por medio de R con otros elementos pertenecientes al concepto C
$\forall R.C$	$\{x \mid \forall y.[R^I(x,y) \rightarrow C^I(y)]$	Conjunto de elementos que tienen todas sus relaciones con otros elementos pertenecientes al concepto C por medio de R, y no por otro medio.
$\top$	Δ^I	Todo nuestro espacio (“top”)
$\bot$	$\varnothing$	El conjunto vacío (“botton”)

Tabla 3.8. sintaxis y semántica de ALC

A continuación describimos la semántica relacionada con DL. Dada una interpretación *I*, que vemos en la ecuación 3.28, definimos una Teoría del Modelo (Model Theory, MT).

$$I = \left(\Delta^I, \cdot^I\right) \qquad \text{[Expr. 3.28]}$$

La Teoría del Modelo indica las reglas básicas de mi abstracción de la realidad. En esta teoría los únicos elementos simples son las instancias de los objetos y las instancias de parejas de objetos. Tanto los roles simples como los Conceptos simples, son conjuntos de [Δ^I] y [Δ^I x Δ^I] respectivamente. Esto ocurre también en el MT de FOL [IAN03].

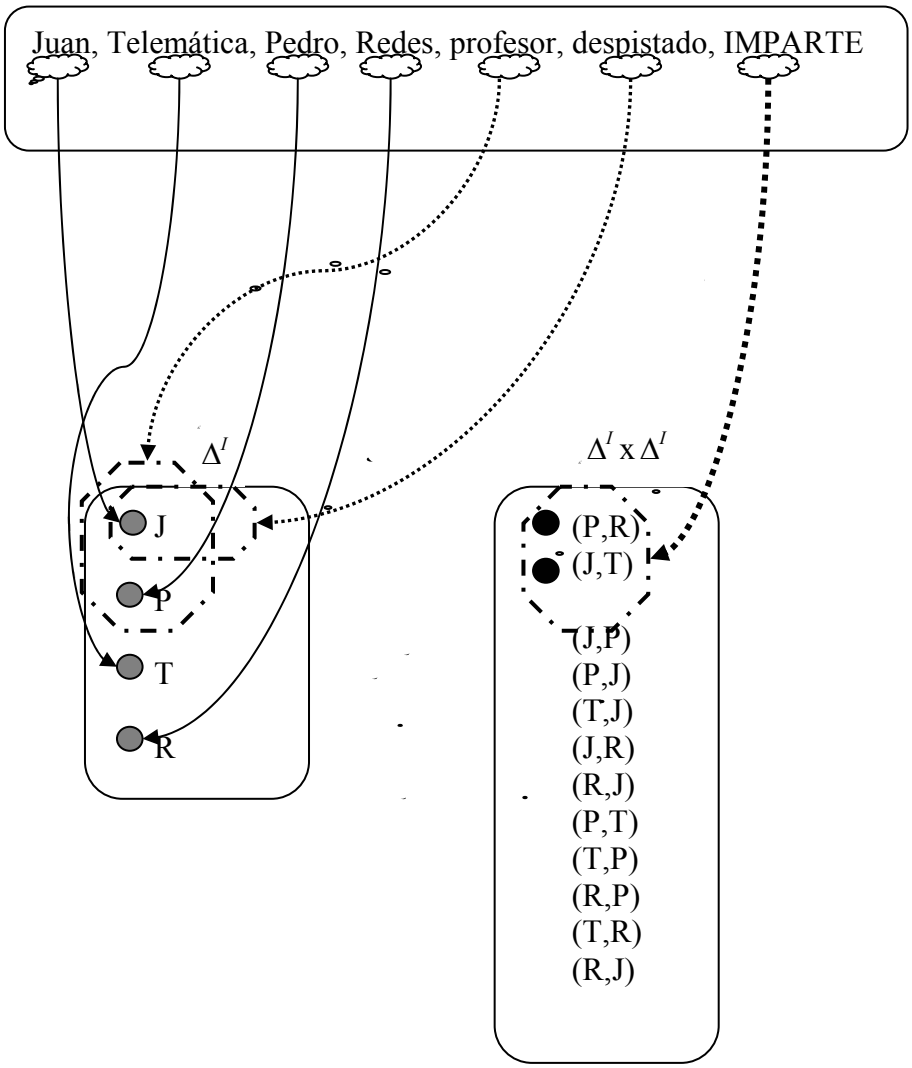


Figura 3.5. Ejemplo de MT

3.2.6.3 Tipos de DL

Se distinguen entre sí por el subconjunto de FOL que abarcan. Cuando más abarcan, más expresivo es el lenguaje, pero a su vez más complicado es razonar en esa lógica. En la tabla 3.9 podemos ver una comparativa de los distintos sistemas, indicando las características de los mismos y ordenados de menor a mayor complejidad [IAN02], [PAN02].

Como veremos más adelante, el lenguaje que utilizaremos en esta Tesis para describir nuestra ontología, OWL, dispone de varios niveles de expresividad. Según su nivel, implementa uno u otro lenguaje de lógica descriptiva, por lo que es importante considerar y ponderar la expresividad del lenguaje y la complejidad de los razonamientos, para elegir el nivel de OWL óptimo.

Nombre	Componentes	Complejidad
$\mathcal{ALC}$	$\sqcup$, $\sqcap$, $\neg$, $\exists$R.C, $\forall$R.C	ExpTime-c
$\mathcal{S}$	$\mathcal{ALC}$ + roles transitivos	ExpTime-c
$\mathcal{SI}$	$\mathcal{S}$ + roles inversos($\mathcal{I}$)	ExpTime-c
$\mathcal{SH}$	$\mathcal{S}$ + jerarquía de roles ($\mathcal{H}$)	ExpTime-c
$\mathcal{SHIQ}$	$\mathcal{SHI}$ + restricciones numéricas (OWL LITE)	ExpTime-c
$\mathcal{SHIQ0}$	$\mathcal{SHIQ}$ + nominales (OWL DL)	NExpTime-hard
$\mathcal{SHIQ0(D)}$	$\mathcal{SHIQ0}$ + Datatypes orden 1	NExpTime-hard
$\mathcal{SHIQ0(Dn)}$	$\mathcal{SHIQ0}$ + Datatypes orden n	NExpTime-hard
$\mathcal{SHIQ}$+	$\mathcal{SHIQ}$ + rest. Numéricas en cualquier sitio	No resoluble
$\mathcal{SH}$+	$\mathcal{SH}$ + rest. Numéricas en cualquier sitio	No resoluble

Tabla 3.9. Lógicas DL

En la tabla 3.9 podemos ver órdenes de complejidad con un coste temporal exponencial respecto a la complejidad de nuestro problema (número de elementos), llegando a no ser resoluble en los lenguajes más expresivos. Los aumentos de expresividad principales son por la inclusión de:

- **Roles transitivos**, cuando a está relacionado con b y b está relacionado con c por la misma propiedad:

$$R(a,b) \wedge R(b,c) \rightarrow R(a,c) \qquad \text{[Expr. 3.29]}$$

- **Roles inversos**, un Rol es inverso a otro cuando al aplicar ambos el elemento sobre el que se aplica da como resultado el mismo elemento.

$$R^{-}: R^{-}(R(x)) \equiv x \qquad \text{[Expr. 3.30]}$$

- **Jerarquía de roles**, cuando un Rol es una especialización de algún otro, por lo que los elementos que se relacionan con el rol hijo también se relacionan con el rol padre. Pongamos como ejemplo que el Rol *serHijoDe* es una especialización de *serFamiliaDe*, por lo que todo elemento que se relacionen con otro por medio del Rol *serHijoDe* se relacionará también por medio del Rol *serFamiliaDe*.

$$R \sqsubseteq P \qquad \text{[Expr. 3.31]}$$

- **Nominales**. Los nominales nos permitirán definir una clase enumerando uno a uno los elementos que la contienen: Por ejemplo, la clase de días de la semana incluiría a {Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo}

- **Restricciones numéricas**: $\leq$ n, n $\in$ {1,2,…}. Lo común es restringirlas a los Roles. Por ejemplo:

$$\leq 3 \text{IMPARTE.Materia} \qquad \text{[Expr. 3.32]}$$

- **Tipos de Dato (Datatypes)**. Los tipos de datos son números, strings, etc, que se estudian como un conjunto disjunto de Δ^I , llamado Δ_D^I , definiendo unos nuevos tipos de Roles (en este caso se denominan propiedades/predicados de tipos de datos (datatype properties)). Por ejemplo, EDAD(JUAN) relaciona a Juan con el tipo de dato entero 35.

3.2.6.4 Razonamiento en DL

Al haber separado la KB en dos bloques, el razonamiento se divide también en 2 partes:

1. **Razonamiento en la T-Box:**

   - **Posibilidad de un concepto**. Ver si un concepto es posible (Concept Satisfiability). Consiste ver si existe un modelo para un concepto (al menos un individuo que lo cumpla).

   - **Inclusión** (Subsumption). Consiste en comprobar si una nueva aserción se puede inferir directamente de las

aserciones de la T-Box. Si es así, se dice que la nueva aserción introducida no aporta nueva información a nuestro sistema.

- **Posibilidad** (Satisfiability). Es ver si existe algún modelo para esa T-Box. Es decir, no hay ninguna incongruencia/inconsistencia en dicha T-Box. Una clase es inconsistente si en todas las interpretaciones de nuestro sistema esa clase es el conjunto vacío. La T-Box se define como inconsistente si en todas las interpretaciones existe alguna clase que no puede tener individuos

Los tres casos se pueden reducir a Posibilidad [BEN03].

2. **Razonamiento en la A-Box:**

- Comprobar si una **instancia pertenece a una clase** determinada (Instance Checking o Inferred Types). Por ejemplo, saber si Juan pertenece a la clase [profesor ⊓ ingeniero].

- **Consistencia**. Si lo que decimos en la A-Box es coherente con ella misma y con lo descrito en la T-Box. Por ejemplo, si decimos que Juan es ingeniero y que Juan no es ingeniero.

Actualmente hay pocos razonadores que manejen razonamiento en la A-Box. Uno de los más utilizados es Racer (Reasoner for Aboxes and Concept Expressions Renamed)[WWW13].

3.2.6.5 Suposición de mundo abierto (OWA)

Una de las razones por las que DL ha sido escogido para la implementación de web semántica es que no supone que el conjunto de aserciones que tiene su A-Box es invariante, y que se contiene toda la información necesaria para resolver cualquier problema.

Por ejemplo, si en nuestra A-Box hablamos de profesor Juan que imparte dos materias pertenecientes a Sistemas de Información, esto no nos permite inferir que TODAS las materias que imparte el profesor son de Sistemas de Información, ya que puede que descubramos en algún momento que imparte otro tipo de materias. Sólo podríamos asegurarlo si:

1. Decimos que un profesor puede dar como máximo 2 materias (si nuestra DL permite restricciones numéricas).

   `Profesor: (≤2 IMPARTE)Materias` [Expr. 3.33]

2. Añadimos directamente la aserción de que Juan sólo da materias de Sistemas de Información.

   `Juan: ∀IMPARTE.Materias_SI` [Expr. 3.34]

Esto nos permitirá trabajar de forma incremental en la web, según vaya apareciendo información nueva o los agentes la vayan descubriendo. Debido al tamaño y la velocidad de cambio de la web, la suposición de un mundo cerrado sería inapropiada [OWL04A].Supongamos que utilizáramos un agente que realizara inferencias bajo la suposición de mundo cerrado. Como la cantidad de información existente en la web es enorme y va creciendo, por razones de espacio y tiempo para el almacenamiento, le sería imposible almacenar toda la información disponible en la web. Además, aunque recogiera una gran cantidad de información, cuando realizara inferencias y apareciera publicada nueva información al respecto en la web, debería de realizar de nuevo las inferencias, que podrían darle resultado diferentes.

Sin embargo, trabajando con la suposición OWA, el agente en principio cogerá la información mínima necesaria hasta llegar a la inferencia que busca, y en caso de aparecer nueva información, sólo ha de comprobar que esa nueva información no contradice a la ya existente en su base de conocimiento. Si no hay contradicción, los razonamientos previos a la introducción de la nueva información son igualmente válidos.

Bajo la suposición OWA, que permite la incorporación de aserciones nuevas, puede darse el caso de que se introduzca nueva información en nuestro sistema que afecte a la T-Box. Se denomina Clasificación a la tarea de insertar nuevos conceptos a la Taxonomía, que puede hacer aparecer clases inconsistentes y relacionar clases entre sí [FRAN02].

3.2.6.6 Suposición de nombres únicos (UNA)

Esta suposición indica que dos instancias con dos URIs distintas son necesariamente instancias diferentes. Esta suposición simplifica los queries y además hace que no se envíe una gran cantidad de información. El ahorro de envío se información se debe a que, si no hay suposición de nombres únicos, he de indicar explícitamente qué instancias son distintas entre sí.

Sin embargo, el lenguaje ontológico que vamos a utilizar (OWL) no permite hacer esta suposición de nombres únicos. Dos URIs distintas pueden, a no ser que se demuestre lo contrario, representar al mismo recurso. Esto nos obliga a realizar grandes cantidades de aserciones cuando tenemos claro que los recursos representan objetos reales completamente diferentes.

Por otro lado, se adapta más a los sistemas web en los que no se dispone de toda la información desde el principio. Al ser la web un sistema distribuido en el que la información sobre una misma instancia puede venir de dos fuentes distintas, sería inviable centralizar la asignación de URIs a instancias. Al no realizar la suposición de nombres únicos, puedo ir recogiendo información de varias fuentes y luego relacionarlas según vaya descubriendo que dos URIs son realmente la misma instancia. Realizando un símil en la vida real, puedo recibir información acerca de un primo lejano, y a su vez información acerca del nuevo novio de una amiga de mi mujer. Meses más tarde puedo llegar a descubrir que ambas personas son en realidad la misma.

Trabajar con un razonador que permita que varios nombres sean la misma instancia nos obliga a ser cuidadosos con la información que introducimos en el sistema, ya que estamos obligados a indicar las clases que son disjuntas y las instancias que son distintas para evitar razonamientos erróneos. Como ejemplo supongamos que pongo una restricción consistente en que un humano sólo puede tener un padre. Si indico que el padre de Juan es Pedro y luego introduzco que el padre de Juan es Ángel, en lugar de indicar que existe una incongruencia en la información facilitada, concluirá que Ángel es la misma persona que Pedro.

3.3 Lenguajes ontológicos

3.3.1 Introducción

Existen múltiples definiciones de ontología, según hablemos de términos filosóficos, lingüísticos o de sistemas de información. Ontología, en términos de sistemas de información (nuestro ámbito) se puede entender como “una especificación explícita de una conceptualización” [GRUB93].

Una ontología “define los términos usados para describir y representar un área de conocimiento” [OWL04A] (como medicina, arte, etc). Las ontologías se utilizan por la gente y por sistemas computacionales para

compartir información de un dominio (entendiendo como dominio una porción determinada de un área de conocimiento).

Las ontologías se componen de

- **Objetos o instancias**, que podrían ser una representación de un objeto en la realidad (por ejemplo, la mesa de mi despacho). También se conocen como instancias, o *individuals*.

- **Clases**, que representan a un conjunto de objetos. Aunque la palabra "concepto" se utiliza a veces como sinónimo [IAN03], otros autores [HOR04] relacionan estas dos definiciones diciendo que las "clases" son concretas representaciones de los "conceptos" (ideas que tenemos en la mente).

- Las **propiedades** que esos objetos pueden tener, siempre dentro de nuestro dominio de interés. También se conocen como roles, slots o atributos.

- **Relaciones** que pueden haber entre dichos objetos, que la mayoría de lenguajes ontológicos agrupan dentro de las propiedades de dichos objetos.

Una ontología se expresará en un lenguaje basado en sistemas lógicos (lenguaje ontológico) que servirá para almacenar nuestro conocimiento sobre el dominio de interés. Algunos autores utilizan el término teoría para referirse a estas ontologías, ya que a veces son relativamente pequeñas y se pueden completar con otras teorías [MAR02]. Las ontologías tienen características diferenciadoras respecto a una base de datos que almacene la información de nuestro sistema, donde las tablas sean clases, las tuplas de las tablas instancias y las relaciones entre instancias relaciones simples o múltiples entre los identificadores de las tablas. Los elementos diferenciadores entre una ontologías y la información almacenada en una base de datos son:

- Permiten que los objetos tengan propiedades (serían los equivalentes a campos de una tabla de base de datos) que no estén en la definición de la clase a que pertenecen. Cuando definimos una tabla que contiene la información de un objeto, hemos de indicar a priori el número de campos y el tipo de datos a almacenar en cada campo.

- Permiten múltiples definiciones de clases y propiedades. En una base de datos normalmente existe una tabla fija para el almacenamiento de información de un objeto.

- La información sobre un objeto no tiene por qué estar solamente en un documento (aunque es cierto que existen bases de datos distribuidas, en estos lenguajes se permite mayor libertad). Además, puede ir apareciendo información en documentos distintos.

- No se supone nunca que se tiene un conocimiento total de nuestro dominio de interés, por eso existen frases como "las pizzas tienen al menos un ingrediente", junto con información de algunas pizzas en las que no sepamos ningún ingrediente.

- Al estar basados en lenguajes lógicos, podrán procesarse inferencias (deducciones), que podrán realizar sistemas no humanos. No se pueden realizar muchas inferencias automáticas basándonos en una base de datos.

Estas características nos permitirán utilizar las ontologías para mejorar las aplicaciones Web que dispongamos, ya sea para generar directamente portales sobre temáticas determinadas (como por ejemplo OntoWeb [WWW3], WordNet [WWW4] u Ontolingua [WWW5]), para describir fuentes en internet (fuentes multimedia, u otras entidades, como objetos de aprendizaje), para describir la capa lógica de un sistema de tres capas, para diseñar cómo ha de ser la documentación de algo (y poder hacer luego consultas a esa documentación por medio de algo más que el índice y buscar las palabras), para su uso por agentes inteligentes (que veremos más adelante), o incluso para describir los propios servicios que un sistema puede ofrecer (OWL-S).

3.3.2 Tipología de lenguajes ontológicos

Existe una gran cantidad de lenguajes ontológicos, lo que nos obliga a realizar una taxonomía de dichos lenguajes. Una primera clasificación sería [BECH03]:

1. **Lenguajes para representaciones gráficas** (denominados *Graphical notations*). Estos lenguajes no se basan en lenguajes lógicos, sólo en la sintaxis de las ontologías (clases, instancias, propiedades que las relacionan entre sí). Si bien en este trabajo no se

considerarán como lenguajes ontológicos, por su deficiencia en lógica para poder realizar inferencias, sí son la base para la gran mayoría de lenguajes ontológicos y para representar ontologías gráficamente. Dentro de este grupo encontramos:

- **las redes semánticas**, entendidas como un conjunto de nodos relacionados entre sí por flechas. Hay sistemas ontológicos fuertemente relacionado con ellos, como WorldNet [WWW4],o sistemas de enseñanza basados en conceptos, como AHA!(Adaptative Hypermedia Architecture) o AIMS (Agent-based Information Management System)[ARO02].

- **los mapas de conceptos** (denominados en inglés *topics maps* [WWW7]), entendidos como una ampliación de las redes semánticas incluyendo el concepto de ocurrencias, que son referencias a fuentes externas al mapa de conceptos (por ejemplo una referencia bibliográfica). Sistemas como Citeseer [WWW9] utilizan estas uniones externas (almacena uniones con los documentos en otros lugares fuera de Citeseer). Ejemplos de lenguajes basados en este sistema son el XML Topic Map (XTM [WWW10]) y el ISO/IEC 13250[WWW11].

- **UML** (Unified Modeling Language [WWW8]), que como lenguaje que permite representar objetos y sus relaciones (aparte de otras muchas posibilidades) se puede entender como uno de estos lenguajes.

- **RDF** (Resource Description Framework[WWW2], que desarrollamos más en profundidad en otros apartados.

2. **Lenguajes basados en lógica**. Los lenguajes lógicos se pueden definir como lenguajes formales para representar la información, de forma que se puedan inferir conclusiones. Conllevan una sintaxis, que define el tipo de sentencias que pueden darse en el lenguaje, y una semántica, que define el significado de dichas sentencias. Los lenguajes utilizados para describir ontologías, y que se basan en lenguajes lógicos son los Lenguajes basados en lógica. Estos lenguajes se considerarán en esta Tesis como leguajes ontológicos, ya que incluyen en su sintaxis definiciones lógicas que permiten

utilizar razonadores lógicos para inferir resultados. Según la lógica utilizada tenemos:

- **Basados en Lógica Descriptiva** (DL *Description Logics*). En este caso tenemos OIL, DAML+OIL, OWL. Estos lenguajes son los que trataremos en la Tesis, por lo que los describiremos con más detalle en otro apartado.

- **Basados en lógica de primer orden** (FOL *First Order Logic*), como es el caso de KIF (Knowledge Interchange Format) y sus sub-lenguajes. Se pude decir que la lógica de primer orden comprende a su vez la lógica descriptiva. También trataremos sobre KIF en los siguientes apartados.

- **Basados en Reglas** (denominadas en inglés como *Rules*), como RuleML [WWW12], LP/Prolog [GUP01]. Una regla en este caso se entiende como "unidades de conocimiento autocontenidas que implican algún tipo de razonamiento" [WAG03]. En cierto modo se puede decir que la lógica clásica es un subconjunto de las posibles reglas. Otros ejemplos de reglas son reglas de reacción (cuando pase X, hacer Y), y otros que veremos más adelante cuando hablemos de RuleML y SWRL.

- **Gráficos conceptuales** (CG *conceptual graphs*), sistemas basados en los gráficos de Charles Sanders Peirce y utilizados para sistemas de inteligencia artificial.

- **Basados en lógicas de mayor orden**, como LBase. El mayor orden en este caso se refiere sintácticamente, es decir poder relacionar conjuntos de elementos de discusión (y no sólo parejas) directamente, sin hacerlo a pares.

- **Lógicas no clásicas** (F-logic, Non-Monotonic). Estas lógicas no se basan en el concepto de verdadero o falso, sino en grado de verdad (probabilidad) de una aserción. Estos lenguajes no serán tratados en este estudio.

- **Lógicas probabilísticas o difusas** (fuzzy logics) Estos lenguajes suelen ser utilizados por sistemas de redes neuronales, que no se estudiarán en esta Tesis.

Como en muchos otros sistemas, cuando más restrictiva es la definición (en este caso la lógica utilizada) más sencillo es su manejo (razonadores más rápidos, descripciones más sencillas, pero más difícil es que nuestros problemas reales se adapten a dichos sistemas. Los lenguajes basados en lógica que se acaban de enumerar se han ordenado de más restrictivo a menos, siendo el más restrictivo los basados en Lógica Descriptiva. Más adelante veremos las distintas lógicas arriba mencionadas.

A continuación veremos los lenguajes ontológicos más importantes utilizados de una u otra forma en internet. Algunos de los lenguajes existían antes de utilizarse en internet y ahora se utilizan como lenguaje intermedio entre sistemas y otros lenguajes se han diseñado desde cero pensando en la comunicón via web.

3.3.3 KIF/SKIF

El Knowledge Interchange Format (KIF) [GEN98] es un lenguaje que se desarrolló con la idea de que fuera un lenguaje de intercambio de conocimiento entre los distintos sistemas desarrollados para tratar el conocimiento (bases de conocimiento y sistemas de inteligencia artificial). SKIF [HAY01] se puede considerar como una versión de KIF más adaptada a su uso en la Web Semántica (permite el uso de URIs, UNICODE y una sintaxis más sencilla, no distinguiendo a priori entre las instancias, las clases y las relaciones entre ellos (todo se agrupa como términos). De este modo podemos decir que algo que es una clase y tiene sus instancias puede ser a su vez una instancia de otras clases. Por ejemplo, en la figura 3.6, vemos el Modelo de Teoría (MT) de SKIF. En este caso tenemos una ontología de enseñanza, donde se tiene lugar el caso en el que el término *Alumno* puede ser una instancia de la clase *Miembro* (de un curso), y a su vez una clase que contenga a los alumnos *Eva* y *Juan*, y además podemos indicar que los alumnos y los profesores se conocen entre sí, por medio de la propiedad *CONOCER*.

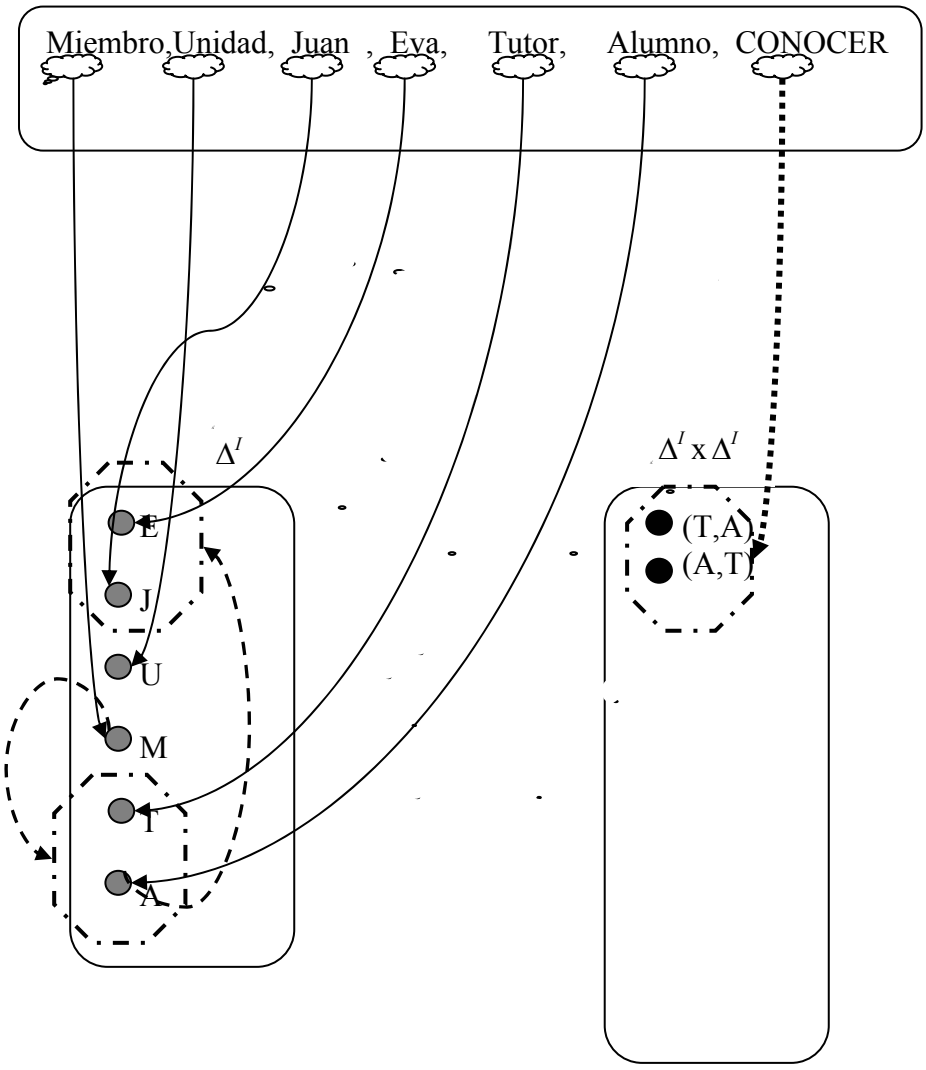


Figura 3.6. Ejemplo de MT de SKIF

Ambos lenguajes se basan en FOL, y utilizan una sintaxis similar a la utilizada en el lenguaje deprogramación declarativa LISP [LUG02], y mientras son un buen leguaje para intercambio, hay pocos razonadores que trabajen con SKIF, y normalmente se ha de hacer una transformación previa a FOL [IAN03].

3.3.4 RuleML

RuleML es un lenguaje basado en reglas, y en principio orientado a su uso en la Web. Las reglas comprenden a la lógica clásica, de forma que puede haber reglas en el lenguaje para definir más profundamente las ontologías. Las reglas se dividen en distintos tipos, pudiendo en algunos casos transformar las reglas de un tipo en reglas de otros tipos.

Los tipos de reglas propuestas en RuleML son:

- **Reglas de Integridad**. Sirven para definir, por ejemplo las restricciones de integridad (integrity constraints) que se colocan en una base de datos relacional (por ejemplo, una fila de la tabla *facturas* tiene un campo *id_cliente* cuyo valor ha de estar obligatoriamente como identificador de alguna fila de la tabla *clientes*).

- **Reglas de Derivación**. Se basan en un conjunto de condiciones y una consecuencia si se dan las condiciones anteriores. Tanto las condiciones como la consecuencia se representa mediante una fórmula lógica. Por ejemplo, decir que "si una factura no está pagada y tampoco está anulada, entonces la factura es una factura por cobrar".

- **Reglas de Reacción**. Estas reglas tienen cierta implicación "temporal", es decir, tienen un disparador, una condición de inicio y una acción a realizar. Hay dos tipos: Evento-Condición-Acción (ECA Rules) y Evento-Condición-Acción-Postcondición (ECAP Rules). ECA se pueden ver como los triggers de una base de datos relacional y los ECAP cuando se expresa de forma abstracta que al terminar una acción ha de modificar siempre algo (casi como si la última sub-acción de la acción sea "modificar X", pero declarándolo de forma que se puedan inferir resultados de ese dato).

- **Reglas de Producción.** Expresa qué se ha de hacer cuando se da una condición. Son muy parecidas a las reglas de reacción, con la diferencia de que una regla de producción no tiene en sí el concepto "temporal" de reacción en el momento que ocurre algo.

- **Reglas de Transformación**. Son reglas que explican como pasar de una instancia de un tipo a su equivalente de otro tipo. Por ejemplo, cómo transformar una instancia compleja de tipo "noticia completa"

a una de tipo "resumen de noticia", en el que le decimos se quede con el titular y el primer párrafo de la noticia.

RuleML se encuentra todavía en un estado de desarrollo, y se está estudiando incluir otros tipos de reglas, como reglas de permisos u prohibiciones, y reglas de asignación de tareas.

3.3.5 OWL

3.3.5.1 Introducción e Historia

El lenguaje ontológico web (Web Ontology Language, OWL) es un lenguaje basado en RDFS (lenguaje ya descrito en otro apartado), como parte de lo que se entiende como Web Semántica.

OWL apareció como recomendación en febrero del 2004 [OWL04A][OWL04B]. Su diseño se ha visto influenciado por más de 10 años de investigación en Lógica Descriptiva (DL) [HOR03]. Su primer predecesor se puede considerar SHOE [HEF99], un lenguaje basado en Frames con características muy apreciadas en la web:

- Una **sintaxis construida sobre XML**, lo que permitía embeberlo dentro de las páginas HTML.

- Los **nombres son URIs**, lo que le permite reutilizar la gran ventaja de los nombres únicos de los recursos, y evitar ambigüedades.

- Suposición de que **las ontologías pueden reutilizarse, cambiar y ampliarse** en el tiempo, por lo que ofrece posibilidades para importaciones, alias locales para los nombres largos, e información sobre versiones y compatibilidades hacia atrás.

Otro lenguaje que influenció de forma destacable a OWL fue DAML-ONT. La iniciativa americana DARPA Agent Markup Language (DAML) nació en 1999 con la idea de ser la base de la Web Semántica. Ya se basaba en RDFS, pero se decidió enriquecer su sintaxis para aumentar su expresividad, por lo que se creó el lenguaje DAML-ONT (también conocido simplemente como DAML)[HEN00].

La alternativa europea fue Ontology Inference Layer (OIL), que buscaba objetivos similares a DAML. OIL fue el primero en basarse en Lógica

Descriptiva (de hecho, fue un lenguaje que representaba $\mathcal{SHIQ}$ DL), a su vez de lenguajes basados en Frames y estándares Web. Sin embargo, su compatibilidad con RDFS no era completa [HOR03].

El antecesor más cercano de OWL estaba a punto de aparecer. Los esfuerzos de DAML y OIL se unieron en una iniciativa entre Europa y Estados Unidos, que se llamó DAML+OIL[CON01], que ya se basó en Lógica Descriptiva y a su vez fue más compatible con RDFS, aunque surgieron algunas incompatibilidades cuando salió la versión definitiva de RDF y RDFS.

OWL es la apuesta de DARPA para la web semántica, ya que permite varios grados de complejidad (varios niveles), así como una sintaxis robusta y probada, y un nivel de expresividad alto.

3.3.5.2 Tipos de OWL

Como hemos visto anteriormente, hay varios tipos de Lógica Descriptiva, en función de su expresividad y la complejidad computacional de realizar razonamientos. Es por esto que OWL se presenta con tres niveles del lenguaje, en función de lo expresivo (y por tanto peor a la hora de inferir) que sea, lo que nos permite para sistemas más sencillo limitar la expresividad buscando una mayor efectividad de cálculo. De menos a más expresivo tenemos:

- **OWL Lite**, es el más sencillo y permite una rápida migración desde otros lenguajes ontológicos más simples. Su lógica tiene unas restricciones numéricas muy limitadas.

- **OWL DL**, que se ha diseñado para soportar el lenguaje $\mathcal{SHIQO}(\mathcal{D})$, más expresivo que el anterior pero con un notable aumento de complejidad (siempre considerando el caso peor), aunque existen razonadores bastante probados para este sistema (Racer). Los tipos de datos introducidos son los permitidos por XMLS. OWL DL es el lenguaje más apropiado para aplicaciones que requieran el máximo de expresividad pero que exijan completitud computacional, ya que todas las conclusiones son computables.

- **OWL Full**, que aunque no aumenta la sintaxis de OWL DL, sí que permite un cambio conceptual grande. Tanto OWL Lite como OWL Full tienen bien separados las instancias (y los datatypes en OWL DL), las clases y las propiedades, OWL Full mantiene la relajación

de RDFS, permitiendo que una URI sea una clase y a su vez la instancia de otra clase, o incluso una propiedad. Esto obliga a que en OWL Lite/DL cuando hablemos de algo tengamos que decir siempre si es una clase, una instancia, una propiedad (y de qué tipo), etc. Este nivel de lenguaje OWL no tiene procedimientos de inferencia completos y seguros.

3.3.5.3 Constructores y axiomas en OWL

Puesto que OWL es el lenguaje que se va a utilizar en esta Tesis, pasamos a describirlo con más detalle. Su sintaxis se divide en constructores, también denominados descripciones de Clase, y axiomas. Los constructores nos permitirán definir Clases, tanto anónimas como no anónimas, y los axiomas serán los elementos por los que podremos realizar aserciones sobre las Clases y sobre las Instancias de las mismas, así como sobre las Propiedades que relacionan a las Instancias. Mientras en la definición formal de OWL [OWL04b] se consideran los constructores de Clase como un subconjunto de los axiomas de clase, en esta Tesis hemos separado los dos términos, ya que el significado semántico de constructores y axiomas es diferente.

Hemos de remarcar que OWL tiene 3 grupos de propiedades (P) (constructores de propiedades) (OWL Lite sólo una):

- **Propiedades de instancia (P_C)**, donde el objeto de la propiedad es un individuo de Δ^I (owl:ObjectProperty). Es la única válida en OWL Lite.

- **Propiedades de tipos de datos (P_D)**, donde el objeto de la propiedad es un individuo de Δ_D^I (owl:DatatypeProperty).

- **Propiedades de anotación y de ontología**, externas a la parte de lógica (owl:AnnotationProperty, owl:OntologyProperty) donde se almacenan breves explicaciones, temas de importación de ontologías, versiones, compatibilidad. Los principales tipos predefinidos son: owl:versionInfo, owl:imports (para definir la ontología, junto con otros sobre la compatibilidad), rdfs:label, rdfs:comment, rdfs:seeAlso, rdfs: isDefinedBy y

Además se les define a las propiedades rango (rdfs:range) y un dominio (rdfs:domain).

Para las instancias tenemos 2 tipos de constructores:

a) De pertenencia a una clase (aserciones sobre conceptos). Por ejemplo:

```
<Profesor rdf:ID="Juan"/>
```
[Ej. 3.7]

En este caso definimos las instancia *Juan* como un elemento de la clase *Profesor*.

b) Cuando le damos el valor a una propiedad (aserciones sobre roles). En el ejemplo 3.8 podemos ver cómo definimos la instancia *Juan* existe indicando que la instancia *Juan* se relaciona con la instancia *Telemática* por medio de la propiedad *imparte*.

```
<rdf: Description rdf:about="#Juan">
<imparte rdf:resource="#Telematica">
</rdf Description>
```
[Ej. 3.8]

En las tablas 3.10 y 3.11 vemos la notación OWL para los constructores de Clases (Tabla 3.10) y los axiomas (Tabla 3.11), siendo C, D una clase, x_i una instancia, P una propiedad cualquiera, P_D una propiedad de tipo de datos, P_C un propiedad de objetos (relaciona un objeto con otro), como ya definimos en la tabla 3.8.

Constructor	Sintaxis DL	OWL Lite	OWL DL	OWL Full
owl:Class	C, $[C^I \subseteq \Delta^I]$	X	X	X
owl:intersectionOf (3)	$C \sqcap D$	X	X	X
owl:unionOf	$C \sqcup D$		X	X
owl:complementOf	$\neg C$		X	X
owl:allValuesFrom	$\forall P.C$	P_C	P	P
owl:someValuesFrom	$\exists P.C$	P_C	P	P
owl:maxCardinality (1)	$\leq nP$	P_C, n={0,1}	P	P
owl:minCardinality	$\geq nP$	P_C n={0,1}	P	P
owl:Thing	T, $[\Delta^I]$	X	X	X
owl:Nothig	$\perp$	X	X	X
owl:oneOf (2)	$\{x_1,\ldots,x_n\}$		X	X

owl:hasValue	P=x ó P ə x, x Datatype		X	X

Tabla 3.10. Constructores de clases en OWL

En la tabla 3.10 se presenta los distintos constructores de clases, es decir, las formas de definir una clase, indicando el nombre de la clase, o como una función lógica entre otras clases. También se pueden definir con los cuantificadores de existencia $\exists$ y universal $\forall$. También se introduce cuantificadores de cantidad ($\leq$nP, $\geq$nP). A modo de aclaración diremos:

- Existe el constructor *owl:cardinality*, que representa = nP, pero es derivable de los constructores (1) *owl:minCardilatity* y *owl:maxCardinality*.

- En el constructor *owl:oneOf*(2), para enumerar los individuos se usan las listas de RDF, utilizando las marcas *rdf:List*, *rdf:first* y *rdf:rest*. Como simplificación se puede utilizar el parseador *rdf:parseType=Collection* para instancias, donde sólo hemos de enumerar a los miembros [OWL04B].

- En OWL Lite el constructor *owl:intersectionOf* es utilizable sólo cuando se intersecta más de una clase y sólo para intersectar clases. Esto se hace para evitar que se haga una enumeración indirecta de instancias.

- Hemos de tener presente que OWL Full relaja las diferencias entre C (clases), P (propiedades) y x (instancias), por lo que algunos constructores comparten el mismo significado.

Axioma	Sintaxis DL	OWL Lite	OWL DL	OWL Full
rdfs:subClassOf (1)(3)	$C_1 \sqsubseteq C_2$	X	X	X
owl:equivalentClass (3)	$C_1 \equiv C_2$	X	X	X
owl:disjointWith	$C_1 \sqsubseteq \neg C_2$		X	X
owl:sameAs	$x_1 = x_2$	X	X	X
owl:differentFrom (2)	$x_1 \neq x_2$	X	X	X
rdf:subPropertyOf	$P_1 \sqsubseteq P_2$	P_C	P	P
owl:equivalentProperty	$P_1 \equiv P_2$	P_C	P	P
owl:inverseOf (4)	$P_1 \equiv [P_2]^-$	P_C	P	P

owl:transitiveProperty	$P[P] \sqsubseteq P$	P_C	P_C	P_C
owl:functionalProperty (4)(5)	$T \sqsubseteq \leq 1P$ (5)	P_C	P	P
owl:inverseFunctionalProperty (4)	$T \sqsubseteq \leq 1P^-$	P_C	P_C	P_C
owl:SymmetricProperty	$P \equiv [P]^-$	P_C	P_C	P_C

Tabla 3.11. Axiomas en OWL

Podemos ver en la tabla 3.11 los distintos axiomas del lenguaje ontológico, junto con las marcas XML utilizadas para cada uno. También se indican las diferencias existentes entre los tres niveles de OWL. A modo aclaratorio podemos especificar los siguiente:

- De nuevo debemos tener presente que OWL Full relaja las diferencias entre C, P y x.(1).

- OWL supone que dos URIs distintas pueden referirse a la misma instancia/objeto/propiedad, por lo que se ha de constatar si son iguales o distintos explícitamente. Se especifica también una notación *owl:allDifferent* para enumerar instancias todas diferentes entre ellas, pero es el mismo axioma que *owl:differentFrom*(2).

- En OWL Lite sólo permite en el uso de los axiomas *rdf:subClassOf* y *owl:equivalentClass*(3) que el sujeto sea un nombre de clase y el objeto un nombre de clase o propiedad.

- En OWL DL, para el uso de los axiomas *owl:inverseOf, owl:functionalProperty* e *owl:inverseFunctionalProperty* (4) no se permite combinar P_C y P_D, puesto que el lenguaje OWL DL distingue entre P_C y P_D.

- Respecto a los tipos de datos, se soportan los de XMLSchema, pero en principio se pueden definir nuestros propios tipos de datos (serían tipos de datos no reconocibles) o restringir un tipo de datos a un rango de un XMLS Datatype, por medio del axioma *owl:DataRange*, colocando dentro una enumeración con el constructor *owl:oneOf*.

- El significado del axioma para una propiedad *owl:functionalProperty* significa que una propiedad para un sujeto x sólo puede tener un objeto. Por tanto, Si tenemos en nuestra base de

conocimiento la aserción P(x,y) y además la aserción P(x,y2), entonces podremos concluir que y=y2 si P es funcional.

Tras estas últimas aclaraciones ya hemos definido el lenguaje OWL, por lo que a continuación veremos un ejemplo en el que se ha utilizado este lenguaje dentro de la web semántica.

3.3.5.4 Ejemplo de uso: OWL-S

OWL-S fue admitido en W3C en noviembre de 2004 como Draft Proposal. El objetivo del lenguaje es permitir la definición de ontologías que proporcionen un modelo de servicio sin ambigüedades[ZHO06]. Se inició como DAML-S, luego pasó a ser DAML+OIL-S [PAO03]. Actualmente ha evolucionado hacia OWL-S [ANK04], siempre siguiendo las últimas mejoras en lenguajes ontológicos par la Web. La última especificación se puede encontrar en [WWW14].

OWL-S pretende enriquecer los actuales servicios web (Web Services) con información semántica. La infraestructura propuesta se puede ver en la Figura 3.7.

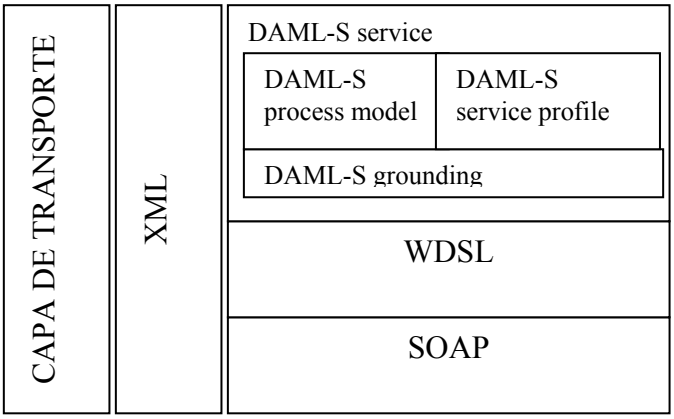


Figura 3.7. DAML-S Infraestructura

El concepto de Servicios Web se genera con la idea de crear una red en la que los programas actúan como agentes independientes que producen y consumen información, automatizando procesos y realizando transacciones

B2B (business to business). La relación entre esta idea y que exista un lenguaje que permita transmitir cierta semántica a estos agentes es directa.

El sistema utiliza ya lenguajes existentes en la web. Se basa en XML/OWL, y ha de comunicarse con WDSL (Web Services Description Language) En la comunicación OWL-WDSL se describe el interfaz del servicio a utiliza o a ofrecer, a modo de descripción de una API). Se utiliza también SOAP (Simple Object Access Protocol), utilizado en la web para pasar mensajes entre servicios web. WSDL y SOAP se basan a su vez sintácticamente en XML.

La Ontología propuesta en OWL-S para un servicio se basa en dividirlo en 3 conceptos:

- Lo **que necesita para ser invocado y qué devuelve**. Este bloque es el Service Profile. Es lo que utiliza para publicarse en la Web.

- **Información acerca del funcionamiento**, qué modelo sigue. Este bloque es el Service Model, y explica cómo funciona el servicio. Esta información se suele usar para poder encadenar varios Servicios y obtener un resultado al final de esa combinación de servicios, o para en un futuro poder monitorizar la ejecución de un servicio. Actualmente existen propuestas [ROU05] en las que se relaja esta parte, permitiendo que no sea parte de OWL-DL, limitándola a lógica de primer orden.

- **Información del modo de uso**, es decir, cómo se relaciona con WSDL y SOAP. Trata de los detalles de formatos de los mensajes, números de puertos, técnicas de serialización.

Es decir, El Servicio presenta (*presents*) un *ServiceProfile*, se describe por medio de (*describedBy*) un *ServiceModel* y soporta (*supports*) un *ServiceGrounding*.

Estos puntos son el inicio de una Ontología, que podemos ver en el ejemplo 3.9:

[Ej. 3.9]

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
```

```
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"

xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns="http://www.owl-
ontologies.com/unnamed.owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
  xml:base="http://www.owl-
ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="ServiceProfile"/>
  <owl:Class rdf:ID="ServiceGrounding"/>
  <owl:Class rdf:ID="Service"/>
  <owl:Class rdf:ID="ServiceModel"/>
  <owl:ObjectProperty rdf:ID="describedBy">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="#ServiceModel"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="supports">
    <rdfs:domain rdf:resource="#Service"/>
    <rdfs:range rdf:resource="#ServiceGrounding"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="presents">
    <rdfs:range rdf:resource="#ServiceProfile"/>
    <rdfs:domain rdf:resource="#Service"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Pero la ontología es más amplia, ya que cada elemento tiene a su vez sus propiedades:

- **ServiceProfile**, tiene nombre (*serviceName*), información de contacto (*contactInformation*) y descripción (*textDescription*), parámetros que pueden ser entradas (*hasInput*) o salidas (*hasOutput*), precondiciones (*hasPrecondition*) y efectos (*hasEffect*), un conjunto de parámetros del servicio (definidos por *serviceParameter* y *sParameter*) y una categorización del servicio (por medio de *categoryName*, *taxonomy*, *value* y *code*). Esto último nos permitirá saber de qué tipo de servicio se trata (reserva de viajes, compra de entradas de cine, etc).

- **ServiceModel**, que describe los procesos por medio de la clase *Process*, y su workflow entre los procesos si el servicio es una

composición de procesos atómicos (por medio de lo que se llama *ControlConstruct*). Cada proceso tiene entradas y salidas.

- **ServiceGrounding**, que conecta con WDSL por medio de la clase *WDSLGrounding*, que contiene la configuración del WDSL (por ejemplo, dice si el servicio realiza una operación o devuelve un documento).

3.3.6 SWRL

Semantic Web Rule Language (SWRL [SWRL04]), es otro lenguaje desarrollado a partir de OWL ampliado con definición de reglas al estilo de RuleML. Las ampliaciones más importantes son:

- **Definición de variables** que representan a instancias (individuals) o a un valor de tipo de datos (datatype). Normalmente se representa para que se entienda la notación no SWRL como ?variable. Por ejemplo, ?x:

```
<swrl:Variable rdf:ID="x"/>
```
[Ej. 3.10]

- **Definición de atom**, como unidad mínima de significado para una regla. Si x e y representan o una variable o una instancia determinada o un valor de tipo de datos determinado, podemos decir que un atom puede ser una definición de clase, C(x), una propiedad, P(x,y) o una aserción de igualdad, sameAs(x,y) , o desigualdad, differentfrom(x,y).

- **Definición de conjunto de atoms**, por medio de <swrl:Atomlist>, a modo de lista RDF.

- **Definición de reglas,** como un conjunto de atoms que generan el antecedente (body) de la regla y otro conjunto de atoms que generan el consecuente (head), que representa la siguiente regla:

$$antecedente \Rightarrow consecuente \qquad \text{[Expr. 3.35]}$$

Si actualmente las herramientas para OWL DL son muy limitadas, las pocas herramientas para SWRL existentes se encuentran, como se indica en [CRE05], aún en un estado menos maduro y habiendo algunas variaciones entre la sintáctica y la sintaxis entre distintas versiones e implementaciones

de herramientas (aunque hay algún ejemplo teórico de desarrollo como en [GUE05]), por lo que su uso ha sido descartado para esta Tesis.

Tras haber revisado los lenguajes ontológicos, sus posibilidades expresivas y su complejidad de computación, concluimos que el lenguaje óptimo actualmente para desarrollar nuestra Tesis, siendo el más utilizado en la web semántica, es OWL, y en este caso especificaremos como óptimos el nivel OWL DL, que conjunta una gran expresividad con una computación costosa pero posible.

Capítulo 4: Primeras interacciones: Evaluaciones

4.1 Introducción

En los capítulos anteriores se ha realizado un estudio del estado del arte en los ámbitos que abarca esta Tesis, relativos a teleeducación y web semántica, que incluyen a su vez el estado de la web actual, las ontologías de teleeducación, los conceptos de lógica descriptica y de racioncinio, y por último los lenguajes ontológicos actuales. A partir de este punto y a lo largo de los restantes capítulos se empezarán a describir los desarrollos y resultados de esta Tesis.

Se comenzará en este capítulo 4 por tratar los sistemas de evaluaciones y ontologías relacionadas, tratando a fondo los estándares más utilizados y presentando una ontología desarrollada (TeML) que contiene algunos conceptos innovadores. Dichos conceptos, junto con las principales aportaciones ontológicas del estandar más utilizado para la exportación de

evaluaciones (IMS QTI) serán utilizados para una parte de la ontología que especificaremos en el capítulo 6. En el capítulo 5 se propondrá una arquitectura que permita las búsquedas semánticas en la web por parte de los agentes inteligentes, así como el acceso a los objetos de aprendizaje. Por tanto la arquitectura nos permitirá explotar la ontología desarrollada en esta Tesis de una manera eficiente para la distribución de contenidos educativos. Por último en el capítulo 7 se realizará una prueba de concepto cuyos resultados demuestren y validen el funcionamiento de distintas partes de la arquitectura propuesta así como de la ontología desarrollada en esta Tesis, aportando además resultados de eficiencia.

Una evaluación es un conjunto ordenado de elementos (ítems) que se usan para evaluar los conocimientos de un candidato en un campo determinado (outcomes). Dichos conocimientos pueden representar por ejemplo el nivel de dominio de una materia. Una evaluación contiene todas las instrucciones necesarias para presentar correctamente los elementos y para realizar el cálculo de los valores de los resultados (por ejemplo, la nota final de un test). Esta definición está basada en IMS QTI[QTI06].

Después de las interacciones básicas de las página HTML (links), se puede decir que las evaluaciones fueron el primer grupo de interacciones que apareció en la Teleformación, y hoy en día son las interacciones más comunes en un proceso de aprendizaje vía web. Se fundamentan en la existencia de un conjunto de preguntas o actividades sencillas que el estudiante debe completar basándose en sus conocimientos, y normalmente permiten una corrección automática o semi-automática de las mismas. El resultado de esta corrección nos permitirá evaluar si el alumno ha alcanzado los objetivos de aprendizaje necesarios.

Las evaluaciones se desarrollaron con anterioridad a la aparición de la teleformación, y vienen sobre todo aparejadas con cualquier formación a distancia, que es el caso en el que el tutor/docente tiene más dificultad para evaluar directamente y de forma continua al alumno.

El uso de los ordenadores para la evaluación y autoevaluación de los estudiantes no es una novedad. Existen algunas aplicaciones desarrolladas desde hace mucho tiempo como pueden ser TRIADS [MAC98], QUIZIT [TIM97] [QUIT] o Questionmark [QMK06], y cada día aparecen aplicaciones nuevas que se suman a la gran cantidad de aplicaciones actuales. Además, la mayoría de plataformas de e-Learning ofrecen servicios de creación y uso de evaluaciones o tests, como ocurre en

WebCBT [WCT] (en 2006 absorbido por Blackboard [WWW72]), SAKAI, .LRN y Moodle. Se pueden encontrar comparativas como en [GAM01] [PAL99], o incluso realizar comparativas de las últimas versiones de las plataformas de e-Learning como Edutools [EDT].

Hoy en día las evaluaciones on-line han llegado a un nivel de madurez que ha permitido desarrollar especificaciones muy completas como es el caso de la especificación IMS Question & Test Interoperability (QTI) V2 [QTI06].

4.2 Motivación de las evaluaciones

El objetivo principal de los tests es evaluar de forma sencilla el progreso del estudiante, tanto por parte de él mismo (autoevaluaciones) como por parte de profesor. En el caso de que los evalúe el profesor, suele ser un inicio para interactuar con el estudiante (vía e-mail, chat o foro) y para orientar si es necesario el proceso de aprendizaje.
Los beneficios que conlleva la introducción de este servicio en nuestros sistemas (tests on-line) de teleformación son principalmente los siguientes [KLEE98], [LLO96]:

- La evaluación on-line, en caso de poder ser corregida automáticamente, permite un rápido feed-back entre profesores y estudiantes, para poder analizar los resultados y mejorar la enseñanza. Esto permite que tanto el profesor como el alumno (en autoevaluaciones) se centren más en los resultados y no en la parte mecánica de corrección de exámenes.

- El feed-back al contestar a un test o una pregunta del mismo puede ser inmediato. Esto aporta grandes ventajas al proceso de aprendizaje, ya que es en el momento que se contesta a un test cuando más interés se tiene sobre el mismo y permitirá que el alumno repase los conceptos que necesita en el momento.

- Al poder manipular los datos de forma electrónica, podemos obtener informaciones generales estadísticas también más rápidamente y de una forma más flexible, permitiendo, por ejemplo, evaluar la dificultad de una pregunta o localizar preguntas que necesitan una modificación en su formulación.

- Permite la inclusión de contenidos multimedia, lo que es una mejora respecto a las evaluaciones en papel.

- Permite presentar exámenes diferentes para cada alumno, según algún algoritmo (que normalmente es obtener las preguntas aleatoriamente de un superconjunto de las mismas). Dicho algoritmo puede basarse en datos del alumno en cuestión, generando un examen personalizado a las características de cada alumno.

- Permite asignar metadatos a las preguntas, exámenes, contestaciones y otros objetos lógicos de un test. Estos metadatos pueden almacenarse en lenguajes propietarios y dependientes de la aplicación o pueden ser metadatos en lenguajes estándar o siguiendo ciertas recomendaciones en el campo.

- Permite correcciones masivas de exámenes, si la corrección se puede automatizar. Esto permite la realización de exámenes masivos y ofrecer los resultados de los mismos en un tiempo razonable, como se ha hecho con los test de sistemas de lectura óptica.

- Permiten medir el progreso realizado tras una experiencia educativa, por medio de una evaluación antes de la experiencia y otra después.

- Permiten medir los conocimientos iniciales de un alumno antes de comenzar la experiencia educativa, para evaluar si cumple los requisitos o se necesita algún material de apoyo para nivelarse.

Como podemos apreciar, obtenemos ventajas debido al tratamiento y almacenaje automático de las evaluaciones, como son el rápido feed-back, ya sea porque la realimentación es automática o porque la rápida corrección del examen permite saber la nota al alumno y al profesor, y reaccionar al respecto. Esta facilidad de tratamiento permite también realizaciones y correcciones de exámenes de forma masiva, ya que no requieren gran carga computacional en los servidores. Además, la facilidad de acceso y tratamiento a los datos almacenados permite, respecto a las preguntas, preparar exámenes personalizados y distintos a cada alumno, generando incluso evaluaciones de acceso para evaluar el nivel de conocimiento inicial. Respecto a las contestaciones, la facilidad de acceso y tratamiento permite realizar dataminig sobre las mismas para extraer información estadística que nos permita obtener conclusiones.

Pero el punto que más nos interesa en la Tesis es la asignación de metadatos a los distintos elementos que participan en una evaluación, de modo que podamos marcar y luego buscar elementos con unas características que haya

definido. Si los metadatos incluyen información ontológica, como puede realizarse con OWL, dicha información permitirá a los agentes realizar inferencias que permitan realizar búsquedas más sofisticadas. Por ejemplo, supongamos que un conjunto de exámenes, sobre la misma materia, tienen una dificultad con un valor 7 sobre 10 (suponiendo que definimos en nuestro sistema dificultades con valores entre 0 y 10). Si mi agente ha de generar un examen de dificultad 4 sobre 10, puede tener definido que todos los exámenes con todas las preguntas de dificultad X sobre 10 son exámenes con dificultad X sobre 10. Entonces escogería las preguntas de mis exámenes iniciales que tuvieran la dificultad que deseamos y los juntaría en un nuevo examen de la dificultad esperada.

4.3 Limitaciones de la Evaluaciones

Una de las mayores limitaciones de toda evaluación a distancia es el control que puede llevarse a cabo del estudiante durante la misma. La propia libertad que nos ofrece la web, que en muchas ocasiones es una ventaja, nos permite realizar las operaciones desde cualquier lugar y en cualquier momento. En evaluaciones este aspecto puede ser contraproducente ya que es muy difícil comprobar que el alumno realiza la prueba en las condiciones exigidas.

En una evaluación presencial, los alumnos se encuentran ubicados en recintos definidos, de forma que se pueden utilizar mecanismos de vigilancia y control para que la prueba se realice cumpliendo con ciertos requerimientos. En el caso de las evaluaciones on line la vigilancia y el control son difíciles de llevar a cabo, de forma que nos encontramos con los siguientes riegos:

- **Suplantaciones**. En este caso es otro individuo distinto al estudiante evaluado el que realiza la prueba, y este hecho es ocultado a los evaluadores.

- **Uso de ayudas externas**. Como por ejemplo el uso o consulta de material no permitido durante la evaluación.

- **Colaboración con otras personas**. En esta situación el alumno evaluado es el que realiza la prueba, pero recibe ayuda externa de otras personas que no están siendo evaluadas, lo cual falsea el resultado.

Actualmente, en las evaluaciones que se realizan vía web, para resolver este inconveniente se recurre a soluciones del tipo que presentamos a continuación:

- **Confiar en la buena voluntad del estudiante**. Un estudiante debidamente motivado no debería perder la oportunidad de comprobar los conocimientos adquiridos.

- **No dar una gran importancia a las evaluaciones** más que para información del alumno. De este modo no existe móvil para recurrir a algún tipo de manipulación de las mismas.

- **Recurrir a pruebas vigiladas** (por ejemplo, realizadas en aulas de academias acreditadas). Se puede incluso realizar vigilancias a distancia por medio de webcams y Netmeeting, por ejemplo. El uso de aulas académicas con ordenadores para todos los exámenes genera problemas logísticos como los planteados en [HOL04].

Otro problema muy común en las evaluaciones vía web se presenta en los sistemas en los que se decide que la computación de la respuesta correcta se haga en el ordenador del alumno, por tanto la información sobre cómo se va a corregir se envía al alumno con una cierta encriptación u ocultación. Este es un problema del que adolece SCORM, donde se define que el SCO procesa (con javascript) si los objetivos de aprendizaje se han cumplido o no[ADL04b].

La solución para este problema es, además de las anteriores planteadas para el caso de evitar fraudes, que no se envíe ninguna información sobre el proceso de cálculo de los resultados de una evaluación al ordenador del cliente, o que se envíe al computador del alumno un valor encriptado por respuesta, que sólo el servidor pueda decodificar y entender si es una respuesta correcta o incorrecta[ROM06].

Y una última dificultad, que supone un riesgo principalmente en evaluaciones oficiales, es el poder verificar que las respuestas obtenidas en la prueba fueron las que efectivamente contestó el alumno y que no ha surgido un problema en el sistema que haya podido modificarlas.

Este problema no se presenta en las evaluaciones presenciales, ya que cuando se valora un documento escrito por el alumno, el propio manuscrito constituye una prueba de su autenticidad.

En el caso de las evaluaciones on line, incluso aunque se utilice algún mecanismo de seguridad como la firma digital del alumno para verificar el envío, no existe una prueba fehaciente de la autenticidad de las respuestas, ya que el alumno podría argumentar que hizo el examen (hecho que probaría la firma digital) pero que esas no fueron sus respuesta, lo que plantearía la posibilidad de un fallo en el sistema..

Para resolver este problema se plantean dos posibles soluciones:

- Cuando la evaluación del alumno tiene lugar en las dependencias de la entidad examinadora, se puede obtener una copia impresa de su examen, que el alumno firma y entrega. Este documento nos servirá como prueba en el caso hipotético de que el alumno planteara alguna duda sobre la autenticidad de la evaluación electrónica que ha realizado.

- Independientemente del lugar donde tenga lugar la evaluación del alumno, se le puede facilitar un documento electrónico con sus respuestas, visado mediante firma digital por la entidad examinadora. Si el alumno no muestra disconformidad, este documento también servirá como prueba en el caso hipotético de que se planteara alguna duda sobre la autenticidad de la evaluación electrónica realizada.

Por tanto, la mayoría de las limitaciones de las evaluaciones en la teleformación parten de la dificultad de asegurar la autoría y las condiciones ambientales en el momento de la realización de las contestaciones, tanto por parte de la entidad, como por parte del mismo interesado a la hora de reconocer el resultado de las mismas. La autoría se puede subsanar por medio de un sistema de firmas digitales, pero el aseguramiento de las condiciones de contorno en el momento de contestación del mismo es un punto aún por resolver si deseamos libertad en el emplazamiento del alumno.

4.4 Metadatos y Ontologías en las evaluaciones

Uno de los puntos clave para el futuro desarrollo de la web semántica es la incorporación de metadatos a los objetos utilizados, de modo que los agentes inteligentes puedan procesarlos y entenderlos para poder realizar inferencias sobre los mismos. De hecho, mientras para los usuarios finales el contenido del texto de una pregunta es la información con la que trata, para los agentes que encuentren esta pregunta puede que les sea más útil conocer

la temática de la pregunta, las posibles respuestas a presentar y el autor. Los agentes no podrán realizar funciones más avanzadas que presentar un texto obtenido mediante un identificador sin la existencia de metadatos que nos indiquen detalles de las evaluaciones con las que tratemos, así como sin una ontología implícita para agentes de uso especializado o explícita para agentes más generalistas. Los metadatos permiten de esta forma facilitar la interoperabilidad de dichas evaluaciones, tanto a nivel del tests como a nivel de preguntas individuales, de modo que podamos exportar cursos completos con sus evaluaciones o montar un curso reusando evaluaciones y preguntas de repositorios heterogéneos.

Como ya hemos comentado, actualmente hay una gran gama de soluciones que cubren las evaluaciones on-line, cada una con su propia definición y ontología implícita. Pese a la gran variedad de soluciones similares e incompatibles entre sí, la incorporación de metadatos tiene lugar en pocas ocasiones, con definiciones propietarias de los mismos y con un único estándar de intercambio limitado como es IMS QTI (basado en XML/XHTML/XMLS).

En este apartado haremos una breve descripción de IMSQTI y sus metadados relacionados, así como sobre la implementación de un modelo completo de evaluaciones vía web, desarrollado en el Departamento de Comunicaciones, y que permite ampliar y comparar el espectro de información disponible en los metadatos de IMSQTI.

4.4.1 IMS QTI

La especificación de IMS sobre Question & Test Interoperability (QTI) describe un modelo para la representación de preguntas (AssessmentITem), tests (AssessmentTest) y sus correspondientes resultados.Esta especificación pretende el intercambio de preguntas, tests y resultados de los tests entre herramientas de autor, pools de contenidos educativos, herramientas de explotación de tests y herramientas de explotación de recursos educativos (LMS). La especificación se basa en un modelo de datos descrito en UML y con una traslación directa a XML para su intercambio[IMS06].

La especificación comenzó en sus primeras versiones en 1999, incorporándose a la especificación de la estructura abstracta de IMS[IMS03b]. En 2006 se publica la versión 2.1 de la especificación, en la que incluye ya el concepto de tests y de presentación de los resultados, con la posibilidad de relacionar preguntas por medio de secciones y reglas.

4.4.1.1 Elementos principales de IMS QTI

Los elementos principales de la especificación se basan en el estudio de la gran mayoría de sistemas de exámenes on-line, de forma que cubra de forma general la mayoría de los sistemas y por tanto cualquier sistema pueda exportarse. También intenta resumir los conceptos principales, actores y funcionamiento general de un sistema de evaluación. La especificación se basan en la definición de unos conceptos (que nosotros asimilaremos a clases) relacionados entre sí, así como la definición de un comportamiento determinado y una representación de esas clases y relaciones en XML[QTI06b].

Los conceptos principales son los de la pregunta en sí, con sus componentes, la agrupación de preguntas para componer una evaluación determinada y la agrupación de preguntas a nivel de repositorio de las mismas. También abstrae el concepto de respuesta de un alumno, así como el tratamiento de la misma para evaluar al alumno. Su estructura se puede describir de la siguiente forma (Fig. 4.1):

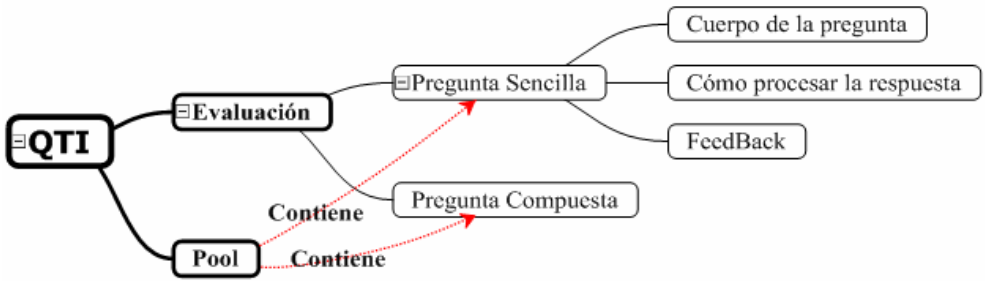


Fig. 4.1. Estructura IMS QTI

La base principal es la pregunta (clase *assessmentItem*), considerada como la unidad indivisible a la hora del intercambio de contenidos. Es el equivalente intuitivo de una pregunta de un test. Puede ser Sencilla o Compuesta (conjunto de preguntas sencillas que comparten alguna parte). Para su intercambio se agrupan en Pools o repositorios (clase *pool*) de preguntas.

Luego tenemos la Evaluación (clase *assessmentTest*), que representa a un conjunto de preguntas que se configuran para ser respondidos por un estudiante. Pueden relacionársele resultados globales y notas globales.

La pregunta contiene los siguientes datos:

- **Cuerpo de la pregunta** (Texto o imágenes), definido dentro del elemento *itemBody* que indica a su vez los recursos que necesita (imágenes, textos, html, etc). Para la presentación de la pregunta se permite la utilización de hojas de estilo y en el texto se permite cierto XHTML (eXtended HTML) para subrayados, negritas, elementos para interactuar, etc., así como MATHML. Dentro del cuerpo de la pregunta se incluye a su vez una descripción de las posibles interacciones que puede realizar el usuario sobre la pregunta. Más adelante se describirá con más detalle el concepto de interacción.

- **Información de cómo se ha de procesar la respuesta**, de forma que cuando se conteste a una pregunta, por medio de unas reglas definidas en ese punto, se obtenga el resultado de la evaluación de conocimientos (representado en la clase *outcome*) a través de un conjunto de variables de resultado. Normalmente dichas variables son numéricas y se suelen ponderar luego a la hora de evaluar un test en conjunto.

- La pregunta puede contener también un **posible feedback** para el alumno, representada en la clase *feedback*. La clase *feedback* incluye información también de en qué casos se ha de presentar al alumno. Se distingue entre el feedback modal y el incrustado. El modal se presenta al alumno cuando ya se ha procesado la respuesta a la pregunta y tenemos el resultado, de modo que la interacción se ha terminado, pues se basa en el resultado obtenido. Por otro lado tenemos el feedback incrustado (definido por la propiedad *inline*), que se asocia a una respuesta del usuario, antes de procesar dicha respuesta. Por ejemplo, un feedback modal podría decir qué repasar si no has acertado y uno inline te explicaría por qué la respuesta que has dado en particular es incorrecta.

La especificación permite también la posibilidad de que un test o un ítem sean adaptativos, es decir que varíen según los resultados de los usuarios. Por ejemplo, un test que cuando se contesta de forma errónea a una pregunta básica de un tema ya no sigue preguntando sobre ese tema o una pregunta compuesta que según lo que se haya contestado pida alguna aclaración.

4.4.1.2 Interacciones en IMS QTI

IMS define respuesta como el dato que nos indica el usuario por medio de su interacción con la pregunta. Esta interacción será la base para poder

describir cómo han de comportarse las preguntas tras una contestación.. Las respuestas pueden tener un conjunto de variables de respuesta (representadas en la clase *responseVariable*) que serán tratadas en el procesamiento de la respuesta.

Para hablar del comportamiento de las preguntas en tiempo de ejecución, es necesario definir el concepto de Sesión de usuario, como el contexto en el que tienen sentido las variables de las que hablamos y en el que el estudiante realiza sus interacciones. Cada vez que un usuario interactúa con las preguntas, se genera un conjunto de variables que son específicas de un periodo de interacción usuario-sistema. El conjunto de esas variables definen la Sesión de usuario, y nos constituyen el sistema de memoria de las relaciones desarrolladas durante ese periodo de tiempo.

El comportamiento principal es el siguiente (Fig. 4.2):

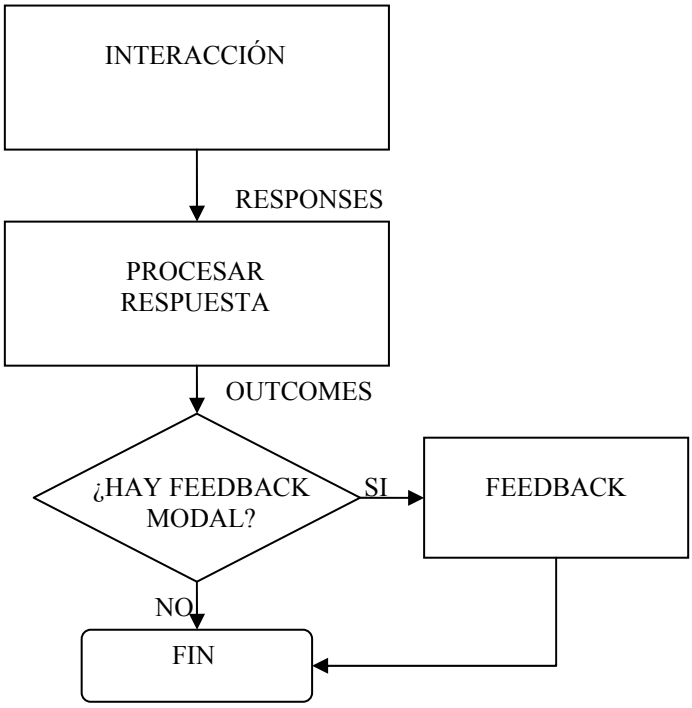


Fig. 4.2. Comportamiento en IMS QTI

1. Al alumno se le presenta la pregunta, e interactúa con ella dependiendo del tipo de pregunta. Como ejemplo de interacciones tenemos la elección de una de las respuestas si es una pregunta de opciones simple, o la elección de varias si es múltiple. Otra posible interacción sería la escritura de cierto texto, o el desplazamiento de un objeto gráfico a un punto determinado.

2. Las interacciones son convertidas por el sistema en respuestas. Por ejemplo, en una pregunta de opciones simple, se define una variable de respuesta como un valor único que se le asignará el identificador de la opción de respuesta seleccionada.

3. Al obtener las variables de respuesta arranca el procesado de las mismas. Para realizar dicho procesamiento se utilizan las reglas definidas(que vendrán descritas en las instancias de la clase *Rule*), que normalmente son condicionales (descrita por medio de instancias de las sublases *responseIf* , *responseElseIf* y *responseElse*, aunque existen otras subclases posibles) y expresiones de esas condicionales. Un ejemplo de regla que se puede describir es "Si la respuesta coincide con la respuesta correcta, coloca un 1 en la variable de resultado denominada NOTA, si no asignale 0". Puede que ocurra que existan feedbacks definidos, que se ofrecerán al alumno durante el procesamiento si son incrustados.

4. Tras procesar las respuestas, el sistema obtiene las variables de resultado, que pueden conducir a feedbacks modales si existen. Si no es así, se reporta el resultado obtenido al sistema y, resetea las variables de respuesta y se espera a siguientes interacciones.

Por tanto el comportamiento en tiempo de ejecución tras una interacción de un usuario en la realización de una evaluación consiste en convertir esa interacción en variables de la sesión, procesar dichas variables en función de sus reglas definidas y obtener variables de resultado, pudiendo presentar alguna realimentación en caso de estar definida.

En caso de preguntas condicionales, las variables de resultado no se borran entre una interacción y otra, ya que esta información puede afectar a preguntas futuras. Además, en las preguntas condicionales, la contestación a una pregunta puede provocar algún cambio en la presentación de preguntas de test, ampliando o reduciendo la cantidad de preguntas pendientes.

4.4.1.3 Empaquetado en IMS QTI

La especificación indica cómo se han de agrupar los conjuntos de preguntas o exámenes de forma que puedan ser exportados. Básicamente sigue la filosofía de IMS Content Packaging (CP) que se basa en un archivo comprimido que contiene:

- **Las preguntas en sí ,los tests y las estadísticas de uso** de un test de o un conjunto de preguntas . Los tests se representan mediante la clase *assessmentTest*, donde se ofrece la información necesaria para poder presentar y ejecutar un test completo. Los valores estadísticos se agrupan en la clase *bankProfile*, que contiene elementos de la clase *usageData* para cada grupo de estadísticas obtenidas en un contexto determinado . Este conjunto se agrupa como uno o varios archivos XML).

- **Los archivos de imágenes y extras** necesarios. Estos archivos estarán referenciados en los distintos cuerpos de las preguntas, dentro del XHTML de las mismas. Mediante estos archivos podremos introducir elementos multimedia dentro de las preguntas.

- **El manifiesto resumen del paquete**, a modo de archivo XML denominado *imsmanifest.xml*). Es aquí donde se almacena la metainformación sobre la posible estructura de las preguntas (en tests), así como información más general sobre las preguntas, los exámenes y sobre el paquete en sí.

En caso de agrupar no sólo preguntas sino una unidad de aprendizaje, se recomienda el uso del resto de especificaciones de IMS: IMS Learning Design (LD, para la definición de unidades de aprendizaje y para definir reglas de paso de objetivos), IMS Simple Secuencing SS (para indicar la estructura y navegabilidad del curso) y IMS CMI (para indicar la comunicación en tiempo de ejecución de variables entre el cliente y el servidor).

4.4.1.4 Metadatos en IMS QTI

IMS QTI permite añadir metainformación en el momento de empaquetar un conjunto de preguntas y/o exámenes, así como indicar el modo de comportamiento y los elementos que componen el conjunto empaquetado. Esto se hace por medio del manifiesto (*imsmanifest.xml*), como ya adelantamos en el apartado anterior.

Un hito importante en la definición de los metadatos fue la aparición de la versión 2.0 de IMS QTI. A partir de esta versión se llegó a un acuerdo para utilizar IEEE LOM 1484.12.1-2002 (en particular su representación en XML) como base para describir los recursos educativos en el manifiesto, ampliando alguno de los apartados y recomendando algunos valores determinado para algunos de los apartados de IEEE LOM

El archivo resumen donde se encuentra la metainformación en los paquetes de IMS QTI, que como vimos anteriormente denominaremos manifiesto, contiene en líneas generales (Fig 4.3):

- **Metainformación sobre el paquete en sí**, situado entre las marcas <*cp:metadata*>). Este apartado suele llevar la información sobre la versión del IMS CP (definido en <*cp:schema*> y <*cp:schemaversion*>), y una descripción del paquete en general como recurso educativo en conjunto, usando IEEE LOM (por medio de la marca <*imsmd:lom*>).

- **Descripción de la organización del paquete** , utilizando la marca <*cp:organization*>. En este punto se suele colocar la información de IMS LD [CAE05] si es necesaria. En la mayoría de los casos, cuando se trata de la agrupación de un conjunto de preguntas sin estructura educativa, la marca existe pero no contiene ningún elemento.

- **Descripción de los recursos unitarios** que hay dentro del paquete, que en este caso serán principalmente las preguntas. La descripción de los recursos unitarios se realiza dentro de la marca <*cp:resources*>. Además, por cada recurso habrá una etiqueta <*cp:resource*>.

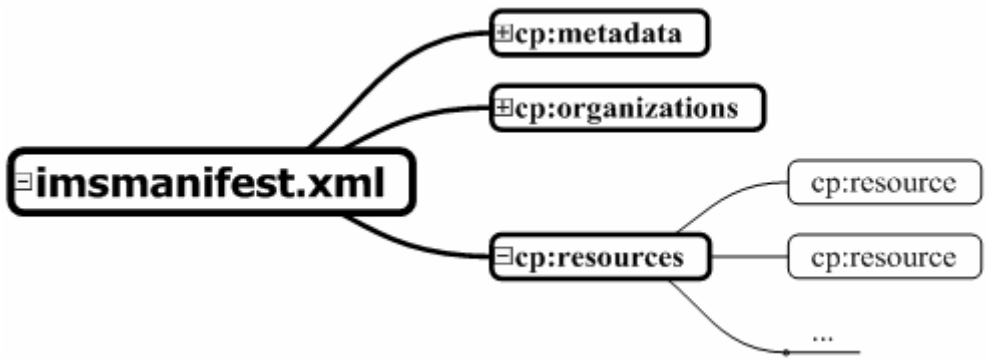


Fig. 4.3. Descripción general del manifiesto

En cada uno de las marcas *cp:resource* se describen los recursos que se incluyen en el paquete: las preguntas, los tests, los resultados estadísticos y los ficheros auxiliares que se comparten entre varias preguntas. A partir de este punto nos centraremos en el elemento que describe cada uno de los recursos, describiendo primero sus atributos y a continuación los elementos que contiene.

Los atributos de los recursos incluidos dentro de la marca <*cp:resource*> son los siguientes:

- **Atributo de identificación**, denominado *identifier*, que contiene el identificador único del recurso. Este identificador ha de ser único obligatoriamente dentro del contexto del manifiesto, de forma que en todo el manifiesto nos podamos referir al recurso con ese identificador.

- **Atributo de localización física**, denominado *href*, que indica la ubicación física del recurso (URL), siendo la ruta relativa dentro del paquete. Este apartado será utilizado a la hora de desplegar el paquete en algún sistema de aprendizaje vía web.

- **Atributo de descripción del tipo de recurso**, denominado *type*, que nos describe del tipo de recurso. Por ejemplo, las preguntas de IMS QTI versión 2.0 son del tipo *imsqti_item_xmlv2p0*. Este atributo sólo indica que el elemento es uno de los descritos en IMS QTI, sin llegar a describir si es una pregunta, un test u otro elemento.

Podemos ver en el ejemplo 4.1, que indica un elemento con identificador *ID-23435632456ac3*, siendo dicho elemento uno de los definidos en la especificación IMS QTI 2.0 y que está ubicada en el fichero *pregunta.xml*.

[Ej. 4.1]

```
<manifest
xmlns:cp="http://www.imsglobal.org/xsd/imscp_v1p1">
  <cp:metadata/>
  <cp:organizations/>
  <cp:resources>
    <cp:resource identifier="ID-23435632456ac3"
type="imsqti_item_xmlv2p0" href="pregunta.xml">
.....
    </cp:resource>
  </cp:resources>
```

```
</manifest>
```

Dentro del manifiesto, los elementos principales que contiene cada uno de los recursos son (Fig. 4.4):

- **Elemento de localización de ficheros**, denominado <*cp:file*>, elemento que indica la dirección de cada uno de los ficheros que necesita: el mismo recurso en sí. Por tanto habrá un elemento de este tipo para el fichero xml de la pregunta, y uno por cada uno de los ficheros auxiliares (imágenes, etc). Esto permitirá desagregar el paquete de contenidos para separar cada uno de los recursos.

- **Elemento de metadatos del recurso** , por medio de la marca <*cp:metadata*>, donde se describe el recurso unitario. Para realizar esta función utiliza de nuevo los elementos <*cp:schema*> y <*cp:schemaversion*>, siendo en este caso su valor la descripción de la versión de IMS QTI utilizada. También se incluyen el elemento de IEEE LOM <*imsmd:lom*> y una extensión de IEEE LOM dentro del recurso, denominada <*imsqti:qtiMetadata*>y que incluye la metainformación específica de las preguntas que no está contemplada en la versión de IEEE.

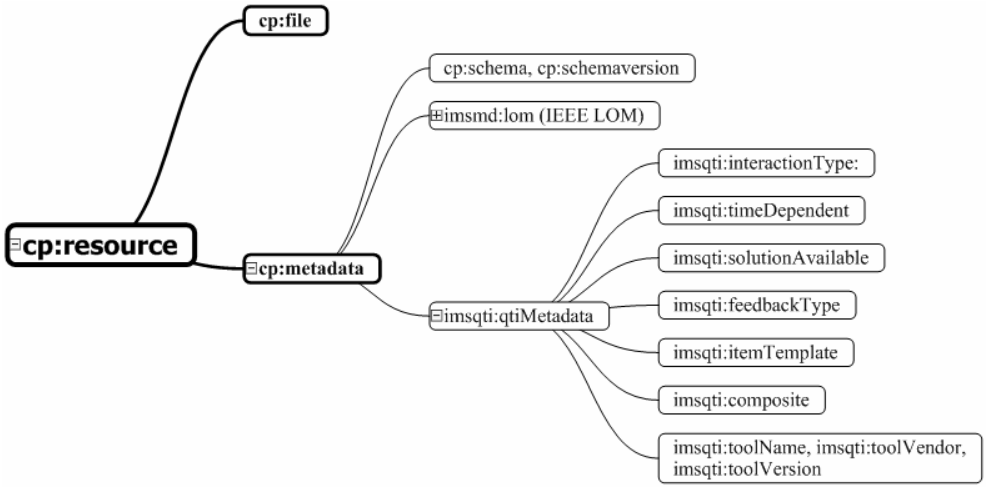


Fig. 4.4. Elementos principales de cp:resource

En el elemento de metadatos de las preguntas encontramos las peculiaridades de las mismas. Para poder describir de forma completa a las preguntas, en IMS QTI se ha optado por extender la ontología de IEEE LOM, tanto extendiendo elementos dentro de la especificación IEEE LOM

como colocando otro elemento fuera de IEEE LOM de descripción de ontología denominado *imsqti:qtiMetadata*, como podemos apreciar en la figura 4.4.

Dentro de la ontología definida por medio de IEEE LOM, la extensión más destacable es la inclusión de nuevos valores para el elemento *kind* de la categoría <*relation*>. Los nuevos elementos permiten indicar qué preguntas excluyen a otras, por medio de la marca <*precludes*> y qué preguntas necesitan de preguntas anteriores, por medio de la marca <requires>. También existe un elemento que permite indicar qué preguntas están agrupadas en secciones dentro de un test, mediante el uso de la marca <*issetmemberof*>.

La extensión externa a la ontología definida en IEEE LOM nos permitirá incluir información específica de los elementos de una evaluación. Como se ha indicado anteriormente se realiza por medio del elemento <*imsqti:qtiMetadata*>. Este elemento permite:

- **Definir el tipo de interacción**, con lo que indicaremos el tipo de pregunta que es. Cada tipo de interacción define unos parámetros diferentes en el archivo .xml específico de la pregunta, de modo que en el momento de realización de un test el sistema ha de presentar la pregunta de una forma determinada para que cumpla sus objetivos. El tipo de pregunta también condicionará el tipo de respuesta que se puede presentar, que podrá representarse en una o varias variables de tipo identificador, parejas de identificadores, numérico, textual, de localización de un fichero, etc) Los tipos posibles son:

  - choiceInteraction, cuando es una pegunta en la que se selecciona entre un grupo de posibles respuestas.

  - inlineChoiceInteraction, parecida a choiceInteraction pero las opciones las presenta incrustadas dentro de un texto.

  - gapMatchInteraction, donde hay un texto con huecos y un conjunto de palabras que se pueden utilizar.

  - associateInteraction, cuando hay que asociar por parejas elementos (normalmente textos).

- textEntryInteraction, usado para rellenar huecos de texto pequeños. Normalmente se usa para preguntas de tipo "rellena los huecos".

- extendedTextInteraction, donde el alumno rellena un texto a enviar, de cierta longitud (por ejemplo, si se pide escribir una redacción sobre algo).

- hottextInteraction, donde el alumno ha de elegir si marca o no un conjunto de frases o partes de frases definidas como marcables. Un ejemplo es cuando tenemos un texto y hay una partes subrayadas que hemos de decir si son correctas o no.

- matchInteraction, donde se relacionan los elementos por parejas. La diferencia con associateInteraction es que en este caso se define un conjunto de elementos origen y un conjunto de elementos destino, y se empareja siempre un origen con un destino. En el otro caso no se distingue entre elementos de un tipo o de otro.

- orderInteraction, donde el alumno ordena un conjunto de elementos.

- graphicAssociateInteraction, como cuando asociamos nombres pero asociamos regiones de una imagen con otras, dos a dos. (por ejemplo, para un ejercicio de unir los puntos).

- drawingInteraction, pregunta en la que se ha de dibujar algo. El sistema presenta una imagen inicial (puede ser un lienzo en blanco) y el sistema ha de presentar mecanismos para que el alumno pueda dibujar y enviar el fichero de resultado para ser corregido.

- graphicGapMatchInteraction, cuando el alumno ha de arrastrar imágenes pequeñas dentro de una imagen grande, y hay una zona sobre la que ha de soltar esa imagen para que se considere la respuesta como correcta.

- graphicOrderInteraction: donde se han de ordenar correctamente un grupo de regiones de una imagen.

- hotspotInteraction, donde el alumno ha de elegir qué región de una imagen pulsar dentro de un conjunto definido de regiones. Por ejemplo, un dibujo con un perro y un gato, con una región que delimita el perro y otra al gato y se pide que señale al perro.

- positionObjectInteraction, donde hay unas imágenes que hemos de colocar dentro de otra imagen de fondo en su posición correcta. Por ejemplo, situar la silueta de un país europeo en su posición correcta dentro de Europa.

- selectPointInteraction, donde el alumno marca un punto de una imagen, y dependiendo de la coordenada se da una puntuación. En este caso se definen áreas de la imagen con una puntuación, para que no haya que marcar el punto exacto al pixel.

- sliderInteraction, donde al alumno elige un valor por medio de desplazar un puntero a lo largo de una barra de desplazamiento. Suele utilizarse para poder dar valores aproximados.

- uploadInteraction, en el que el alumno carga un fichero al sistema como resultado.

- customInteraction, que se utiliza para extensiones de sistemas que tienen tipos de preguntas no representables con el conjunto anterior,

- endAttemptInteraction, es una interacción auxiliar que se usa cuando el alumno desea terminar de probar una pregunta (por ejemplo, si se rinde y no sabe la respuesta).

- **Definir si el recurso es dependiente del tiempo** , por ejemplo, si para una pregunta hay un límite temporal desde su presentación al alumno hasta la obtención de la respuesta. Esta definición se hace por medio de la marca <*TimeDependent*>.

- **Definir si la respuesta correcta está incluida en la definición de la pregunta**, por medio de la marca <*solutionAvailable*>. Las preguntas que no la tengan incluida exigirán un tratamiento manual de corrección.

- **Definir si tiene feedback y si la pregunta es adaptativa o no**, por medio de <*feedbacktype*>. Una pregunta adaptativa es la que cambia su representación en función de la interacción del usuario.

- **Definición de templates**. La utilidad de los templates surge cuando vamos a hacer un conjunto de preguntas en las que hay partes, como la gestión de la respuesta, que son exactamente iguales, por lo que se colocan en un fichero separado. Se indica mediante la marca <*itemTemplate*>si el recurso es un template o no.

- **Indicación de si el elemento es compuesto o simple**. Si el elemento es una composición de interacciones se indica por medio de la marca <*composite*>.

- **Información sobre la herramienta de autor** con la que se ha generado la pregunta, por medio de las marcas <*toolname*>,<*toolversion*> y <*toolvendor*>.

Toda esta información introducida en el elemento *imsqti:qtiMetadata* podría inferirse por medio de una introspección de los distintos ficheros XML de las preguntas, por lo que se podría considerar que la información está repetida en el manifiesto y en los elementos en sí.. Sin embargo, esto permite separar lo que se considera como metadatos en IMS QTI de los ficheros de las preguntan, permitiendo que el manifiesto sea susceptible de ser publicado para describir las preguntas, sin tener que publicar el detalle de las preguntas.

Otro punto donde se pueden definir metadatos es en los recursos que describen estadísticas de exámenes o preguntas. En ellos se describe el contexto de la estadística -número de muestras, fechas de realización,...- el tipo de estadística realizada -por medio de una catalogación de estadísticas no definido aún en la especificación- y el valor de la estadística, con valores de las desviaciones si se desea. IMS QTI recomienda indicar en los valores estadísticos la media del resultado obtenido por cada pregunta, aunque se permite en su lugar colocar el porcentaje de gente que ha contestado a una respuesta determinada.

4.4.2 TeML: Ontología basada en XML para evaluaciones

En este apartado describiremos una ontología de descripción de las evaluaciones en Teleformación desarrollada en el Departamento de Comunicaciones de la Universidad Politécnica de Valencia. Esta ontología y

su implementación del modelo ontológico se terminó en el año 2002, siendo su mayor novedad el uso de XML como lenguaje para almacenar las evaluaciones, así como su desarrollo totalmente por web, sin ningún tipo de aplicación local. Se puede ampliar la información en [MAN02], [MAN02B] y [PAL03].

4.4.2.1 Elementos principales de la ontología

La ontología está enfocada alrededor del concepto Pregunta, de modo que un autor introduce peguntas que luego utiliza para definir exámenes. Las preguntas tratan de ciertos Conceptos, y esos conceptos pertenecen a una Asignatura. Las asignaturas tratan de más de un concepto, como veremos más adelante.

Los estudiantes se consideran estudiantes de una Asignatura, por lo que podemos asimilar Asignatura por Curso como se entiende en Teleformación, y los profesores podían serlo de varias asignaturas. De esta forma definimos propiedades de objeto que relacionan a un estudiante con una o varias asignaturas, y propiedades de objeto que relacionan un profesor con una o varias asignaturas.

Los estudiantes obtienen, a partir de las evaluaciones, notas sobre cada examen, y un informe de los conceptos que hayan acertado. Podemos considerar que generamos una propiedad de tipo de datos que relaciona una contestación al examen con una nota, y un conjunto de relaciones de objeto que relaciona la contestación al examen con un conjunto de conceptos que el alumno ha demostrado dominar por medio de la contestación del examen.

La ontología no se desarrolló por medio de lenguajes ontológicos, ya que en 2001/2002 utilizar XML ya se puede considerar como un avance considerable, aunque sí que existe una ontología implícita como podemos apreciar en la figura 4.5, donde vemos parte de la misma.

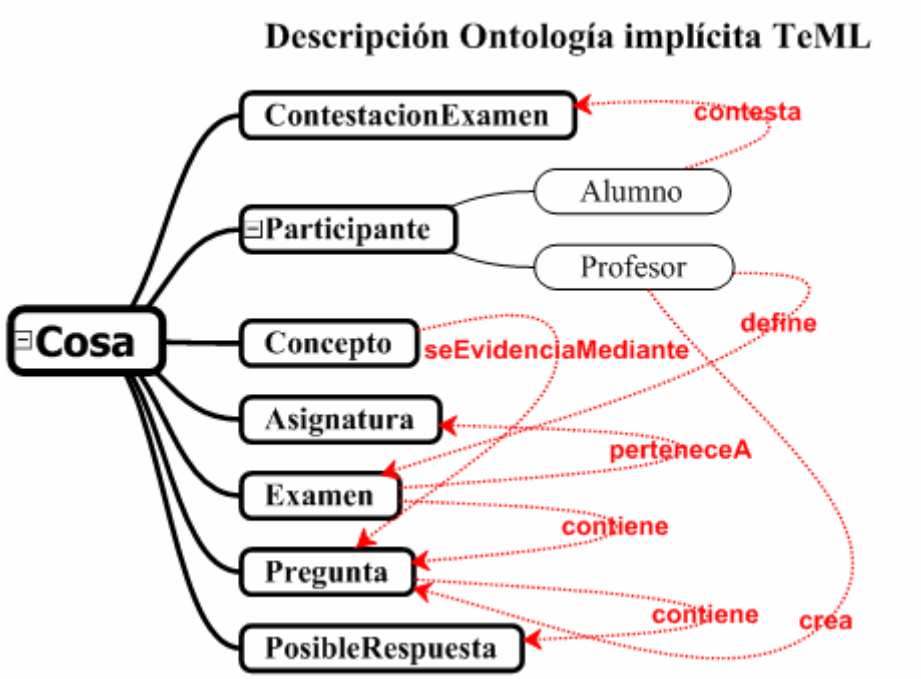


Fig. 4.5. Resumen de la ontología implícita

El funcionamiento del sistema consiste en lo siguiente:

- El Profesor crea preguntas en el sistema, indicando el enunciado y las posibles respuestas, marcando las respuestas correctas. Tanto el enunciado como las respuestas pueden tener un fichero multimedia asociado, bien sea una imagen, un sonido, un video, u otro formato visualizable con un navegador. Se definen tipos de preguntas según los ficheros multimedia que contengan, para facilitar la introducción y previsualización de los mismos.

- El Profesor define un Examen y lo relaciona a un conjunto de preguntas. En el proceso de generación de un Examen, se define la dificultad del mismo por medio de unos valores de dificultad máxima y mínima, que nos indicarán los límites de dificultad de las preguntas escogidas. Además se define la cantidad de preguntas y el número de respuestas mínimo que tiene que tener cada pregunta. De esta forma el sistema, a modo de agente, le presenta un pool de posibles preguntas que cumplen esos requisitos y el profesor escoge las que desea utilizar. También se define en el proceso si el examen se puede corregir de forma automática, de forma que sólo contiene

preguntas cuyo proceso de corrección no precisa de una intervención humana.

- Cuando el profesor termina de definir el examen, se almacenará en el sistema junto con una descripción XML del mismo y una versión de presentación por defecto del mismo en JSP. De esta forma podemos considerar que tenemos una metainformación del examen en el archivo XML y una localización del recurso listo para ser presentado al alumno, a modo de archivo JSP.

- El Alumno contesta al examen, y si el examen está definido como examen que permite autocorrección directa, presenta la nota. Existen casos en los que, aunque el sistema pueda corregir automáticamente, se considera más adecuada una revisión por el profesor, de forma que pueda modificar la nota según algún criterio determinado, como puede ser la evolución e interés del alumno a lo largo del proceso de aprendizaje. Por esta razón se debe indicar explícitamente si el sistema debe presentar directamente al alumno al nota. La contestación del alumno se almacena junto con una descripción XML de dicha contestación al examen. De esta forma el objeto de contestación al examen se almacena a modo de metadatos en el fichero XML.

- El profesor, tras la realización de un examen puede ver las contestaciones de los alumnos, modificarles las notas, modificar la dificultad de las preguntas o lanzar un proceso de recálculo de la dificultad de la pregunta. Trataremos este tema más adelante.

4.4.2.2 Clasificación pedagógica

La clasificación pedagógica utilizada en el sistema está orientada a su uso dentro de la Universidad, por lo que habla de Asignaturas, Preguntas y Conceptos.

Una instancia de la clase Asignatura puede tener n Conceptos, y cada Concepto puede pertenecer a m Preguntas. De esta forma hay una propiedad de clase que relaciona una instancia de Asignatura con una instancia de Concepto, y dicha propiedad es inversa funcional. Esta propiedad indica que una asignatura abarca una serie de conceptos únicos para la misma. Las Asignaturas nos sirven además para marcar a los alumnos y profesores , y asignarles permisos.

Una instancia de la clase Pregunta está asignada obligatoriamente a una única instancia de la clase Asignatura y contempla n Conceptos, representados como relaciones entre la instancia de la clase Pregunta e instancias de la clase Concepto. Esta propiedad indica que si se contesta correctamente a la pregunta se evidencia que se dominan los conceptos con los que se relaciona.

Como posible mejora al sistema se podría realizar una modificación que permitiera compartir preguntas entre asignaturas, ya que en realidad hay asignaturas que comparten conceptos, suposición que no se permite en la ontología TeML. Para resolver esta restricción deberíamos permitir relacionar a las instancias de Pregunta y a las instancia de Concepto con más de una instancia de Asignatura . De esta forma se podría compartir preguntas de conceptos comunes entre asignaturas. El sistema propuesto se puede ver en la Fig. 4.6.

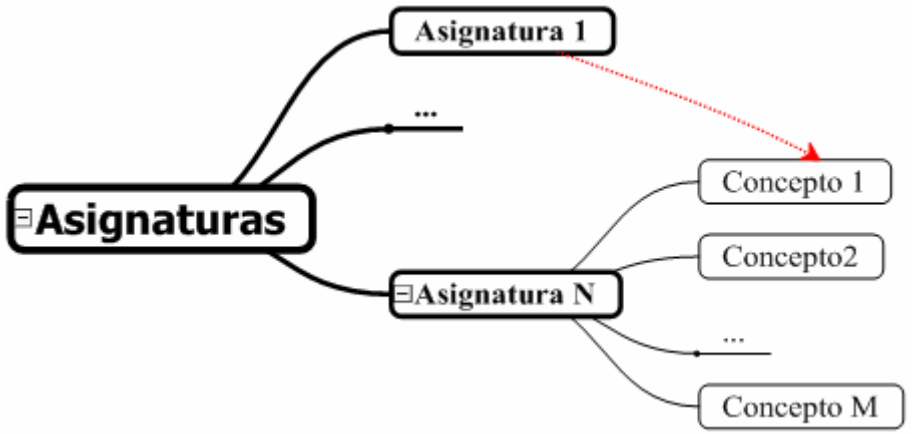


Fig. 4.6. Propuesta de modificación de la clasificación pedagógica

4.4.2.3 Realimentación del sistema: Cálculo de la dificultad

Otro punto interesante de la propuesta TeML desarrollada es la propuesta de utilización de las distintas ejecuciones de los exámenes para modificar la información que tenemos de las preguntas. En particular TeML propone que se utilice para poder recalcular la dificultad de las preguntas.

Como comentamos en el capítulo de introducción, este tipo de realimentación de los sistemas permite modificaciones de la información existente por medio de las interacciones de los usuarios. Además, es esta posibilidad de adaptación del sistema en función de la información nueva que recoge de los usuarios la que diferencia la formación on-line de los

simples CDs multimedia que contienen contenidos de cursos, por lo que estudiaremos con más detalle dicha adaptación a continuación.

En la propuesta realizada, el cálculo de la dificultad se arranca de forma manual, es decir, el profesor ha de indicar explícitamente al sistema que recalcule la dificultad de las preguntas. Tras la orden de recálculo de dificultades de las preguntas, el sistema aplica un algoritmo que se basaba en lo siguiente:

- El profesor especifica un valor de dificultad inicial a cada pregunta, que introduce antes de que la pregunta pueda ser utilizada en algún examen. Dicha dificultad se especifica por medio de un valor entre 0, que indica dificultad mínima, al valor 10, que indica la máxima dificultad.

- El profesor decide la dificultad del examen necesaria, que implica la dificultad de las preguntas utilizadas, con conocimiento del público al que iba dirigido, es decir, suponiendo que se plantea una dificultad que permita que, si la destreza de los alumnos a la hora de realizar un examen se distribuye como una distribución normal, no aprueben los alumnos que se consideran del extremo inferior de la normal, los que sacan peores notas. En [Kel39] se propone que el porcentaje que se ha de considerar para coger el extremo pero coger suficientes muestras para que no sean resultados anecdóticos es el 27%, permitiendo el margen 25-33%. Este sistema adopta el 33% (1/3).

- Tras realizar el examen se revisa la cantidad de gente que ha aprobado esa pregunta. En caso que sea mayor que 2/3 significa que incluso el extremo inferior aprueba, por lo que se debe rebajar la dificultad. El recálculo se hace restando uno a la dificultad. Para aumentar la dificultad se supone que han de aprobar no sólo el extremo superior, sino al menos la mitad de la franja media. Por eso si los aprobados no llegan al 50%, se aumenta la dificultad en una unidad.

Como podemos contemplar el algoritmo es sencillo, si bien hoy en día se añaden, a parte del nivel de dificultad, conceptos como si la pregunta está definida correctamente y el nivel de discriminación, que indica si llega a diferenciar realmente entre la gente que saca las mejores calificaciones y la que saca las peores. Si quisiéramos añadir estos parámetros y modificar la dificultad podríamos hacer como específica [CHA04]:

- Suponer que cada vez que se hacen los exámenes el **perfil de los alumnos es similar** y está distribuido de igual forma. Si no es así deberíamos mantener las estadísticas diferenciadas por perfil.

- De nuevo **realizar un análisis EGA** (Extreme Groups Approach), separando los alumnos que han obtenido la mejor calificación en el examen y los que han obtenido la peor calificación. Se considera en este caso el 25% de alumnos con las mejores notas y el 25% con las peores. P_H es la media en una pregunta de los de mejor calificación y P_L la media de los de peor. Para calcular la media se supone que la pregunta se puntúa entre 0 y 1 (por ejemplo, una de elección simple es 1 si se acierta y 0 si se falla).

- **Calcular la dificultad** como se indica en la expresión 4.1:

$$P_{NCAL} = \frac{(P_H + P_L)}{2} \qquad \text{[Expr. 4.1]}$$

  Esta fórmula considera que la media de la nota sacada en la pregunta es la dificultad, de modo que cuanto más nos acerquemos al 0 será una pregunta más difícil y cuanto más nos acerquemos al 1 una pregunta más fácil. Este algoritmo deja de ser un algoritmo adaptativo como el algoritmo inicial propuesto en TeML, donde la dificultad sube o baja un punto. Como la dificultad se calcula de nuevo sin basarse en los valores de dificultad anteriores, pueden darse grandes variaciones de valores de dificultad si durante la realización de algún examen se produce alguna anomalía, como puede ser un cambio puntual de perfil de los alumnos.

  Por esta razón será más recomendable hacer como se indica en la ecuación 4.2, donde la nueva dificultad se recalcule a partir de la antigua, por medio de una variable de memoria $\alpha \in [0\text{-}1[$:

$$P_N = \frac{P_{N-1} \cdot (\alpha) + P_{NCAL} \cdot (1-\alpha)}{2} \qquad \text{[Expr. 4.2]}$$

- **Calcular la discriminación** como se indica en la ecuación 4.3:

$$D = P_H - P_L \qquad \text{[Expr. 4.3]}$$

De forma que una buena pregunta ha de saber distinguir entre la gente que ha superado un objetivo de aprendizaje y una que no. [CHA04] especifica que una D<0.3 significa que esa pregunta debe ser revisada, ya que no discrimina los que más saben de los que menos, y por tanto no sirve a priori para evaluar. En este punto debemos añadir que esto sólo es cierto cuando la dificultad de la pregunta no es muy alta o muy baja, como se indica en [SIM06]. Podemos considerar que el margen para utilizar la discriminación es para dificultades entre 0.3 y 0.7. Denominaremos este margen como el margen de dificultad aceptable.

- **Evaluar en las preguntas con opciones las respuestas posibles**. De esta forma revisaremos los enunciados de las posibles respuestas, tanto si son las correctas como si no. De esta forma evitaremos plantear opciones que son obviamente incorrectas aunque no se domine la materia y opciones que aunque se domine la materia no se entienden como correctas. Consideraremos que una pregunta de opciones tiene una de sus posibles respuestas incorrecta para revisar si su dificultad está en el margen aceptable y además alguna de las respuestas no la ha contestado ningún alumno de los de peor calificación (los que sirven para el cálculo de P_L). Por otro lado, una pregunta tiene una respuesta correcta a revisar si su dificultad está en el margen aceptable y además la han contestado correctamente más alumnos de los de peor calificación que los de mejor calificación.

Los métodos de cálculo de dificultad y discriminación se pueden aplicar también a nivel de examen, utilizando en este caso medias de los valores de dificultad de las preguntas. Sin embargo en nuestra especificación no contemplaremos esta posibilidad y nos centraremos en el cálculo de dificultad de la pregunta para permitir introducir esta información en caso de pretender la interoperabilidad a nivel de pregunta.

Por tanto, vemos que TeML permite una adaptación de la dificultad de las preguntas en función de las contestaciones de los alumnos, y proponemos en esta Tesis una modificación del algoritmo de recálculo de la dificultad de TeML, basándonos en el modelo EGA con un factor de memoria α. Esta modificación del algoritmo de cálculo de dificultad nos permite localizar además preguntas que han de ser revisadas, bien porque no permiten

discriminar correctamente o bien porque alguna de sus posibles respuestas ha de revisarse.

4.4.2.4 Implementación del modelo

Para implementar el modelo en el año 2001, se decidió implementar la ontología de forma fija por medio de una base de datos relacional, de modo que los datos de la misma se almacenaban en la base de datos, almacenando algunos elementos complejos, como la definición de un examen o la contestación de un alumno, en formato XML.

La implementación se basa en que todos los elementos que actúan en el sistema conocen a priori la ontología implícita, no estando preparados, como ocurre en los agentes inteligentes, para adaptarse en caso de ampliación de conceptos en la ontología.

En la Fig. 4.7 podemos ver la descripción de la Base de Datos relacional.

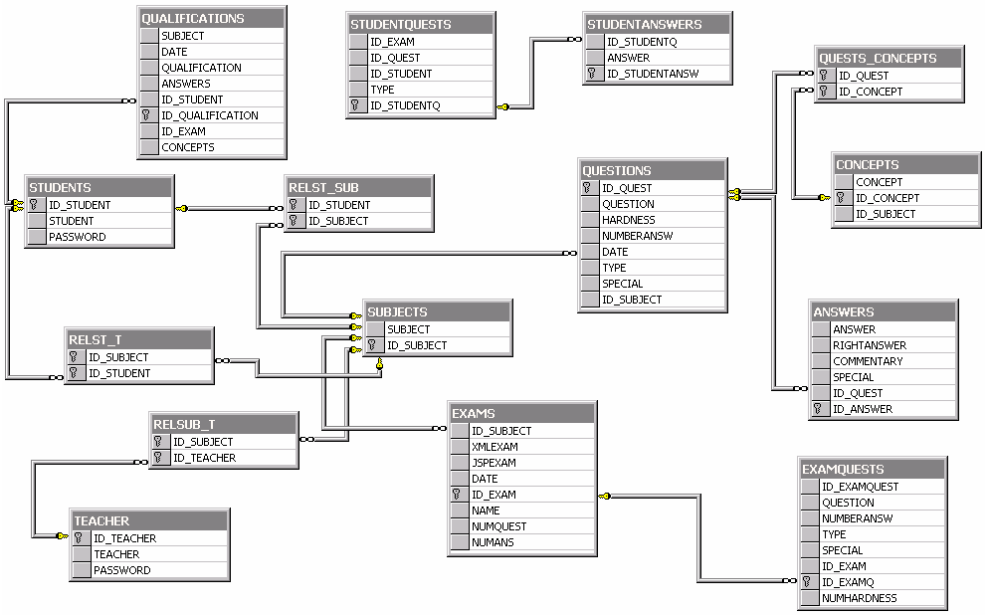


Fig. 4.7. Descripción de la Base de Datos utilizada

Como podemos observar, es la propia base de datos, en su descripción entidad-relación, la que indica qué conceptos se pueden relacionar con qué

otros y si la cardinalidad de dichas relaciones, que pueden ser [1:N] o [M:N].

Por ejemplo, una pregunta se representa en la base de datos como un elemento de la tabla *QUESTIONS*, y se puede relacionar con múltiples respuestas, ya que tiene una cardinalidad [1:N] con la tabla *ANSWERS*. Sin embargo, tiene una cardinalidad [N:M] con los conceptos, almacenados en la tabla *CONCEPTS*.

También podemos apreciar que tanto un alumno, representado como un elemento en la tabla *STUDENTS* como un profesor representado en *TEACHER* están relacionados por sendas relaciones de cardinalidad [N:M] con las asignaturas, representadas como objetos de la tabla *SUBJECTS*.

La implementación del modelo se realiza de modo que incluye todos los subsistemas para que funcione de forma autónoma, de modo que el sistema de exámenes incluye:

- **Herramienta de autor** para el profesor, que permite la introducción de preguntas. Aunque la ontología permite más de una respuesta correcta, la herramienta de autor sólo permite preguntas con una respuesta correcta. Esta herramienta permite, a su ver, la definición de exámenes y la consulta de resultados de los mismos.

- **Herramienta para el alumno**, en la que puede contestar a exámenes que se hayan definido para sus asignaturas y consultar la nota en caso de corregirse automáticamente en el momento.

- **Sistemas auxiliares** de autentificación, seguridad, corrección automática, etc.

Su implementación fue novedosa también por las tecnologías utilizadas, realizando un sistemas independientes de las plataformas y basado en software libre. La única excepción a este punto ge la Base de Datos, que se eligió SQL Server(r), aunque el ataque a la base de datos se hace por medio de SQL estándar y por medio de Java Database Connection (JDBC).

Su implementación se basa en una separación en tres capas, que son la encargada del almacenamiento -realizada con SQL Server-, la capa de lógica de negocio -implementada por medio de Java y JavaBeans- y la capa de presentación -implementada en Java Server Pages (JSP)-. En la capa de

lógica de negocio se añaden las restricciones de la ontología que se refrendan en la base de datos, como las restricciones de relación de uno a muchos y de muchos a muchos, añadiéndoles denominaciones a dichas relaciones entre objetos. Por ejemplo:

```
public class Examen{                                    [Ej. 4.2]
...
  public Pregunta[] getPreguntas() {
...
  }
...
}
```

En este ejemplo vemos que no sólo se define la relación de uno a muchos, sino que se definen el rango y de la propiedad, siendo una propiedad que relaciona a un único examen con varias preguntas.

En cualquier caso, al no utilizar lenguajes ontológicos para la web, la ontología no aparece explicitada en el sistema sino diluida en el código java, no permitiendo su uso por parte de agentes inteligentes generales que puedan entender la ontología y realizar consultas sobre la base de conocimiento, como sí se realizará en la nueva ontología de teleeducación propuesta en los capítulos siguientes.

4.4.2.5 Metadatos de la Aplicación: TeML

Como hemos comentado en los apartados anteriores, en la ontología descrita hay un sistema de marcado de los exámenes y las contestaciones de los exámenes por medio de XML. A esta descripción en XML se le denominó también TeML, que se basa en la definición de los metadatos y su expresión en XML (en este caso por su definición del DTD).

La metainformación de TeML se puede dividir según dos criterios:

1. **Según al objeto al que se refiera**. En este caso tenemos la definición de un examen (lo que nos permitirá obtener información de cómo presentar el examen a los alumnos o si ya se ha realizado, en qué condiciones se realizó), definición de los resultados de un examen y definición de una pregunta en particular.

2. **Según a qué característica se refiera**. En este caso tenemos:

- Descripción de la estructura física y su ciclo de vida, así como datos generales. Equivaldría en parte a los apartados <General>, <Technical> y <Educational> de IEEE LOM.

- Descripción de la temática que trata. En este caso contempla en parte las categorías <Classification> de IEEE LOM.

- Descripción de la dificultad. Este punto se trata aparte de la descripción de la estructura física, ya que la dificultad en sí es un resultado que como hemos visto varía con las interacciones de los alumnos.

A continuación nos basaremos en el primer criterio, y dentro de cada objeto utilizaremos el segundo criterio para agrupar las características.

4.4.2.6 Metadatos de los exámenes

En el caso de la definición de los exámenes, en su descripción de la estructura tenemos:

- **Definición de las preguntas** de los exámenes, que veremos más adelante.

- **Título del examen**. Una cadena de caracteres que nos servirá de cabecera en la presentación del mismo.

- **Autor del Examen**. Permite además añadir un pequeño comentario de tipo texto por parte del autor, para aclarar un poco la pregunta. Este apartado de tipo texto es demasiado abierto para situarlo en un apartado determinado.

- **Indicación del Parcial** (examen parcial) al que se refiere. No se considera como descripción temática ya que cada parcial puede contener unos contenidos distintos a lo largo de los años.

- **Indicación de creación aleatoria del examen**. Cuando se marca esta opción indica que el examen es aleatorio, y por tanto no incluye las preguntas determinadas del examen. Para cada persona a la que se le indica el examen presenta un examen cogiendo las preguntas aleatorias del pool general de preguntas, fijada la dificultad y la cantidad de preguntas.

- **Indicación de los comentarios activados**. Si se marca esta opción, por cada pregunta, si hay algún comentario de feedback, se presentará al alumno cuando realice el examen. Esta opción debe servir sólo para exámenes de autoevaluación y tiene un cierto parecido con el apartado de feedbackType de IMS QTI.

- **Cantidad de preguntas** del examen.

- **Tipo de preguntas**, en caso que el examen esté compuesto por un único tipo de preguntas.

Respecto a la temática sólo habla de la asignatura a la que pertenece y respecto a la dificultad habla de la dificultad definida del examen , que es la dificultad de las preguntas a buscar si es de tipo aleatorio o la dificultad seleccionada por el profesor cuando buscó dentro del pool de preguntas. Como posible mejora proponemos realizar una media de la dificultad de cada pregunta y colocarlo en este apartado, ya que el profesor, cuando busca preguntas por una dificultad, le permite elegir también un margen de variación de la dificultad (como por ejemplo: dificultad 4 ±2).

Hay otros campos que sí contempla el sistema general, y se almacenan en la base de datos, pero que no se consideraron interesantes como metadatos, como son la fecha de definición del examen, que nos podría ser de gran ayuda para evaluar contextualmente el examen. Por ejemplo, podríamos distinguir el examen que se definió en el curso académico 2001/02 para el primer cuatrimestre.

4.4.2.7 Metadatos de las contestaciones a los exámenes

Para definir la contestación de los exámenes, para cada respuesta se almacena en XML la contestación al examen, pregunta por pregunta. No se almacena ni variables de tiempo que ha tardado el alumno ni posibles feedbacks de tipo texto que pudiera indicar el alumno (como “no entiendo la pregunta”) y que podrían ser de utilidad para la depuración de las preguntas. Su único valor es el almacenamiento del resultado, y podría mejorase por medio de una firma de ese resultado que permitiese evitar que modificaciones posteriores de la base de datos corrompieran la información.

4.4.2.8 Metadatos de las preguntas

En el caso de la definición de las preguntas, en su descripción estructural tenemos:

- **Tipo de pregunta**, que permite definir los siguientes tipos de enunciados:

    a) TipoTexto, donde el enunciado de la preguntas es un simple texto.

    b) TipoImagen, donde el enunciado incluye una imagen. Incluye datos de presentación como la posición a enseñar la imagen, y de localización con la URI de la imagen.

    c) TipoVideo, similar a TipoImagen pero con un Video.

    d) Tiposonido, similar a los anteriores pero sin datos de posición. En este caso por defecto aparece un icono que al pulsarlo reproduce el sonido.

    e) TipoFlash, en el que se permite incluir un archivo de flash en el enunciado.

    f) TipoApplet, en el que el enunciado incluye un applet java que realice una presentación completa del enunciado.

- **Tipo de posibles respuestas**. Este apartado es similar a los tipos de preguntas definidos en IMS QTI. Permite definir (aunque sólo se implementa completamente en el sistema las preguntas de opción múltiple y simple):

    a) Respsimple, para los tipos de pregunta de respuesta simple.

    b) Respmultiple, para cuando hay más de una respuesta correcta.

    c) RespPalabra, cuando la solución es una palabra.

    d) Resptexto, cuando el alumno ha de escribir un texto.

    e) RespBarra, cuando el alumno ha de dar un valor desplazando una barra.

    f) RespOrdenar, cuando se ordenan elementos presentados.

g) RespAsociar, cuando se asocian parejas.

En la Tabla 4.1 podemos ver la relación de tipos entre las interacciones de IMS QTI y los tipos de respuestas de TeML.

TeML	IMS QTI
Respsimple	choiceInteraction, (con un ResponseProcessing de respuesta única)
Respmultiple	choiceInteraction (ResponseProcessing de respuesta múltiple)
RespPalabra	textEntryInteraction
Resptexto	extendedTextEntryInteraction
RespBarra	sliderInteraction
RespOrdenar	orderInteraction
RespAsociar	associateInteraction, matchInteraction

Tabla 4.1. Relación de Tipos de preguntas

- **Indicación de si las respuestas tienen feedback** (comentario) o no. Sólo se presentan si el examen lo permite.

Por tanto podemos ver claras similitudes con IMS QTI a nivel de la preguntas, si bien TeML añade a la metainformación datos relativos al tipo de enunciado de la pregunta, por medio del tipo de pregunta. Sin embargo consideramos que esta información se encuentra también en IMS QTI por medio del listado de los ficheros que utiliza, mediante la marca <*cp:file*> y por medio de IEEE LOM, donde se define la herramienta y versión necesaria para poder visualizar correctamente el recurso.

4.5 Conclusiones

En este capítulo hemos visto la importancia de las evaluaciones como interacciones del estudiante en la teleformación. Además de enumerar las ventajas y limitaciones de las evaluaciones, se han descrito dos especificaciones de evaluaciones y su ontología relacionada. Una de las descritas es la especificación de referencia para la interoperabilidad actualmente y la otra es una especificación realizada dentro del Departamento de Comunicaciones de la Universidad Politécnica de Valencia.

Como hemos podido ver, la ontología TeML guarda gran similitud con la ontología utilizada en IMS QTI, si bien IMS QTI es una especificación más completa en variedad de preguntas y respuestas y descripción de reglas de procesamiento de las preguntas. También la definición actual es posterior a la definición de TeML.

Sin embargo, TeML define dificultad de las preguntas y los exámenes, así como un algoritmo adaptativo para el cálculo de la misma. Dicho algoritmo se modifica en esta Tesis, así como algunos puntos de la ontología de forma que se mejora la interoperabilidad en TeML. Es interesante también su clasificación pedagógica en conceptos, de forma que permite que las cuestiones sean herramientas que evidencien el dominio o no de uno o varios conceptos.

Sin embargo, aunque ambas tienen una ontología en la especificación, ninguna de las dos dispone de una representación en un lenguaje ontológico como pueda ser OWL, por lo que hemos de concluir que incluso IMS QTI adolece de una falta de definición formal ontológica, ya que como en el caso de TeML su ontología se define en XML, no utilizando ni tan siquiera lenguajes que permiten relaciones como RDF.

Es por esto que se realizará en esta Tesis una propuesta ontológica que intentará abarcar las dos ontologías estudiadas en este capítulo, definiéndola formalmente y realizando una representación en OWL, que permitirá que agentes de la web semántica obtengan esa información y puedan realizar razonamientos que faciliten la interoperabilidad de los recursos educativos, en este caso los tests o evaluaciones.

Capítulo 5: Arquitectura

5.1 Introducción

A lo largo de este capítulo elaboraremos una arquitectura que permita el funcionamiento de la ontología para teleeducación que propondremos en el siguiente capítulo. De este modo la arquitectura permitirá explotar la información semántica propuesta en la Tesis, y por tanto se generará un sistema completo de búsqueda y obtención de objetos educativos. Entendemos como arquitectura [IEEE00] a la organización básica de un sistema, compuesto por sus componentes, las relaciones entre ellos y las relaciones entre los componentes y el entorno, además de los principios que rigen su diseño y evolución.

El objetivo de la ontología será permitir la interoperabilidad de los recursos educativos, por lo que la arquitectura será una arquitectura de repositorio de contenidos educativos. Dicha arquitectura ha de permitir además el uso y búsqueda por medio de agentes inteligentes, que utilicen la ontología

propuesta en OWL para la búsqueda y obtención de recursos educativos. Como hasta donde llega nuestro conocimiento no existe ninguna arquitectura que cumpla estas especificaciones, definiremos una arquitectura nueva, que nos permita alcanzar nuestros objetivos.

Para generar la nueva arquitectura, veremos primero las tendencias actuales respecto a las arquitecturas de repositorios de contenidos, lo que nos permitirá seguir la línea que mejor se adapte a nuestras necesidades, que estudiaremos con más profundidad. También ampliaremos algunos de los conceptos de las mismas, como las DHT o Tablas Dinámicas Distribuidas, generando lo que denominaremos DHT Semántica.

La DHT Semántica es una modificación de las DHT de modo que permite búsquedas de términos semánticos y además optimiza la localización por medio de la separación de elementos de definición de las clases de la ontologías (TBox) y los elementos de definición de las instancias de las clases (ABox). De este modo adaptamos las DHT para su uso por OWL DL, lenguaje que como vimos se basa en las Lógicas Descriptivas que realizan una separación ABox/TBox. Por tanto la DHT Semántica será un elemento clave y novedoso de nuestra arquitectura, y por esta razón le dedicaremos un apartado de este capítulo.

5.2 Arquitecturas de repositorios de contenidos educativos

Definiremos un repositorio de contenidos [FED05] como un sistema que permite a los gestores de contenidos almacenar, localizar, gestionar y ofrecer recursos de algún tipo. Microsoft sigue la misma línea, [WWW77] definiendo un repositorio como una tecnología que permite compartir información sobre artefactos de ingeniería, entendiendo como artefactos de ingeniería a cualquier información con una estructura compleja y que se desee compartir. La comunidad de desarrollo en Java define un repositorio de contenidos [JSR140] de una forma más restrictiva, limitándolo a un conjunto de espacios de trabajo, en los que cada uno contiene un árbol de elementos, y en que cada elemento puede ser un nodo (elemento no terminal del árbol, que es lo que entendemos en esta Tesis como recurso) o una propiedad (elemento terminal del árbol). Podemos considerar que la especificación en Java de un repositorio de contenidos [JSR140] también es similar, ya que incluye una API que permite el almacenamiento, localización, gestión y descarga de recursos. Sin embargo la definición que seguiremos en la Tesis [FED05] es más generalista al no restringir los modelos de información de los recursos a estructuras de tipo árbol.

Si además, el repositorio de contenidos almacena recursos educativos, lo denominaremos repositorio de contenidos educativos. Veremos a lo largo del capítulo que aunque hablaremos todo el tiempo de recursos educativos u objetos de aprendizaje, el desarrollar una arquitectura que trabaje con la información semántica definida en OWL hará que pueda trabajar con cualquier definición de recursos educativos y además permitirá su uso en repositorios con otros propósitos, siempre que se publique definición ontológica correspondiente.

Existen actualmente una gran cantidad de soluciones propuestas para los distintos repositorios de contenidos, pero podemos agrupar las arquitecturas en tres líneas o tendencias principales:

- **Grandes repositorios de contenidos** o solución monolítica, donde recae al menos la tarea de localización del recurso en un sólo elemento. Como ejemplo de repositorio de contenidos educativos tenemos el caso de Merlot [WWW37]. Merlot (Multimedia Educational Resource for Learning and Online Teaching) es una iniciativa de la Universidad el Estado de California que recoge e indexa metainformación sobre recursos educativos. No contempla más que la localización del mismo y no su distribución, que delega en los servidores en los que se generó el recurso. Uno de los puntos más interesantes de Merlot, sobre todo tratándose de recursos educativos, es su sistema de revisión de calidad de los contenidos por revisores voluntarios y la recolección de opiniones de personas que lo han utilizado, a modo de acreditación de la calidad de los mismos.

  Para implementar el gran repositorio de contenidos podemos basarnos en arquitecturas de un único servidor (cliente servidor) como se propone en [DEH06][KIM05],o en arquitecturas Cluster. Los Clusters normalmente son uniones de elementos homogéneos (igual CPU, forma de comunicarse, sistema operativo) pertenecientes a un sólo dueño. El número de elementos del cluster suele ser fijo y muy limitado, ya que hay una gran centralización de los sistemas de gestión de recursos. Los elementos están mayoritariamente dedicados a servir al Cluster.

- **Sistemas federados de grandes repositorios de contenidos**. Es el caso del repositorio de contenidos educativos Merlot Federated Search [WWW34], así como de otras arquitecturas en las que se

definen y explotan ontologías educativas[SAN05]. El sistema propuesto de búsqueda es parecido al utilizado por buscadores de referencias bibliográficas (como el polibuscador de la Universidad Politécnica de Valencia, basado en SFX/Metalib [WWW35]), que por medio de servicios web convierte la petición de un cliente en peticiones para los distintos repositorios federados y luego junta los resultados para presentarlos de forma uniforme al cliente. En este caso se ataca a los repositorios ofrecidos por Merlot, ARIDANE [WWW21] y EdNA (Education Network Australia [WWW36].

Otra propuesta de sistema federado para un repositorio de recursos educativos lo encontramos en la arquitectura propuesta por ADL para la interoperabilidad de SCORM: CORDRA (Content Object Repository Discovery and Registration/Resolution Architecture), que es, como SCORM, un modelo basado en otros estándares para el diseño de federaciones de repositorios [WWW40].

Una arquitectura que podríamos incluir también en el apartado de sistemas federados es la arquitectura Grid. Mientras los Cluster se basan en elementos iguales, los Grids se basan en una unión de elementos heterogéneos -Normalmente computadores de gran capacidad con CPUs distintas, distintos sistemas operativos, ubicados en distintos lugares- de forma que utilizan toda o parte de sus recursos para obtener un único objetivo común, como puede ser realizar cálculos astrofísicos. Se basan en protocolos y estándares (como los planteados por el Global Grid Forum GGF) aceptados por todos los elementos, que han llegado a un cierto acuerdo para obtener el objetivo común. La tecnología Grid tiene como objetivo aprovechar las capacidades infrautilizadas de los sistemas y grandes sistemas de las empresas y organizaciones, y se denomina Grid Semántica cuando se convierte en su conjunto en un servicio web y que además internamente cada elemento del Grid se relaciona con los otros por medio de servicios web[ROU05][TUR05].

- **Sistemas Peer to Peer (P2P) de compartición de contenidos**. En este apartado se incluyen los sistemas basados en la tecnología JXTA como Splash [WWW39] y Edutella [WWW38].

  Edutella sea quizá la arquitectura P2P más conocida para educación [NEI02]. Implementa un sistema distribuido de búsqueda sobre nodos heterogéneos utilizando RDF, permitiendo 5 niveles de

complejidad de su lenguaje de búsqueda en RDF (RDF-QEL) e incluso permitiendo compatibilidad con otros lenguajes de búsqueda en RDF (como RQL).

En realidad Edutella es válido para cualquier sistema de marcado con metadatos en RDF. La aplicación educativa fue la primera y la más famosa de Edutella, dedicada a intercambiar información RDF entre Universidades y permitir un sistema de búsqueda distribuido, que permite búsquedas como las que se realizan en la red e2K (eMule 2000): locales (en el propio repositorio al que estamos conectados) o búsquedas globales (preguntando al resto de repositorios).

También se están estudiando estructuras mixtas [TAL03][FOS03][TAL04][BHA05], como por ejemplo Grids que contienen algunos elementos que son Clusters, y utilizan técnicas P2P para la búsqueda de recursos en el Grid y para las aumentar su resistencia a fallos. Incluso dentro de las arquitecturas P2P, en función de lo iguales que sean todos los nodos –entendiendo por igualdad que puedan asumir todas las funciones del sistema- se habla de sistemas más P2P puros o sistemas mixtos, en los que se desarrolla una cierta arquitectura Grid entre supernodos.

En la tabla 5.1 siguiente podemos realizar una comparación de las arquitecturas mencionadas en los grupos de arquitecturas anteriores.

	Cliente/Servidor	Cluster	Federados	Grid	P2P
Propietarios	1 por cliente 1 del servidor	1	$<10^3$	<100	$1<x<10^8$
Elementos	>=1 Clientes Único Servidor	$<10^3$	$<10^3$	$<10^4$	$1<x<10^8$
Tipo de elementos	Servidor altas prestaciones	Optimizados y homogéneos	Gran capacidad y heterogéneos	Gran capacidad y heterogéneos	Mayoría PCs particulares (edge of the web)
Localización Elementos	Único servidor, clientes distribuidos	Única localización	Distribuidos con localización determinada	Distribuidos con localización determinada	Distribuidos y localización variable
Dedicación	Total (servidor)	Total	Media	Alta	Baja
Tolerancia a fallos	Baja	Baja	Media/Baja	Media	Alta
Capacidad	según servidor	Fija	Determinista	Determinista	Variable
Sistema de gestión	Centralizado	Centralizado	Cierta distribución.	Cierta distribución	Distribuido
Confianza	según servidor/Total	Total	según servidor	Alta	Baja
Coste de arranque	Precio del servidor	Alto	Medio (utiliza elementos que ya existen)	Medio (utiliza elementos que ya existen)	Bajo

Tabla 5.1. Comparativa de arquitecturas

Como podemos apreciar en la tabla 5.1, las arquitectura cliente/servidor y cluster son las más fáciles le gestionar, al existir un único propietario. Sin embargo, son las menos tolerantes a fallos y las que exigen una mayor inversión inicial, en la compra del servidor o el cluster. Los sistemas federados y los grids, por otra parte, utilizan normalmente elementos ya existentes, por lo que reaprovechan recursos. Estos dos últimos tipos de arquitectura contienen elementos distribuidos, pero son localizables y a los que se les puede suponer cierta confiabilidad, ya que normalmente existe un acuerdo entre los dueños de los elementos participantes para seguir unas reglas. Sin embargo, si queremos llegar a grandes niveles de escalabilidad, de forma que podamos servir a una gran cantidad de agentes distribuidos por la web debemos de recurrir a otro tipo de arquitectura.

Las arquitecturas más escalables y que permiten la mayor entrada de elementos son las arquitecturas P2P. Además estas soluciones son más tolerantes a fallos y tienen un coste de arranque más bajo. Sin embargo, es en estos sistemas en los que existe menos confianza entre sus miembros, y al ser arquitecturas tan distribuidas su control es complicado, siendo sus valores de rendimiento más difíciles de evaluar, ya que se basan en muchos casos en valores probabilísticos.

Para poder diseñar la arquitectura más óptima para nuestros propósitos, primaremos la escalabilidad de las arquitectura P2P y su tolerancia a fallos, por lo que estudiaremos a partir de este punto las arquitecturas P2P y desarrollaremos soluciones que permitan atenuar los inconvenientes de confianza y control.

De hecho, en la actualidad las arquitecturas de red Peer to Peer son las más utilizadas para la distribución de contenidos multimedia. Según CacheLogic [WWW31], en diciembre de 2004 el tráfico de sistemas P2P superaba ya el 60% de todo el tráfico que circula por Internet. La causa de su auge radica en que generan grandes ventajas, sobre todo dirigidas a la supervivencia de dichas redes, ya sea ante grandes crecimientos de demanda, fallos de partes del sistema o ataques informáticos. Dicha supervivencia se manifiesta también ante los ataques legales -la localización y obligación de cerrar nodos por distribución de contenidos ilegales- por medio de la utilización de sistemas de encriptación, anonimato y repartición de responsabilidades entre todos los miembros de la red, que pueden aparecer y desaparecer continuamente. Este último punto hace surgir la duda de hasta qué punto son las arquitecturas P2P beneficiosas para el diseño de una arquitectura

orientada a la distribución de contenidos, pero con la necesidad de mantener y potenciar los derechos de autor. Este será otro de los puntos que intentaremos resolver en nuestra arquitectura, estudiando las distintas posibilidades que se plantean en las arquitecturas Peer to Peer.

5.3 Arquitecturas P2P

Una arquitectura Peer to Peer se define como una arquitectura en que los nodos tienen el mismo nivel, en el sentido que cada nodo es capaz de realizar cualquiera de las funciones necesarias en la arquitectura, y en la práctica muchas funciones del sistema están distribuida entre muchos de los nodos [WWW30].

Por tanto podemos entender como una red P2P a una red en la que no existen clientes y servidores prefijados, sino un número de nodos iguales que pueden funcionar a la vez como clientes y servidores para otros nodos de la red.

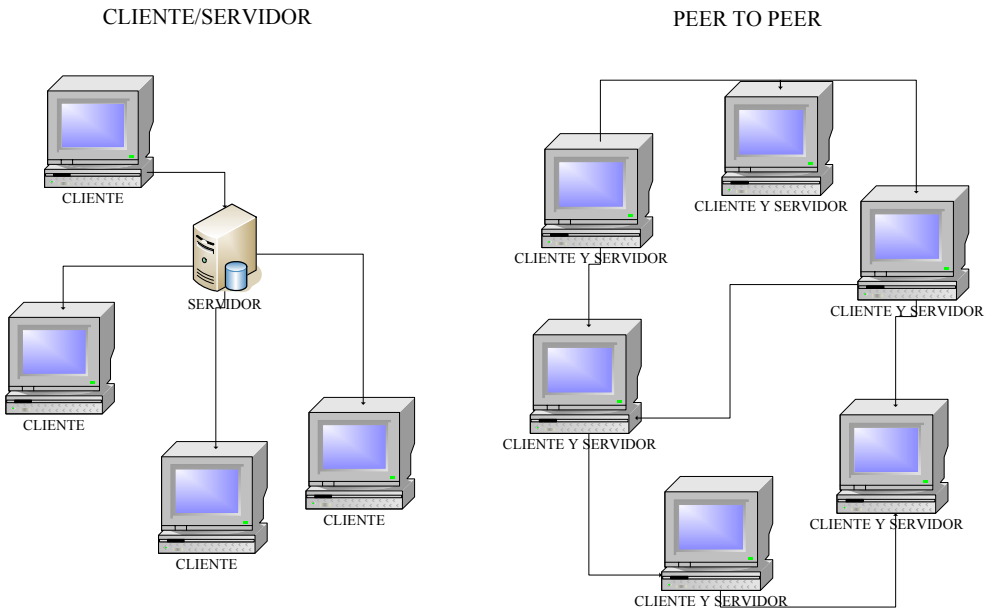


Fig. 5.1. Sistema Cliente-Servidor frente a un Sistema P2P

En la figura 5.1 podemos apreciar la diferencia entre la clásica arquitectura cliente/servidor y la arquitectura P2P. En la primera existe un nodo principal, llamado servidor, que realiza la mayoría de las funciones y al que han de conectarse directa o indirectamente el resto de nodos, que son

clientes de este nodo principal. Por otro lado, en la arquitectura P2P podemos observar que no existe ningún nodo que destaque en un principio de los otros, y no es un punto crítico mantener la conexión con un nodo determinado. De hecho, cada nodo puede ser a la vez cliente y servidor de otros nodos de la red.

Los usos principales de las arquitecturas P2P se pueden agrupar en:

- **Compartición de ficheros**, como utilidad más extendida y famosa. como ejemplo podemos tener una aplicación eMule funcionando en la red Kad. También hay pruebas de compartición de ontologías (a nivel de RDF o no estándar), como son el caso de SWAP[SWAP04] y su continuación Oyster [PAL06].

- **Computación paralela**, como en SETI@home, donde un servidor reparte partes del proceso de datos a los nodos, los nodos procesan los datos y envían los resultados de nuevo al servidor. Aunque existe una centralización en el reparto de datos y la recepción de resultados, el cálculo del los mismos los puede realizar cualquier nodo disponible.

- **Redes de Distribución de Contenidos** (CDNs), como la Coral Content Delivery Network [WWW33], donde se pretende hacer un sistema que cachee páginas de internet para poder ofrecerlas cuando el servidor que las ofrece esté saturado o haya caído. El almacenaje de esas páginas cacheadas se reparte entre los nodos de la red.

- **Comunicaciones entre usuarios**, como por ejemplo Skype o herramientas de chat, donde no es necesario un servidor que centralice las comunicaciones, y existe una búsqueda distribuida en la red.

- **Realizar búsquedas distribuidas en la red**. De hecho, existen muchos sistemas P2P que utilizan estos sistemas de búsquedas distribuidas para localizar nodos o recursos dispersos.

- Para implementar **herramientas de interacción directa entre usuarios**, como juegos y herramientas de colaboración. De nuevo la inexistencia de servidores intermedios mejoran los rendimientos y permiten mayor independencia y tolerancia a fallos.

Las arquitecturas P2P se desenvuelven en sistemas altamente heterogéneos, con grandes probabilidades de caída de los elementos que lo componen, lo que hace que se diseñen en gran medida como sistemas extremadamente tolerantes a fallos. Además los elementos, en lugar de buscar un objetivo común, buscan su propio objetivo particular, lo que hace que no se puedan planificar las tareas y existan grandes variaciones de demanda del sistema. Se basa normalmente en su mayoría en ordenadores personales donde los dueños no se conocen.

5.3.1 Desarrollo de las arquitecturas P2P

Se pueden clasificar las arquitecturas en varias generaciones [FOS03], [XIE03], que van consiguiendo ser cada vez más independientes de sistemas centrales y a su vez optimizando los algoritmos de búsqueda de recursos. En nuestro caso dividiremos las arquitecturas en tres generaciones:

1. **Arquitecturas P2P de primera generación**, con elementos centralizados, como es el caso de Napster (cuya búsqueda de recursos se hacía por medio de un sistema de indexación centralizado) o SETI@home, sistema que se basa en un servidor que envía a los nodos trozos del ruido detectado por las antenas que apuntan al espacio para intentar buscar patrones que puedan indicar inteligencia. Ya en esta primera generación, en caso de descarga de contenidos se realiza entre el interesado y el propietario del recurso (peer to peer).

   Estas arquitecturas de primera generación sirvieron para probar el potencial de cálculo y almacenamiento de los PCs de los usuarios de internet –el potencial de los elementos terminales de la web- así como para demostrar que los sistemas centralizados no son los más adecuados cuando hay un gran crecimiento en nodos, ya que por muy rápidos en el cálculo, mucha memoria que tengan y mucho ancho de banda, es insuficiente ante crecimientos exponenciales de miles de nodos.

2. **Arquitecturas de segunda generación**, donde ya se distribuyen las tareas, pero se basan en protocolos de enrutamiento y comunicación del tipo inundación -mandar el mismo mensaje por todos los nodos- u otros protocolos poco eficientes para un número grande de nodos. Para evitar saturaciones de red por el reenvío de mensajes, se utilizan sistemas de TTL -saltos de un mensaje antes de que desaparezca-y

en otros casos se permite la función de búsqueda sólo entre supernodos. Se entiende como supernodos a un subconjunto de los nodos de la arquitectura que son los que se utilizan para indexación, es decir, nodos en donde se indexan las búsquedas. Como ejemplos de estas arquitecturas tenemos las generadas por el protocolo FastTrack, protocolo usado por Kazaa [WWW32] o la arquitectura por defecto de JXTA, con sistemas de descubrimiento basados en multicast [JXTA1],[JXTA2]. Un sistema más avanzado que FastTrack es la red Gnutella2 y eDonkey 2000 (cerrada en septiembre de 2006 por Recording Industry Association of America, -RIAA-, aunque sigue en versiones como eMule), basados en nodos que tienen la libertad de decidir si participan como servidores de búsqueda –supernodos- o no.

3. **Arquitecturas de tercera generación**. Estas arquitecturas se basan actualmente en Distributed HashTables (DHTs), que son algoritmos que permiten que una tabla dinámica –Hashtable que trabaja con pares [clave, valor]- se encuentre distribuida en la red y permita búsquedas y almacenajes en la misma, aunque el número de nodos crezca. Estos algoritmos se utilizan en la red Kad, red utilizada por las nuevas versiones de eMule y por las nuevas versiones de BitTorrent. Mientras en otros sistemas la clave era el resultado de aplicar una función hash al contenido del documento, en este caso se indexa por funciones hash de las palabras de las que se compone el nombre del documento, permitiendo una simple búsqueda por palabras clave.

   BitTorrent es destacable sobre todo por su algoritmo de distribución de los contenidos. En este caso ya no se descarga el contenido de un nodo fuente, sino que se elabora una subred -llamada enjambre- dedicada a la distribución de un único fichero, entre uno o varios nodos que contienen la fuente, llamados semillas (seeds) y los nodos que se están descargando la fuente, de modo que se divide el gran fichero en fragmentos y cuando un nodo que hace de cliente tiene completo un fragmento ya puede compartirlo con los otros miembros de la subred.

Como hemos visto las generaciones han ido avanzado, de modo que al principio las funciones que podían realizar los nodos anónimos eran la de descarga de un contenido a otro nodo, y poco a poco han ido distribuyendo también entre todos los nodos la función de indexación y localización de

recursos, distribuyendo también la tarea de descarga de un contenido a un gran número de nodos receptores. En el desarrollo de nuestra arquitectura nos basaremos en la tercera generación, puesto que buscaremos una gran escalabilidad a la hora de indexar los contenidos ontológicos que nos permitan luego el acceso a los objetos educativos que pretendemos compartir.

5.3.2 Ventajas de las arquitecturas P2P

Aunque las hemos comentado por encima en el apartado anterior, las mayores ventajas de las arquitecturas P2P son [COO02][HAA04][ALO05],[SIE05]:

- **Alto grado de escalabilidad**. Al no haber centralización de funciones, los sistemas P2P están diseñados para que cada nodo nuevo que entra pueda aportar a su vez recursos utilizables por la red. Los protocolos a utilizar se diseñan de forma que al crecer a n nodos las necesidades del sistema para los encaminamientos sean del orden de *O*log(n).

- **Gran capacidad de adaptación** a las dificultades, ya que existe una cierta duplicidad de datos y posibilidad de absorber tareas de otros nodos que por alguna razón caen. En un sistema P2P puro no deben existir cuellos de botella ni tareas críticas asignadas a nodos determinados.

  Encontramos un ejemplo de adaptación a las dificultades la solución encontrada para utilizar nodos que están detrás de un subsistema que impide a otros nodos que contacten con ellos directamente. Estos nodos -conocidos como nodos LOWID en eMule- tienen cortadas las conexiones de entrada por un cortafuegos o no conocen su dirección IP porque están detrás de un traductor de direcciones de red (NAT). En estos casos, algunos sistemas -como eMule o Skype- se basan en utilizar nodos repetidores que actúan dejándose conectar por los dos nodos que quieren comunicarse, y así el nodo que era destino de la conexión no está obligado a ofrecer puertos de entrada abiertos. Podemos ver el esquema de funcionamiento en la Fig. 5.2.

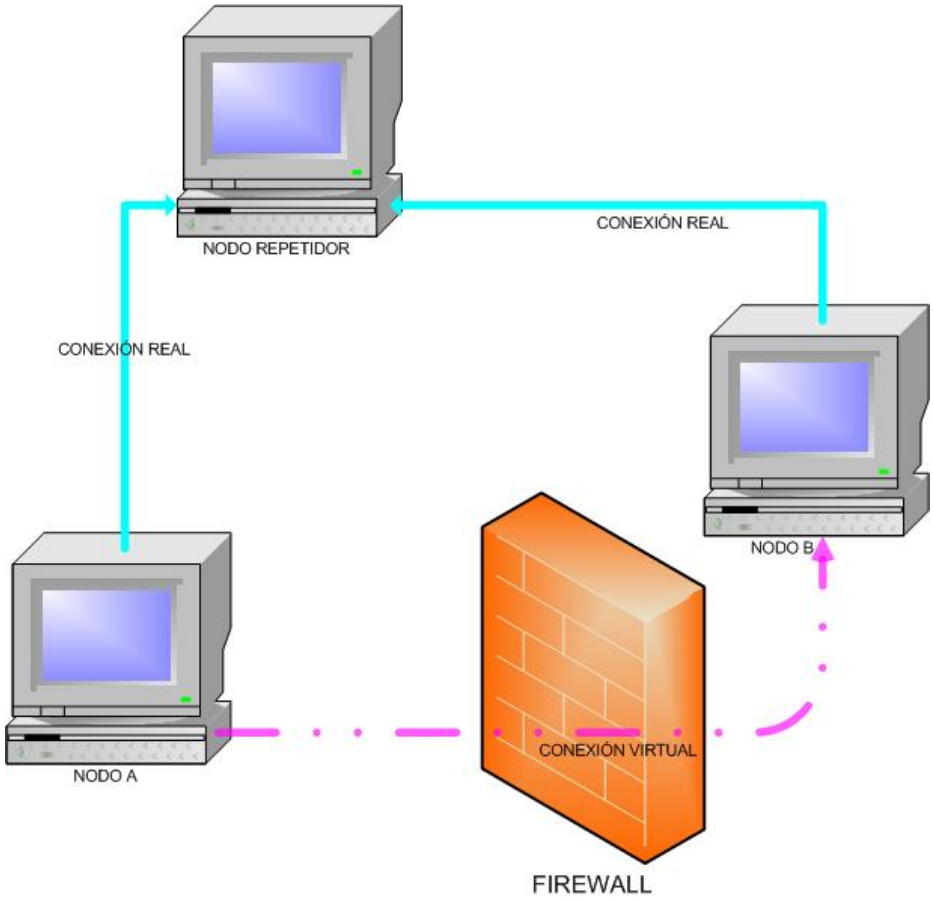


Fig. 5.2. Nodo repetidor

Como podemos apreciar en la figura 5.2, se genera una conexión virtual donde el nodo B permite la entrada de la conexión virtual. Dicha conexión virtual se lleva a cabo por medio de dos conexiones reales con un nodo repetidor, donde el nodo repetidor recibe las conexiones del nodo A y B.

Podemos ver otra adaptación que tiene como propósito evitar la avaricia de recursos, es decir, nodos que se aprovechan de la red y no ofrecen a la red recursos suficientes. Considerando como recurso la posibilidad de recibir conexiones inesperadas de entrada, vemos que los nodos LOWID no pueden actuar como nodos repetidores y por tanto ofrecen menos recursos que los que permiten ser utilizados como nodos repetidores. En este caso se penaliza a los nodos LOWID con menores capacidades de uso de la red, por ejemplo, un ancho de banda de bajada muy bajo.

- **Resistencia los ataques a la red**, ya que normalmente los ataques a la red se hacen buscando los lugares más débiles y sobrecargándolos. En este caso no hay puntos fuertes ni débiles. De todas formas, los sistemas distribuidos han de tener algoritmos que eviten en todo momento realimentaciones positivas que hagan que el propio sistema entre en bucles de crecimiento de carga. Como ejemplos de estas protecciones a las realimentaciones positivas tenemos el hecho de no volver a consultar nodos que ya se han consultado, y la utilización de paquetes con un número determinado de saltos antes de que desaparezcan.

- **Adaptación a los picos de demanda**, ya que la carga está distribuida entre todos los nodos de la red. De esta forma, por ejemplo, un aumento de demanda a la red en un valor X se convierte normalmente en un aumento de demanda X/N para cada uno de los N nodos que satisfacen dicha demanda. Además, cuando el aumento de demanda se debe a la aparición de nuevos nodos, estos nuevos nodos se unen normalmente a la red para prestar sus servicios. Un ejemplo de esta adaptación es el protocolo BitTorrent, donde cada nodo que solicita un fichero se convierte a su vez en posible servidor de la parte del fichero que ya se ha descargado.

- **Bajo coste de arranque y mantenimiento**, de forma que si conseguimos que todo nodo que se conecte se beneficie de alguna forma, no hace falta comprar grandes ordenadores que lleven un seguimiento centralizado. Llegados a este punto es importante destacar que es mucho más eficiente invertir en un cluster que comprar una gran cantidad de nodos para implementar una red P2P, en el caso que no pueda utilizar nodos de otros propietarios. Toda red P2P consume en conjunto más recursos que un ordenador monolítico realizando la misma función, ya que existe una gran redundancia, que es una de las razones por las que el sistema es tan resistente. Sin embargo, cuando hablamos de órdenes de magnitud de 10^8 nodos, aunque se produzca redundancia y no se utilice el potencial de cada nodo al cien por cien, la capacidad total es mayor que cualquier sistema monolítico existente.

5.3.3 Inconvenientes de las arquitecturas P2P

Las propias características de las arquitecturas P2P llevan aparejadas ciertas desventajas que pueden ser aceptadas o no según la función del sistema que queramos desarrollar. Podemos enumerar las siguientes [ALO05][VER04][GER03]:

- **Falta de determinismo**. La gran escalabilidad de los sistemas hace que para una gestión de un número tan elevado de posibles nodos, se recurra a técnicas probabilísticas. Esta falta de determinismo acarrea desventajas, como que los tiempos de respuesta de la red sean muy variables y difícilmente asegurables.

- **Imposibilidad de una localización fija de recursos**. Cada vez que se accede de nuevo a un recurso no se puede asegurar que se encuentre en el mismo nodo en el que estaba, ya que existe una reconfiguración y relocalización de recursos dinámica. Lo que es una ventaja para proteger el sistema ante caídas de un nodo se convierte aquí en una desventaja. En cualquier caso una práctica habitual es las redes P2P es acceder primero a los que nos ofrecieron servicio con anterioridad, de modo que los utilizamos como puntos de acceso a la hora de entrar de nuevo en la arquitectura.

- **Dificultad de imponer autoridades en el sistema para control**. La propiedad de disponer de sistemas de control distribuidos es una ventaja contra ataques o caídas, pero para determinados casos puede ser un gran problema. Por esta causa se han de diseñar sistemas que permitan regular y controlar a los nodos que no se comportan como se espera de ellos. Algunos sistemas realizan la distribución del control, por medio de sistemas de créditos -como eMule-, y elaboran rankings de mejores nodos a los que el sistema ofrece mejores servicios.

- **Dificultad de detectar los nodos maliciosos**. No solamente es difícil actuar sobre los nodos maliciosos, sino que muchos sistemas, que priman el anonimato y la confusión respecto a los nodos que solicitan un recurso, hacen que sea complicado la localización de algún nodo que está dañando al sistema, por ejemplo por medio de envíos masivos de peticiones y negativa de servir al resto de nodos.

- **Falta de confianza**. Al no conocer los otros nodos de la red, es de esperar cierta desconfianza respecto a los servicios que nos

proporcione. Como ya hemos visto, en otras arquitecturas esto no ocurre. Por ejemplo, en las arquitecturas Grid, conocemos a los proveedores de los nodos. Encontramos otro ejemplo en las arquitecturas clientes/servidor, donde todo depende de la confianza que tengamos en ese único servidor. En arquitecturas P2P, la estimación de confianza en un nodo se suele realizar en algunos sistemas también de forma distribuida –por medio de sistemas de créditos- y hemos de apoyarnos en firmas digitales y códigos hash de los recursos para asegurar la autenticidad de los mismos.

- **Limitación de los servicios ofrecidos**. Una aplicación P2P ha de estar funcionando en una gran cantidad de nodos antes de que el sistema sea realmente útil para todos los participantes, por lo que es ciertamente difícil generar aplicaciones P2P que sean exitosas en la web. Además, normalmente se exige una cierta apertura del código (por ejemplo GPL) que permita comprobar a las comunidades que la aplicación P2P no es dañina para los propios nodos y que no realiza ninguna acción maliciosa.

- **Necesidad de políticas win-win**. Para que el sistema sea aceptado por una gran cantidad de nodos, los nodos han de estar individualmente interesados de alguna forma en los servicios que se ofrecen. Su interés ha de ser tan alto que permitan el uso de sus recursos a cambio de esos beneficios. No todos los sistemas pueden ofrecer una ventaja que haga que entren nodos. De hecho los sistemas P2P normalmente se enfocan en sistemas no comerciales en los que el pago por los servicios es la misma participación en los mismos.

5.4 Distributed HashTables (DHT)

Como hemos comentado, la tercera generación de sistemas P2P se basan en el desarrollo de algoritmos que permiten la búsqueda distribuida de los recursos, por medio de la generación de tablas dinámicas distribuidas.

Definimos una tabla dinámica distribuida (DHT) como una Hashtable que trabaja con pares [clave,valor], encontrándose distribuida en la red y permitiendo búsquedas y almacenajes en la misma, aunque el número de nodos crezca o disminuya dinámicamente. Para realizar esta tarea las DHT se fundamentan en unos algoritmos que veremos a continuación.

Normalmente las claves de las DHTs suelen ser una transformación por medio de una función hash del recurso que queremos buscar, y el valor de la tabla indica la dirección URL del nodo que puede ofrecerlo. En los casos de redes de distribución de ficheros, la clave suele ser la función hash del fichero que buscamos y el valor la dirección URL de donde podemos descargarnos el fichero en el que estamos interesados.

Como podemos ver en la figura 5.3, las DHTs nos permiten demandar los siguientes servicios:

- **Unirnos a la subred** P2P que configura la DHT, ya que en cierto modo la DHT es una red P2P que permite la localización de recursos en otra red P2P.

- **Salir de la red**, cuando estamos unidos a la red. El sistema permite que haya salidas inesperadas y no notificadas de la red.

- **Buscar contenidos según la clave**, normalmente cuando estamos unidos a la red, de forma que colaboramos con la misma. El resultado de la petición es el valor asociado a esa clave en la tabla dinámica o la incapacidad de encontrar dicho valor.

- **Insertar un par (clave, valor)** en la DHT, de nuevo normalmente cuando estamos unidos a la red, de modo que cedemos recursos a cambio de obtener los servicios de la DHT. El resultado es que el par (clave, valor) queda insertado en la DHT y podrá ser consultado por otros nodos

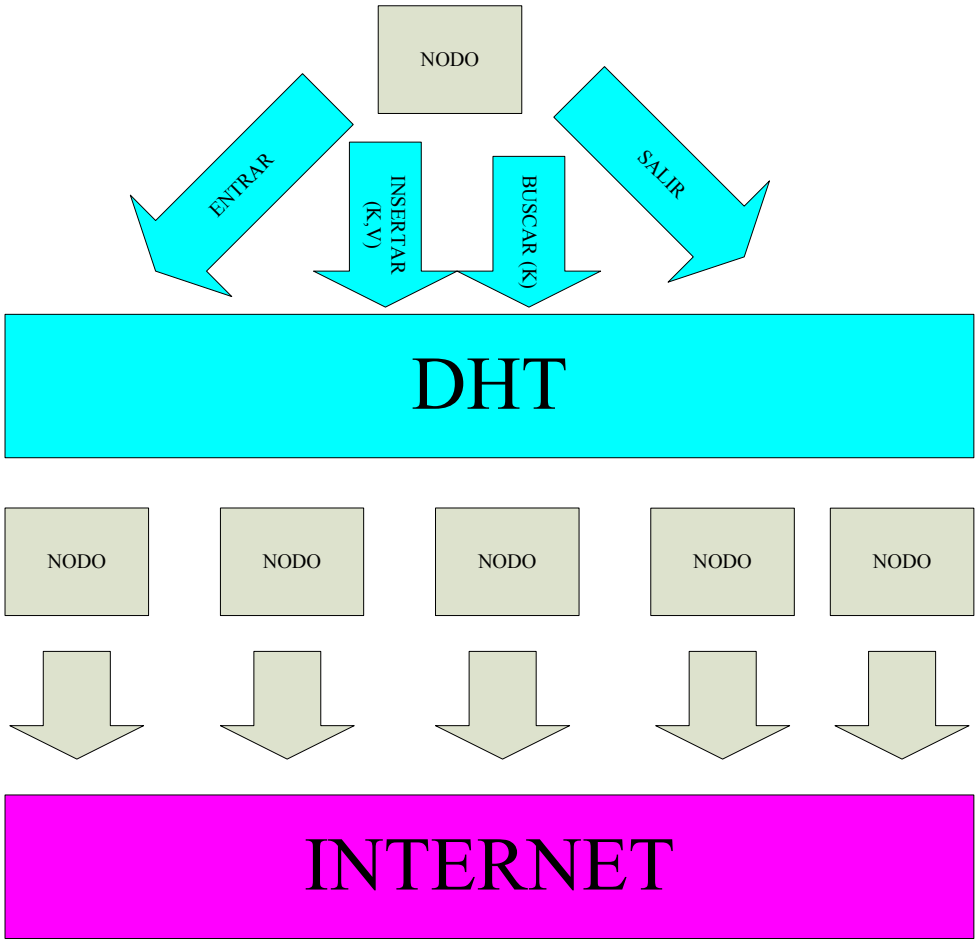


Figura 5.3. Servicios ofrecidos por las DHT

En la figura 5.3 podemos apreciar a su vez que las DHT se sustentan normalmente en un conjunto de nodos que se han unido a la DHT y que normalmente se unen entre sí por medio de la red internet. Un nodo puede realizar las peticiones de entrada si no pertenece a la DHT, de salida si sí que pertenece y luego las peticiones típicas a una tabla dinámica.

Las bases de los DHTs son:

1. **Las claves han de estar distribuidas aleatoriamente y de forma uniforme, siendo únicas**. Esto se suele hacer por medio de pasar una función hash a la fuente (nombre de la fuente, contenidos del fichero, etc). Para evitar repeticiones se usa un espacio de valores de 20 bytes (160 bits) que nos da de orden de 2^{160} (10^{48}) valores. Para hacernos una idea de la magnitud del espacio de valores, en la Tierra

hay del orden de 2^{170} átomos. Algunos sistemas, como e-Mule, utilizan espacios de 128 bits, considerándolos suficientemente amplios.

2. **Los identificadores (IDs) de los nodos han de estar distribuidos a su vez aleatoriamente y de forma uniforme, siendo únicos**. Esto se suele hacer a su vez pasando una función hash (normalmente SHA-1 que genera 160 bits) a la dirección del nodo.

3. **Se define una topología**, donde se define una función distancia entre nodos, y el número de nodos conectados a un nodo determinado. En este caso el concepto de conexión se refiere al caso en el que el nodo conoce las direcciones IP de los nodos con los que está conectado, y por tanto se consideran como conexiones directas sobre la red P2P superpuesta a internet.

4. **Se hace que los nodos almacenen los valores de la DHT (clave, valor) cuyas claves estén cerca (según la topología elegida) del ID del nodo**. Al estar distribuidos aleatóriamente tanto las claves como los nodos, la carga se reparte por igual entre los nodos.

5. **Se proveen algoritmos para que un nodo pueda entrar conociendo un nodo que esté ya en la red P2P de la DHT**, se coloque en el lugar que le corresponde y almacene los valores de la hashtable que le correspondan. Además se proveen mecanismos distribuidos para que en caso de abandono o caída inesperada de un nodo, se pueda recuperar el sistema. Este último punto se lleva a cabo normalmente haciendo que cada nodo cachee la parte de la hashtable de j nodos que estén cerca de él.

Como podemos ver en la tabla 5.2, según topologías y definición de distancias se obtienen unos algoritmos u otros. Se consideran mejores sistemas los que necesitan menos saltos en las consultas de la DHT y menos tiempo y menos mensajes generados a la hora de que un nodo entre o salga de la DHT.

Nombre	Topología	Saltos en búsqueda	Conexiones a recordar
CHORD	circular	Olog(n)	Olog(n)
CAN	toroidal de d dimensiones	O(dn1/n)	d

PASTRY	malla	Olog(n)	Olog(n)
TAPESTRY	malla	Olog(n)	Olog(n)
PLAXTON	malla	Olog(n)	Olog(n)
DB2	De Bruijn	Olog(n)	O(1)
CRN	Mariposa	Olog(n)	Olog(n)
KADEMLIA	malla	Olog(n)	Olog(n)

Tabla 5.2. Comparativa de algoritmos para DHT. Basada en [XIE03]

En la tabla 5.2 podemos apreciar los diferentes algoritmos de DHT en función de la topología elegida. Los primeros algoritmos que surgieron fueron [MAY02] CHORD (utilizado en CCDN), CAN (basado en una topología d-dimensional pero con una complejidad de ejecución del algoritmo alta y con peor escalabilidad). Los algoritmos Pastry, Tapestry y Plaxton y Kademlia se basan en una topología de malla (según los "1s" o "0s" de su id de nodo) aunque el más utilizado es Kademlia [MAY02], al ser más óptimo para redes muy grandes. Kademlia es actualmente la base de la red Kad sobre la que funciona eMule.

El algoritmo Kademlia se basa en una distancia definida como la or exclusiva entre los puntos.

$$\partial(a,b) = a \oplus b \qquad \text{[Ec 5.1]}$$

Las ventajas de la topología derivada de la distancia en Kad respecto a los otros algoritmos son la simetría y la unidireccionalidad. La simetría permite que, en el caso en que un nodo tenga cerca a otros nodos, éstos lo tengan a él también como nodo cercano. La unidireccionalidad de la distancia permite que sólo haya un nodo a una distancia D del nodo X, de forma que las búsquedas de la misma clave suelen acabar pasando por los mismos nodos según nos acercamos al nodo buscado.

La simetría nos permite aprovechar los mensajes que nos llegan de búsqueda para comprobar que los contactos del nodo son correctos, y por tanto no necesitamos cargar la red con más mensajes de control. La unidireccionalidad nos permite hacer que los nodos por los que ha pasado una petición cacheen el resultado de la petición, ya que es posible que cuando se pida el mismo recurso de nuevo pasen por ellos.

En el algoritmo Kademlia, cada nodo n_i tiene una lista –llamada bucket- de otros nodos por cada bit del tamaño de las claves, de modo que tiene 160 buckets cuando tratamos con claves de 160 bits. En cada bucket de la

posición j (b_j) almacena información para contactar con un máximo de k nodos (k normalmente=20) que distan de n_i el orden de magnitud 2^j (es decir, que coinciden en los bits j+1 a 159 (si son claves de 160 bits) y difiere al menos en la posición j, pudiendo diferir o no en los bits 0 a j-1. Puede que algunos buckets estén vacíos, sobre todo los de valores i más bajos. Para elegir los mejores nodos a conectarse dentro de un bucket se premia los que llevan más tiempo en la red, ya que se basa en la suposición de que cuanto más tiempo lleva un nodo en la red más probable es que continúe en ella.

El algoritmo Kademlia contempla también que cada nodo lance múltiples búsquedas paralelas al siguiente salto de búsqueda. El número máximo de peticiones paralelas por nodo se fija por medio de una variable α, a la que se recomienda se le asigne el valor de α=4.

5.5 Arquitectura propuesta

A partir de este punto, tras ver las distintas arquitecturas, haber decidido definir una P2P por razones de escalabilidad y robustez, y después de haber visto con detalle las distintas posibilidades de arquitecturas P2P pasamos a definir con detalle la arquitectura sobre la que funcionará la ontología OWL y que permitirá su uso por parte de agentes inteligentes que busquen recursos educativos.

Pretendemos que la arquitectura propuesta cumpla los siguientes requisitos:

- **Permitir búsquedas por agentes inteligentes** de los contenidos ontológicos. De esta forma podrán actuar los agentes inteligentes y obtener la información en OWL o sistema equivalente. Esto significa que los agentes inteligentes no disponen en un principio de toda la información lógica necesaria para realizar sus razonamientos, sino que puede que pidan al principio y luego vayan ampliando su base de conocimiento, tanto de su TBox como de su ABox.

- **Los sistemas serán sistemas conectados a Internet**, por lo que se diseñará una red sobre Internet (overlay net). Se define una red sobre internet [KER04][AKE04] como una red que consiste en una serie de nodos preseleccionados, localizados en una o varias subredes, y conectados entre sí por medio de un enrutamiento a nivel de aplicación, aunque se sigue conectando por medio de la red IP. Al realizar el enrutamiento a nivel de aplicación, existe un mayor control sobre las conexiones y se pueden aplicar algoritmos de selección de conexiones particularizados.

- **Permitir una distribución rápida de contenidos**, evitando en la medida de lo posible la distribución de contenidos con copyright de forma libre.

- **Soportar cierto grado de escalabilidad**. En este caso trabajaremos con soluciones P2P adaptadas, que permitan grandes picos de tráfico, si bien se realizará un cierto control.

- **Permitir un funcionamiento mínimo del sistema**, de forma que si acude un primer cliente al sistema pueda realizar búsquedas y obtener los objetos educativos deseados.

- **Identificación de las fuentes**. Si en otros sistemas peer to peer el anonimato es premiado, al menos en nuestro sistema identificar los orígenes de los objetos educativos es positivo para evaluar su posible calidad a priori.

Todos estos requisitos se contemplarán en una arquitectura P2P con ciertas modificaciones, principalmente modificaciones que faciliten el trabajo con la web semántica y agentes inteligentes y permitan la identificación de las fuentes de los contenidos.

5.5.1 DHT Semántica

Para permitir el uso de la arquitectura por parte de agentes inteligentes que trabajan en la web semántica, tendremos que adaptar los sistemas de localización de recursos para que puedan funcionar con términos lógicos y permitan una rápida adquisición de conocimientos por parte de los agentes. Para hacer este sistema de localización de la arquitectura que diseñamos, y al no existir ningún sistema en la actualidad que resuelva esta cuestión, definimos en esta Tesis un sistema que denominaremos DHT Semántica, que describiremos a continuación y que permite el trabajo con la web semántica en redes P2P de tercera generación.

La arquitectura propuesta se fundamenta en un sistema de búsqueda P2P basado en las DHT explicadas anteriormente. En este caso se escoge también el algoritmo Kademlia. Una de las diferencias principales radica en que las claves de la tabla dinámica distribuida serán la función hash de las URIs utilizadas para describir ontologías, clases, propiedades e instancias en OWL. De esta forma, en lugar de realizar búsquedas de URLs donde se pueda localizar un fichero determinado, los agentes podrán buscar URLs

donde se encuentren ficheros OWL que ofrezcan información acerca de un elemento de la ontología –clase, instancia o propiedad- determinado.

Para poder realizar estas búsquedas, la información que debe almacenar cada nodo de la DHT semántica será la información para poder localizar el servidor de contenidos que generó cada uno de los ficheros relacionados. Además, un nodo deberá almacenar si el fichero OWL del que almacena su localización contiene información de descripción de clases y propiedades (TBox) y si contiene información sobre instancias de la ontología (ABox). Aunque en OWL DL la información de la TBox y de la ABox no se mezclan (una clase no puede ser a su vez una instancia de otro tipo de clase), sí que es posible que un mismo fichero contenga información de los dos tipos.

Esta peculiaridad de marcar los contenidos según el tipo de información que contiene sería otra de las modificaciones importantes respecto a las redes existentes basadas en Kademlia (como la red KAD), por lo que se puede colocar esta información en alguno de los campos no utilizados de la red particular sobre la que construyamos. La causa por la que separamos explícitamente la información de la TBox y la información de la ABox radica en que la descripción de clases y sus relaciones (TBox) será la información a la que primero accederán los agentes –ya que asienta las bases de la ontología- y por regla general tendrá muchos menos elementos a indexar que la información ABox, que trata de las instancias y las relaciones entre las mismas. Por ejemplo, una ontología de cursos no debería tener más de un centenar de elementos (URIs distintas) en su TBox, mientras que instancias de esos elementos (como por ejemplo, comentarios unitarios de foros que pertenezcan a todos los cursos registrados en la red) puede haber más de 10^5.

Otra diferencia destacable entre esta red y una red DHT normal, a parte de que indexa contenidos semánticos y los marca como TBox y/o ABox, es que son los servidores de contenidos los que publican (técnica push) en la DHT Semántica su información. Para realizar esta publicación, los servidores de contenidos han de basarse en una arquitectura que les permita que cuando tengan nueva información semántica que publicar, la analicen y den de alta como claves de la DHT Semántica todas las URIs utilizadas como clases, propiedades e instancias del documento OWL.

Por ejemplo, si un servidor de contenidos desea publicar una edición antigua de un curso, describiendo tanto los contenidos como las

interacciones de los alumnos y profesores, generarán el archivo .owl y luego lo transformará por medio de capa de publicación en un conjunto de claves y valores de URLs que insertará a modo de entradas en la DHT. Por medio de este mecanismo, por cada URI que tenga algún significado en la ontología se publicará en la DHT Semántica un par (hash(URI), URL) que permitirá a los agentes localizar la URL donde se encuentra el fichero OWL que describe de alguna forma el elemento ontológico identificado por la URI. No es necesario utilizar un archivo para describir cada objeto educativo o experiencia educativa, sino que se pueden agrupar, aunque para mejorar las prestaciones se recomienda no utilizar archivos muy grandes con información de TBox (ya que es la que más se accede a priori) o incluso intentar mantener los archivos con información de TBox separados de cualquier información de la ABox..

En la Fig. 5.4 podemos ver la arquitectura de DHT Semántica, detallando las funciones a realizar por parte del servidor de contenidos.

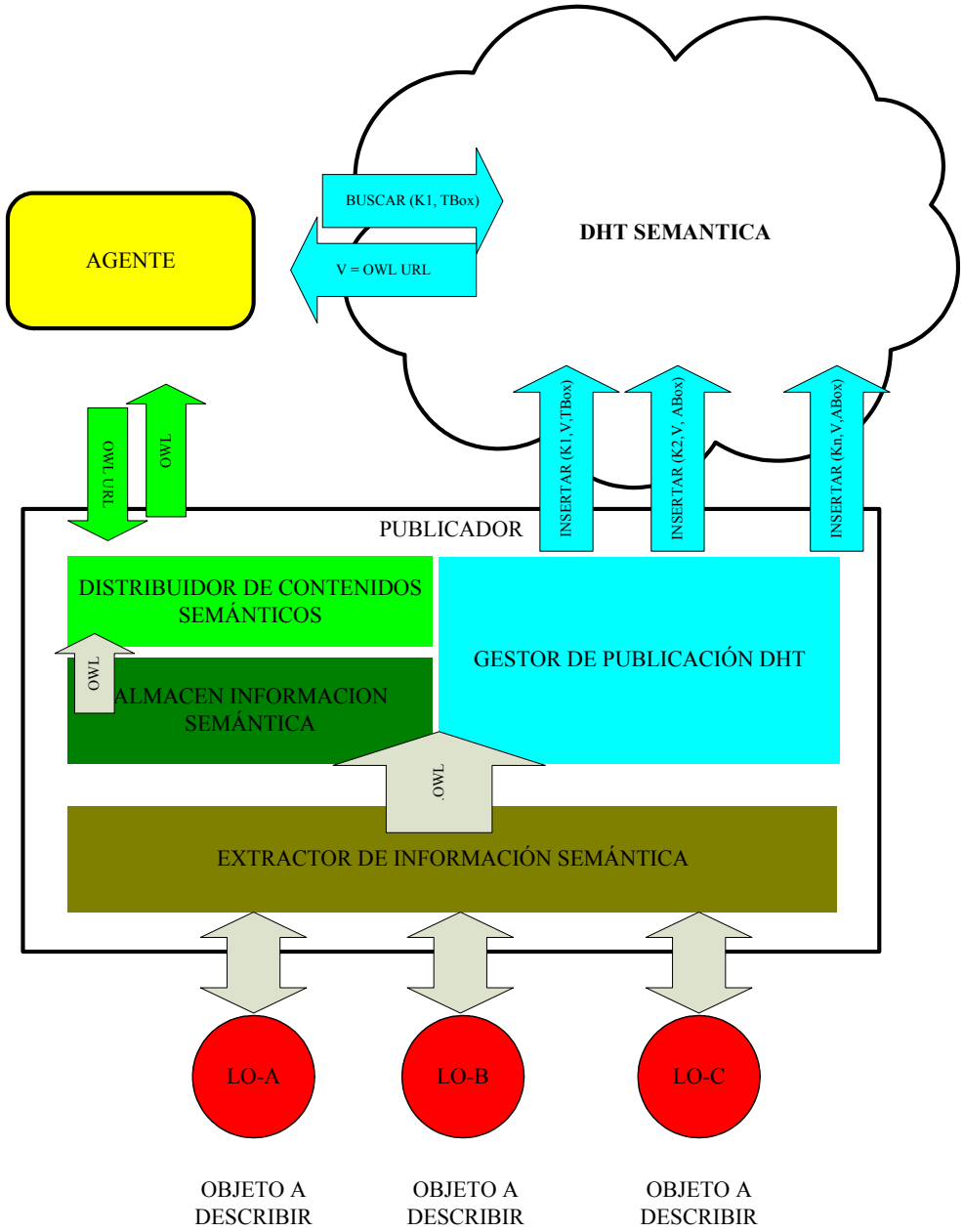


Fig. 5.4. DHT Semántica

Como podemos apreciar en la figura 5.4 el servidor de contenidos dispone de un elemento funcional encargado de la publicación en la DHT Semántica y facilitación de documentos OWL a los agentes. A este elemento lo denominaremos publicador. El publicador primero extrae la información de descripción de los objetos educativos a describir, en este caso LO-A, LO-B

y LO-C. Con esa información genera el archivo OWL que almacena en su almacén de información semántica. Además de almacenarlo lo transfiere también al gestor de publicación DHT, que convierte el archivo OWL en valores a almacenar en la tabla dinámica distribuida de la DHT Semántica. En este caso se introducen como claves (K1,K2,.., Kn) las distintas funciones hash de las URIs de los elementos descritos en el archiv OWL. Como valor de la publicación (V) se ofrece la dirección URL que permita localizar el archivo OWL por parte de los distintos agentes. En la DHT Semántica ha de indicarse a su vez si el par (clave, valor) se refiere a información ABox o TBox.

Por otro lado, los agentes consultan en la DHT Semántica localizaciones de archivos OWL donde se describa un elemento ontológico determinado. Para realizar esta función, el agente ha de consultar a la DHT Semántica utilizando como clave de búsqueda la función hash de la URI que identifica a nuestro elemento ontológico (como puede ser el nombre de una clase) y si la información que buscamos es para la TBox o para la ABox de la base de conocimiento del agente. Tras esa petición, la DHT semántica le contesta con el valor V (o valores) que consiste en la URL del archivo OWL (o archivos OWL) donde puede localizar esa información.

Una vez el agente ha obtenido la dirección URL donde localizar el fichero OWL, intentará obtener dicho fichero de esa dirección. Para servir el fichero OWL, el publicador dispone de un distribuidor de contenidos semánticos, que obtiene del almacén el archivo que se ha solicitado. La razón por la que no se sirve el fichero OWL directamente y se pasa por un distribuidor de contenidos semánticos radica en que el almacén de información semántica puede no ser un simple almacén de ficheros OWL, sino una base de conocimiento que realice chequeos de consistencia de la información ofrecida y actualice la información según se la vaya ofreciendo el extractor de información semántica.

Una vez descrito el proceso de localización y obtención de información semántica para los agentes inteligentes, pasamos a describir el funcionamiento de la arquitectura de distribución de contenidos.

5.5.2 Descripción del funcionamiento de la arquitectura

Nuestro objetivo en este apartado es describir cómo debe funcionar todo el sistema para permitir a los agentes la localización por medio de información semántica de objetos educativos a compartir. Como ya hemos visto en otros

capítulos, la web semántica se basa principalmente en describir URIs y relacionarlas entre sí por medio de propiedades

Las fases que definimos son las siguientes, que describiremos a fondo en los siguientes subapartados:

1. **Fase de definición y creación de ontología a utilizar**. En esta fase los agentes absorben la información ontológica básica (entendida como la información para su TBox). En el caso que ya tenga información, intentará ampliarla por si hay ampliaciones ontológicas.

2. **Fase de formación de las queries ontológicas**. En este punto el agente interactúa con el usuario y le muestra las distintas posibilidades de búsqueda de las que dispone.

3. **Fase de inferencias de búsqueda**, donde el agente utilizará su información y obtendrá más información para su ABox para poder localizar elementos que cumplan las restricciones impuestas por el usuario.

4. **Fase de contacto con el servidor de contenidos educativos**. En este punto el agente pide más información al servidor de contenidos para presentarlos al usuario y que negocie con el servidor si desea o no descargar los objetos educativos relacionados.

5. **Fase de descarga**, donde se obtiene el objeto u objetos deseados.

Como podemos apreciar, estas fases engloban todas las acciones necesarias para poder localizar y descargar los objetos educativos que necesite un usuario para poder componer una experiencia educativa que pueda ser utilizada por él mismo o por un conjunto de alumnos.

5.5.2.1 Fase de definición y creación de la ontología a utilizar

En esta fase el usuario elige una URI donde se encuentra la parte principal ontología sobre la que realizar la búsqueda. Un ejemplo de URI de una ontología puede ser “http://dcom.upv.es/owl/rorollo/tesis.owl#", que será la ontología propuesta en esta Tesis y desarrollada para describir una experiencia educativa.

Al usuario se le deberían presentar las últimas ontologías utilizadas o permitirle introducir la URI de la ontología a mano. Esta parte podría quedar eliminada si obligamos que siempre parta de la misma ontología, facilitando el uso pero limitando las posibilidades de realizar las mismas búsquedas pero con otras URIs.

El agente utiliza el arranque de la búsqueda (que en principio será el archivo .owl con la descripción de la ontología en la versión inicial). Sin embargo, OWL, como la mayoría de los lenguajes ontológicos de la red semántica, permite que la definición de la ontología se encuentre distribuida en distintos documentos, por lo que hará es buscar en la DHT semántica los documentos que se relacionan con la URI de definición, y que no se refieren exclusivamente a definir instancias de la ontología.

El agente obtiene la ontología por medio de localizar los ficheros que definen la ontología <owl:Ontology rdf:about="(la URI)"> y están marcados como "TBox" en la DHT Semántica. Además, si encuentra etiquetas de importación de ontologías owl (<owl:imports rdf:resource="(NEW_URI)">) continua añadiendo a su TBox la información de la NEW_URI encontrada. Si encuentra ontologías más modernas que la que se busca (es decir, que referencian a nuestra URI en <owl:priorversion>), debería indicarlo al usuario por si decide utilizar la nueva versión de la ontología (en caso de ser compatible con la anterior por medio de <owl:backwardCompatibleWith>). Este sistema es parecido al definido para RDF en [BID04].

Cada vez que el agente añade información semántica de un fichero OWL a su TBox, debe comprobar la congruencia del sistema para evitar que la TBox contenga contradicciones. En principio una ontología que se publique en la DHT Semántica no debería contener contradicciones, pero se ha de evitar realizar inferencias sobre ontologías inconsistentes, ya que pueden llevarnos a resultados dispares.

Tras esta fase el agente ya ha almacenado en su TBox la información básica de la ontología: definición de clases y propiedades, lo que denominaremos la ontología bases (a falta de incluir instancias de las clases).

5.5.2.2 Fase de formación de las queries ontológicas

Una vez almacenada la ontología base, el agente presenta al usuario las posibilidades de búsqueda que se presentan, que dependerá de las clases y propiedades que se han encontrado.

Pongamos por ejemplo la ontología definida en esta Tesis para la teleformación (*http://dcom.upv.es/owl/rorollo/tesis.owl#)*. El agente le presentará al usuario la posibilidad de buscar foros que cumplan ciertas condiciones. Una de las posibilidades de búsqueda sería buscar las instancias de la clase *LearningExperience* que contengan instancias de *ForumThread* cuyo *hasFirstTreeItem* contenga la palabra "Socket" en el String que devuelve su propiedad *getText*. Estas clases ontológicas se verán en el siguiente capítulo de esta Tesis.

Para aplicaciones específicas con TBoxes fijas, el usuario puede pasar directamente a realizar unas ciertas búsquedas predefinidas que facilitan las tareas de búsqueda. De todas formas esto haría perder parte de la flexibilidad del uso de la web semántica y pasaría a ser un sistema predefinido basado en XML o en RDF.

Al final de esta fase el usuario decide el tipo de búsqueda que desea realizar, y ordena al agente que comience la búsqueda en la web semántica, es decir, basándose en su TBox ampliar la ABox con instancias e ir realizando inferencias hasta encontrar elementos que cumplan los requisitos de la búsqueda planteada.

5.5.2.3 Fase de inferencias de búsqueda

En esta fase el agente realiza las inferencias necesarias y decide qué tipos de instancias de qué clases ha de buscar. Su primera acción es asegurarse que la búsqueda que pedimos no es incongruente según su base de datos, para detener la búsqueda en ese caso.

Empieza la búsqueda obteniendo información para la ABox de la DHT semántica. De la DHT obtiene la localización de los archivos .owl a descargar del servidor de contenidos, no volviendo a descargar un fichero si ya lo ha obtenido, ya que se le supone un cacheo de ficheros obtenidos. Como las DHT sólo permiten búsquedas por palabras exactas, en caso de filtrar por partes de palabras de un campo, debe ser el propio agente en el momento de obtener las fuentes, generando internamente una subclase de la clase a filtrar y poblándola con las instancias que cumplan la restricción. Por ejemplo, en la búsqueda de la fase anterior, debería buscar instancias de la clase *FirstTreeITem* y generar una subclase que se poblaría con las instancias de *FirstTreeItem* que contuvieran el subString "Socket" en su propiedad *getText*.

5.5.2.4 Fase de contacto con el servidor de contenidos educativos

El agente va encontrando resultados, y se los muestra al usuario, junto con la información del recurso que ha encontrado. Hasta este punto el agente ha servido como un buscador semántico, es decir, ha ido poblando su base de conocimiento y realizando inferencias sobre la misma para poder devolver instancias que cumplan los requisitos exigidos por el usuario. Hasta aquí llega la búsqueda en la web semántica.

El usuario decide si quiere obtener más información de ese recurso educativo, y si lo hace se accede al servidor que contiene esa fuente (un servidor de objetos educativos). El agente entonces pedirá al servidor de objetos educativos toda la metainformación sobre ese objeto educativo. En este caso el Servidor puede ofrecer solamente la información en owl que ya ha ofrecido en el periodo de búsqueda o puede enseñar demostraciones, que permitan conocer más el objeto, siendo esta parte más publicitaria y dirigida a ser entendida por el usuario directamente.

Como vemos esta parte es más comercial, ya que se ha localizado algún objeto educativo de los que busco pero ha de ser el usuario quien realice la validación final y negociación para decidir si adquiere o no el objeto educativo.

5.5.2.5 Fase de descarga

Cuando el usuario, que normalmente será un ensamblador de contenidos, está interesado en alguno de los objetos, pasará a pedir la obtención del contenido del objeto educativo. En este caso se especifica un sistema de entrega de contenidos inteligente de forma que el servidor de contenidos pueda servir directamente al cliente o, si detecta que va a servir un archivo grande (normalmente los que tienen una gran carga multimedia), pase a ejecución del protocolo de distribución BitTorrent, arrancándolo por medio de envío del archivo .torrent al agente que espera la descarga.

En este segundo caso el agente que pasa a la descarga, debe estar preparado para ejecutar el protocolo BitTorrent si está obtenido en obtener la información.

5.5.3 Elementos del sistema

Los nodos mínimos existentes en el sistema para que se realice un funcionamiento normal de la arquitectura serán (ver figura 5.5):

- **Los servidores de objetos educativos** (servidores de contenidos). En este caso se supone que son repositorios en entidades que decidan entrar a compartir contenidos, por lo que son sistemas ya existentes. Estarán distribuidos por distintas entidades productoras de contenidos educativos (universidades, empresas, colegios, etc). Se define el sistema con servidores dedicados a la distribución de estos contenidos que estén activos y conectados al sistema la mayoría del tiempo, lo cual se cumplirá por el propio interés de las entidades que ofrecen los contenidos.

  No se especifica una estructura P2P entre las distintas entidades que tuviera como objetivo compartir las tareas de descarga de objetos educativos, ya que estas entidades pueden ser competidoras y no es muy probable que servidores más potentes y con mejores conexiones cedan recursos a otras entidades competidoras que han invertido menos en recursos.

  Sin embargo, como hemos comentado en la descripción del funcionamiento, sí que se genera una estructura P2P –BitTorrent- entre el servidor de contenidos y los agentes que están descargándose un fichero grande. Se especifica que sólo se ofrezca al agente el documento .torrent cuando esté ya permitido el acceso a los contenidos (por ejemplo, cuando haya aceptado el pago por el contenido).

- **Los servidores de búsqueda semántica** (superpeer semánticos) Estos servidores permitirán buscar contenido semántico para alimentar las TBoxes y Aboxes de los agentes inteligentes de búsqueda. El espacio de identificadores posible se define de 128 bits como en la red Kad que usa el eMule.

  El sistema tendrá un conjunto de servidores dedicados a mantener la DHT semántica (al menos para dar una mínima capacidad), aunque puede que sean servidores virtuales ofrecidos por cada uno de los servidores físicos de contenidos, ya que en este caso sí que están todos interesados en mejorar la red de búsqueda de contenidos.

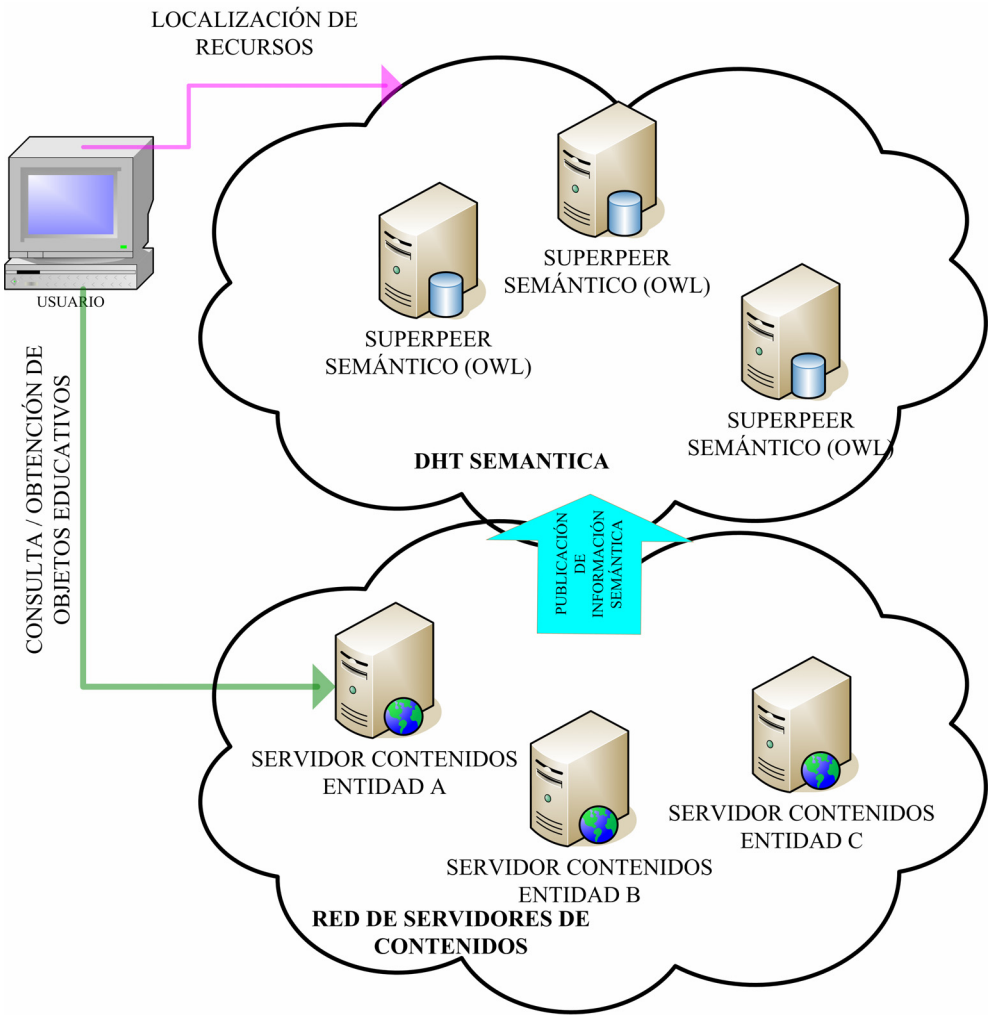


Fig. 5.5. Elementos del sistema

Como vemos en la figura 5.5, los servidores de contenidos, por medio de su publicador semántico, publicarán a la DHT Semántica su información, que buscarán los agentes de los usuarios para localizar objetos educativos en los que estén interesados. Cuando el usuario elige un objeto educativo, se contacta directamente con el servidor de contenidos que lo ofrece, para poder verlo con más detalle y descargarlo si así lo consideran ambas partes (el usuario consumidor y el servidor de contenidos).

5.5.4 Escalabilidad de la arquitectura

En este apartado definiremos los distintos mecanismos que defenderán a la arquitectura definida frente a los grandes aumentos de tamaño, tanto a nivel de información semántica que se guarda en la DHT Semántica como a nivel

de grandes demandas de descarga de un mismo archivo. En estos aspectos será donde veremos las ventajas de una arquitectura P2P cuando hay grandes variaciones de necesidad de recursos.

5.5.4.1 Escalabilidad respecto a la DHT Semántica

Como hemos comentado, la arquitectura se basa en unos superpeers semánticos dedicados que aseguran un mínimo de servicio en el arranque de la DHT Semántica.

La escalabilidad se ve comprometida por dos causas principales:

- **Aumento desmesurado de los contenidos a indexar en la DHT Semántica**. En este caso, para aumentar la potencia a la vez que se aumentan contenidos se define el comportamiento de los nodos de forma que cuando un nodo se encuentre saturado, obligue a alguno de los servidores de contenidos a los que indexa -cuyas direcciones tiene como valores de las claves que controla- a que entre a participar en la red de búsqueda con un servidor de búsqueda virtual. Si se niega, libera (borra) sus entradas y si ya es servidor de búsqueda busca el siguiente.

  Este nuevo nodo virtual para la DHT entrará con un valor de hash también virtual, de forma que se sitúe topográficamente al lado del nodo saturado y pueda así descargarlo. Es muy poco probable que aparezca un nodo con un valor de hash de su dirección igual al de nuestro nodo virtual, pero en ese caso el servidor virtual se daría de baja automáticamente a sí mismo.

  Es importante destacar que por cada fichero .owl que publiquemos, generaremos n entradas en la DHT Semántica. Sobre un estudio de más de 14.000 documentos RDF, se han encontrado una media de 65 aserciones por documento (una aserción tiene tres URIs, los dos recursos que relaciona y la propia relación), lo que nos da 65*3 URIs por documento [BID04], con lo que podemos suponer que la cantidad de entradas en la DHT Semántica unos dos órdenes de magnitud superior al número de documentos que indexa.

- **Aumento desmesurado de las peticiones de búsqueda**. Este caso se da cuando muchos agentes inteligentes atacan a la DHT Semántica. Para absorber los picos de peticiones, se define un mecanismo parecido al anterior, pero en dos fases, consistiendo la

primera fase en obligar a los servidores de contenidos a realizar funciones de servidores virtuales de búsqueda.

En una segunda fase, si sigue habiendo saturación, se obliga a los propios agentes de búsqueda que están realizando búsquedas a que cedan recursos a modo de nodo de búsqueda en la red, siendo estos nodos muy volátiles que desaparecerán al poco tiempo de realizar las búsquedas completas.

Para realizar de forma distribuida este segundo control de saturación, especificamos que cuando un Servidor de búsqueda detecte que está saturándose para servir todas las peticiones que le llegan, marque todas sus respuestas con una petición de colaboración (como si mandara un SOS con la respuesta).

Cuando un agente detecte una marca de petición de colaboración, ha de agregarse como servidor de búsqueda. El nodo que le ha servido la última búsqueda (por el que accede el agente de búsqueda) sólo aceptará de él una petición de unión a la red DHT Semántica hasta que se integre en la misma, no permitiéndole ninguna otra búsqueda hasta que sea un miembro de la DHT Semántica.

Algunos autores [TRA03] indican que la aparición de nuevos nodos, si son volátiles, cargan más la red que el beneficio que generan. Para evitar esto se le exige al agente a permanecer ofreciendo servicios de búsqueda durante una hora consecutiva.

Por lo tanto, el sistema de búsqueda semántica (la DHT Semántica) se protege ante los picos de demanda de recursos, basándose en algoritmos similares a los de otras redes P2P, en las que empiezan a colaborar primero los servidores de contenidos (al ser más estables y confiables) y en última instancia los propios nodos de los usuarios.

5.5.4.2 Escalabilidad respecto a la distribución de contenidos

En este apartado nos referimos a las herramientas que evitan la saturación del sistema a la hora de distribuir los contenidos educativos.

Según nuestra arquitectura, serán los propios servidores de contenidos los que decidan cuál de las dos maneras de distribución prefieren, en función del tamaño de la fuente:

- Para distribución de **ficheros pequeños**, se servirán de forma normal vía HTTP, por una sencilla arquitectura cliente-servidor

- Para **ficheros grandes**, para evitar una posible congestión si se piden el mismo fichero por parte de muchos agentes, se pasa a la distribución BitTorrent, generando una especie de subred entre el servidor de contenidos y los agentes como indica la figura 5.6, donde se ve con colores cada uno de las partes del archivo general y cómo fluyen esas partes entre algunos nodos que no son el servidor.

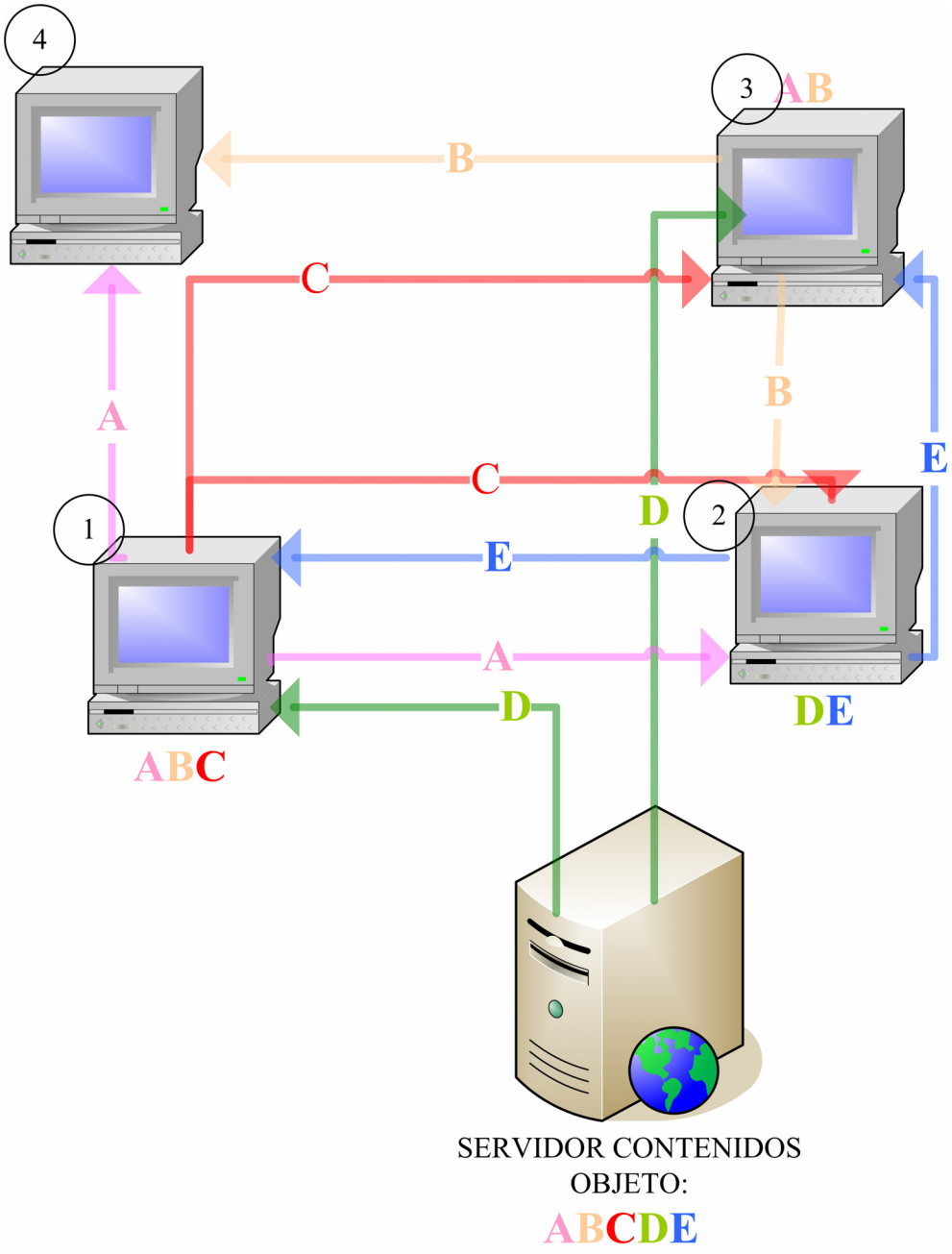


Fig. 5.6. Subred BitTorrent

En la figura 5.6 podemos ver como un objeto educativo de gran tamaño, se divide en paquetes (en este caso [A,B,C,D,E]). El servidor de contenidos va distribuyendo paquetes del objeto, y a su vez cada nodo que se descarga un paquete ha de estar dispuesto a ofrecerlo a otros nodos que lo necesiten. En

este caso, podemos ver como el nodo 1 contiene los paquetes [A,B,C], y en este instante está sirviendo el paquete A a los nodos 2 y 4 y el paquete C a los nodos 2 y 3. A su vez en este instante está recibiendo el paquete D del servidor de contenidos y el paquete E del nodo 2. Los nodos 2 y 3 están también ofreciendo los paquetes que contienen y recibiendo los paquetes que le faltan en este momento. El nodo 4 es un nodo recién incorporado que no tiene aún paquetes para ofrecer y está recibiendo el primer paquete. En el ejemplo queda patente que los paquetes no tienen por qué servirse y recibirse en orden y que es beneficioso que varios nodos reciban paquetes complementarios para luego poder compartirlos entre ellos y descargar al nodo servidor. Como vemos la figura 5.6, al tener cada nodo paquetes distintos los pueden compartir con mayor facilidad. Menos el nodo 4, que acaba de conectarse, todos los otros nodos están ya recibiendo todos los paquetes que les faltan, mientras ningún nodo –incluyendo al servidor- está sirviendo más de 4 paquetes.

Como indica el protocolo de BitTorrent, cuando un agente termina de descargarse un archivo ha de mantenerse en la red durante un tiempo para seguir colaborando en la distribución de contenidos. En nuestra arquitectura en particular, es el propio servidor del fichero (seed) el que ofrece el fichero .torrent para comenzar la descarga, no existiendo servidores que ofrecen ficheros .seed de forma exclusiva.

5.5.5 Aspectos de seguridad de la arquitectura

Como ya hemos visto uno de los puntos críticos de las arquitecturas P2P, es el de la seguridad, ya que no existe control sobre los nodos por los que pasan los datos, quién los pide realmente ni hay un sistema de control centralizado.

Los aspectos principales a tratar son:

- **Confidencialidad**, es decir, que toda la información que envíe el servidor de contenidos le llegue al agente de destino y sólo a él. Como no tenemos control sobre la red al ser Peer to Peer, la solución propuesta consiste en un sistema de cifrado de clave pública/privada, de forma que cuando el agente solicite el fichero incluya a su vez su clave pública para que sólo el agente destino pueda leer el contenido.

  Este procedimiento no es directo en el caso de BitTorrent, ya que el archivo va dirigido a muchos agentes. Para resolver este punto, se realizará un sistema de cifrado de clave pública/privada para cada

descarga de cada paquete de un nodo a otro, ya que cada paquete puede proveer de cualquier nodo del enjambre que lo tenga disponible. Para realizar esta acción, en el protocolo de solicitud de un paquete a uno de los nodos del enjambre de descarga, se enviará la clave pública del receptor. El emisor contestará con el paquete codificado con la clave pública del receptor, de modo que sólo podrá descifrarlo el receptor con su clave privada.

- **Autentificación**, de forma que sepamos que quien nos envía una información es realmente quien dice ser. Para resolver esto utilizaremos autentificaciones como las definidas en la arquitectura JXTA, es decir, un sistema de autentificación SSL o equivalentes (STL o IPSec).

- **Integridad de los datos**, resuelto por medio de una firma de los datos encriptados enviados, de forma que podamos conocer si son los datos originales o se han corrompido.

- **Autorización**, de forma que sólo los nodos que tienen en cada caso permitida una función la realicen. Para resolver esta situación nos basamos en el escaso interés de nodos ajenos a ceder sus recursos para estas redes, así como en el sistema de protección contra boicots del punto siguiente.

- **Protección contra boicots e usos indebidos**. Nos referimos en este apartado a la protección frente a nodos que sobrecargan la red o que envían mensajes falsos, de forma que consumen recursos de la misma para un fin distinto al proyectado. Para resolver este problema definimos un protocolo de lista negra, de forma que sólo los nodos fiables puedan publicar en la lista y que no se permita la entrada a agentes que estén en la lista negra. En este caso los nodos que pueden ser fiables serán todos los nodos que no sean agentes, ya que se les supone de mayor confianza.

  Para evitar la centralización de la lista negra, se define un sistema distribuido de confianza, de modo que cuando un nodo detecte a otro malicioso, se lo apunte en su lista y lo publique a sus nodos conectados directamente (en la DHT semántica se considera que los nodos a los que está un nodo conectado son los que tiene en sus buckets). Un nodo sólo aceptará información para actualizar su lista negra de nodos en los que confía (por ejemplo, nodos que han estado

funcionando bien durante bastante tiempo). Además, sólo marcará un nodo como malicioso cuando reciba más de una notificación proveniente de distintos nodos en los que se confía.

5.6 Conclusiones

En este capítulo se ha presentado una arquitectura que nos permite que los agentes inteligentes puedan buscar la información semántica necesaria para encontrar fuentes de objetos educativos, así como descargarlos de los servidores de contenidos correspondientes.

Un punto importante a destacar en esta arquitectura es que funciona sobre cualquier otro tipo de ontología, ya sean educativa o de otro tipo, sin modificar ninguna parte de la misma. Incluso puede dar servicio a la distribución de otros tipos de contenidos con metainformación en forma semántica. Además, la DHT Semántica que hemos definido en este capítulo nos permite realizar la publicación y explotación de contenidos semánticos por parte de agentes inteligentes de cualquier ontología, sea educativa o no, implementan una gigantesca base de conocimiento distribuida en la propia DHT Semántica, a la que consultan los agentes para localizar fragmentos OWL de la misma.

Al describir la arquitectura, se ha realizado a su vez una descripción funcional de la misma, indicando los mecanismos para conseguir gran escalabilidad y los aspectos relacionados con la seguridad necesarios para un funcionamiento correcto de una red P2P como la que planteamos en este capítulo.

Se ha mostrado también que, por la naturaleza de los contenidos a distribuir -normalmente con derechos de autor, de tamaño muy variable, y además, actualizables por los servidores de contenidos- es más aconsejable que se almacenan los objetos educativos y los ficheros OWL en los mismos servidores de contenidos, no realizando un almacenaje completamente distribuido en la red. Por tanto la arquitectura es Peer to Peer respecto a la localización de información semántica y respecto a la descarga de grandes ficheros por varios agentes, pero no en las tareas de almacenamiento de los objetos educativos.

Sin embargo hay una parte importante para que funcione correctamente una red P2P: el interés de los nodos en participar de forma interesada o desinteresada en la red. Es por esto que en caso de ser necesario se solicitan recursos a los servidores de contenidos, ya que son ellos los primeros

interesados en promocionar sus productos. En caso de no ser suficiente dicha colaboración, se solicitan recursos después a los agentes de búsqueda, puesto que ellos están interesados en el servicio que les ofrece la red.

.

Capítulo 6: Especificación de Ontología para Interoperabilidad

6.1 Introducción

A lo largo de este capítulo se definirá en profundidad y de forma rigurosa la ontología que permite la interoperabilidad de las experiencias educativas en mejores condiciones que mediante el uso de las ontologías actualmente existentes.

El objetivo de esta ontología es permitir al usuario un acceso eficaz, rápido, organizado, fácil y completo a las experiencias educativas disponibles, para obtener resultados ajustados, precisos y clasificados para ser incorporados y reutilizados con otros fines educativos distintos de los previstos inicialmente.

En este sentido, el proceso permite al usuario determinar mejor las búsquedas que desea realizar, y ajustar con más precisión los resultados que desea obtener, gracias a la utilización de criterios ontológicos específicos.

Además, el proceso se puede realizar de forma rápida ya que el uso de ontologías en la web semántica permite que sean los agentes inteligentes los que realicen las búsquedas y filtrados previos, para presentar al usuario los resultados más relevantes de forma automática.

Otra de las ventajas de este sistema es que permite el acceso a información de las experiencias educativas a las que hasta el momento no era posible acceder a través de las ontologías existentes. Nos referimos, por ejemplo, a la posibilidad de acceder a las interacciones entre los distintos actores de un proceso formativo como pueden ser los foros, lo que supone también un aspecto novedoso de esta especificación.

La ontología implementada permite también organizar los resultados de las búsquedas realizadas, de forma que se pueden agrupar las soluciones bajo una serie de criterios específicos predeterminados que el usuario puede elegir. Por ejemplo, si se realiza una búsqueda de objetos educativos que traten sobre la temática 'computación paralela' los agentes podrán presentar resultados agrupados en cursos específicos de computación paralela, cursos en los que los usuarios realizan comentarios sobre computación paralela, y cursos en los que se realizan preguntas sobre computación paralela, entre otros. Esta organización de los resultados facilita la obtención de una visión clara del material disponible, lo que permite una mejor y más rápida planificación del trabajo a realizar.

Asimismo, esta ontología se caracteriza porque la intermediación de agentes inteligentes en búsquedas muy específicas reduce la posibilidad de olvidar una fuente, ya que los agentes realizan búsquedas más exhaustivas que las que pueda realizar un usuario, lo que evita que el usuario tenga que definir cada una de las fuentes en las que está interesado. Puede ocurrir que el usuario no estuviera inicialmente interesado en obtener información sobre una experiencia educativa en concreto o no hubiera evaluado correctamente su posible utilidad, pero al obtener un resultado sobre ésta descubra que también puede servir para sus objetivos. De esta forma, la ontología propuesta aprovecha las posibilidades de la Web Semántica, ya que está basada en un leguaje OWL, que se caracteriza por su idoneidad este contexto.

La implementación de esta ontología está basada en estándares y modelos de referencia que gozan de una amplia aceptación, como son los desarrollados por IEEE, ADL e IMS. Esta característica facilita el uso de la ontología a usuarios familiarizados con el uso de estos sistemas, pues permite ganar rapidez en las búsquedas al utilizar conceptos ya conocidos. A su vez, facilita la generación de metainformación para ser publicada, siguiendo la ontología propuesta en esta Tesis.

Para la especificación de la ontología se seguirán las pautas metodológicas indicadas en [NOY01][REC05]. De esta manera, se enumeran los distintos conceptos que son objeto de estudio, luego se agrupan por similitud y utilidad y, por último se realiza una especificación formal de los mismos que permita traducción directa a un lenguaje ontológico, como puede ser OWL.

Fruto de esta metodología, presentaremos primero una ontología general para Teleeducación, que se dividirá en tres ontologías diferenciadas. La primera de las ontologías tratará la conceptualización de los contenidos estáticos. La segunda de las ontologías versará sobre las interacciones del alumno con el sistema, por medio de las evaluaciones. La tercera y más novedosa ontología se centrará en las interacciones entre los usuarios (basado en una visión constructivista del aprendizaje), escogiendo en este caso los foros, ya que son, junto con el correo electrónico, uno de los elementos más utilizados para estas interacciones. Los foros son, además, un campo no tratado hasta el momento dentro de las ontologías para la interoperabilidad.

La posibilidad de compartir la información introducida a través de la interacción de los usuarios es interesante y útil para generar una base de conocimiento que podríamos aprovechar en otras imparticiones del curso. De esta forma podemos ir enriqueciendo nuestro repositorio de experiencias educativas con las impresiones descritas en los foros. Esto nos permitirá ampliar las búsquedas, ya que se podrá buscar entre todas las interacciones de los foros, para dar con una información específica o para elegir qué objeto educativo reutilizar.

El procedimiento que se va a seguir para realizar la especificación, una vez identificadas las tres ontologías, es el siguiente: en primer lugar se analizarán los conceptos de cada una de las tres ontologías Una vez se han definido los conceptos y antes de pasar a especificar las ontologías, se ha

considerado conveniente introducir un elemento auxiliar que permita especificar las estructuras en árbol y que se utilizará en las especificaciones siguientes. A continuación se realizará una especificación formal para cada una de las ontologías. Finalmente, se especificarán los nexos de unión entre las tres ontologías descritas.

6.2 Visión General

La especificación general la separaremos en tres ontologías (ver Fig. 6.1):

- **Ontología para los contenidos estáticos**, donde se indique su estructura.

- **Ontología de la de la interacción del alumno con alguno de los contenidos**. Nos referimos en este punto a las evaluaciones, sean del tipo que sean.

- **Ontología de descripción de las interacciones entre usuarios**. Esta ontología será la que proporcione la visión constructivista a nuestra ontología general.

Desarrollaremos las tres ontologías en este capítulo, realizando primero las definiciones de los conceptos de cada una de ellas y luego realizando la especificación formal con el lenguaje de lógica descriptiva que nos servirá para poder plasmarla en OWL.

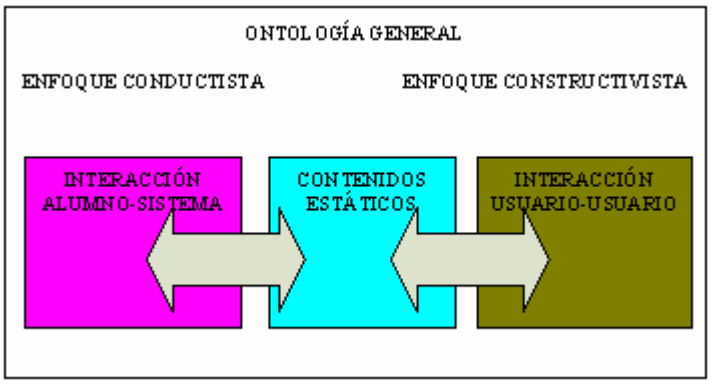


Fig. 6.1. Descripción de la ontología General

Como se indica en la figura 6.1, las tres ontologías que forman la ontología general se relacionan entre sí, utilizando la ontología de contenidos estáticos como puente para conectar las interacciones alumnos-sistema y las

interacciones entre los usuarios. También se indica en la figura que las interacciones alumno-sistema serán las que apoyen un aprendizaje según las teorías conductistas, ya que premian un comportamiento del alumno. El comportamiento que se premiará será en este caso la realización de una prueba o test, que servirá para determinar si han alcanzado los objetivos de aprendizaje. Por otro lado, se muestra que las interacciones entre usuarios pretenden apoyar el aprendizaje por medio de un enfoque constructivista, en el que por medio de la relación de un alumno con otros miembros de la experiencia educativa se produce también un aprendizaje. Por último, aunque no siempre será así y dependerá de los contenidos en sí, podemos considerar que los contenidos estáticos pretenden fomentar el aprendizaje por medio de definiciones de conceptos y relaciones entre los mismos, por lo que es esta ontología la que permite el aprendizaje por medio de teorías cognoscitivistas. En esta Tesis se considera que el aprendizaje se produce por medio de las tres formas anteriormente indicadas, y es por esto que se desarrollan las tres ontologías agrupándolas en una general.

6.2.1 Ontología para los contenidos estáticos

Entendemos como contenidos estáticos a los contenidos que han sido creados antes de la fase de impartición del curso y no generan información sobre la adquisición de conocimientos por parte del alumnos más allá de la comunicación al sistema de la navegación del discente por los contenidos. Incluye desde las páginas html donde se describen los apuntes del curso hasta los documentos multimedia donde al alumno puede observar ejemplos reales (por ejemplo, por medio de vídeos) o representaciones. Todos estos contenidos se podrían almacenar directamente en un CD y enviarlos en este formato a los potenciales alumnos.

La ontología de contenidos estáticos es un tipo de ontología muy desarrollada por gran cantidad de autores, siendo las más aceptadas y extendidas IMS Content Packaging Specification [IMS04] y SCORM Content Aggregation Model (basado a su vez en IMS CP) [ADL04a]. De hecho, la mayoría de plataformas (como Moodle) aceptan paquetes SCORM o IMS. Algunas de las herramientas de autor y sistemas propietarios están dirigiendo sus esfuerzos para poder trabajar con alguno de los dos estándares a nivel de contenidos estáticos [BAR04], [ROM06]. De hecho, las ontologías arriba mencionadas son ontologías implícitas, entendiendo implícitas como ontologías no plasmadas en un lenguaje ontológico para la web como OWL, por lo que dicha ontología se ha de programar previamente en los distintos agentes que utilizan los datos. También existen actualmente otros desarrollos de ontologías de contenidos estáticos ya

implementados en lenguaje OWL para sus metadatos [SAN05][GUO06][ULL04]. Sin embargo, los desarrollos actuales de ontologías sobre contenidos estáticos son desarrollos en los que se abarcan conceptos parciales, como puede ser la implementación parcial de IEEE-LOM de [SAN05] o la enumeración de distintos tipos de actividades instruccionales de [ULL04]. La ontología en OWL sobre educación más completa sobre los contenidos estáticos la encontramos en [GU06], donde se distingue entre la ontología de la materia que se trata, la ontología sobre el contexto o tipo de actividad instruccional y por último la ontología sobre la estructura de los contenidos, pero sin seguir en ningún caso las pautas marcadas por IMS CP o SCORM.

Si bien existen ejemplos de traducciones parciales de SCORM en OWL (usando OWL-S[ARO04] o una versión OWL del lenguaje DyLOG[BAL04]), se ha preferido realizar una implementación propia para simplificar el modelo debido que no trataremos la ontología para definir la secuenciación y navegabilidad de los contenidos. La razón por la que no se incluyen estos aspectos en la ontología es la poca utilidad que puede aportar la inclusión de esta información para realizar búsquedas de recursos educativos que podamos compartir. Los aspectos de navegabilidad y secuenciación sí son tratados en las ontologías comentadas anteriormente[ARO04], [BAL04], siendo de hecho la parte de SCORM implementada. Tanto en [ARO04] como en [BAL04] se conceptualiza cada actividad de aprendizaje como un proceso de servicio web. Dicho proceso es compuesto y descrito por algún lenguaje descriptivo de procesos como puede ser OWL-S, que nos indica los distintos subprocesos de aprendizaje por donde va pasando la actividad de aprendizaje global.

Por estas razones se decide en esta Tesis generar una ontología nueva para los contenidos estáticos basada en estándares e implementaciones de referencia globalmente reconocidos. En el desarrollo que especificaremos nos basaremos en el modelo de referencia SCORM. Para definir la ontología basada en SCORM podríamos habernos basado en la versión XML de SCORM tal como aparece en el modelo de referencia y transformar los objetos de XML en clases de OWL, de forma que nuestra ontología fuera una simple enumeración de clases y subclases sin restricciones definidas explícitamente. Sin embargo, en esta Tesis se opta por ampliar más la ontología por medio de la introducción de las restricciones principales del modelo de referencia en la descripción OWL. Cuanto más relaciones entre instancias y axiomas traslademos a nuestro lenguaje ontológico, más útil va a ser ya que son estas restricciones las que van a entender los agentes

inteligentes cuando realicen búsquedas. Por ejemplo, si por medio de relaciones y axiomas somos capaces de describir una estructura en árbol entre los elementos, podremos realizar búsquedas en las que indiquemos que nos devuelva el objeto raíz de un árbol que contenga otro elemento en su interior que cumpla unos requisitos que le impongamos. Como podemos apreciar, son estos tipos de relaciones y axiomas los que permiten realizar búsquedas más inteligentes que una simple búsqueda en documentos XML, y por tanto uno de los puntos más novedosos del uso de lenguajes ontológicos. No tendría sentido utilizar lenguajes ontológicos sólo para enumerar objetos y características de los mismos que podría realizar con lenguajes de más bajo nivel como puedan ser XML o RDF.

Para comenzar la definición de nuestra ontología abstraeremos los contenidos estáticos en la siguiente estructura:

- **Learning Content Package** . Es el elemento que abstrae la información relativa a un paquete exportable SCORM. Podemos decir que la clase equivale a la abstracción del la etiqueta <*imscp:manifest*> de la representación XML de SCORM.

- **Organization**. En cada instancia de la clase *Learning Content Package* existe al menos una instancia de la clase *Organization*, que indica las diferentes organizaciones de los contenidos de paquete según el contexto al que se dediquen. De este modo, existirá una relación de un objeto *Learning Content Package* con un objeto *Organization* por cada forma distinta de presentar los contenidos. Por ejemplo, un paquete de un curso sobre matemáticas puede tener todos sus recursos en una organización que formación integral y sólo los avanzados en una formación integral pero para estudiantes con conocimientos en la materia. Su equivalencia con la etiqueta XML <*imscp:organization*> de SCORM es directa. A partir de él aparece una estructura jerárquica de árbol.

- **Learning Activity**. Es el equivalente a una unidad de aprendizaje, y permite una división en otras instancias de la clase *Learning Activity* a n niveles. La etiqueta XML en la que se representa este concepto en SCORM es <*imscp:item*>. Este concepto se basa en la suposición que indica que los contenidos educacionales on-line, estén generados con la herramienta que sea, pueden representarse como un árbol de contenidos de n niveles. Cada sistema tiene un conjunto distinto de niveles. En la Tabla 6.1 podemos ver ejemplos.

Canadian Armed Forces	US Marine Corps	US Army	Buddha
Course	Course	Course	Minicurso
Performance Objective	Phase	Module	Unidad de Aprendizaje
Enabling Objective	SubCourse (Annex)	Lesson	Segmento
Teaching Point	Lesson	Learning Objective	Pantágina
	Task	Learning Step	Contenido
	Learning Objective		Recurso/Ampliación
	Learning Step		

Tabla 6.1. Ejemplo de estructuras de contenidos de aprendizaje, con distintos niveles [ADL04a]

Como indica la Tabla 6.1, los distintas estructuras de aprendizaje de los distintas definiciones de contenidos de teleformación se pueden dividir en elementos repartidos en n niveles de una estructura en árbol. Por ejemplo, la formación de los cuerpos de Marina de Estados Unidos utiliza contenidos estructurados en una estructura de árbol a 7 niveles.

SCORM define una estructura de árbol sin fijar el número máximo de niveles, pero distingue entre las actividades que contienen otras actividades de aprendizaje (llamadas actividades de tipo *Cluster*) y las actividades de aprendizaje que no contienen a otras actividades (denominadas como actividades de tipo *Leaf Activity*). Una de las restricciones que plantea SCORM es que sólo las actividades *Leaf Activity* puedan contener recursos educativos, mientras que las actividades *Cluster* sólo sirven a nivel lógico para realizar una organización de los contenidos. Esta restricción es necesaria en SCORM para una correcta representación de los contenidos en orden en el Sequence & Navigation Model, y se respetará en esta ontología pese a no tratar la navegabilidad para mantener la mayor compatibilidad entre SCORM y nuestra ontología.

- **Resource** (recurso) conceptualiza el archivo o archivos en los que se puede dividir una experiencia educativa. En este caso representaremos nuestro recurso por medio de la URI que lo

identifica de forma única. SCORM diferencia entre objetos *Resource* de tipo *Assets* y *SCOs*.

Un objeto *Asset* representa el recurso más básico. Son la forma más simple de recurso, y representa cualquier documento (texto, imagen, sonido o cualquier otro tipo de dato) que puede ser representado directamente por un cliente Web.

Por otro lado, un objeto *Sharable Content Object (SCO)* conceptualiza también un recurso pero que realiza cierta comunicación entre él mismo y el LMS, en función de la interacción del usuario.

Como resumen de los descrito hasta este punto, hemos abstraído en cuatro clases principales y cuatro subclases la ontología de contenidos estáticos. Las denominaciones y definiciones se han realizado para que sean lo más similares al modelo de referencia SCORM, para permitir mayor compatibilidad y facilitar la extracción de la ontología OWL de paquetes SCORM ya existentes.

Como se ha descrito en este apartado, la estructura de los contenidos se basa en una estructura en árbol. Al no existir predefinida la estructura de árbol en OWL, se definirá formalmente la abstracción de una estructura en árbol para aprovecharla en esta y otras ontologías.

6.2.2 Ontología para las interacciones alumno-sistema

En este apartado trataremos de la ontología para las interacciones alumno-sistema, que conocemos como evaluaciones. Nos volveremos a basar a su vez en el modelo propuesto por SCORM, en el que hay ciertas partes (denominadas SCOs) que interactúan con el sistema de gestión del aprendizaje (LMS) para hacer un cierto seguimiento de los avances formativos del alumno. Si bien existen algunas ontologías desarrolladas en OWL [KIM05][ULL04] que incluyen en su ontología las evaluaciones, ninguna de las desarrolladas en este lenguaje ontológico para la Web se basa en IMSQTI, que es el estándar más aceptado para implementar la interoperabilidad de las evaluaciones.

Para definir la ontología de las evaluaciones nos basaremos en el modelo de referencia SCORM[ADL04c] y en IMSQTI [QTI06], de forma que conceptualizaremos en clases los modelos de referencia. Como se realiza en SCORM y en IMSQTI, separaremos la ontología de resultados obtenidos

(basada principalmente en SCORM) de la ontología de descripción de las actividades de evaluación (que basaremos en IMSQTI). La relación entre las dos ontologías la podemos ver en la Fig. 6.2.

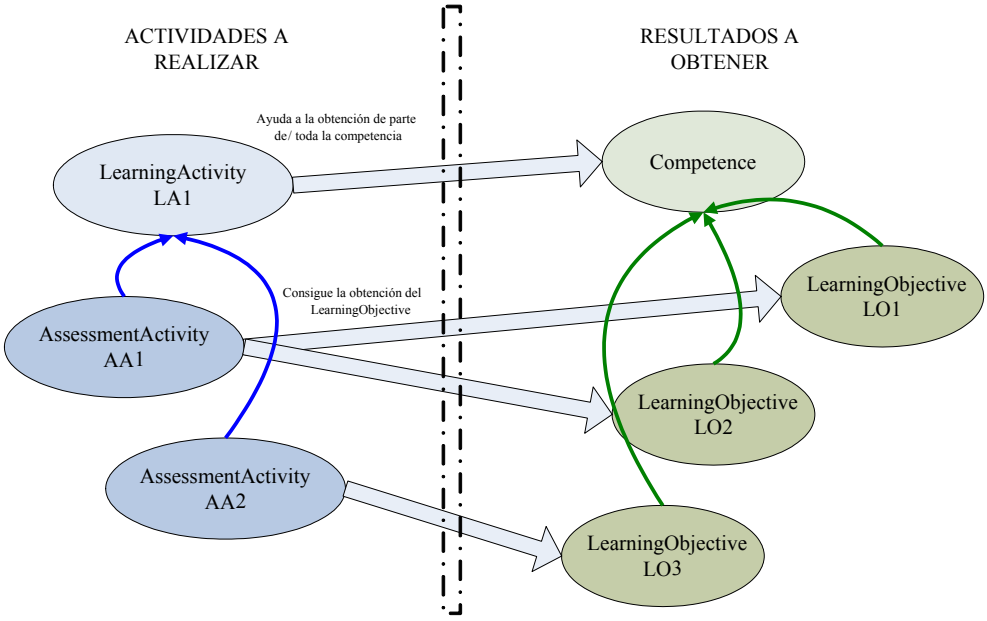


Fig. 6.2. Relación entre las ontologías de descripción de actividades y de resultados

Como se describe en la figura 6.2, tenemos por una parte unas actividades que sirven para evaluar al alumno, que en caso de ser resueltas de forma satisfactoria por parte del alumno evidenciarán que el alumno ha alcanzado un objetivo de aprendizaje determinado. Si el alumno supera un conjunto definido de actividades de evaluación, que a su vez evidencia una obtención de un conjunto de objetivos de aprendizaje, podremos concluir que el alumno ha obtenido una competencia determinada.

La descripción en detalle de las actividades de evaluación, que representaremos por medio de la clase *AssessmentActiviy*, se basará en la definición del modelo de información de IMS QTI [IMSQTI] y en parte en los tipos definidos por IEEE.1484.11.1 [IEEE05]. La parte de descripción de los distintos tipos será expresada en OWL aunque no se pretende que sirva para describir completamente el comportamiento deseado y el procesado de las respuestas (que necesitaría un leguaje de descripción de procesos como OWL-S). La razón por la que no se profundiza en el comportamiento deseado y el procesado de las respuestas se debe a que

nuestro interés para lo ontología no es la ejecución del curso, sino facilitar su búsqueda e indexación en la web semántica.

La parte de consecución de objetivos se basa en que una actividad puede servir para evaluar si se han adquirido ciertos objetivos de aprendizaje, como define SCORM [ADL04c] [ADL04b]. La actividad en sí es suficiente para evaluar ese objetivo totalmente, y puede que haya otras actividades que también evalúen la consecución de ese objetivo de aprendizaje. Un objetivo se consigue de forma total, es decir, no hay distintos grados de consecución de los objetivos de aprendizaje.

Aunque el modelo de referencia SCORM permite asignar elementos *LearnigObjective* a elementos *Asset* (de forma que se cumple cuando un alumno ha visualizado dicho elemento *Asset*), en esta Tesis consideraremos que no se ha obtenido un objetivo de aprendizaje hasta que no se ha evaluado de alguna forma. La razón por la que se realiza esta distinción se justifica porque, por elemplo, el hecho de haber descargado un fichero no significa ni tan siquiera que se haya leído, por lo que no se considera adecuado poder obtener un objetivo de aprendizaje únicamente por descargarse un recurso. El modelo SCORM realiza esta concesión porque utiliza el concepto de objetivo de aprendizaje para describir la navegabilidad, mezclando ambos propósitos.

Por tanto, se considerará el objetivo de aprendizaje, representado por la clase *LearningObjective*, como un objetivo de aprendizaje indivisible y que se puede constatar que se ha obtenido por medio del elemento de evaluación al que pertenece. Además, puede definir la temática sobre la que se desarrolla la instancia de la clase *LearningObjective* así como la clasificación de tipo de objeto *LearningObjective* obtenido. Esta definición de temática y clase de objetivo de aprendizaje se especifica siguiendo el estándar IEEE-LOM-1484.12.1, es decir, con los atributos *catalog* y *entry*. El atributo *catalog* indica sobre qué catálogo se clasifica y el atributo *entry* el valor para identificarlo dentro de la clasificación. Ambos atributos son propiedades que relacionan nuestro objeto *LearningObjective* con una cadena de texto.

En la especificación realizada en este capítulo no se diferenciará entre objetivos locales y globales, como sí hace SCORM S&N [ADL04c], pero sí entre el concepto de *LearningObjective* y el concepto de *Competence*. La clase *Competence* representa un nivel más complejo de resultado adquirido y se compone de un conjunto de instancias de *LearningObjective*, aunque

puede necesitar más pruebas a parte de la obtención de los objetivos de aprendizaje para considerar que se ha adquirido la competencia. Se define para poder marcar los contenidos estáticos como facilitadores para obtener una cierta competencia. En este caso los contenidos estáticos pueden otorgar la competencia por completo o necesitar otros elementos para evidenciar la consecución de la misma. Las instancias de la clase *Competence* contendrá las mismas propiedades de datos que la especificación IMS Reusable Definition of Competency or Educational Objective Specification (y su normalización en IEEE 1484.20.1), es decir, título, definición y metadatos, pero no desarrollará la propiedad definición por medio de frases (elementos *statements* de IEEE 1484.20.1), sino se definirá por medio de unas nuevas propiedades *catalog* y *entry*.

Permitiremos que un objeto *LearningObjective* pueda ser compartido por varios objetos *Competence*, de forma que podremos relacionar distintos objetos *Competence* por medio de las instancias de la clase *LearningObjective* que contienen. Otra forma de relacionar elementos *Competence* se realiza por medio de la propiedad *catalog*, de forma que dos objetos *Competence* que pertenecen al mismo catálogo de competencias están de algún modo relacionados. No se aborda una relación directa entre competencias para simplificar el modelo y evitar relaciones complejas entre las mismas. La razón por la que no se relacionan las instancias de la clase *Competence* entre sí de forma directa se debe a que no se puede asegurar que las competencias tengan algún tipo de estructura detallada, como una jerarquía en forma de árbol.

Por ejemplo, en la Fig. 6.3 podemos apreciar dos competencias que comparten el mismo objeto educativo.

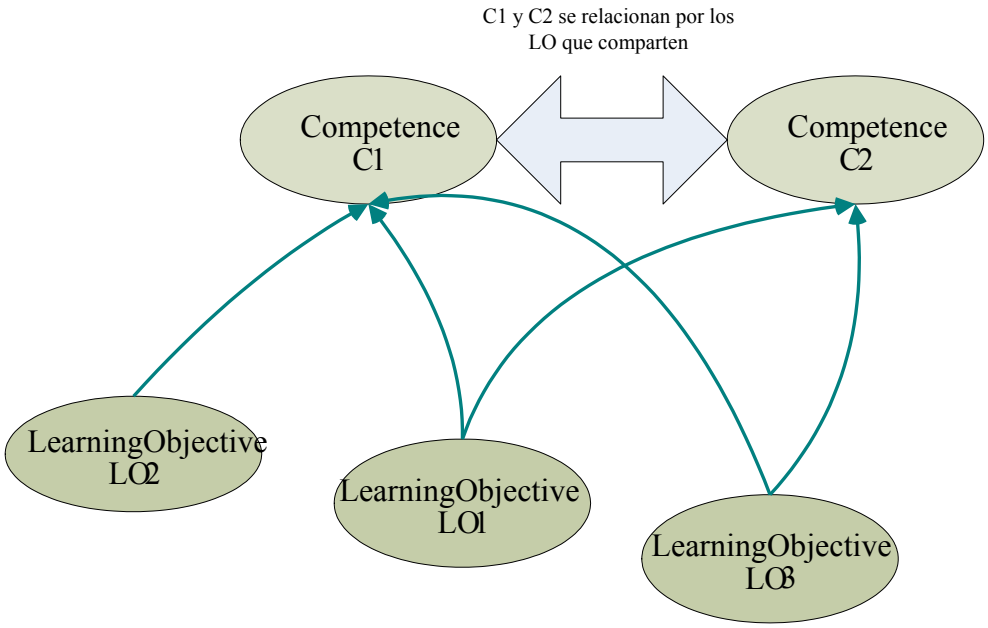


Fig. 6.3. Relación entre objetos *Competence*

De la figura 6.3 podemos inferir también que si ambas competencias contienen sólo los objetos *LearningObjective* de la figura, la instancia de *Compentence* C1 contiene a la instancia C2.

6.2.3 Ontología para las interacciones entre usuarios

Esta ontología es la menos desarrollada y contemplada por los modelos de referencia, normas y recomendaciones sobre la formación on-line. Algunos modelos la contemplan como simple medio a la hora de la impartición, pero en ningún caso se trata la posibilidad de potenciar la interoperabilidad de estas interacciones. En los Sistemas de Gestión de la Formación (LMSs) actuales se contemplan estas interacciones como un servicio añadido a la experiencia educativa. Dicho servicio y la información que ha recibido normalmente desaparece cuando ya se ha impartido dicha experiencia. Sin embargo, se propone desde esta Tesis la recolección y tratamiento de estas interacciones para poder realizar búsquedas semánticas dentro de las mismas, aunque ya haya terminado la acción formativa para la que se crearon. Por tanto podemos considerar que la ontología especificada en esta Tesis es pionera en el campo de las interacciones entre usuarios en un proceso de teleformación.

El modelo de referencia SCORM [ADL04b] contempla de forma muy limitada y unidireccional la posible comunicación en el sentido aprendiz-

instructor por medio de los elementos *cmi.comments_from_learner*, permitiendo que los elementos *SCO* almacenen (aunque se profundiza más en este tema dentro de la especificación del modelo de referencia) de alguna forma un conjunto n de comentarios (al menos se debe poder almacenar n=250 por SCO), basados en un texto (almacenado en el elemento *cmi.comments_from_learner.n.comment*, que ha de soportar al menos strings de 4000 caracteres), una fecha de creación (*cmi.comments_from_learner.n.timestamp*) y un indicador de a qué parte del elemento *SCO* se refiere el comentario (elemento *cmi.comments_from_learner.n.location*, con una estructura que no está definida y depende de cada implementación del SCO). Se define de forma similar los comentarios del instructor a los alumnos (comunicación uno a muchos) por medio del elemento *cmi.comments_from_lms* (en este caso permitiendo un máximo de n=100). En la figura 6.4 podemos ver los dos tipos de comentarios y los sentidos de la comunicación.

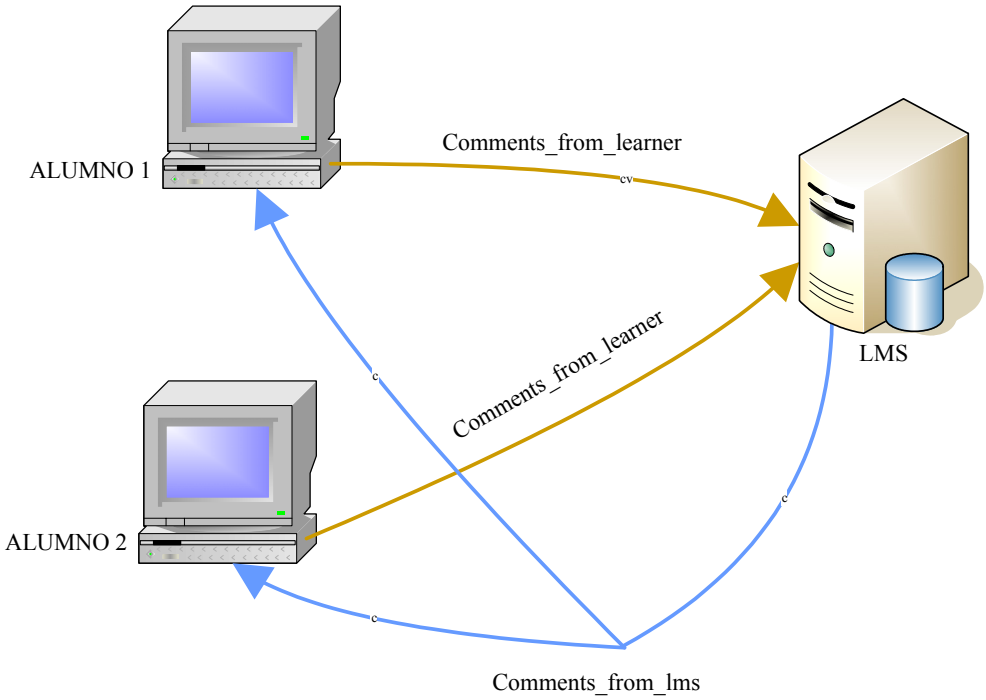


Fig. 6.4. Esquema de *comments_from_learner* y *comments_from_lms*

En la figura 6.4 se describen los nombres de los elementos que participan en este sistema. Además podemos apreciar los distintos sentidos de la comunicaciones, y que no existe ningún tipo de relación entre los distintos comentarios, más allá de pertenecer al mismo elementos *SCO*.

IEEE.1484.11.1 introduce los mismos elementos para las interacciones entre usuarios, ya que, como el modelo de referencia SCORM, IEEE.1484.11.1 se basa en AICC Computed Managed Instruction [AICC04]. Incluso mantiene los mismos valores para los tamaños mínimos y los campos de comentario, fecha y localización. La única diferencia radica en que en este caso no se utiliza el prefijo *cmi*., aunque sí que se indica en la descripción que pretende definir sin ambigüedades la comunicación definida por AICC CMI.

En ninguno de los casos se relacionan los distintos comentarios entre sí (lo que daría lugar a conversaciones) ni se permite los comentarios entre los propios aprendices. La utilidad de esta comunicación es simplemente generar un feedback desestructurado sobre posibles errores tipográficos encontrados y no permite profundizar más, por medio de una conversación sobre las posibles mejoras o temas interesantes no cubierto completamente por los contenidos estáticos del curso.

Podemos ver un esquema común de las propuestas de IEEE.1484.11.1 y SCORM en la Fig. 6.5.

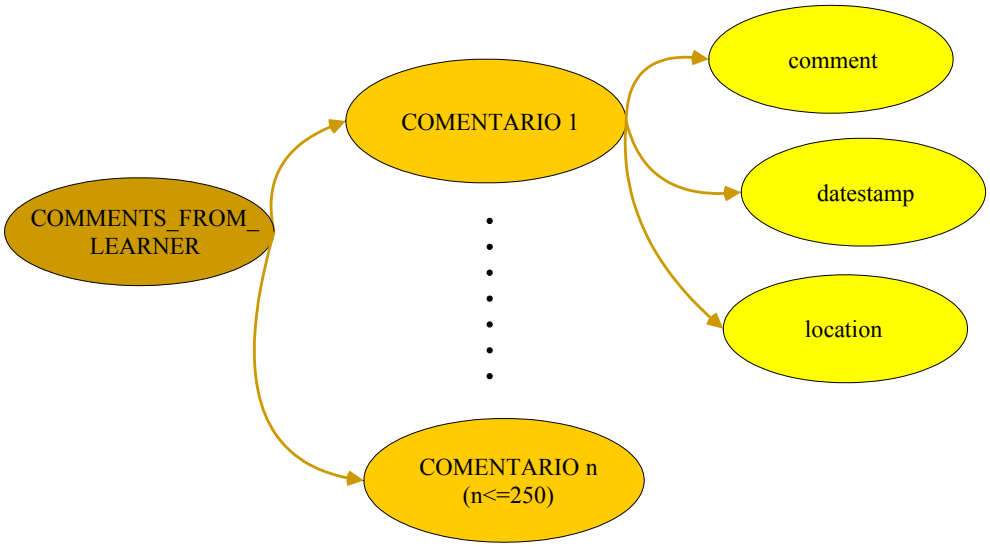


Fig. 6.5. Esquema común de los comentarios en IEEE y en SCORM

En la figura 6.5 podemos ver la conceptualización de las dos especificaciones, de IEEE y de SCORM, de modo que se abstraen los conceptos comunes, que nos servirá para la implementación de nuestra ontología.

Nuestra especificación pretende basarse en estos modelos pero ir más allá de simples comentarios. Para hacer esto definiremos la interacción constructivista (representada por la clase *ConstructivistInteraction*) como la abstracción de las posibles interacciones que puede hacer un participante de la experiencia educativa con otros participantes (sean alumnos, instructories, tutores u otras figuras). El concepto de interacción en este apartado, es completamente distinto a la mayoría de sistemas de aprendizaje, donde se asimila interacción a una acción del alumno con el sistema a modo de contestación de un test (como se hace [AICC04]).

En nuestro sistema aseveramos que las interacciones se pueden hacer por medio de comentarios (con una estructura similar a la de IEEE-1484.11.1) o por medio documentos compartidos, ya que este método es otro método que permite la interacción entre los distintos usuarios.

Además conceptualizaremos los recursos constructivistas (que pertenecerán a la clase *ConstructivistWidget*) refiriéndonos a los elementos que nos sirven en la formación on-line a compartir experiencias con otros participantes, como pueden ser Foros, Listas de Correo (entendidas como Foros en los que responder a un correo es responder a ese comentario y enviar un correo nuevo es como generar un nuevo Thread del Foro), una actividad de generación y modificación de un recurso en grupo (como puede ser por medio de la herramienta Wiki [WWW51]). También se incluyen los sistemas de preguntas frecuentes, que se generan a partir de las interacciones de los usuarios que plantean los mismos tipos de dudas.

Esta especificación permitirá cubrir la mayoría de las posibilidades constructivistas, profundizando en la de tipo foro y expresar cómo se podrían añadir otras, de modo que la ontología sea ampliable con nuevas técnicas.

En la figura 6.6 podemos ver un resumen de esta parte de la ontología.

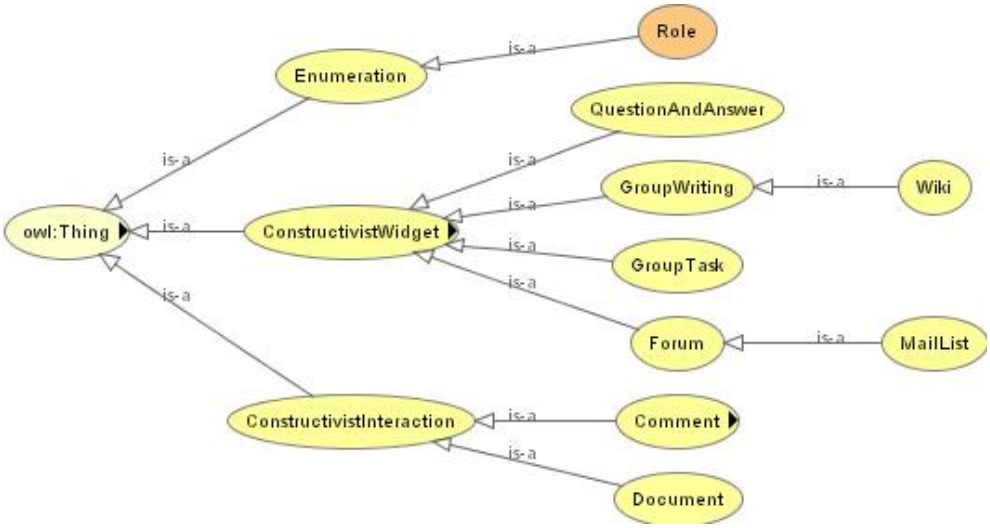


Fig. 6.6. Generalización de la ontología de interacción entre usuarios

Como podemos ver en la figura 6.6, definimos las interacciones constructivistas y los distintos recursos constructivistas, realizando una taxonomía de los mismos.

Para definir los foros empezaremos definiendo con un poco más de detalle los comentarios (instancias de la clase *Comment*), definiendo preguntas (instancias de *Question*), respuestas a esas preguntas (representadas por la clase *Answer*) y comentarios anidados en forma de árbol (clase *TreeComment*) que nos servirán para los foros. También describiremos características de los Threads (abstraidos en la clase *ForumThread*) y de los foros (elementos de la clase *Forum*).

También definiremos los roles de los autores. La razón por la que no se identifica al autor con un autor individual, es para proteger el anonimato de los autores de los foros, ya que esa información pretendemos que sobrepase el contexto de la impartición del curso y nos sirva para búsquedas y para añadirlo a otras experiencias educativas. Por esta razón se definen los distintos roles de la experiencia educativa (profesor, alumno, tutor, etc) que permitirán marcar de alguna forma el origen de las distintas interacciones. De esta forma se podrán realizar búsquedas de recursos educativos con un texto en alguno de los comentarios de un tutor.

6.3 Especificación de una estructura de árbol

En este apartado desarrollaremos la lógica de una estructura de árbol, ya que nos servirá para las partes de nuestra ontología que se basen en una estructura de este tipo. OWL no implementa directamente este tipo de estructuras, por lo que tendremos que definir su lógica inherente.

En nuestro caso haremos una clase *TreeStructure* que sea subclase de otra más genérica llamada *Structure*. Después definiremos una clase *TreeItem* que abstraerá todo objeto de una estructura en árbol. Las clases *TreeStructure* y *TreeItem* se relacionarán por medio de la propiedad *hasFirstTreeItem* (propiedad funcional) y su inversa *isFirstTreeItemOf*

Indicaremos la clase *TreeItem* como un conjunto de elementos pertenecientes a la unión de dos clases disjuntas: *ClusterItem* (elemento que aún tendrá otros elementos por debajo de él en su rama del árbol) y la clase *LeafItem* (elemento terminal de árbol).

$$TreeItem \subseteq (ClusterItem \cup LeafItem) \qquad \text{[Expr. 6.1]}$$
$$ClusterItem \subseteq \neg LeafItem$$

Definimos a su vez una propiedad de objeto (relación) *hasChild* (y su inversa *isChildOf*) que relaciona un objeto *TreeItem* con otro objeto *TreeItem*, siendo la misma propiedad inversa-funcional (es decir, que un elemento *TreeItem* puede tener como máximo un padre). Además definimos otra propiedad de objeto para relacionar una instancia de la clase *TreeStructure* con su primer elemento por medio de la propiedad funcional (una estructura de árbol sólo puede tener un primer elemento) *hasFirstElement* (y su propiedad inversa *isFirstElementOf*). Como vemos en la figura 6.7, cualquier instancia de la clase *TreeItem* puede ser el primer elemento de una estructura de árbol, pero en todo árbol sólo existe una estructura de árbol que contiene a todas las demás (árbol principal). Además, por cada estructura de árbol sólo puede un elemento raíz *TreeItem.*

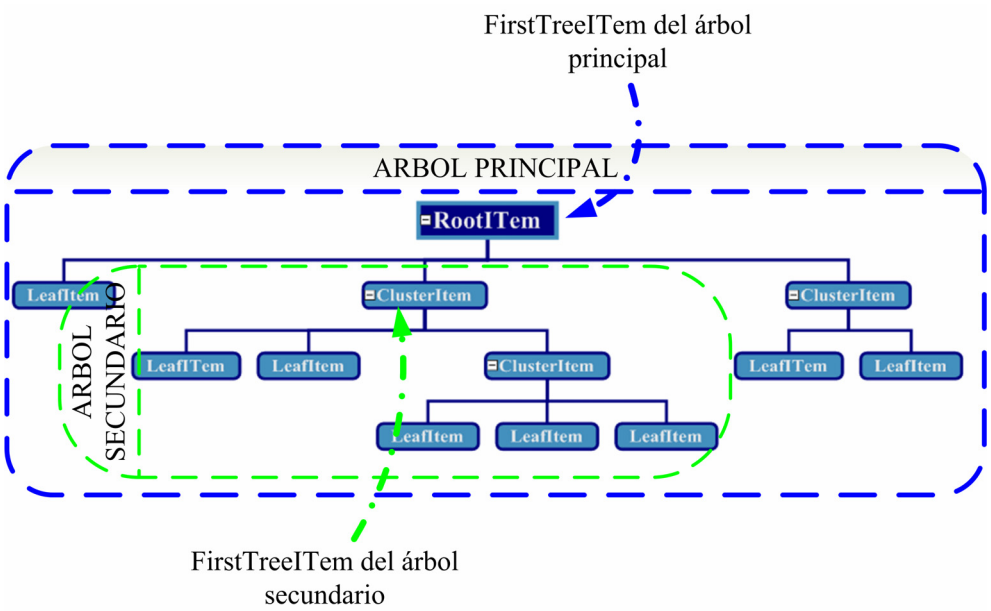


Fig. 6.7. Árboles y subárboles

En la figura 6.7 podemos observar que existe sólo un árbol principal. Por otro lado, una estructura de árbol puede tener tantos árboles secundarios como se desee.

Definiremos una propiedad de datos *isLeafItem* que nos relacionará los objetos *TreeItems* con un booleano (de valores *true*/*false*) por razones que veremos a continuación.

Además incluiremos las propiedades transitivas *hasDescendent* y *isDescendetOf*, que nos serán de utilidad para los foros y para la ubicación de un elemento de la clase *TreeItem* dentro de una instancia de la clase *TreeStructure*.

Definiremos también la clase *ClusterItem* como el conjunto de elementos *TreeItem* que tiene por definición al menos una relación del tipo *hasChild* con un elemento de tipo *TreeItem* . Es decir, una instancia *ClusterItem* se caracterizará por tener al menos alguna propiedad *hasChild* que lo relaciona con un elemento de la clase *TreeItem*.

$$ClusterItem \equiv (TreeItem \cap \exists hasChild.TreeItem) \qquad \text{[Expr. 6.2]}$$

A su vez se define un elemento de la clase *LeafItem* como un objeto *TreeItem* que no es un elemento *ClusterItem* y tiene además una propiedad de datos *isLeafItem* con el valor *true*.

$$LeafItem \equiv (TreeItem \cap (isLeafItem \ni true)) \qquad \text{[Expr. 6.3]}$$

La razón por la que se incluye la propiedad de datos *isLeafItem* es para poder marcar un elemento *TreeItem* directamente como terminal. Al trabajar con la suposición de OWA (Open World Assumption), podríamos decidir que un objeto *TreeItem* es un *ClusterItem* porque encontramos una relación del tipo *hasChild* con otro objeto *TreeItem*, pero no podríamos decidir si un TreeItem pertenece a la clase *LeafItem* por el simple hecho de no haber encontrado aún ninguna relación *hasChild* con otra instancia de *TreeItem*. De esta forma cuando encontremos que esa propiedad es *true* sabremos que es *LeafItem*, y si la encontramos con el valor *false* o encontramos algún hijo, es una instancia de la clase *ClusterItem*.

Definimos también las instancias de la clase *RootItem* como objetos que sean *TreeITem* (*ClusterItem* o *LeafItem*) y que no tiene ningún padre:

$$RootItem \equiv (TreeItem \cap \neg(\exists isChildOf\,.TreeItem)) \qquad \text{[Expr. 6.4]}$$

La expresión 6.5 nos servirá para definir a su vez una clase *TopTreeStructure* (estructura de árbol que no está incluida en otros árboles) diciendo que son las que su primer elemento es un elemento *RootItem*:

$$TopTreeStructure \equiv \qquad \text{[Expr. 6.5]}$$
$$(TreeStructure \cap (\exists hasFirstTreeItem.RootItem,))$$

La razón por la que aparece el concepto de *TopTreeStructure* es para permitir que la rama de un árbol sea a su vez un árbol, pero que se pueda distinguir entre el árbol principal y los subárboles (ver Fig. 6.7.).

Otra de las razones por las que se introduce el elemento *RootItem* es para intentar evitar los bucles de referencias. Puesto que un objeto *TreeItem* puede tener como mucho un padre, si ya tiene padre no puede tener otro distinto, y si existe en el árbol un elemento *RootItem* (que no tiene ningún padre), será imposible que exista una referencia circular (ver Fig. 6.8).

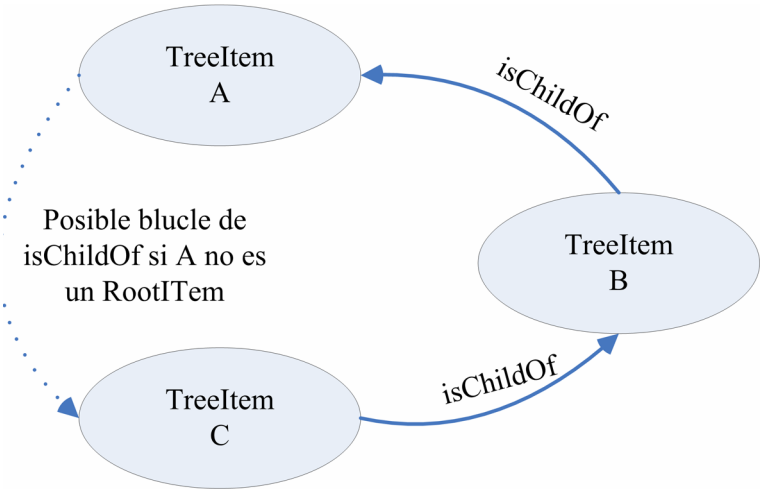


Fig. 6.8. Existencia de referencias circulares

En la figura 6.8 podemos ver el caso en que no se definieran elementos que no puedan ser descendientes de otros. Si la instancia de la clase *TreeItem* C tiene a B como padre y la B tiene a A como padre, en el caso que la instancia de la clase *TreeItem* A tuviera como padre a la instancia C, se produciría un bucle.

En la Fig 6.9 podemos ver la jerarquía de la estructura de árbol, y en la figura 6.10 las distintas propiedades que relacionan los objetos.

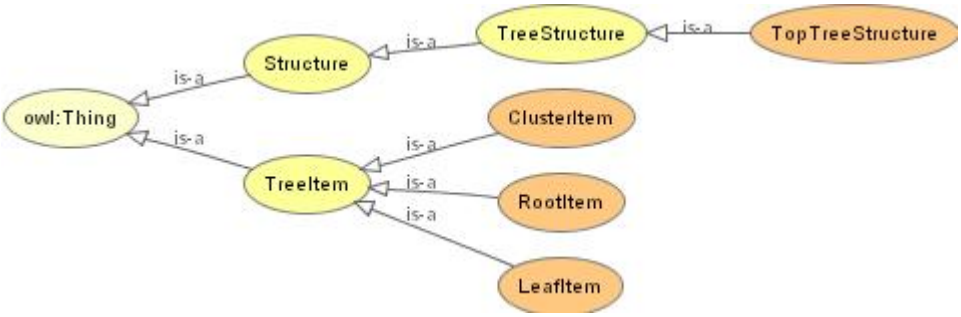


Fig. 6.9. Jerarquía de la estructura de árbol

En la figura 6.9 destaca la definición de las subclases no excluyentes de *TreeItem*: *ClusterItem*, *RootITem* y *LeafItem*. En paralelo podemos ver los conceptos de estructura definidos: *Structure*, *TreeStructure* y *TopTreeStructure*.

A partir de la propiedad *isChildOf* podríamos haber pensado en definir la propiedad *hasSibling* como propiedad simétrica que indica que si dos

instancias de la clase *TreeItems* distintas tienen el mismo padre (relación por medio de la propiedad *isChildOf* con el mismo padre) entonces se relacionan por medio de la propiedad *hasSibling*:

$$\begin{aligned} &\forall X \forall Y \forall Z \in TreeITem, \\ &owl : differentFrom(X,Y) \wedge \\ &isChildOf(X,Z) \wedge \\ &isChildOf(Y,Z) \Rightarrow hasSibling(X,Y) \end{aligned} \qquad \text{[Expr. 6.6]}$$

En la expresión 6.6 tenemos la regla derivada de la definición de la propiedad *hasSibling* definida en el párrafo anterior, y que se corresponden con la definición común en la mayoría de estructuras en árbol.

Sin embargo OWL DL no permite la aserción de reglas, por lo que no podríamos a priori realizar este tipo de afirmación. Este tipo de debilidad viene indicado por autores como [CRE05], y por eso existe una propuesta de combinación entre OWL DL y RuleML: SWRL [SWRL04]. En el caso de utilizar SWRL la regla quedaría como se indica en el ejemplo 6.1:

```
  <swrl:Variable rdf:ID="x"/>
  <swrl:Variable rdf:ID="z"/>
  <swrl:Variable rdf:ID="y"/>
  <swrl:Imp rdf:ID="ReglaSibling">
    <swrl:body>
      <swrl:AtomList>
        <rdf:first>
          <swrl:IndividualPropertyAtom>
            <swrl:argument1 rdf:resource="#x"/>
            <swrl:propertyPredicate
rdf:resource="#isChildOf"/>
            <swrl:argument2 rdf:resource="#z"/>
          </swrl:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest>
          <swrl:AtomList>
            <rdf:rest>
              <swrl:AtomList>
                <rdf:first>
                  <swrl:DifferentIndividualsAtom>
                    <swrl:argument2 rdf:resource="#y"/>
                    <swrl:argument1 rdf:resource="#x"/>
                  </swrl:DifferentIndividualsAtom>
                </rdf:first>
                <rdf:rest
rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
```

```
ns#nil"/>
              </swrl:AtomList>
            </rdf:rest>
            <rdf:first>
              <swrl:IndividualPropertyAtom>
                <swrl:propertyPredicate
rdf:resource="#isChildOf"/>
                <swrl:argument1 rdf:resource="#y"/>
                <swrl:argument2 rdf:resource="#z"/>
              </swrl:IndividualPropertyAtom>
            </rdf:first>
          </swrl:AtomList>
        </rdf:rest>
      </swrl:AtomList>
    </swrl:body>
    <swrl:head>
      <swrl:AtomList>
        <rdf:first>
          <swrl:IndividualPropertyAtom>
            <swrl:argument1 rdf:resource="#x"/>
            <swrl:argument2 rdf:resource="#y"/>
            <swrl:propertyPredicate
rdf:resource="#hasSibling"/>
          </swrl:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest  rdf:resource="http://www.w3.org/1999/02/22-
rdf-syntax-ns#nil"/>
      </swrl:AtomList>
    </swrl:head>
  </swrl:Imp>
```

Ejemplo 6.1. Ejemplo de regla en SWRL

Como podemos destacar de las partes resaltadas del texto del ejemplo 6.1, SWRL sí que permite realizar aserciones de reglas (conocidas como también como implicaciones). En este caso separa entre un cuerpo, indicado por la marca <*body*> y una cabecera, indicado por la marca <*head*>. El cuerpo contiene los condicionantes de la regla y la cabecera la consecuencia de dicha regla.

Sin embargo en esta Tesis no se contemplarán las reglas, ya que están fuera del alcance de OWL DL, lenguaje en el que expresaremos nuestra ontología. De todas formas SWRL permite utilizar ontologías OWL, por lo que se podría en un futuro ampliar esta ontología con reglas y producir una ontología en lenguaje SWRL. Para los objetivos de esta Tesis no se considera necesario.

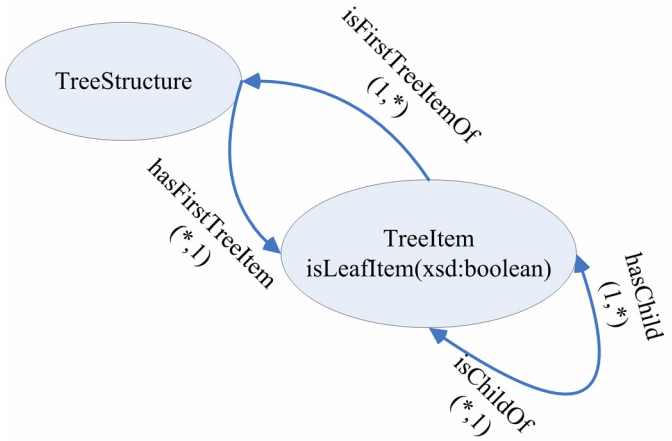


Fig. 6.10. Propiedades de relación de los objetos

En la figura 6.10 podemos ver las propiedades que relacionan entre sí a las instancia de *TreeItem* y de la clase *TreeStructure*. Se basan en una relación de padre/hijo entre objetos *TreeItem* y una propiedad *isLeafItem* que permite marcar a los elementos terminales. Por otro lado tenemos la relación de una estructura de tipo árbol con su primer elemento del árbol, por medio de la propiedad *hasFirstTreeItem* y *isFirstTreeITemOf*. Además vemos en la figura la cardinalidad de las distintas relaciones, indicando con "*" cuando dicha cardinalidad no está limitada. También indicamos dentro de las distintas clases las propiedades de tipos de datos que relacionan a cada instancia con un dato numérico, booleano, de texto o similares.

6.4 Especificación de la ontología para los contenidos estáticos

En este apartado realizaremos la especificación formal de los contenidos que han sido creados previamente a la fase de impartición de una experiencia educativa y que no generan información sobre la consecución de objetivos educativos por parte del alumno.

La especificación se basará en la concreción por medio de expresiones lógicas y posterior traslación a OWL de los conceptos de *Learning Content Package*, *Organization* y *Learning Activity*, y utilizará la ontología auxiliar de estructura de árbol planteada en el apartado anterior. Puesto que esta especificación se basa en el modelo de referencia SCORM, incluiremos el sufijo *SCORM* a nuestras denominaciones.

Las clases definidas son las siguientes:

- **SCORMContentPackage**. Es la representación del concepto de *Learning Content Package* y se relaciona al menos con un elemento *SCORMOrganization* por medio de la propiedad de objeto *hasOrganization*, función inversa-funcional (una instancia de *SCORMOrganization* sólo puede estar en un único *SCORMContentPackage*).

$$SCORMContentPackage \subseteq \exists hasOrganization.SCORMOrganization \quad \text{[Expr. 6.7]}$$

En la expresión 6.7 podemos ver se indica que los objetos de la clase *SCORMContentPackage* están incluidos dentro del conjunto de los objetos que tienen al menos una relación por medio de la propiedad *hasOrganization* con una instancia de la clase *SCORMOrganization*.

- **SCORMOrganization**. Es la representación del concepto *Organization* y está relacionado con múltiples instancias de la clase *SCORMLearningActivity* por medio de la propiedad *hasChild*. Esto significa que todo elemento *SCORMOrganization* es a su vez el primer elemento (elemento raíz) de una estructura de tipo árbol.

$$SCORMOrganization \subseteq RootItem \cap \forall hasChild.SCORMLearningActivity \quad \text{[Expr. 6.8]}$$

Como vemos en la expresión 6.8, la clase *SCORMOrganization* está incluida dentro de la clase *RootItem* y además si tiene alguna relación por medio de la propiedad *hasChild* es con una instancia de *SCORMLearningActivity*. Esto nos permitirá definir a partir de aquí una estructura de tipo árbol con instancias de la clase *SCORMLearningActivity* que vemos a continuación.

- **SCORMLearningActivity**. Es la representación del concepto de *Learning Activity* y se caracteriza por ser un elemento no raíz de la estructura del árbol, como se describe en la expresión 6.9.

$$SCORMLearningActivity \subseteq TreeItem \cap \neg RootItem \cap \forall hasChild.SCORMLearningActivity \quad \text{[Expr. 6.9]}$$

En la expresión 6.9 se aprecia que las instancias de la clase *SCORMLearningActivity* son también instancias de la clase *TreeItem* que no son del tipo *RootItem* y además que sólo se relacionan por

medio de la propiedad *hasChild* con instancias de su misma clase *SCORMLearningActivity*.

- **SCORMLeafLearningActivity** son Learning Activities que se caracterizan por ser elementos terminales de la estructura de árbol y por estar relacionados con una *SCORMResource* (relación funcional). Tiene como subclases disjuntas *SCORMAsset* y *SCORMSCO*.

$$SCORMLeafLearningActivity \equiv SCORMLearningActivity \cap LeafItem \cap \exists hasSCORM\,\mathrm{Re}source.SCORM\,\mathrm{Re}source \quad \text{[Expr. 6.10]}$$

$$SCORMLeafLearningActivity \equiv SCORMAsset \cup SCORMSCO$$

$$SCORMAsset \subseteq \neg SCORMSCO$$

Como indica la expresión 6.10, una instancia de la clase *SCORMLeafLearningActivity* se define por ser una instancia de la clase *SCORMLearningActivity* que es además una instancia de la clase *LeafItem* y que tiene al menos una relación con una instancia de la clase *SCORMResource* por medio de la propiedad *hasSCORMResource*. Además cualquier instancia de esta clase es o una instancia de la clase *SCORMAsset* o una instancia de la clase *SCORMSCO*.

- **SCORMResource**. Es una descripción simple de un recurso, que identifica por medio de su URI. Contiene una propiedad funcional de datos *hasURI* que relaciona el objeto con un tipo de datos *xsd:anyURI*.

$$SCORM\,\mathrm{Re}source \subseteq \exists hasURI.xsd : anyURI \quad \text{[Expr. 6.11]}$$

En la figura 6.11 podemos contemplar la jerarquía resultante de la ontología de contenidos estáticos. En la figura 6.12 podremos observar un resumen de sus propiedades principales.

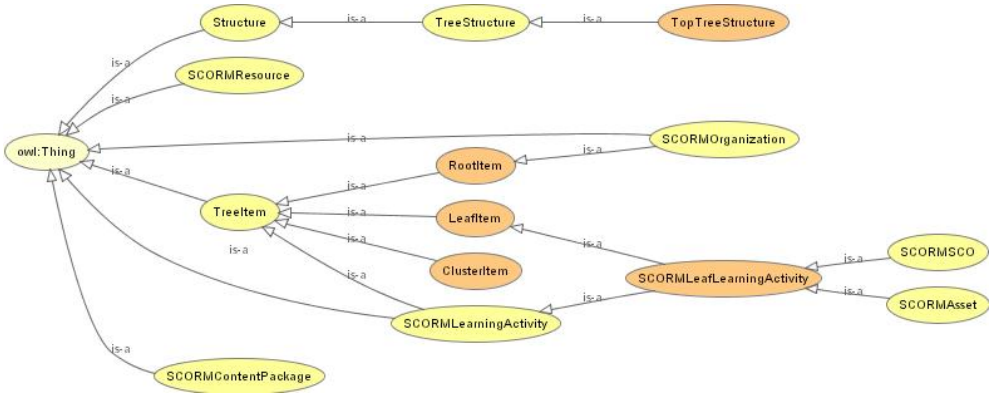


Fig. 6.11. Jerarquía de la ontología para los contenidos estáticos

En la figura 6.11, extraída de la herramienta de generación de ontologías de Protégé-OWL[WWW78] (en particular su ampliación para visualización de ontologías OWLViz[WWW79]) observamos unas clases nuevas como la clase *Structure*, que únicamente indica que sus subclases son estructuras de algún tipo, y de esta forma permitir ampliaciones de la ontologías con otras estructuras que se definan en un futuro. En la figura 6.11 se representan, además, las clases que especificamos para la estructura de árbol, ya que se utilizan en esta ontología. En amarillo tenemos las clases primitivas, es decir, las clases en las que sólo indicamos propiedades que son condiciones necesarias para pertenecer a la misma, y en naranja las clases definidas, en las que definimos propiedades que son necesarias y suficientes para que un objeto se pueda considerar perteneciente a esa clase.

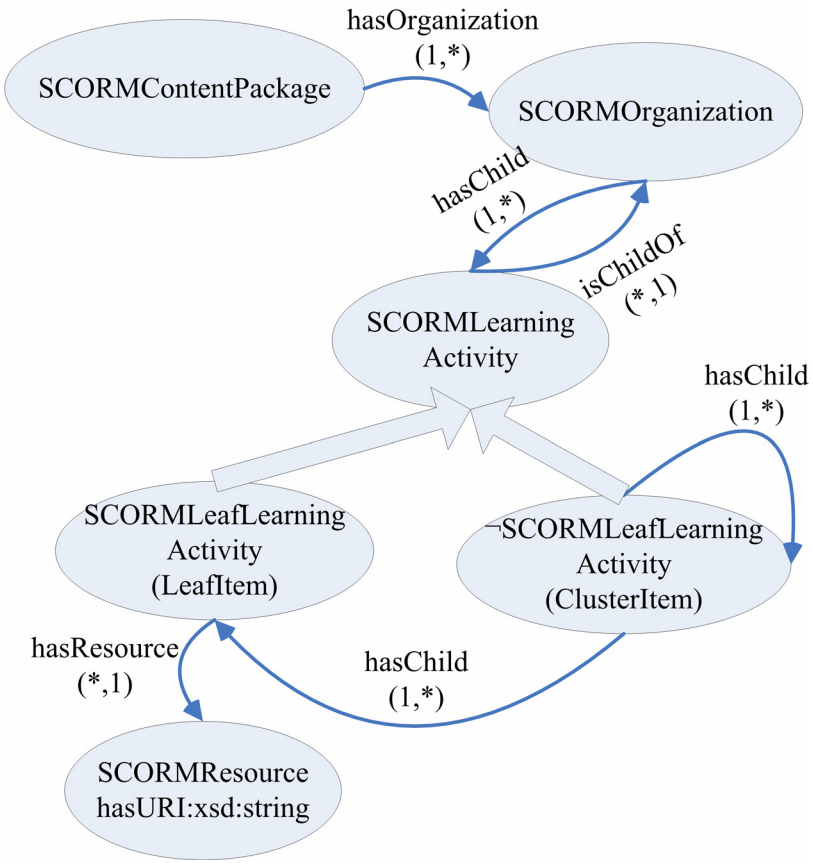


Fig. 6.12. Propiedades de relación para los contenidos estáticos

Como apreciamos en la figura 6.12, podemos decir a modo de resumen que cada objeto *SCORMContentPackage* está relacionada al menos con una instancia de *SCORMOrganization*, que a su vez el nodo raíz de un árbol de objetos *SCORMLearningActivity*. La clase *SCORMLearningActivity* se divide en dos subclases (indicado en la figura por medio de flechas más gruesas), en función de si es subclase de *LeafItem* o no (y por tanto subclase de *ClusterItem*). Los objetos terminales del árbol (instancias de *LeafItem*), están relacionados con un *SCORMResource*, que es que un objeto que encapsula una URI. Ese objeto *SCORMResource* puede ser el recurso de más de un *LeafItem*, permitiendo que esta forma que un recurso educativo sea referenciado por más de una estructura de árbol.

6.5 Implementación de la ontología para las interacciones alumno-sistema

El objetivo de esta ontología es representar por medio de lenguajes ontológicos la lógica existente en las interacciones descritas en el capítulo *Primeras interacciones: Evaluaciones*. De este modo, la ontología especificada en este apartado será la que nos servirá para permitir a los agentes inteligentes la realización de búsquedas dentro de las distintas evaluaciones de una experiencia educativa.

Para llevar a cabo la implementación de la ontología se distingue entre la parte de descripción de las actividades de evaluación a realizar y la descripción de los resultados a obtener, siguiendo la filosofía conceptual del modelo de referencia SCORM, donde se separa entre las actividades *SCOs* que pueden comunicarse con el LMS e indicar los distintos *LearningObjective* que se han alcanzado. No mezclaremos la navegabilidad con los objetivos de aprendizaje, ya que no se considera importante para permitir la interoperabilidad. En la figura figura 6.2 se pudo observar los distintos conceptos descritos en esta ontología. A continuación veremos por separado la parte dedicada a la descripción de las actividades de evaluación, basándonos en IMSQTI y después la descripción en lógica descriptiva de los distintos conceptos utilizados para la descripción de los objetivos a conseguir por el alumno, siguiendo la filosofía de TeML, donde cada evaluación permitía evidenciar el dominio de un concepto determinado.

6.5.1 Descripción de las actividades de evaluación

Las actividades para la interacción alumno-sistema representan en esta ontología las herramientas conductistas del aprendizaje, y comprenderá los distintos tests de evaluación que se pueden realizar en un proceso de aprendizaje. Dichas actividades se representarán en nuestra definición de ontología por medio de instancias de la clase *AssessmentActivity*. La clase *AssementActivity* se define como una especialización de la clase ya definida *SCORMSCO*, que se definió en la ontología de contenidos estáticos. Las instancias de la clase *AssessmentActivity* contienen al menos una propiedad de los relaciona con un elemento evaluable, que representaremos por medio de instancias de la clase *AssessmentItem*. Las instancias de la clase *AssementActivity* se caracterizan también por tener una propiedad obligatoria *completionThreshold*, con un valor comprendido entre [0,1] que define el valor final que han de sumar todas las respuestas del alumno como

mínimo para que se considere la actividad como superada satisfactoriamente (y por tanto, sus objetivos de aprendizaje obtenidos):

$$\begin{aligned} &AssesmentActivity \subseteq \\ &(SCORMSCO) \cap \\ &(hasAssessmentItem \geq 1) \cap \\ &(completionThreshold = 1) \end{aligned} \qquad \text{[Expr. 6.12]}$$

En la expresión 6.12 se indica además que las instancias de la clases *AssessmentActivity* deben contener al menos una elemento unitario de evaluación, representado por medio de la propiedad *hasAssessmentItem*. Los elementos evaluables *AssessmentItem* tienen una estructura parecida a la de IMS QTI, es decir, tienen un cuerpo del elemento (equivalente al enunciado de una pregunta), un proceso de evaluación de la respuesta del alumno y un posible Feedback en caso necesario. Podemos ver las relaciones en la figura 6.13.

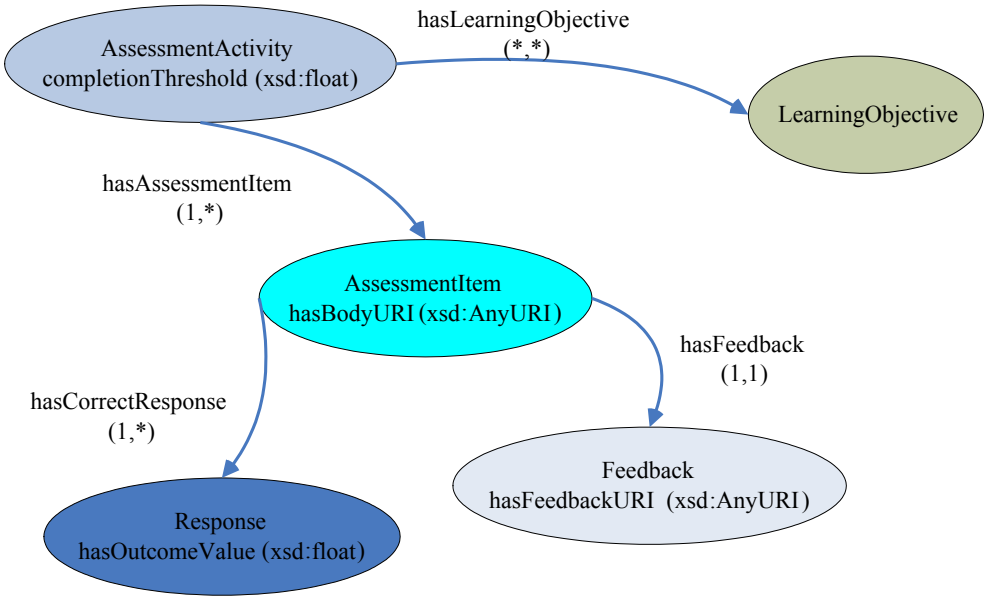


Fig. 6.13. Relaciones de los elementos evaluables

En la figura 6.13 podemos apreciar que una instancia de la clase *AssementActivity* tiene una propiedad de tipo de datos *completionThreshold*, que puede tener un valor de coma flotante, Además, se puede relacionar por medio de una relación muchos a muchos con distintas instancias de la clase *LearningObjective*, lo que indica que una instancia de *AssessmentActivity*

puede hacernos conseguir más de un objeto *LearningObjective*, y que a su vez un mismo *LearningObjective* se puede obtener desde distintos objetos *LearningActivity*. Las instancias de *AssessmentItem* se relacionan con distintos objetos *Response* por medio de la propiedad *hasCorrectResponse*. Esta relación indica que esa posible respuesta representada por medio de una instancia *Response*, si la contesta el alumno, se ha de puntuar con el valor de su propiedad de datos *hasOutcomeValue*. Las distintas respuestas relacionadas con un objeto de la clase *AssessmentITem* por medio de *hasCorrectResponse* no tienen por qué indicar que son correctas. Es el valor de la propiedad *hasOutcomeValue* lo que indicará la exactitud de la misma. De esta forma si, por ejemplo, un elemento de la clase *Response* tiene un valor en *hasOutcomeValue* menor que cero, indica que si el alumno elige esa contestación, se le restará puntuación a la hora de evaluar si se ha conseguido aprobar el objeto *LearningAssessment* y por tanto haber obtenido los objetivos de aprendizaje correspondientes. Si no hay una relación por medio de *hasCorrectResponse* con el valor de la respuesta del alumno, se supone que se asigna un valor 0 a esa respuesta.

El proceso de evaluación de la respuesta no se define para cada caso como en IMS QTI, sino que se considera más apropiado definir tipos de elementos evaluables. Esta parte se basará en parte en el modelo de datos de IEEE para las comunicaciones entre el alumno y el sistema, IEEE 1484.11.1[IEEE05]. Se puede considerar que los distintos tipos de elementos evaluables abstraen un conjunto de posibles *processingTemplates* de IMS QTI, uno por cada tipo de elemento evaluable. Por ejemplo, podemos considerar que todas las preguntas de test con sólo una respuesta correcta se procesan de igual forma, por lo que pueden compartir el mismo *processingTemplate* en IMS QTI. Por tanto, cada tipo de elemento evaluable puede tener un tipo definido de respuesta. Esta decisión sacrifica una mayor flexibilidad que nos ofrece IMS QTI, ya que evalúa la respuesta de forma abstracta por medio de reglas, a cambio de una mayor claridad y sencillez en la metainformación. Otra razón por la que no se realiza una descripción del procesamiento es de nuevo la necesidad de ampliar OWL a OWL-S (u otro lenguaje superior de descripción de procesos), no siendo de gran interés esta descripción para las búsquedas en repositorios.

Como se indica en la figura 6.13, puede haber varias relaciones *hasCorrectResponse*, donde cada respuesta de las posibles definidas tenga una puntuación distinta, incluso negativa (respuestas que restan). Esto difiere de IEEE 1484.11.1, donde no se especifica la forma de convertir una respuesta de un alumno en un resultado (numérico o no). Además, este valor

de la propiedad *hasOutcomeValue* nos permite prescindir de otra parte de ponderación entre elementos evaluables, ya que se puede realizar por medio de este valor.

$$\mathrm{Re}\,sponse \subseteq \qquad \text{[Expr. 6.13]}$$
$$\forall\, hasOutcomeValue.xsd : float \cap hasOutcomeValue = 1$$

Como se distingue en la expresión 6.13, las respuestas han de tener exactamente una propiedad *hasOutcomeValue* que lo relaciona con un tipo de datos de coma flotante. La propiedad *hasOutcomeValue* ha de contener un valor obligatoriamente para todas las respuestas, incluso las que no se corrigen automáticamente, ya que incluye el concepto de ponderación de la pregunta.

Para ser capaces de indicar explícitamente el tipo de pregunta y procesado de la respuesta, se definen subclases de *AssessmentItem* por cada tipo de interacción de IEEE 1484.11.1, y se definen a su vez el formato de sus posibles respuestas. En la figura 6.14. podemos ver las diferentes subclases de la clase AssessmentItem y en la figura 6.15 podemos ver los tipos de respuesta relacionados.

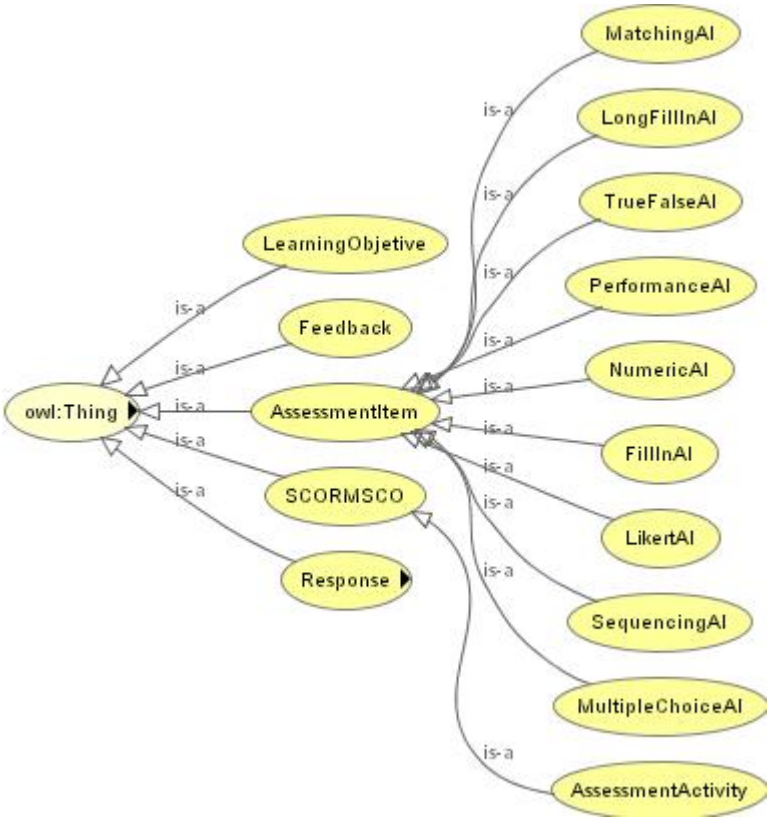


Fig. 6.14. relaciones de herencia entre las clases

La definición de los distintos tipos de respuestas se hace indicando el tipo de valor que puede tener la respuesta, y permitiendo una o múltiples valores por respuesta en función del tipo de pregunta que sea. Por ejemplo:

- **TrueFalseAI** es una subclase de la clase *AssessmetItem* que sólo permite respuestas correctas del tipo *TrueFalseResponse*. Además, indicamos que los distintos subtipos de la clase *AssessmentItem* son disjuntos entre sí, es decir, que una instancia de la clase *TrueFalseAI* no puede ser a su vez instancia de ningún otro tipo de pregunta..

[Expr. 6.14]

$$TrueFalseAI \subseteq (AssessmentItem) \cap (\forall hasCorrectResponse.TrueFalseResponse)$$
$$TrueFalseAI \subseteq \neg FillInAI \cup \neg MatchingAI \cup \ldots \cup \neg MultipleChoiceAI$$

Como vemos en la expresión 6.14, si un elemento pertenece a la clase *TrueFalseAI*, tendrá todas sus respuestas como instancias del tipo *TrueFalseResponse* y no podrá pertenecer a otra subclase de la

clase *AssessmentItem* (no puede ser un elemento de FillInAI, por ejemplo).

La clase *TrueFalseResponse* es una especialización de la clase *Response* que tiene una propiedad funcional *TFResponseValue* que lo relaciona con un booleano, y además sólo se da esa propiedad una vez por respuesta, como indica la expresión 6.15.

[Expr. 6.15]

$$TrueFalse\,\mathrm{Re}sponse \subseteq \mathrm{Re}sponse \cap \left(\forall TF\,\mathrm{Re}sponseValue.xsd : boolean\right)$$
$$TrueFalse\,\mathrm{Re}sponse \subseteq TF\,\mathrm{Re}sponseValue = 1$$

- **FillInAI**, será también una subclase de *AssessmentItem*, sólo que en este caso se permite rellenar hasta 10 huecos de texto por respuesta (como indica el modelo de datos de IEEE 1484.11.1 para las preguntas de rellenar huecos), e indica si es importante el orden de los strings y si diferencia entre mayúsculas y minúsculas:

[Expr. 6.16]

$$FillInAI \subseteq \left(AssessmentItem\right) \cap \left(\forall hasCorrect\,\mathrm{Re}sponse.FillIn\,\mathrm{Re}sponse\right)$$
$$FillInAI \subseteq \neg MatchingAI \cup \neg LongFillInAI \cup \ldots \cup \neg TrueFalseAI$$

$$FillIn\,\mathrm{Re}sponse \subseteq \mathrm{Re}sponse \cap \forall FI\,\mathrm{Re}sponseValue.xsd : String$$
$$FillIn\,\mathrm{Re}sponse \subseteq \left(FI\,\mathrm{Re}sponseValue.xsd : String \leq 10\right)$$
$$FillIn\,\mathrm{Re}sponse \subseteq \forall caseMatters.xsd : boolean \cap caseMatters = 1$$
$$FillIn\,\mathrm{Re}sponse \subseteq \forall orderMatters.xsd : boolean \cap orderMatters = 1$$

En la expresión 6.16 vemos reflejado en lógica descriptiva tanto las restricciones de la clase *FillInAI* como las aserciones respecto a la clase de respuesta que define *FillInResponse*. La clase FillInAI se define de forma muy similar a la clase *TrueFalseAI*, y para la clase de respuesta *FillInResponse* podemos destacar su propiedad *FIResponseValue* que lo relaciona con una cadena de caracteres, pudiendo tener hasta 10 relaciones con cadenas de caracteres por medio de la propiedad *FIResponseValue*. También vemos las propiedades únicas de valores boolean *caseMatters* y *orderMatters*.

- **SequencingAI**. Para respuestas que necesitan tipos de datos complejos (como por ejemplo las instancias de la clase *SequencingAI* que necesita para cada elemento de la respuesta el orden y el identificador del elemento de secuencia) se definen clases auxiliares aparte de la clase de respuesta. En el caso de *SequencingAI* definimos, además de la subclase de respuesta *SequenceResponse*, una nueva clase denominada *SequenceItem*.

[Expr. 6.17]

$$SequenceAI \subseteq (AssessmentItem) \cap (\forall hasCorrect\,\mathrm{Re}sponse.Sequence\,\mathrm{Re}sponse)$$
$$SequenceAI \subseteq \neg FillInAI \cup \neg MatchingAI \cup \ldots \cup \neg TrueFalseAI$$

$$Sequence\,\mathrm{Re}sponse \subseteq \mathrm{Re}sponse \cap \exists S\,\mathrm{Re}sponseVaule.SequenceItem$$
$$Sequence\,\mathrm{Re}sponse \subseteq S\,\mathrm{Re}sponseValue \leq 36$$

$$SequenceItem \subseteq \forall SOrder.xsd : \mathrm{int}eger \cap SOrder = 1$$
$$SequenceItem \subseteq \forall SValue.xsd : ID \cap SValue = 1$$

En la expresión 6.17 vemos que, además de definir la subclase de elemento de evaluación *SequenceAI*, el tipo de respuesta *SequenceResponse*, que permite tener hasta 36 instancias de la clase *SequenceItem* (límite que coincide con la especificación de IEEE 1484.11.1). relacionadas por medio de la propiedad *ResponseValue*. También se indica que los elementos de la clase SequenceItem tendrán las propiedades únicas *SOrder* y *SValue* que los relacionan con un entero y un identificador respectivamente.

El resto de tipos de respuesta definidos en IEEE 1484.11.1 se definen de la misma forma que los tres ejemplos anteriores. En la figura 6.15 podemos ver los tipos de Respuesta posible así como las clases que abstraen los valores de respuesta compuestos. Estas clases auxiliares que añadimos son:

- **NumericInterval**, que abstrae el concepto de un intervalo entre un valor mínimo y máximo. Las instancias de la clase *NumericInterval* tienen dos propiedades de datos xsd:float funcionales y obligatorias maxInterval y minInterval, que nos definen el intervalo inclusivo (como define IEEE.1484.11.1) de posibles valores, [minInterval, maxInterval]. No se contempla la posibilidad de límites superior o inferior infinito, que en IEEE.1484.11.1 se hace por medio de la no

definición del parámetro superior o inferior. La razón por la que no se sigue en este punto IEEE.1484.11.1 estriba en que al trabajar con una suposición OWA, el hecho de no tener en nuestra base de conocimiento uno de los dos extremos del intervalo no significa que no exista, por lo que nunca podremos concluir que hay una ausencia de ese extremo del intervalo, sino simplemente que no lo hemos localizado de momento.

- **PerformanceStep**, clase que nos servirá para definir un conjunto de pasos o acciones de un proceso, teniendo los parámetros de nombre (definido en la propiedad *stepName*) como obligatorio y el orden (propiedad *stepOrder*). Puede que los pasos no tengan por qué seguir un orden definido, por lo que habrá otra propiedad booleana *orderMatters*. También contiene el valor del paso, entendido como la forma de describir la acción o valor de la acción del proceso. Dicho valor de paso puede ser un xsd:string (definido en la propiedad *stepString*) o un objeto de la clase *NumericInterval*, dependiendo de si es una instancia de la subclase *StringPerformanceStep* o de la otra subclase *NumericPerformanceStep* respectivamente.

- **MatchItem**, siendo esta clase la que nos sirve para valores de respuesta que relacionan dos opciones entre sí, y tiene una fuente (indicada en la propiedad *hasSource*) única y definida por su xsd:ID y un conjunto de posibles opciones destinto (relacionadas con la instancia de la clase *Matchtem* por medio de la propiedad *hasTarget*) también definido por su xsd:ID.

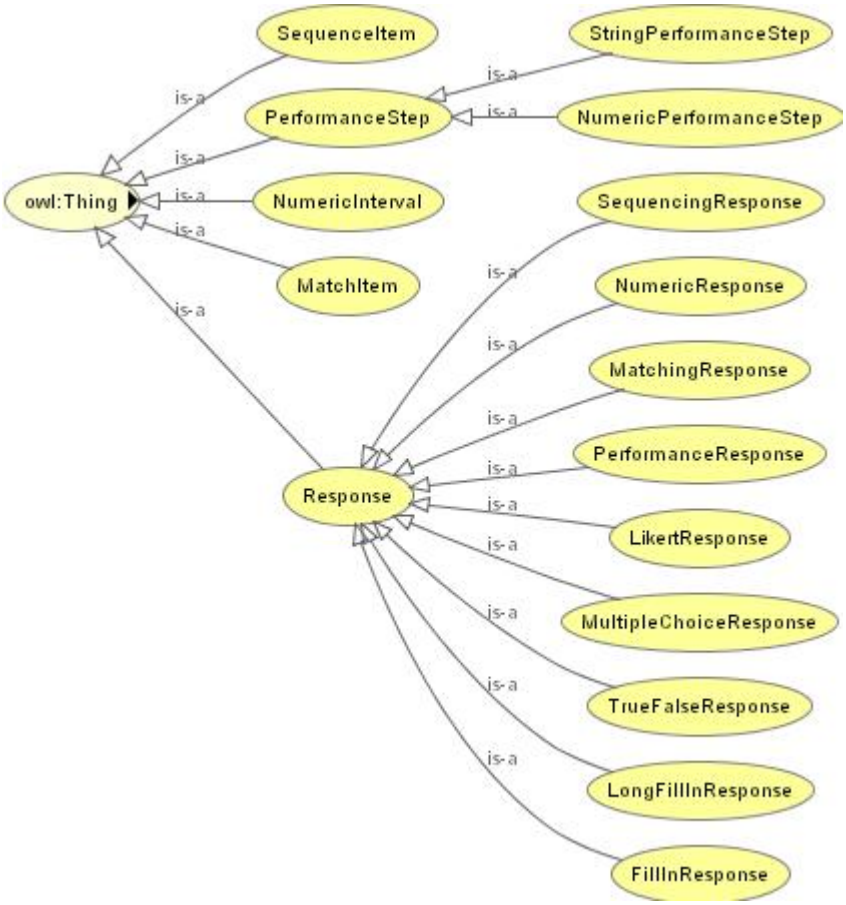


Figura 6.15. Distintos tipos de respuesta y clases que representan valores de respuesta compuestos

6.5.2 Descripción de los resultados

Los resultados que se pueden obtener por medio de la realización de actividades de evaluación se representa por medio de los objetivos de aprendizaje (abstraidos en la clase *LearningObjective*), que a su vez se agregan en competencias (abstraidos en la clase *Competency*), no siendo a veces suficiente un conjunto de instancias de *LearningObjective* para obtenerla (por ejemplo, se puede necesitar como evidencia al menos tres años de experiencia laboral en una determinada materia, aparte de aprobar algunos exámenes).

A continuación describiremos el proceso de obtención de objetivos de aprendizaje por parte de un alumno por medio de la contestación de los tests. A la respuesta de un alumno a una instancia de *AssessmentActivity* se le asignará un valor para la propiedad *hasOutcomeValue* igual al que haya

en la instancia de *CorrectResponse* de la actividad cuyos valores de respuesta sean iguales a los contestados por el alumno. Para poder evidenciar que un alumno ha conseguido un objetivo de aprendizaje, ha de superar satisfactoriamente un elemento *AssessmentActivity* que conduzca a evidenciar dicho objetivo de aprendizaje. Esta relación entre una instancia de *AssessmentActivity* y una instancia de *LearningObjective* se implementa por medio de la propiedad *hasLearningObjective*. Como hemos visto antes, para superar satisfactoriamente un AssessmentActivity el alumno ha de conseguir una puntuación suma de sus valores de resultado para cada elemento *AssessmentItem* (que se almacenará en la variable *completionStatus* cuando se realice una evaluación a un alumno) que supere al valor de la propiedad *completionThreshold* definida en la actividad.

[Expr. 6.18]

$$completionStatus \equiv \sum_i AssessmentItem_i.Student\,\mathrm{Re}sponse.hasOutcomeValue$$

$$completionStatus \geq AssessmentActivity.completionThreshold$$

En la expresión 6.18 vemos cómo se calcula en una interacción si se ha completado con éxito o no una elemento *AssessmentActivity*. Se sumaran todos los resultados a las respuestas del alumno a los elementos *AssementItem* de la actividad, y si dicho valor es superior al valor almacenado en la propiedad *completionThreshold* de la actividad, se considerará como superada y el alumno habrá obtenido todos los objetivos de aprendizaje relacionados con dicha actividad. Este apartado es un inciso de cómo debe actuar un LMS en tiempo de ejecución y por tanto no está incluido en la ontología, ya que la lógica descriptiva no permite representar operaciones matemáticas ni reglas de actuación.

Las instancias de la clase *LearningObjective* contienen una propiedad de tipo String (propiedad *description*) que define una pequeña descripción del objetivo de aprendizaje. Además, un elemento de la clase *LearningObjective* puede estar relacionado con múltiples instancias de la clase *Competency* por medio de la propiedad de objeto *contributesToCompetency*.

La clase *Competency* se define de forma similar a IMS RDCEO [IMS02], incluyendo los campos de título (propiedad *hasTitle*), su ubicación en algún catálogo de competencias (por medio de las propiedades *hasCatalog* y *hasEntry*) y un conjunto de listas (representadas por medio de rdf:List), con al menos una lista, que define de forma cerrada todos los objetos de la clase *LearningObjective* que son necesarios para la competencia. La razón por la

que se utiliza una lista cerrada es para poder definir de forma clara conjuntos completos de instancias de la clase *LearningObjective* que consiguen hacen que se considere como probada esa competencia abstraída en la instancia de la clase *Competency*. De esta forma se podrá asegurar que se han superado todas los objetivos de aprendizaje necesarios para evidenciar la adquisición de la competencia. Habrá competencias que necesiten objetivos de aprendizaje que no se puedan obtener por medio de exámenes, como puede ser un número de años de experiencia en un sector. En la Fig. 6.16 podemos las relaciones entre los distintos objetos y sus propiedades básicas.

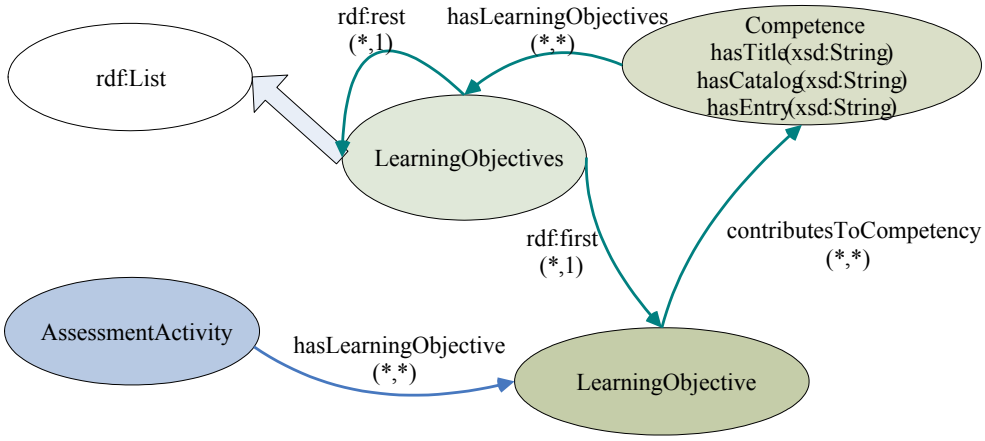


Figura 6.16. Relaciones de los objetos *LearningObjective* con el resto de objetos

En la figura 6.16 apreciamos que existe una clase que es una especialización de la lista cerrada rdf:List, que se denomina *LearningObjectives*. Esta lista contiene un conjunto cerrado de elementos de la clase *LearningObjective*. Por otro lado, tenemos la clase *Competence* que puede estar relacionado con varias listas de objetivos por medio de la propiedad *hasLearningObjectives*. Además, los objetos de la clase *LearningObjective* se relacionan con los objetos de la clase *Competency* por medio de la propiedad *contributesToCompetency*. Aunque en un principio podemos considerar que existe redundancia al relacionar los objetos *LearningObjective* con los objetos *Competency* de forma directa e indirecta por medio de la clase *LearningObjectives*, dicha relación es necesaria ya que en nuestra lógica descriptiva representada en OWL DL no se pueden realizar aserciones de reglas, como sí permite SWRL.

6.6 Implementación de la ontología las interacciones entre usuarios

En este apartado trataremos de la ontología que representa el aprendizaje constructivista, es decir, el que se produce por medio de la interacción informal y poco estructurada entre los distintos actores del proceso formativo, ya sea el tutor u otros estudiantes. Estos conceptos son los menos desarrollados y contemplados por los distintos modelos de referencia, normas y recomendaciones sobre la formación on-line, por lo que será una de las partes más innovadoras e inéditas de la Tesis, y se basará menos en conceptos subyacentes en estándares, como se ha realizado en las otras dos ontologías. Algunos modelos, como SCORM[ADL04b] e IEEE.1494.11.1[IEEE05], contemplan las interacciones entre actores del proceso formativo como un simple medio a la hora de la impartición, pero en ningún caso se trata la posibilidad de potenciar la interoperabilidad de estas interacciones, mientras está demostrado que en estas informaciones se almacena una gran cantidad de formación específica, ya que existe una gran cantidad de bases de conocimiento como[WWW73][WWW74], que se basan en foros de discusión en los que el usuario puede buscar conversaciones relacionadas con su temática de interés. En los Sistemas de Gestión de la Teleformación (LMSs) actuales se contemplan estas interacciones como un servicio añadido a la experiencia educativa. Dicho servicio y la información que ha recibido normalmente existe únicamente durante el periodo de aprendizaje de una edición de un curso, desapareciendo cuando ya se haya impartido la experiencia. Sin embargo, se propone desde esta Tesis el desarrollo de un grupo de conceptos en la ontología que facilite la recolección y tratamiento de estas interacciones, para poder realizar búsquedas semánticas dentro de las mismas, aunque haya terminado la acción formativa para la que se crearon. Por tanto podemos considerar la ontología desarrollada en esta Tesis como pionera en el campo de las interacciones entre usuarios en un proceso de teleformación.

La ontología de relaciones entre usuarios arranca de un concepto primitivo que denominaremos *ConstructivistWidget*. Esta clase contendrá a todos los elementos que potencien la comunicación constructivista entre los usuarios. Como especializaciones de esta clase encontraremos la clase que agrupa a todos los foros, denominada *Forum*, otra clase que abstrae las baterías de cuestiones y respuestas, también conocidas como FAQs, y que se representará por medio de la clase *QuestionAndAnswer*. También se contempla la escritura colaborativa, por medio de los objetos de la clase *GroupWriting*, y las tareas en grupo por medio de la clase *GroupTask*. La

interacción constructivista que especificaremos a partir de este punto es la más utilizada en las experiencias educativas por internet: los Foros.

Por tanto, la clase *Forum* será una subclase de la clase *ConstructivistWidget* con una propiedad de datos para su título (propiedad denominada *forumName*) y otra propiedad de objeto denominada *hasForumThread* para relacionarlo con múltiples instancias de la clase *ForumThread*. Esta última propiedad será inversa funcional, de forma que un objeto *ForumThread* sólo tenga un único foro del que dependa. Además, un objeto de la clase *Forum* estará relacionado con la ontología de contenidos estáticos, de forma que todo foro esté relacionado con una instancia de la clase SCORM*LearningActivity* por medio de la propiedad *relatedTo*.

$$Forum \subseteq ConstructivistWidget \qquad \text{[Expr. 6.19]}$$
$$Forum \subseteq \forall hasForumThread.ForumThread$$
$$Forum \subseteq \forall relatedTo.SCORMLearningActivity \cap relatedTo = 1$$

Como vemos en la expresión 6.19, un objeto de la clase Forum es una especialización de la clase *ConstructivistWidget* y además se relaciona con una conjunto de objetos *ForumThread* por medio de la propiedad hasForumThread. También podemos apreciar que un objeto de la clase *Forum* está relacionado de forma única con un objeto de la clase *SCORMLearningActivity*.

Las instancias de la clase *ForumThread* estarán relacionadas por medio de la propiedad *hasForumThread* y su inversa *isThreadOf* con un único objeto de la clase *Forum*. Además, cada elemento de la clase *ForumThread* será una estructura de árbol principal (clase *TopTreeStructure*). Como última restricción diremos que tendrá como primer elemento de esa estructura un tipo de comentario *TreeComment* (que definiremos a continuación) de forma obligatoria.

$$ForumThread \subseteq isThreadOf.Forum \qquad \text{[Expr. 6.20]}$$
$$ForumThread \subseteq TopTreeStructure$$
$$ForumThread \subset \exists hasFirstElement.TreeComment$$

Los objetos de la clase *ForumThread* no tendrán propiedades como título u otro identificador, ya que serán descritos por su primer elemento , que será una instancia de la clase *TreeComment*. Es decir, un objeto de la clase *ForumThread* cuyo primer elemento tenga el título “problemas al

implementar SingleTones en java" se representará con ese título, ya que consideramos que una conversación trata sobre su primer comentario.

Describiremos en la ontología también las distintas interacciones básicas en un proceso constructivista, agrupadas en la clase ConstructivistInteraction, y que tendrán en esta ontología dos subclases iniciales, una primera clase que representa las interacciones por medio de un comentario, denominada *Comment* y otra interacción que represente la publicación de algún documento que se trabaje entre un grupo de usuarios, que denominaremos *Document*.

La representación de los comentarios por medio de la clase *Comment* tendrá las siguientes propiedades (ver Fig 6.17):

- **Texto** (propiedad *text*), propiedad de datos funcional que tiene el mismo cometido que la propiedad *comments_from_learner/lms.n.comment* de IEEE.1494.11.1. Esta propiedad almacena la cadena de caracteres con el comentario escrito por el usuario.

- **Fecha de creación** (propiedad *dateOfCreation*), propiedad también funcional y con el mismo objetivo a su vez que la propiedad *comments_from_learner/lms.n.timeStamp* de IEEE.1494.11.1. En este caso será un valor del tipo xsd:dateTime.

- **Rol del autor del comentario** (propiedad *authorRole*), propiedad de objeto que lo relacionará de forma funcional con un tipo de autor (instancia de la clase *Role*, que definiremos más adelante).

- **Documento adjunto,** por medio de la propiedad *hasAttachment*, propiedad de objeto que permitirá que un comentario tenga ningún o más de un documento relacionado (instancia de la clase *Document*, que describiremos más adelante).

Como podemos ver los comentarios definidos en esta ontología añaden información relevante, como el tipo de autor y los posibles attachments, campos no contemplados por otras especificaciones. Sin embargo, no contiene la localización del comentario, ya que ésta se encontrará en la estructura en la que esté el comentario (por ejemplo, en el objeto de la clase *Forum* si es un foro).

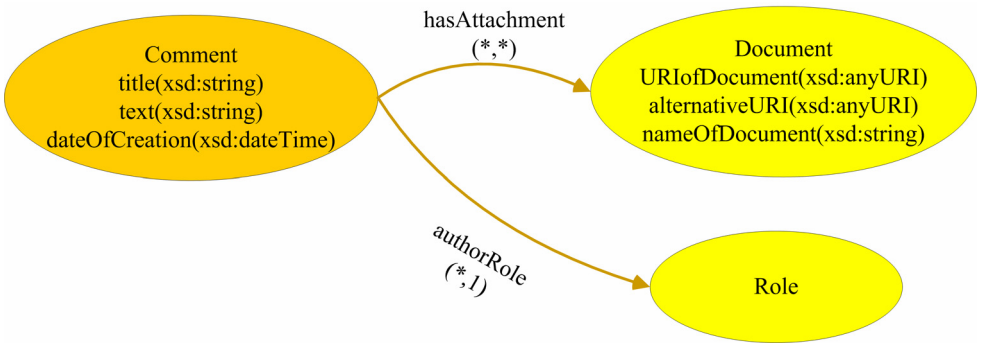


Fig. 6.17. Propiedades de Comment

Definiremos a su vez tres tipos de comentarios, que no serán disjuntos, que serán las preguntas (clase *Question*), las respuestas (clase *Answer*) (relacionados entre ambos por las propiedades *isQuestionOf* y *hasAnswer*) y el tipo que nos interesa para la definición de los foros que será el comentario en estructura de árbol, representado mediante la clase *TreeComment*.

$$Question \subseteq Comment \cap \forall hasAnswer.Answer \qquad \text{[Expr. 6.21]}$$
$$Answer \subseteq Comment \cap \forall isAnswerOf.Question \cap isAnswerOf = 1$$

$$TreeComment \subseteq Comment \cap TreeItem$$

En la expresión 6.21 vemos como se especifican las preguntas y las respuestas, así como la aserción que indica que la clase *TreeComment* pertenece a la intersección entre la clase *Comment* y la clase *TreeItem*, por lo que una instancia de *TreeComment* tendrá las propiedades de un comentario más las de un elemento de árbol. Por tanto, los objetos de la clase *TreeComment* tendrán las siguientes propiedades heredadas:

- Las propiedades heredadas por ser un TopTreeComment: *hasChild*, *isFirstTreeItemOf*, *hasDescendent*, *isLeafItem*. Estas propiedades nos permiten configurar los comentarios en una estructura de árbol, como cualquier foro.

- Las propiedades heredadas por ser un *Comment* (las descritas anteriormente). Estas propiedades nos permitirán almacenar los textos y documentos del comentario en cuestión, así como una indicación del rol que tenía el usuario que realiza el comentario, ya que no es igual un comentario realizado por un tutor que uno relacionado por un aprendiz.

La clase *Document* será una clase simple con propiedades de tipo de datos sobre su nombre (propiedad *nameOfDocument*), URI (propiedad *URIofDocument*) y múltiples posibles URIs alternativas (por medio de la propiedad *alternativeURI*).

La clase *Rol* es una clase definida por sus elementos, es decir, consta de una enumeración de elementos. Se ha definido en ella los posibles roles que pueden participar en el foro:

$$Rol \equiv \{LearnerRole, TutorRole, TechnicalSupportRole, FacilitatorRole, ContentAuthorRole, GuestRole\} \quad \text{[Expr. 6.22]}$$

De esta forma, en la figura 6.18 podemos ver las propiedades principales de las clases que nos permiten definir los foros.

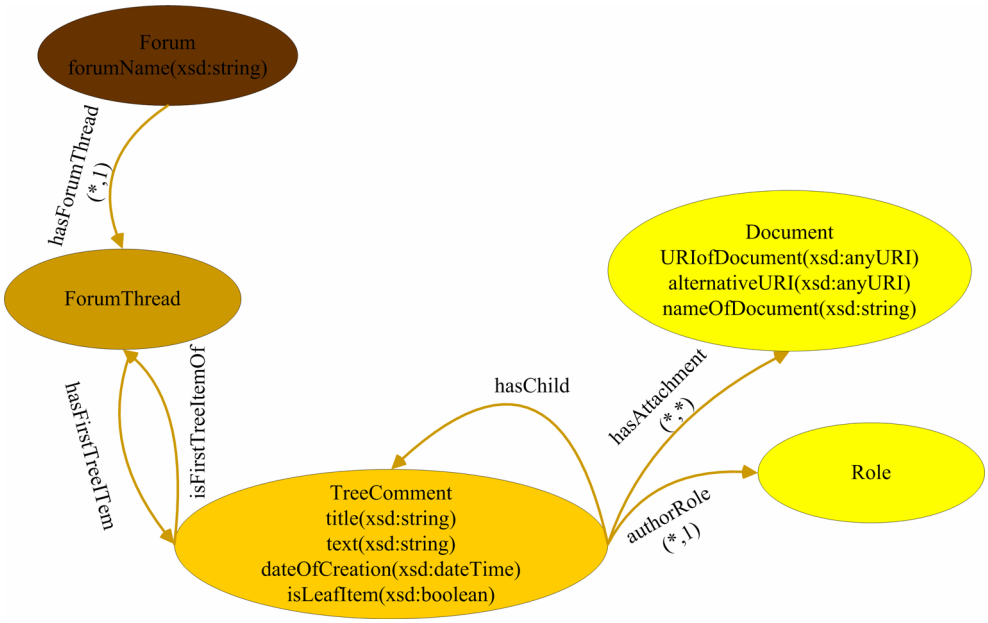


Fig. 6.18. Propiedades principales de la ontología de definición de foros

Como apreciamos en la figura 6.18, el elemento principal de los foros es el comentario con estructura de árbol (clase *TreeComment*). Los comentarios iniciales de los foros estarán relacionados con un objeto de la clase *ForumThread*, que a su vez estará relacionado con un objeto de la clase *Forum*, de forma que un objeto *Forum* pueda tener múltiples objetos

ForumThread. También podemos observar sus relaciones con el Rol que generó el comentario y con el posible documento anexado.

6.7 Conexión de las tres ontologías

En este punto veremos los puntos de unión entre las tres ontologías especificadas anteriormente para generar una única ontología que cubra todos los aspectos generales de una experiencia educativa sobre la que queramos potenciar la interoperabilidad. Aunque lo hemos visto a lo largo de las descripciones de las subontologías, los puntos de conexión son:

- **Conexión de los contenidos estáticos con las interacciones alumno-sistema**. La subontología de contenidos estáticos y la de interacción alumno-sistema se relacionan al definir la clase *AssessmentActivity* como una subclase de la clase *LeafActivityItem* de tipo *SCORMSCO*. De esta forma algunas de las instancias de la clase *SCORMSCO* podrán ser actividades de evaluación.

  Esta relación sigue la filosofía que se especifica en SCORM, con diferencia que aquí toda instancia de la clase *AssessmentActivity* es un SCO y no puede ser un Asset. De este modo cuando un autor quiere introducir un punto de interacción alumno-sistema, como puede ser un test, lo hace en la definición de los contenidos estáticos, añadiendo objeto de la clase *SCORMSCO* especial que será la actividad de evaluación, es decir, un objeto de la clase *AssementActivity*.

- **Conexión de la ontología de contenidos estáticos con la ontología de interacción entre usuarios**. La ontología de contenidos estáticos y la de interacción entre usuarios (foros) se relacionan por medio de la propiedad *relatedTo*, que relaciona un foro con una instancia de la clase *LearningActivity* (que puede que sea un cluster o un elemento terminal).

  No se especifica en la ontología si los distintos foros de la experiencia educativa se crean al principio con el contenido estático del curso o si se pueden crear a petición de un usuario en el periodo de impartición. De esta forma damos mayor flexibilidad ya que puede haber casos en que se quiera un sólo foro o un conjunto definido antes de la impartición y otros casos en que los foros se dan de alta durante la ejecución del curso.

- **Conexión de las dos subontologías de interacciones**. Estas dos ontologías se pueden relacionar entre sí por medio de la ontología de contenidos estáticos, de modo que un objeto de la clase *AssessmenItem* que sea un *SCORMSCO* puede estar relacionado con un *Forum* por medio de la propiedad *relatedTo* de una instancia de la clase *Forum*.

De esta forma las tres subontologías se unen en una única que define los tres bloques definidos al principio del capítulo (ver Fig 6.1).Una de las ventajas de unir las ontologías radica en que nos permitirá, por ejemplo, que cuando hagamos búsquedas en repositorios, podamos localizar la actividad de aprendizaje que tiene un foro sobre el que se trata un tema que me interesa, o que cuando importemos un curso importemos también los foros de una explotación anterior, lo que nos permitirá aumentar la base de conocimientos del curso.

Además, de esta forma la interacción de los actores no queda fuera de lo que se entiende como la información a exportar de un curso y esta información nos servirá para realizar búsquedas inteligentes a los foros relacionados, lo cual no es posible con los desarrollos actuales, ya que los foros son una herramienta que nace y muere en una única impartición del curso.

6.8 Conclusiones de la especificación

Como hemos visto a lo largo del capítulo hemos especificado una ontología utilizando lógica descriptiva para facilitar la interoperabilidad de los objetos educativos. La especificación en lógica descriptiva se ha realizado de forma que su transformación a OWL DL es directa como veremos en el capítulo siguiente.

Al limitar nuestra lógica a OWL DL nos ha obligado a prescindir de las aserciones de reglas, pero no nos ha limitado en ningún apartado, ya que como hemos visto no hemos necesitado ninguna regla para definir los conceptos necesarios para facilitar la búsqueda y compartición de objetos educativos. Son mayores los beneficios, ya que es el lenguaje OWL DL el que se está utilizando mayoritariamente para la web semántica y es el que tiene razonadores optimizados.

El hecho abordar la especificación en tres bloques diferenciados ha permitido realizar un enfoque modular que aporta claridad a la especificación, ya que la división se ha realizado entre los contenidos estáticos, las interacciones alumno-sistema y por último las interacciones

entre los distintos usuarios. Como ya hemos visto en el capítulo, esta división pretende separar los distintos enfoques de aprendizaje: el cognoscitivista, el conductista y el constructivista. Aunque se ha abordado de forma modular se ha tenido en cuenta en todo momento el conjunto, indicando los puntos de unión entre las tres ontologías que hace que tenga coherencia el conjunto.

En cada uno de los elementos de la ontología se ha preferido seguir, en caso de que existan, conceptos de las distintas normas, especificaciones y modelos de referencia como son ADL SCORM[ADL04a][ADL04b], IMS QTI [QTI06][QTI06b], IMS CP[IMS04] e IEEE 1484[IEEE05][IEEE02] e IEEE RDCEO[IEEE02]. Ninguna de las especificaciones cubre totalmente la ontología propuesta, por lo que se ha ido indicando qué partes de cada una se han ido siguiendo en cada caso. Para seguir estas normas ha sido necesario extraer los conceptos de las mismas, ya que la mayoría de las normas están definidas en XML y llevan documentación explicativa donde se indican las distintas restricciones y comportamientos, no estando ninguna descrita en un lenguaje ontológico para la web. La ventaja de representar la información en lenguaje ontológico radica en que podrá introducirse en la web semántica para que la información sea explotada por los agentes inteligentes, que puedan ayudar a los usuarios a realizar búsquedas más eficaces y mejores para localizar objetos educativos,

Sin embargo, dado el enfoque novedoso de la ontología, ya que no se ha encontrado un enfoque similar en ninguna de las ontologías estudiadas, se ha tenido que ampliar los conceptos de los estándares actuales para permitir la búsqueda y localización de objetos educativos de una forma más eficaz. Es el caso de las relaciones directas entre las actividades de evaluación y los objetivos de aprendizaje, o toda la implementación de ontología para las interacciones entre usuarios, como son los foros y comentarios. Estas ampliaciones facilitan que la información introducida a través de la interacción de los usuarios durante una experiencia educativa se almacene e indexe de la misma forma que los contenidos estáticos. Esta información introducida es una fuente para generar una base de conocimiento que podríamos aprovechar para localizar un objeto educativo o incluso para utilizar en otras imparticiones del curso. De esta forma podemos ir enriqueciendo un repositorio de experiencias educativas con la base de conocimiento generada por medio de la interacción de los usuarios en los foros. Este extremo nos permitirá ampliar las búsquedas, como ya se ha indicado a lo largo de este capítulo.

Tras realizar la especificación formal de la ontología, en el capítulo siguiente realizaremos una validación de la misma, es decir, la traduciremos directamente al lenguaje OWL DL y realizaremos una prueba de concepto, donde validaremos el sistema y veremos ejemplos del funcionamiento de las distintas partes de la arquitectura.

Capítulo 7: Validación de la Ontología OWL DL y Resultados

7.1 Introducción

Como se explica en los primeros capítulos de esta Tesis, actualmente la información publicada en la web no contiene información semántica para que pueda ser tratata por agentes inteligentes, y por tanto las búsquedas que se pueden realizar en esta información publicada se basa en búsqueda de palabras clave. Actualmente, con el desarrollo de la web semántica, van apareciendo algunas ontologías que pretenden subsanar la carencia de información entendible por los futuros agentes.

Por otro lado, en el campo de la teleeducación se está empezando a adoptar el paradigma de la reutilización, tras haber experimentado los altos costes de

la generación de materiales desde cero. Para que tenga lugar la reutilización de objetos educativos, tenemos que ser capaces de publicar y buscar dichos objetos de la forma más sencilla y automática posible. Es aquí donde entra la web semántica y la necesidad de generar ontologías en el campo de la teleeducación, que nos faciliten la interoperabilidad de los recursos educativos, preferentemente basándonos en estándares y especificaciones ya existentes.

Otra de las carencias de las iniciativas de teleeducación para la interoperabilidad es el desaprovechamiento de las interacciones de los usuarios de una experiencia educativa para enriquecerla, de forma que se pueda almacenar esa información para otras ocasiones, e incluso utilizar como elemento para la realización de búsquedas.

La ontología especificada en esta Tesis pretende suplir estas carencias, ya que dicha ontología facilita la interoperabilidad al estar elaborada para la web semántica. Incluye también conceptos novedosos hasta el momento como las interacciones entre los usuarios de una experiencia educativa. Esta ontología, apoyada en la arquitectura propuesta para su explotación, pretende además facilitar el uso de la información por parte de agentes inteligentes. Gracias al sistema definido, los usuarios interesados en buscar recursos educativos para su reutilización podrán apoyarse en agentes inteligentes que les facilitarán resultados más precisos, completos y organizados.

Tras desarrollar los puntos principales de la Tesis, que han sido la especificación de la ontología que potencie la interoperabilidad de los recursos educativos para la teleformación, y el diseño de una arquitectura basada en la web semántica para soportar el uso de esa ontología por agentes inteligentes; se procederá en este capítulo a validar el sistema implementando algunas de las partes del mismo. Esta prueba de concepto, seguirá los siguientes pasos:

- **Descripción en OWL DL de la ontología global**, tratando con especial atención los conceptos relacionados con los foros. Esta parte se basará en el capítulo anterior, dentro del cual se realiza la especificación de la ontología. Para realizar este paso se estudiaran distintas posibilidades y herramientas que facilitan esta labor de transformación, cuando ya está especificada una ontología en lenguaje lógico no web y se pretende traducir a OWL.

- **Importación a nuestro sistema de datos reales de foros** de experiencias formativas on-line siguiendo la ontología especificada en el capítulo anterior. Los datos que se importarán serán una recopilación de información relacionada con foros pertenecientes a experiencias educativas de los últimos años. Este tipo de extracción de información de los foros no se realiza en los sistemas actuales de formación on-line, ya que es una información no aprovechada que desaparece cuando se acaba una experiencia educativa. Para realizar esta importación nos basaremos en Jena, una infraestructura desarrollada en Java orientada a facilitar el trabajo para la web semántica.

- **Modelización de un agente inteligente**, que realice búsquedas, alimente su base de datos y realice también inferencias. Para realizar este modelo se compararán los distintos motores de inferencia para OWL DL para elegir el más adecuado para nuestra validación.

7.2 Generación de la ontología propuesta en OWL

A lo largo del capítulo 6 se ha especificado la lógica de la ontología propuesta, de forma que sólo nos resta implementarla en OWL, comprobar que cumple las restricciones OWL DL y validarla. Por desgracia, la novedad de los lenguajes utilizados hace que no existan muchas herramientas para este fin y las que existan sean en su mayoría componentes de proyectos de investigación que se encuentran aún en una fase de pruebas.

Por tanto, para plasmar la ontología en OWL DL se han estudiado las herramientas de autor para generación de ontologías más importantes y referenciadas en el campo de estudio. Dichas herramientas son:

1. **Generación directa** por medio de un editor de texto o editor XML estándar. Esta opción se basa en utilizar herramientas de capas de la web semántica inferiores, como puede ser XMLSpy para generar un documento OWL.

2. **Altova SemanticNetworks 2006**, la única herramienta comercial existente que ha sido desarrollada por los creadores de XMLSpy, uno de los editores de XML más utilizados. Su objetivo es facilitar a los autores de documentos RDF y OWL la elaboración y modificación sintáctica de los mismos.

3. **SWOOP 2.3 beta 3,** herramienta opensource para el diseño y compartición de ontologías OWL DL en la web.

4. **Protégé 3.1.1**, herramienta que permite el diseño de ontologías de forma generalizada y dispone de un plug-in para particularizarlas a la lógica OWL.

A continuación explicaremos por encima sus características y realizaremos una evaluación subjetiva sobre los entornos.

7.2.1 Generación directa

Esta opción realmente no es el uso de una herramienta de generación de ontologías, sino el uso de herramientas de más bajo nivel para generar los documentos. Se puede comparar con el uso del bloc de notas de Windows para generar documentos html. Si bien esta opción es válida para ejemplos sencillos y permite modificaciones muy puntuales, se descartó desde el principio debido a la complejidad de generación y depuración con esta técnica. Esta dificultad proviene del hecho de que, al no ser herramientas que entiendan la sintaxis de las ontologías, se fuerza al usuario a preocuparse más por la sintaxis que por la ontología en sí. Además, no disponen de herramientas directas de validación y chequeo de las ontologías, así como añadidos para realizar pruebas de inferencias. Sin embargo estas herramientas son importantes, ya que la mayoría de herramientas de generación de ontologías se construyen sobre éstas, debido a la estructura en capas de la web semántica.

7.2.2 Altova SemanticNetworks 2006

Esta herramienta está desarrollada en el 2006 por la compañía que desarrolló el famoso XMLSpy para el desarrollo de documentos XML [WWW52]. Para evaluarlo, al ser una herramienta propietaria se ha utilizado la licencia gratuita de 30 días que se ofrece en la descarga. Es una herramienta visual diseñada para documentos RDF, RDFS y OWL (Ver Fig. 7.1).

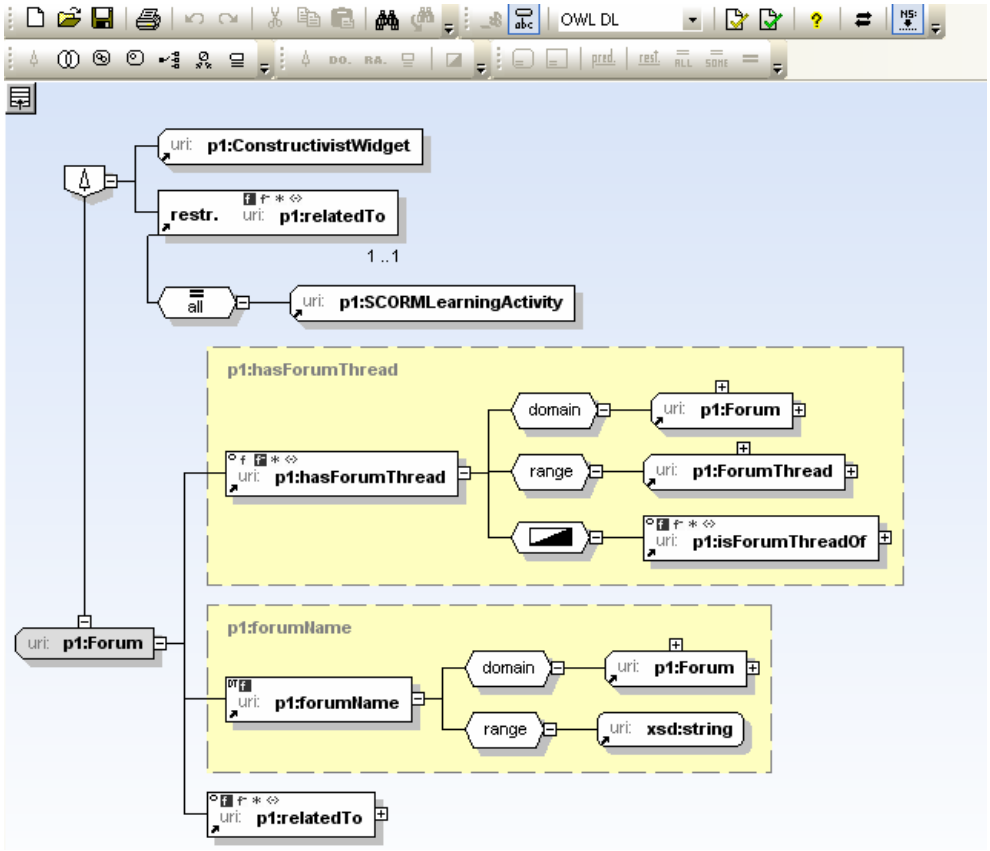


Fig. 7.1. Altova SemanticNetworks

Destaca de la herramienta que es muy intuitiva, probablemente debido a que se basa en la experiencia de otros productos de la compañía. Además consta de una ayuda bastante completa, y de una opción de visualización gráfica (como podemos ver en la figura 7.1) muy descriptiva.

Sin embargo es una herramienta muy orientada a RDF que dibuja sintácticamente todo como relaciones entre entidades (por ejemplo, una propiedad está relacionada por medio de "range" con la clase rango de la misma) sin permitir ampliar la ontología ni realizar inferencias.

Además, su estructuración en árbol se basa en poder desplegar todos los grafos RDF que contiene el recurso que desplegamos, sin realizar gran distinción (sólo un pequeño gráfico en la relación y un recuadro con borde distinto si es clase, instancia o propiedad) entre los casos en que es una relación de herencia (subclase) o es una propiedad, o cualquier otra relación. No chequea tampoco si una relación ya se ha mostrado en la estructura de

árbol, por lo que se puede llegar a un bucle de relaciones ya que podemos repetirla si hay propiedades inversas.

También se ha de de destacar que aunque permite revisar sintácticamente OWL en las versiones lite y DL, no está diseñado para realizar ningún tipo de inferencia ni para conectarse directamente con un motor de inferencia.

Todas estas características hacen que no haya aún empresas que se hayan decantado por esta herramienta, mientras que las herramientas de XML de la compañía, como XMLSpy sí son utilizadas en muchos casos. Si bien se evaluó la versión 2006, se ha comprobado que la versión 2007 no incluye grandes variaciones.

7.2.3 SWOOP 2.3 beta 3

SWOOP (Semantic Web Ontology Overview and Perusal) [WWW53] es una herramienta opensource desarrollada a partir del 2004 por MINDSWAP (Mariland INformation and Network Dynamics lab Semantic Web Agent Project) y que está actualmente (desde marzo 06) en su versión 2.3 beta 3.

Se basa en un interfaz web (Fig 7.2), es decir, SWOOP actúa como un navegador y permite navegar entre las clases (aprovechando las URIs de los recursos para acceder a ellos).

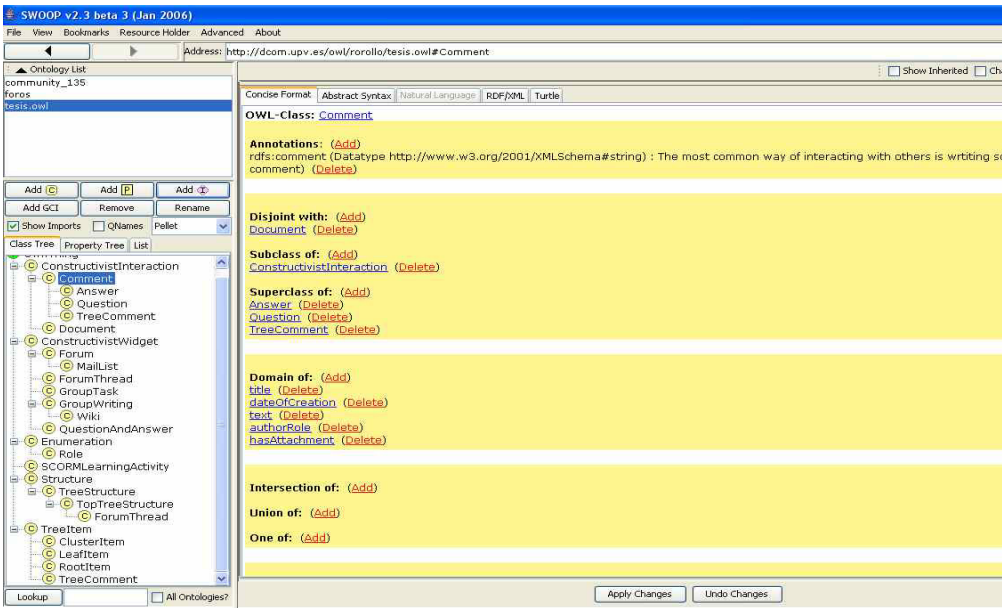


Fig. 7.2. SWOOP

En la figura 7.2 podemos apreciar el interfaz web gráfico. Al ser desde el principio una herramienta diseñada para OWL DL, diferencia totalmente entre clases, propiedades e individuos, teniendo a la izquierda una estructura en árbol para la navegación entre los recursos, separándolos en tres pestañas según sean clases, propiedades o individuos. Las estructuras en árbol se generan en función de las relaciones de herencia entre las clases y propiedades.

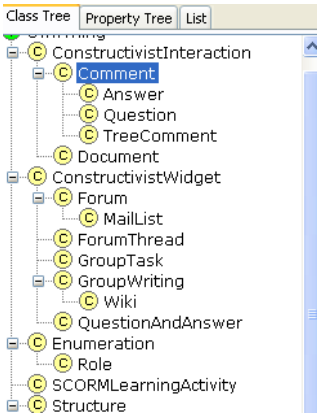


Fig. 7.3. Estructura de navegación por los recursos

Por ejemplo, en la figura 7.3 podemos apreciar que la clase *Answer* es una especialización de la clase *Comment*. La navegación es sencilla, ya que la estructura en árbol nos sirve de menú para buscar y mostrar a la derecha la clase, objeto o instancia que deseemos tratar.

SWOOP permite además, editar directamente los recursos por un sistema parecido a wiki [KAL04], de forma que realiza un control de versiones del recurso. La forma de editar los recursos está muy orientada a la sintaxis de OWL, en lugar de estar orientado a un sistema general de representación del conocimiento por medio de ontologías [KAL05].

SWOOP está orientado al desarrollo colaborativo de ontologías, de forma que se puedan realizar pequeñas ontologías entre distintos autores y conectarlas entre sí por medio de referencias o importaciones. Para facilitar esto dispone de un plug-in (denominado Annotea) para compartir anotaciones entre distintos autores sobre el mismo recurso, así como un sistema de control de versiones realizado desde un servidor de una ontología donde guarda todos los cambios realizados y permite actualizar la ontología para obtener la última versión.

7.2.4 Protégé 3.1.1

Prótegé es una herramienta opensource que permite la creación y edición de ontologías así como la creación de bases de conocimiento. Está desarrollada principalmente por Stanford Medical Informatics, con la colaboración de la Universidad de Manchester y cuenta con el apoyo de la National Library of Medicine. Aunque existían versiones desde 1995, Protégé 2000 fue la base del desarrollo actual.

Protégé dispone de dos formas de modelar ontologías:

- **Protégé-Frames** editor, que permite trabajar con un modelo de frames compatible con OKBC [CHA98]. No evaluaremos esta forma de operar al no ser la utilizada en la Tesis.

- **Protégé-OWL** (Figura 7.4) editor, que es el preparado para el diseño de ontología para la web semántica, basados en OWL o SWRL. Este editor es realmente un plug-in instalado sobre la versión de frames, que es con la que trabaja internamente Protégé.

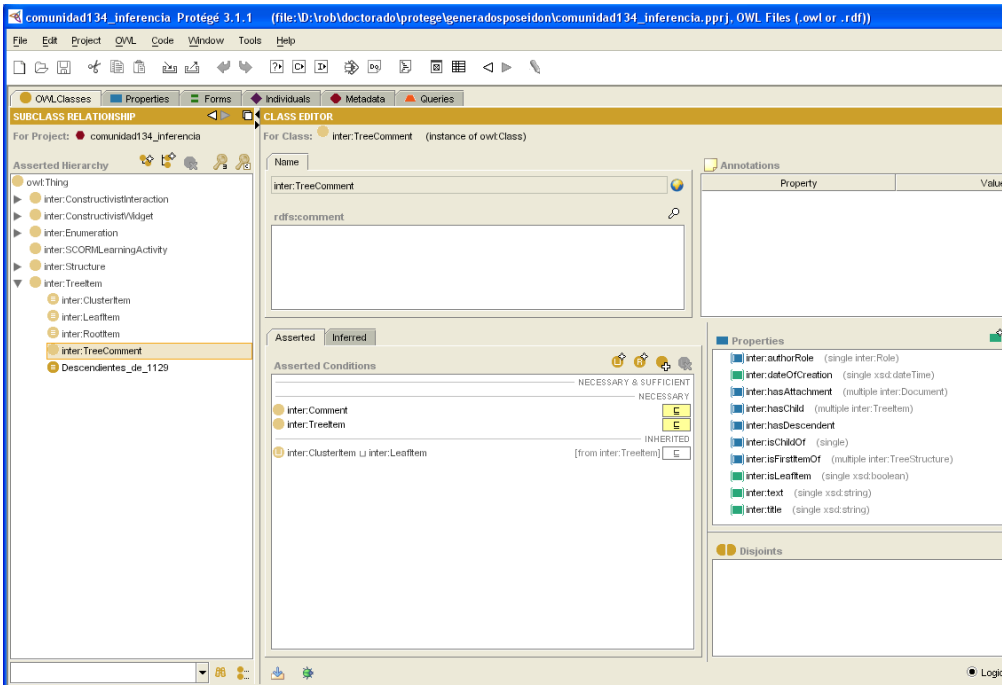


Fig. 7.4. Protégé-OWL 3.1.1

En la figura 7.4 podemos apreciar el espacio de trabajo de la aplicación Protégé, con también una estructura en árbol para clases, instancias y propiedades como ocurría en SWOOP, pero con un interfaz más amigable a la hora de editar un recurso en particular, ya que incluso representa las restricciones con operadores de lógica matemática.

Todas las características evaluadas a partir de este punto sobre Protégé se refieren al editor de OWL, ya que el objetivo de la herramienta es modelar una ontología en este lenguaje.

Como característica más destacada tenemos sus años de desarrollo continuo así como la comunidad de usuarios y desarrolladores que arrastra el proyecto (en julio 2006 más de 52.000 usuarios registrados y más de 2.500 usuarios apuntados a la lista de discusión de Protégé-OWL).

El hecho que haya gran cantidad de desarrolladores, y que se haya facilitado un estándar para añadir plug-ins a Protégé, hace que exista una gran cantidad de plug-ins para muchas utilidades, de forma que no sea necesario tenerlos todos, sino sólo importar y ver los que nos interesen. Por ejemplo, en la figura 7.5 podemos ver los plug-ins utilizados en nuestro caso (representados como pestañas), para realizar búsquedas en la sintaxis RDF (pestaña *Queries*) y para visualizar las ontologías (pestaña *Jambalaya*).

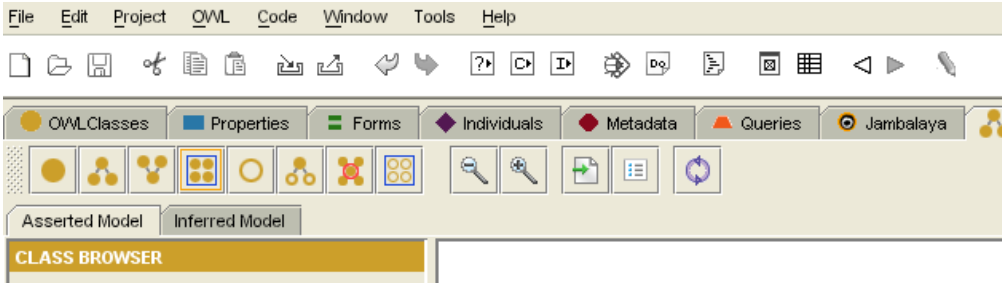


Fig. 7.5. Plug-ins en Protégé

La visualización de los recursos es mucho más visual que en SWOOP, separando completamente las clases de las instancias, las propiedades y las restricciones (Ver Fig 7.6).

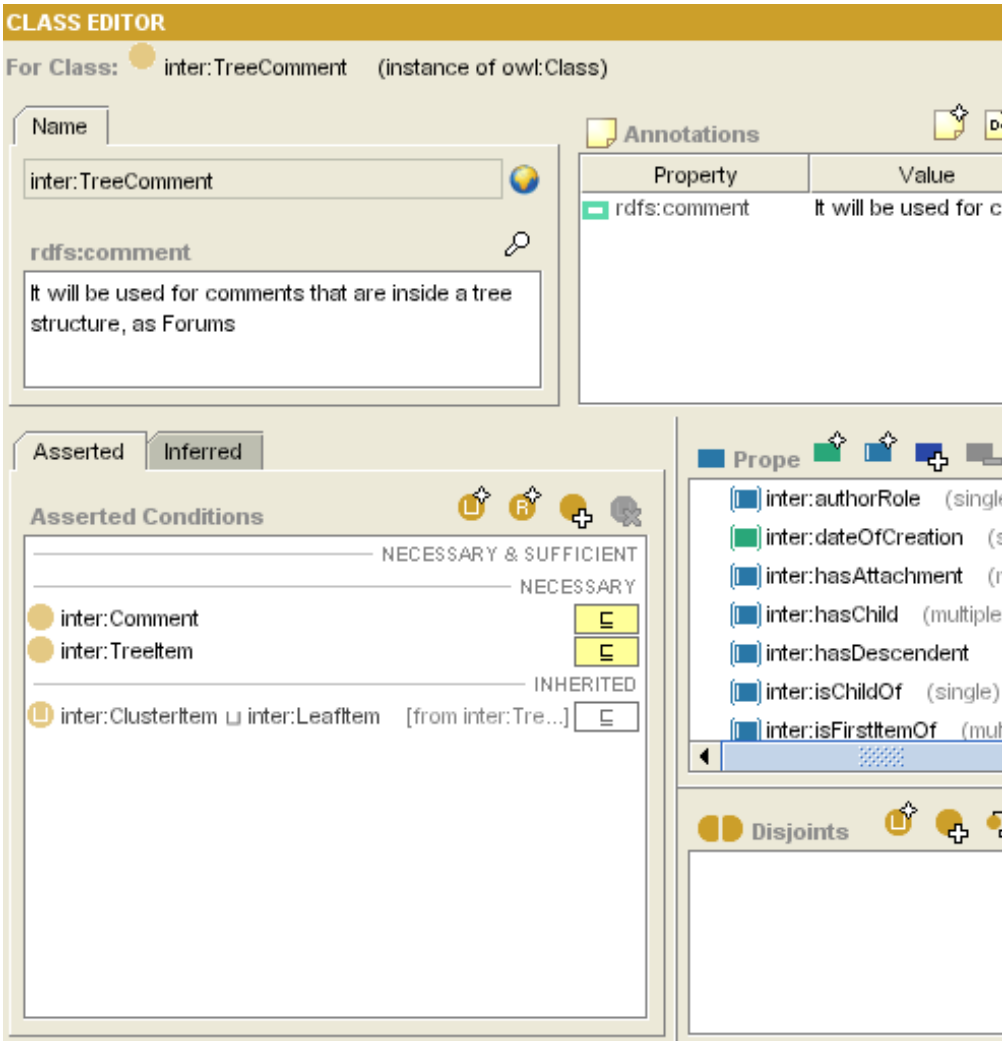


Fig. 7.6. Ejemplo de edición visual de una clase

Como SWOOP, permite la conexión directa con razonadores, pero en este caso separa entre el modelo que hemos diseñado y el inferido (añadiendo las inferencias que ha dado el razonador). Esta separación se da a nivel de herramienta, de modo que los plug-ins (por ejemplo, de visualización) pueden utilizar ambos modelos, como se muestra en la figura 7.7.

Otra ventaja de Protégé, es la presencia de templates y patrones de diseño para generación de algunas acciones que se repiten mucho en la definición de ontologías (y pedidos por la amplia comunidad de usuarios y desarrolladores de la herramienta). Por ejemplo, generar clases disjuntas directamente, generar enumeraciones, listas, etc.

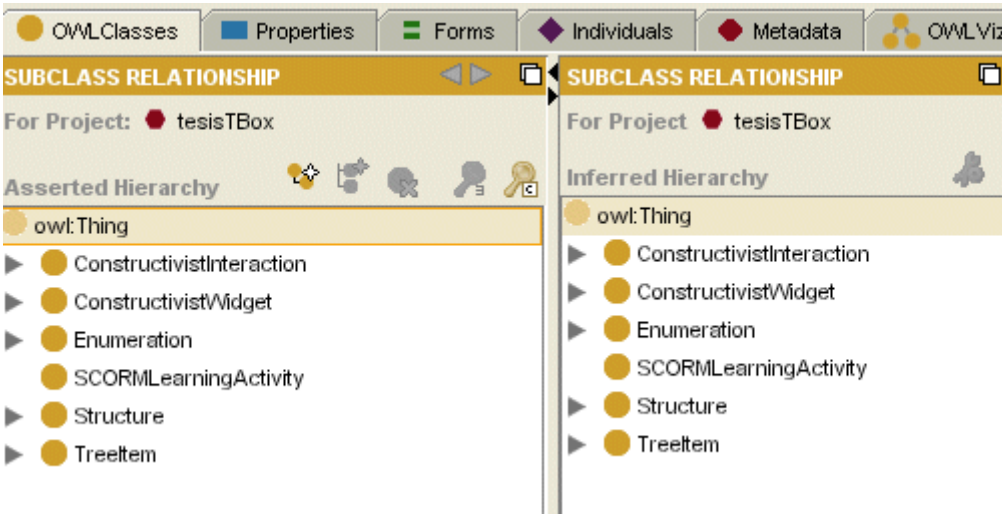


Fig 7.7. Ontología diseñada e inferida

7.2.5 Comparación de las herramientas

Para realizar este punto nos basamos en los resultados comparativos de la tabla 7.1.

En primer lugar descartamos Altova SemanticNetworks por su poca orientación hacia desarrollo de ontologías OWL, así como su poca conectividad con motores de inferencia (razonadores). Suponemos que en versiones más maduras incluirá estas propiedades.

Característica	SemanticNetworks	SWOOP	Protégé (versión OWL)
Evaluación objetiva			
Multiplataforma	NO	SI (Java)	SI (java)
Código abierto	NO	SI (MIT license)	SI (Mozilla PL license)
Freeware	NO	SI	SI
Conexión con razonadores	NO	SI (vía HTTP)	SI (vía HTTP)
API para utilizar	NO	SI	SI
Orientación a OWL	Baja	Total	Total (OWL, SWRL)
Cantidad de plug ins	NO	<5	>20

Visión abstracta de la base de conocimiento	Baja (representación sintáctica)	Media (todas las relaciones se ven iguales)	Alta (visión conceptual)
Evaluación subjetiva			
Facilidad de instalación	Alta	Alta	Media (plug-ins algo más costosos)
Facilidades para aprender el manejo	Alto (ayuda en el programa)	Bajo (videos de ejemplo en la web)	Alto (tutoriales completos)
Facilidad de uso	Alta	baja	media
Comunidad de desarrolladores implicados	Bajo (compañía)	Medio	Alto

Tabla 7.1. Comparativa de herramientas de autor de ontologías

En la Tabla 7.1 se han comparado características objetivas y subjetivas. En las objetivas podemos apreciar que SWOOP y Protégé están bastante igualadas, ya que son plataformas onpensource, muy orientadas al uso de ontologías en OWL y compatibles con el uso de razonadores. Protégé destaca al tener más ampliaciones, y al permitir trabajar con ontologías en un nivel de abstracción mayor, de forma que no es necesario ni conocer la sintaxis de OWL.

Sin embargo en la evaluación subjetiva se aprecia que Protégé obtiene mejores resultados, al ser más fácil e intuitiva de usar, y estar respaldada por una comunidad de desarrolladores más amplia. Sólo Protégé es un poco más complicada a la hora de instalar ampliaciones distintas a las estándar, aunque actualmente incluye en la versión de instalación la mayoría de ampliaciones recomendables para el trabajo con OWL.

Por tanto, entre SWOOP y Protégé elegimos Protégé, ya que es una herramienta más visual, con más recursos de ayuda y con una comunidad de desarrollo y detección de errores considerable que hace que el producto sea un producto bastante estable. SWOOP tiene un grado de madurez más bajo que Protégé, como se indica en [RUT05].

7.2.6 Generación de la ontología

Hemos seleccionado la herramienta Protégé como la más idónea para traducir a OWL DL la ontología para teleeducación especificada por medio

de lógica descriptiva en el capítulo anterior. Para realizar dicha traducción, en la que generaremos un documento OWL DL que describa las clases de la ontología, es decir, la información no relativa a instancias (TBox) de la misma.

La ontología de las clases (TBox) se genera siguiendo los axiomas presentados en el capítulo de especificación de la ontología, siguiendo los siguientes pasos cíclicamente en un proceso de refinamiento iterativo.

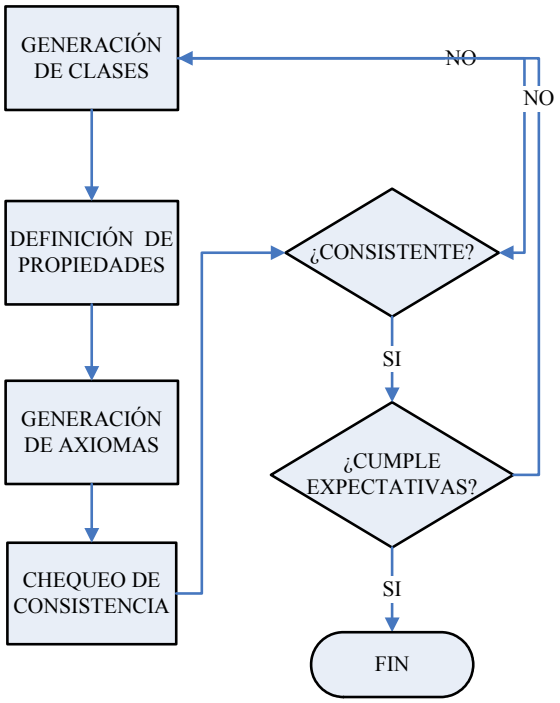


Fig 7.8. Procesos de refinamiento iterativo

En la figura 7.8 podemos ver el esquema del proceso de refinamiento iterativo que consiste en:

1. **Generación de las clases primitivas y su relación jerárquica directa**. Siguiendo la guía de [NOY01], se deja como clases los conceptos que vayan a ser tratados como tales e la ontología y se colocará luego como instancias el nivel más pequeño de granularidad y que representa un elemento real y no un concepto en la ontología. La definición de estas clases se realizó ya en la especificación de la ontología.

2. **Definición de las propiedades de las clases**, basándonos en las posibles necesidades (en este caso de búsquedas y de relación entre objetos) del sistema. De nuevo fue parte de la especificación.

3. **Generación de axiomas y las clases definidas por medio de los mismos** (como es el caso de TopTreeStructure, RootItem, LeafItem, ClusterItem). Es decir, después de ver el sistema de clases es cuando aparece la necesidad de generar axiomas que los definan y restrinjan.

4. **Chequeo de la consistencia de la ontología**. Chequear la ontología significa asegurar que no existen contradicciones en la misma y ver las relaciones de herencia que se pueden inferir de las aserciones. Son operaciones equivalentes a las descritas en el apartado 2.6.4 del capítulo 3, donde se hablaba de los razonamientos en la lógica descriptiva. Para los chequeos de la ontología se han utilizado tanto el razonador Racer 1.9 como Pellet 1.3 beta 2. Si el chequeo muestra incongruencias, volveremos a repasar desde el primer punto. En caso de ser positivo, estudiaremos el conjunto para ver si cumple los requisitos que esperamos (es decir, que representa fielmente los conceptos que deseo describir y utilizar) y si no es así volveremos al inicio para añadir nuevas clases, propiedades o axiomas en ese orden. Si tras chequear la consistencia la ontología cumple nuestras expectativas damos el proceso por concluido.

Este proceso iterativo se ha aplicado para las siguientes ontologías, que son en su conjunto la ontología global de e-learning:

I. **Ontología de la estructura de árbol**, con la consiguiente comprobación de que las propiedades estaban correctamente definidas y que se podían inferir padres, ancestros y elemento raíz de un árbol.

II. **Ontología de los contenidos estáticos**, introduciendo los conceptos de estructura de árbol. Una vez pasada a owl y chequeada no se ha utilizado más, ya que la parte sobre la que realizaremos la prueba de concepto es otra.

III. **Ontología de interacciones entre el alumno y el sistema**, basándose en la ontología de contenidos estáticos (ya que son una especialización de la clase *ScormLeafLearningActivity*), introduciendo las listas de tipo *rdf:List* necesarias para su implementación. Se genera y se chequea su consistencia.

IV. **Ontología de las interacciones entre usuarios**. Esta ontología se ha unido con la ontología de la estructura de árbol y luego se ha generado la conexión correspondiente (mediante la propiedad *relatedTo*) con la estructura de contenidos estáticos.

    En este punto destacamos la generación de la clase enumerada *Role*. La razón por la que esta clase se define por medio de sus instancias es la necesidad de limitarse a OWL DL, por lo que las propiedades sólo pueden ser de tipos de datos o de objeto, por lo que cuando definamos el rol que tiene en el curso el autor de un comentario, podamos relacionarlo con una instancia de la clase *Role*.

Hemos de destacar que la ontología se genera con denominaciones en inglés, para facilitar la publicación de resultados en foros internacionales. La URI que sobre la que se generará la TBox será:

```
http://dcom.upv.es/owl/rorollo/tesis.owl
```
[Expr. 7.1]

Aunque lo más recomendable es que esa URI fuera una URL que nos permitiera obtener el archivo .owl (es decir, que con un navegador pudiéramos acceder al archivo owl con esa URL), no es necesario, ya que sólo indica un identificador único para la ontología.

El resto de ontologías creadas se generarán sobre URIs que compartirán la misma raíz, y que contendrán las instancias de los distintos foros:

```
http://dcom.upv.es/owl/rorollo/
```
[Expr. 7.2]

Aunque podríamos haber asignado la misma URI a todas la ABox (incluso que fuera la misma que la de la TBox), para aclarar mejor los ejemplos preferimos separar las URIs y realizar importaciones para unir las ontologías en una base de conocimientos única.

7.2.7 Chequeo de las ontología

Para realizar el chequeo de las ontologías se sigue el siguiente esquema (Ver Fig.7.9):

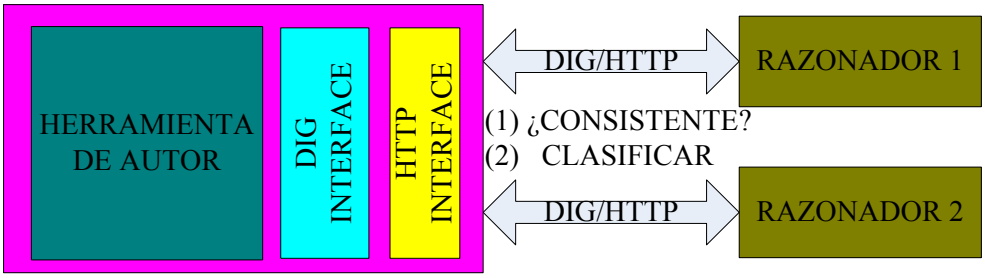


Fig. 7.9. Esquema de trabajo para el chequeo de ontologías

Como podemos ver en la figura 7.9, tras desarrollar la ontología en la herramienta de autor, la misma herramienta ofrece una conexión vía DIG/HTTP con razonadores que estén dando el servicio de chequeos de ontologías que están a la espera a modo de servidores HTTP.

Se aprovecha la ventaja de conexión por medio de DIG/HTTP [BECH03] que nos ofrece Protégé para activar dos razonadores (Racer y Pellet) como servidores HTTP en los puertos 8080 y 8081 de nuestra máquina de pruebas. Se describirán las características de los razonadores más adelante.

En cada momento de comprobación de la ontología se realizan dos pasos:

- **Chequeo de consistencia** de la ontología, es decir, ver que no hemos definido ninguna clase que no pueda tener nunca instancias.

- **Clasificación de las clases de la ontología**. Esto hace que el razonador vea si puede inferir más relaciones de herencia o equivalencia entre clases que los indicados por el autor.

En la figura 7.10 podemos ver ejemplos de chequeos de consistencia y clasificación.

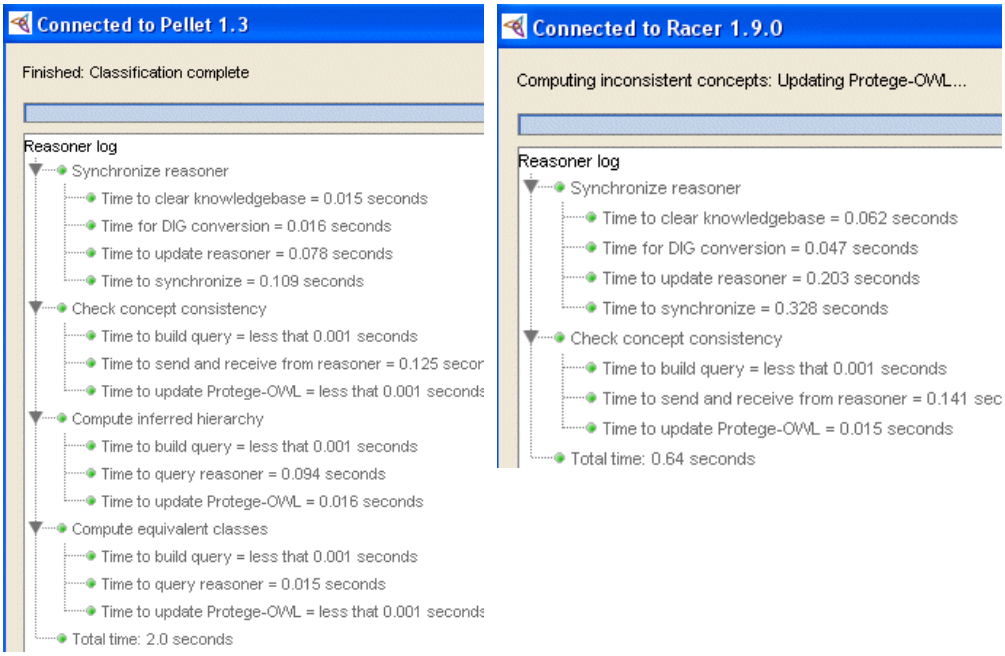
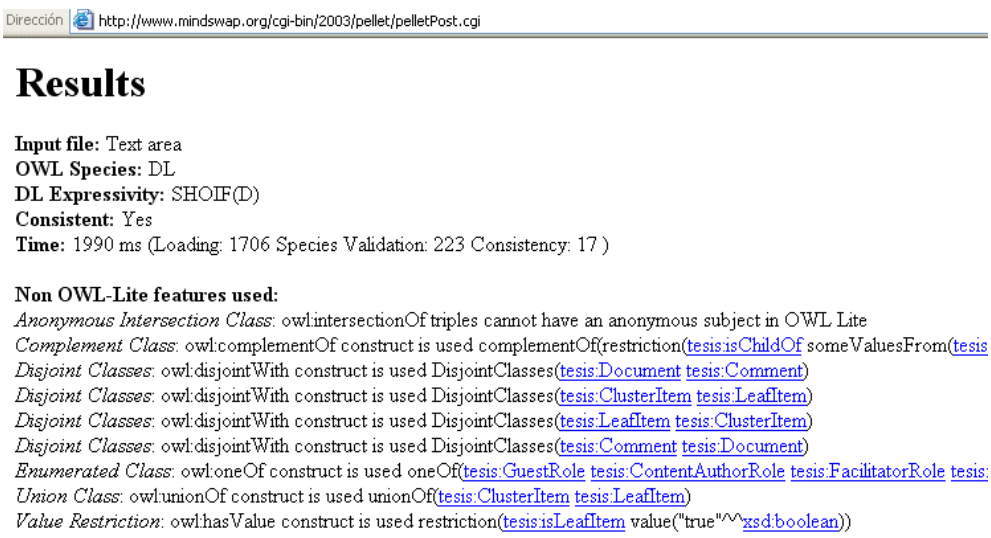


Fig 7.10. Chequeo y Clasificación con distintos razonadores

Como último paso, las ontologías finales OWL se chequean a su vez por medio de la versión web de Pellet [WWW54], para corroborar el nivel de OWL utilizado (Lite, DL o Full). En la Fig 7.11 vemos los resultados de chequear nuestra TBox. Se ratifica que cumplen la sintaxis OWL DL. La razón por la que se chequea la ontología como archivo .owl es la indicación de algunos autores [RUT05] que constatan que el interfaz DIG no permite indicar restricciones cardinales sobre propiedades de tipo de datos. Esto puede hacer que, utilizando DIG, se den ontologías como consistentes cuando puede que haya alguna inconsistencia. De todas formas nuestra ontología no utiliza este tipo de restricciones.


Dirección http://www.mindswap.org/cgi-bin/2003/pellet/pelletPost.cgi

Results

Input file: Text area
OWL Species: DL
DL Expressivity: SHOIF(D)
Consistent: Yes
Time: 1990 ms (Loading: 1706 Species Validation: 223 Consistency: 17)

Non OWL-Lite features used:
Anonymous Intersection Class: owl:intersectionOf triples cannot have an anonymous subject in OWL Lite
Complement Class: owl:complementOf construct is used complementOf(restriction(tesis:isChildOf someValuesFrom(tesis
Disjoint Classes: owl:disjointWith construct is used DisjointClasses(tesis:Document tesis:Comment)
Disjoint Classes: owl:disjointWith construct is used DisjointClasses(tesis:ClusterItem tesis:LeafItem)
Disjoint Classes: owl:disjointWith construct is used DisjointClasses(tesis:LeafItem tesis:ClusterItem)
Disjoint Classes: owl:disjointWith construct is used DisjointClasses(tesis:Comment tesis:Document)
Enumerated Class: owl:oneOf construct is used oneOf(tesis:GuestRole tesis:ContentAuthorRole tesis:FacilitatorRole tesis:
Union Class: owl:unionOf construct is used unionOf(tesis:ClusterItem tesis:LeafItem)
Value Restriction: owl:hasValue construct is used restriction(tesis:isLeafItem value("true"^^xsd:boolean))

Fig 7.11. Validación OWL de Pellet para nuestra TBox

Como vemos en la figura 7.11, el resultado de enviar nuestro fichero OWL al chequeador Pellet nos devuelve que es del tipo DL ("OWL Species= DL"). Pellet indica también qué restricciones tendríamos que eliminar para convertir el fichero en OWL Lite.

Tras realizar esta última comprobación concluimos que la ontología plasmada en OWL DL no tiene incongruencias y además cumple con todas las restricciones definidas en el capítulo de especificación de la ontología. A partir de este punto realizaremos una prueba de concepto que permita demostrar que la ontología especificada y la arquitectura propuesta son válidas para los objetivos planteados, es decir, que se puede extraer la información de las experiencias educativas de forma correcta y que los agentes inteligentes pueden utilizar dicha información para encontrar los recursos educativos adecuados.

7.3 Importación de los datos reales de foros de experiencias formativas on-line

En este apartado describiremos con detalle el sistema diseñado para importar a nuestra ontología propuesta la información sobre los foros de las experiencias formativas on-line. Detallaremos el origen de la información sobre foros reales, el entorno de desarrollo del sistema, algunos puntos importantes de la programación y por último comentaremos los resultados contenidos. Por tanto, en este apartado emularemos el subsistema extractor

de información semántica del publicador propuesto en nuestra arquitectura diseñada en capítulos anteriores (ver Fig. 7.12),

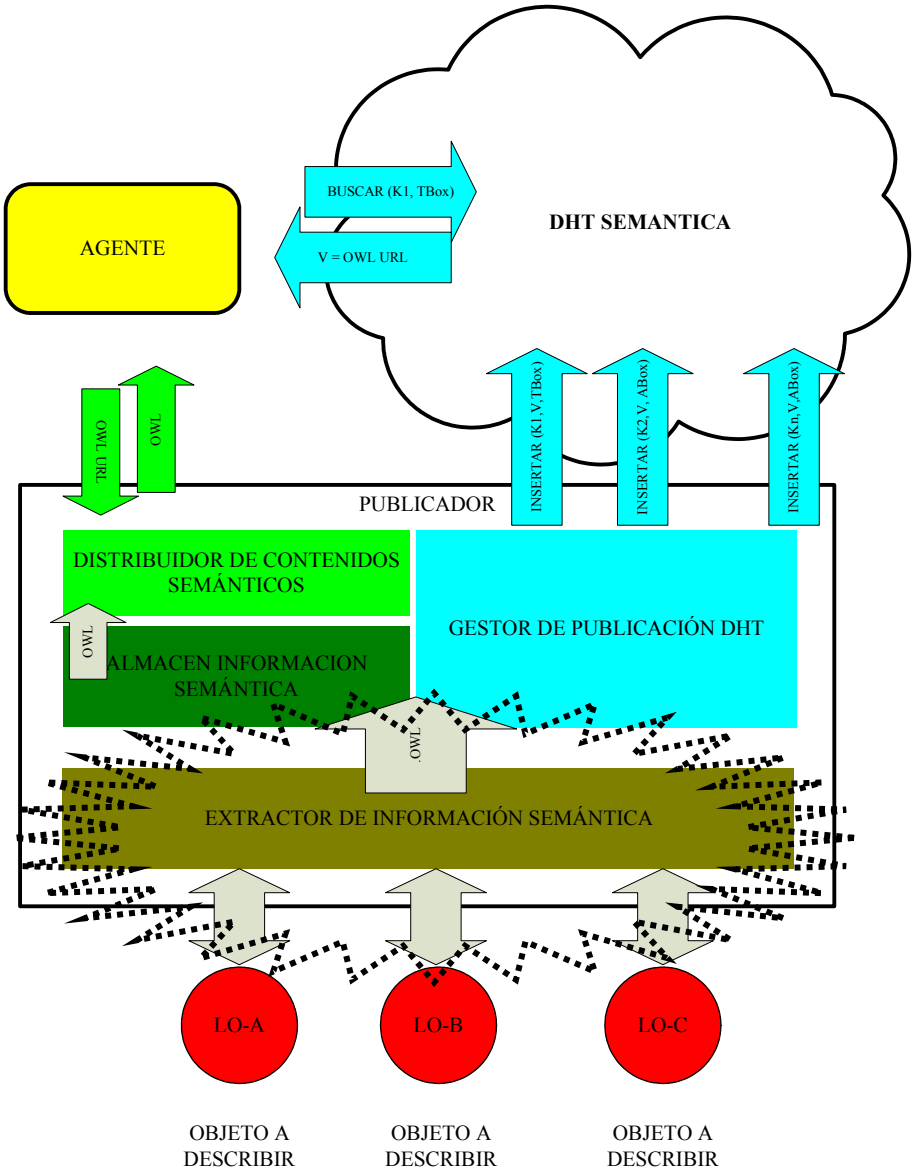


Fig 7.12. Parte de la arquitectura emulada en este apartado

Como se indica en la figura 7.12, la parte que vamos a implementar es la de extracción de información semántica de una experiencia educativa ya existente, de forma que esta información pueda ser publicada en la web semántica y utilizada por los agentes inteligentes.

7.3.1 Origen de los datos

La información utilizada para la comprobación de las ontologías proviene de foros reales de experiencias formativas on-line almacenadas en el Centro de Formación Permanente (CFP) de la Universidad Politécnica de Valencia.

Se utilizará un conjunto de foros terminados hace varios años, siendo todos de ediciones distintas del mismo curso (es decir, foros de cursos realizados con los mismos contenidos estáticos, pero generados en distintas comunidades virtuales). En total han sido 31 ediciones distintas las utilizadas. Las identificaremos por su identificador único de comunidad virtual en el sistema de información del CFP.

Respecto a la implementación que se ha realizado en el CFP de los foros en el sistema destaca lo siguiente:

- **Un curso puede tener múltiples foros**, pero no se contempla el concepto de Thread. Nosotros consideraremos, por tanto, a cada foro como con un único Thread, cuyos datos completaremos con el primer mensaje del foro.

- **Los mensajes de los foros no tienen por qué tener por obligación un título del mensaje**. Lo que haremos en los casos que no exista título será poner los primeros 20 caracteres del texto a modo de sindicación. Los que tenga un título mayor a 20 caracteres se les recortará a 20 caracteres.

- **Los mensajes de los foros identifican a su autor directamente**, no por su rol. Esto lo resolveremos descubriendo el rol del autor y colocándolo en la propiedad del rol.

El curso en cuestión está diseñado para ser un curso que le cueste al alumno unas 45 horas de dedicación, y cuenta con talleres que han de entregar los alumnos de forma individual, no en grupo. Esto debería forzar a los alumnos a interactuar más, al menos con el tutor del curso sobre los trabajos a realizar.

El sistema de información del CFP se basa en una arquitectura de tres capas (ver Fig. 7.13):

- **Capa de almacenamiento**, siendo en este caso una base de datos MS SQL Server 2003. En esta capa se almacenan todas las tablas

necesarias para el funcionamiento del sistema de información del centro.

- **Capa de lógica de negocio**, donde se implementan los conceptos necesarios para el sistema (cursos, foros, matrículas, personas, roles, etc) y los métodos que interactúan entre ellos y el interfaz de acceso a la capa de almacenamiento. Esta capa está desarrollada en Java.

- **Capa de visualización**, donde se presenta al usuario la información y los elementos necesarios para que el usuario interactúe con el sistema. También se encarga de instanciar elementos de la lógica de negocio y ejecutar sus métodos. Esta parte se ejecuta por medio de JSPs y JSFs y se explota por medio de baterías de Apache-Tomcats.

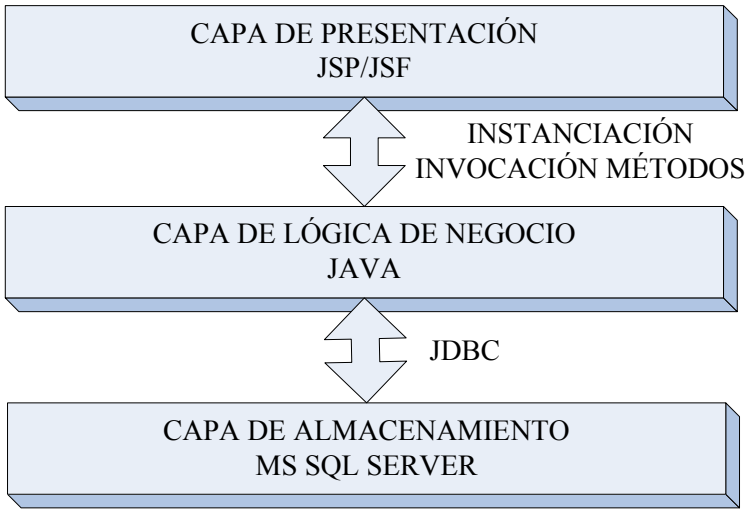


Fig. 7.13. estructura de capas

La capa de almacenamiento se encuentra en un servidor dedicado, y cuenta con otro servidor de reserva (no dedicado) que se actualiza periódicamente con los datos del primero.

La capa de lógica y visualización se ejecutan sobre varios servidores Apache que se encargan del reparto de contenidos por medio de HTTP y HTTPS y una batería de Tomcats sobre varias máquinas (con sistemas operativos linux o windows de forma indistinta), de forma que se encuentran conectados a un único Apache, pudiendo no ser la máquina que ejecuta el Apache la que contiene al Tomcat.

7.3.2 Conexión con el origen de los datos

Para implementar la importación a nuestro sistema se utilizan las mismas herramientas que se utilizan en el CFP para programación, lo que nos facilitará la conexión con las fuentes utilizadas:

- **Programación en java**, lo que nos permitirá reutilizar las clases de lógica de negocio ya utilizadas en el centro. El resultado de la programación se integrará en el paquete cfp.poseidon.util, un paquete de clases de java de tipo genérico que utiliza los paquetes java de lógica de negocio del sistema.

- **El entorno de desarrollo Netbeans 5.0**. Este entorno de desarrollo, junto con el entorno de desarrollo Eclipse es el entorno de desarrollo java utilizado en el centro. Es un entorno opensource que permite el trabajo con la capa lógica y la capa de presentación de forma integrada.

- **Sistema de programación colaborativa** basado en el control de versiones por medio de Subversion 1.3, integrado dentro del entorno Netbeans. Toda la programación del centro se lleva de forma colaborativa, de forma que existe un acceso centralizado a las fuentes, histórico de cambios y autores de los mismos.

La conexión con los foros del centro se hará a través de la capa de lógica de negocio, por medio de clases ya programadas en el centro que son:

- **cfp.poseidon.core.Roles**, clase que abstrae la información sobre los roles disponibles por parte de un cliente.

- **cfp.poseidon.core.Comunidades**, clase que abstrae el concepto de comunidad virtual para la realización de una experiencia formativa. A partir de él parten la información sobre contenidos estáticos, foros, chats, roles, fechas de actividad, etc...

- **cfp.poseidon.core.ComunidadBuscador**, clase que nos permite definir los parámetros de búsqueda en el sistema para buscar un conjunto de comunidades virtuales.

- **cfp.poseidon.core.ComunidadForo**, que se refiere a la información necesaria para describir un foro de una comunidad.

- **cfp.poseidon.core.ComunidadForoMensaje**, que se refiere a la información de un mensaje perteneciente a un foro.

Todas estas clases, excepto la de *cfp.poseidon.ComunidadBuscador*, llevan una clase de tipo *Manager* (por ejemplo, *cfp.poseidon.mgr.RolesManager*) que nos permite conectar con la base de datos para acceder a instancias de la misma buscadas únicamente por medio de campos de la tabla que lo describe. Los buscadores, como *ComunidadBuscador* nos permiten buscar por medio de características que afectan a más de una tabla de la base de datos.

7.3.3 Jena

Jena [WWW57] es una iniciativa de software abierto que cuenta con el apoyo de HP Labs Semantic Web Program. Su primera versión Jena 1 (2001) se centraba en grafos y su transformación a XML/RDF y en su siguiente versión (Jena 2) añade por encima una capa para tratamiento de ontologías (2003) y su transformación a OWL. Jena permite además conectarse a Razonadores por medio de DIG [DICK04]. Por tanto Jena se puede considerar como una infraestructura para trabajar con la web semántica programando con Java.

Para implementar el paso a OWL de la información disponible se barajaron las dos soluciones más referenciadas a la hora de generar archivos OWL de forma automática cuando se dispone de una TBox ya definida:

1. Utilizar la API ofrecida por Protégé para el manejo de ontologías.

2. Utilizar la API de Jena.

Ambas APIs están desarrolladas en Java (lo que permite una integración directa con los sistemas del CFP), son opensource y comparten muchas características comunes. Pese a utilizar Protégé como herramienta de autor para el manejo de ontologías de forma visual, se decide el uso de Jena porque:

- La API de Protégé **incluye mucha más complejidad**, ya que permite realizar muchas más acciones.

- Para el interfaz con RDF/OWL la API de **Protégé utiliza Jena** para la transformación.

Por tanto será mucho más óptimo utilizar directamente Jena que otra API que se construye sobre Jena, ya que lo único que realizaremos en esta fase es la transformación a OWL siguiendo la ontología TBox comentada en apartados anteriores.

Jena proporciona una API que nos permite leer y escribir en los formatos XML/RDF [WIL03], N3 y N-Triples (formatos para grafos abreviados). Además dispone de un motor para poder buscar dentro de los documentos RDF por medio de RDQL. Por último ofrece también mecanismos para leer y escribir en formato OWL. Esta última propiedad es la única que utilizaremos de Jena.

Jena trabaja con la clase *Model*, que es por donde podemos acceder a la información del documento RDF. En el caso de ampliación de RDF a ontologías propiamente dichas (con clases, instancias, propiedades, etc), la clase *Model* se especializa dando lugar a la clase *OntModel*, es decir, la ontología con la que está trabajando el sistema.

Además trabaja con los conceptos (interfaces de Java para permitir polimorfismo) que presentamos a continuación (ver Fig 7.14):

- **Resource**, entendido como cualquier recurso definible en RDF. Es un interface de java de forma que los siguientes conceptos son subclases de este interface.

- **OntClass**, que representa una clase.

- **Individual**, interface que representa una instancia de una clase.

- **Property**, que representa la propiedad de relación entre Resources. Se divide en ObjectProperty (para propiedades con un objeto de tipo instancia) y DatatypeProperty (para propiedades de tipo de datos).

- **Literal**, que es un Resource que representa un elemento de tipo de datos (xsd:).

- **Statement**, que representa la frase que une un sujeto por medio de una propiedad con un objeto.

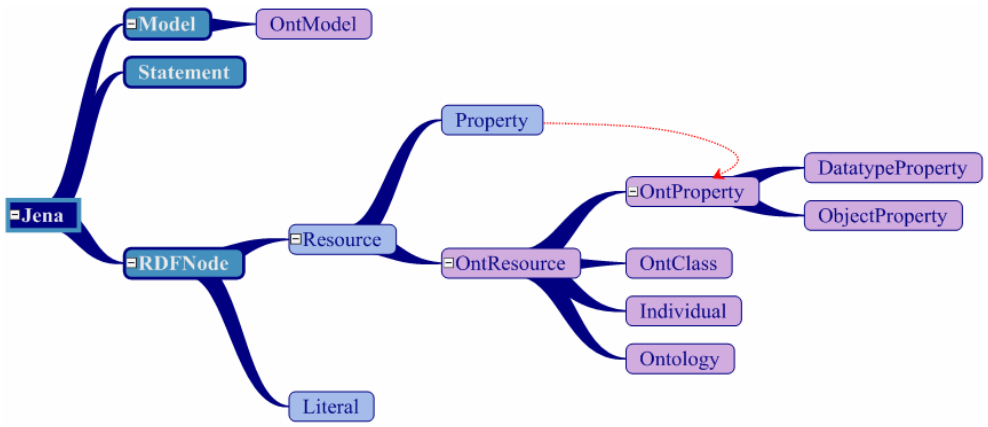


Fig. 7.14. Herencia entre los interfaces de Jena 2

En la figura 7.14 podemos ver la relación de herencia entre los distintos interfaces de Jena 2, siendo las de color azul las relativas a OWL y las de color morado las relacionadas con OWL propiamente dicho.

Jena 2 trabaja con muchos más conceptos, pero como en nuestro caso no se va a utilizar para definir la TBox (que ya se ha hecho con Protégé) no es necesario utilizar la API para indicar propiedades de tipo funcional, restricciones, herencias, etc. En nuestro caso sólo nos interesa la parte de poblar las ontologías con instancias, es decir, crearlas y colocar valores a las propiedades de las clases que ya están definidas.

Los métodos que más utilizaremos son:

- Sobre objetos de **tipo Model**

  a) **Resource createResource(java.lang.String uri)**, que devuelve un objeto *Resource* que es uno nuevo si no existía en el modelo con esa URI o el que existe si ya hay definido alguno en con esa URI. En nuestro caso, al importar la TBox tendremos ya definidas los objetos *Resource* de las clases (que cumplirán también el interfaz de *OntClass*, pero eso no nos afecta).

  b) **OntClass createClass (java.lang.String uri)**, que devuelve la representación de una clase. Este método lo utilizaremos solamente cuando necesitemos algún método de la clase *OntClass* que no se ofrezca desde el interfaz *Resource*.

c) **Property createProperty(java.langString uri)**, que funciona de forma parecida al método anterior pero devuelve ya un objeto que cumple el interfaz *Property*. Esto nos ahorra convertir un objeto *Resource* a uno *Property* por medio de

```
Resource res=...[creamos el Resource].
Property p=(Property)res.as(Property.class)
```
[Expr. 7.3]

d) **Individual createIndividual (java.lang.String uri)**, que igual que el anterior genera exactamente una instancia con esa URI. Si la instancia con esa URI ya existe, no duplica (esto nos servirá para cuando obtengamos los Roles).

e) **Literal createTypedLiteral(String value, RDFDatatype dataType),** que genera un literal con el valor y el tipo de datos (datatype) que le digamos nosotros (en algunos caso es suficiente colocar el objeto y Jena decide el tipo de datos). Por ejemplo, para crear xsd:boolean o xsd:dateTime. Para el tipo de datos xsd:string no es necesario y se puede hacer directamente como vemos a continuación.

f) **Statement createStatement(Resource s, Property p, RDFNode o)**, genera una frase con sujeto s, objeto o y predicado p. Se utilizará este método sólo para las propiedades de tipo de datos donde hemos de colocar como objeto de la frase un Literal (*RDFNode*). Es equivalente al método de las instancias de *Resource* "(*addProperty(Property p, RDFNode o*)". Jena no proporciona métodos directos para añadir propiedades de tipo de datos a los objetos de la clase *Individual*.

g) **Model add(Statement stm)**, método que estamos obligados a utilizar ya que cuando se crea un objeto de la clase *Statement* no se añade al modelo hasta que no ejecutamos este método.

- Sobre objetos de **tipo Individual** (son métodos de su superInterface Resource) utilizaremos:

a) **Individual addProperty(Property p, Individual i)**, que añade de nuestro modelo una relación entre la clase que ejecuta el método y la instancia de *Individual* i por medio de la propiedad (representada por la clase *Property*) p. El método realmente admite cualquier *RDFNode* (que admite una clase Literal) y no sólo instancias de Individual.

b) **Individual addProperty(Property p, String s)**, que añade una propiedad de tipo xsd:string.

Como podemos observar, Jena 2, aunque permite trabajar con ontologías y conectar razonadores que implementen un interface determinado, sigue estando muy orientado a RDF (trata de nodos, statements, grafos, etc). Sin embargo, para la importación es un punto poco importante, ya que la lógica de la ontología de clases (TBox) ha sido desarrollada por medio de otra aplicación y siempre será más rápido generar instancias usando RDF (lo que significa trabajar a un nivel de abstracción más bajo nivel de abstracción).

7.3.4 Programación de la importación

La importación seguirá el siguiente esquema (figura 7.15)

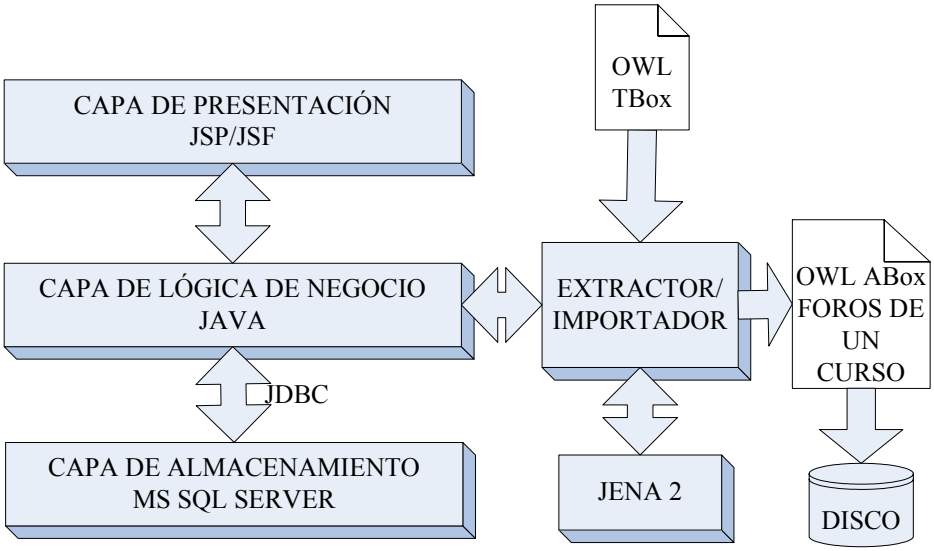


Fig. 7.15 esquema de la importación de foros a OWL

Se programará un importador o extractor de información que utilizará la API de Jena 2 y las clases mencionadas anteriormente de la capa de lógica de negocio del sistema de información del CFP para introducir la

información de nuestra TBox, obtener los datos de los foros y transformarlos a documentos OWL que almacenaremos en ficheros.

El exportador se basará en la clase java *ExportaInteracciones.java*, y los pasos que se realizan por cada exportación son:

1. **Inicialización** del objeto *OntModel* y el objeto *Ontology* con su sus espacios de nombres correspondientes.

[Expr. 7.4]

```
OntModel aBox =

ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,null);
//generamos el namespace
aBox.setNsPrefix(tBoxPrefix,tBoxNs);
String
aBoxURI=rootURI+"community_"+this.getIdComunidad()+".owl";
aBoxNs=aBoxURI+"#";
//creamos la ontologia nueva
Ontology ontABox=aBox.createOntology(aBoxURI);
```

El nombre que se le da a la URI de la ontología de cada comunidad será la raíz común más el identificador de la comunidad dentro del sistema del CFP. Por ejemplo, para la comunidad 128 tenemos:

```
http://dcom.upv.es/owl/rorollo/community_128.owl
```
[Expr. 7.5]

2. **Importamos la TBox**.

[Expr. 7.6]

```
ontABox.addImport(aBox.createResource(tBoxURI));
//marcamos la ontología importada para no importarla más
veces
aBox.addLoadedImport(tBoxURI);
```

3. **Insertamos en la ontología los foros** de la comunidad. Para introducirlos los colocamos con URIs con prefijo “ForumID_” y el identificador único del foro en el sistema.

[Expr. 7.7]

```
ComunidadForo f=...//cargamos de la base datos la información
del Foro
//generamos el foro y le añadimos propiedades
Resource res=aBox.createResource(tBoxNs+FORUM);
Individual ind
    =aBox.createIndividual(aBoxNs+FORUM+"ID_"+f.getId(),res);
//ponemos el nombre del Foro
prop=aBox.createProperty(tBoxNs+FORUMNAME);
ind.addProperty(prop,f.getNombre()+"");
```

Para insertar el resto de instancias se utilizarán prefijos para las URIs de "ForumThreadID_" y "TreeCommentID_", con los identificadores respectivos en cada caso.

Además, en cada caso se cargará, proveniente de la base de datos, la información de la instancia correspondiente, como se ha hecho aquí con la *cfp.poseidon.core.ComunidadForo*.

Es importante destacar en este punto que Jena, en la mayoría de métodos que modifican la ontología (añadir recursos, instancias, propiedades de la mismas,...) devuelve el objeto instancia o recurso al que estamos afectando. De esta forma tenemos una referencia al mismo sin tener que buscar de nuevo en el modelo la instancia que hemos colocado o la instancia a la que le hemos colocado una propiedad. Esta propiedad es muy útil y te permite encadenar aserciones y tener siempre referencias a los recursos que estás generando o modificando. Del mismo modo nuestros métodos de introducir foros, mensajes, etc, devuelven una referencia a la instancia generada, para que podamos utilizarla en las siguientes referencias.

4. **Para cada foro encontramos el primer mensaje** (el que no tiene padre) y utilizamos sus datos para generar el ForumThread y el RootItem correspondiente. En los métodos correspondientes que introducen en nuestra ontología estos conceptos se añade información sobre su padre, que llamaremos en cada caso con la variable *indIn* (el objeto *Forum* para el *ForumThread* y el objeto *ForumThread* para el *RootItem*), para poder colocarle la propiedad correspondiente que los relaciona.

[Expr. 7.8]

```
ComunidadForoMensaje cfm=...//cargamos de la BBDD el primer
mensaje
Resource res=aBox.createResource(tBoxNs+FORUMTHREAD);
Individual ind

=aBox.createIndividual(aBoxNs+FORUMTHREAD+"ID_"+cfm.getId(),r
es);
//le colocamos el Forum (indIn) al que pertenece
Property prop=aBox.createProperty(tBoxNs+ISFORUMTHREADOF);
ind.addProperty(prop,indIn);
//de forma redundante, aceleramos busquedas poniendo
propiedad inversa
prop=aBox.createProperty(tBoxNs+HASFORUMTHREAD);
indIn.addProperty(prop,ind);
```

Para insertar el primer comentario del Foro (el objeto *RootItem*), utilizamos un método que coloca sus valores y entramos en un método recursivo que se encarga de ir insertando los hijos de cada comentario del Foro hasta que se llega a comentarios sin hijos (objetos de la clase *LeafItems*). En este caso se añade un límite a la recursividad para evitar bucles infinitos, llegando como máximo a una profundad de foros de 18 niveles, más que suficiente para la mayoría de foros (se considera una profundidad alta cuando se llega hasta el sexto nivel).

5. **Para cada comentario del foro lo generamos como Comment** y le colocamos sus propiedades como objeto *Comment*:

[Expr. 7.8]

```
//método que genera el Comment y le coloca sus propiedades
//como comentario
ind=insertaMensaje(cfm);
//lo relacionano con su ForumThread (indIn)
prop=aBox.createProperty(tBoxNs+ISFIRSTITEMOF);
ind.addProperty(prop,indIn);
//coloco la propiedad inversa para acelerar busquedas
prop=aBox.createProperty(tBoxNs+HASFIRSTTREEITEM);
indIn.addProperty(prop,ind);
//metodo recurso para meter sus hijos y colocar las
//propiedades como TreeItem
insertaHijos(cfm,ind,profundidad);
```

También habrá que extraer el texto del comentario e insertarlo como una de las propiedades. Se insertan también los literales de fechas y que se inserta una propiedad de tipo objeto que es el Rol del autor:

[Expr. 7.9]

```
...
//pongo la fecha de creacion ej 2006-03-17T00:00:00
prop=aBox.createProperty(tBoxNs+DATEOFCREATION);
Calendar cal=Calendar.getInstance();
cal.setTime(cfm.getFechaCreacion());
lit=aBox.createTypedLiteral(cal);
Statement stment= aBox.createStatement(ind,prop,lit);
aBox.add(stment);
...
//pongo el tipo de autor. Por defecto alumno (learner)
//añado los roles
OntClass oc=aBox.createClass(tBoxNs+ROLE);
Individual
rolLearner=aBox.createIndividual(tBoxNs+LEARNERROLE,oc);
Individual
rolTutor=aBox.createIndividual(tBoxNs+TUTORROLE,oc);
Individual rolFacilitator
    =aBox.createIndividual(tBoxNs+FACILITATORROLE,oc);
prop=aBox.createProperty(tBoxNs+AUTHORROLE);
if (//Tiene Rol de tutor de la comunidad){
    ind.addProperty(prop,rolTutor);
}else{
    if (//Tiene Rol de facilitador){
        ind.addProperty(prop,rolFacilitator);
    }else{
    //supongo que en otro caso es alumno
        ind.addProperty(prop,rolLearner);
    }
}
...
```

Aunque hay muchos roles definidos en la TBox, la implementación del sistema de información del CFP limita los roles y sólo tendremos el de facilitador, el de tutor y el de alumno (roles utilizados para la impartición del curso).

6. **Para cada comentario, de forma recursiva generamos su información específica por ser TreeItem**. El detalle de la recursividad se puede ver aquí, y destacamos la generación de propiedades de tipo xsd:boolean y la colocación de relación entre los objetos TreeComments:

[Expr. 7.10]

```
...
if (//el mensaje tiene hijos){
    for (//recorremos todos los hijos){
```

```
        Individual ind=insertaMensaje(cfms[k]);
        //le coloco su padre
        prop=aBox.createProperty(tBoxNs+ISCHILDOF);
        ind.addProperty(prop,indPadre);
        //al padre le coloco su hijo. Redundante pero acelera
queries
        prop=aBox.createProperty(tBoxNs+HASCHILD);
        indPadre.addProperty(prop,ind);
        //parte de recursividad. Insertamos sus hijos
        insertaHijos(cfms[k],ind,--profundidadRestante);
    }
}else{
    //no tiene hijos, es un terminal del arbol LeafItem
    Literal verdadero

=aBox.createTypedLiteral("true",XSDDatatype.XSDboolean);
    prop=aBox.createDatatypeProperty(tBoxNs+ISLEAFITEM);
    Statement
stm=aBox.createStatement(indPadre,prop,verdadero);
    aBox.add(stm);
}
...
```

7. Por último, **pasamos nuestro modelo a un archivo OWL**:

[Expr. 7.11]

```
File f=//definición del fichero donde insertaremos el
       //documento OWL
FileOutputStream os=new FileOutputStream(f);
RDFWriter writer=aBox.getWriter("RDF/XML-ABBREV");
writer.setProperty("xmlbase",aBoxURI);
writer.write(aBox,os,aBoxNs);
```

Y con esto concluimos los detalles de programación de la extracción de información de los foros de una de las comunidades, para poder importarlos en nuestro sistema.

7.3.5 Ejecución de la importación

Para emular el proceso de importación de información a nuestro sistema, realizamos un proceso que recorre todas las experiencias educativas de nuestro caso de estudio y genere un fichero .owl para cada una, basándose en la programación del apartado anterior. Los ficheros OWL tendrán la información de los foros y mensajes de cada curso al que pertenecen. El proceso almacenará información sobre tiempos de generación, número de foros, threads y mensajes y profundidad.

El esquema de los equipos para realizar las pruebas es el de la Fig 7.16, donde todo el sistema del extractor corre sobre un ordenador con sistema operativo Windows XP Profesional, 2GHz y 1GB de RAM, a excepción del acceso a la base de datos del CFP, que se accede vía Ethernet a un servidor con Windows 2000 Server con 2GHz de RAM reservados para la base de datos que corre en Microsoft SQL Server 2003.

Se realiza la emulación en un periodo de tiempo en el que apenas hay acceso a la base de datos (20:30 PM) y actividad en la red, ejecutando el programa que genera cada uno de los 31 ficheros owl (uno por cada curso o comunidad virtual). Este experimento se repite ocho veces para intentar minimizar los errores de medida por causas puntuales. A cada una de las repeticiones las denominaremos *importación 1* a *importación 8*.

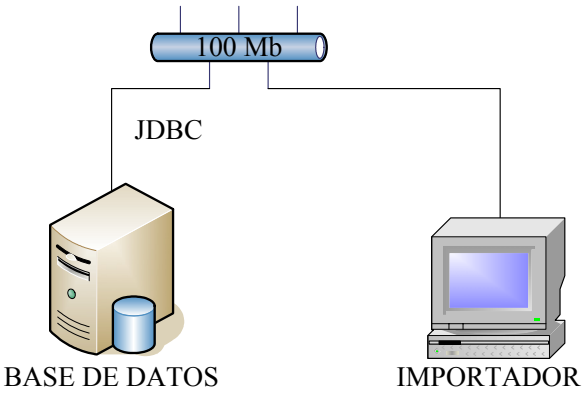


Fig. 7.16. Esquema de montaje de equipos implicados en la importación

A los ficheros generados se les ha chequeado la consistencia, clasificado y comprobado que son OWL DL como se ha explicado para la TBox (Fig 7.9 y 7.10), dando resultados satisfactorios en todos los casos.

En el apartado siguiente ser verán algunos resultados del proceso que hemos concluido, como tiempos en función del tamaño y profundidad del foro, y la ocupación de disco de cada ontología de un curso.

7.3.6 Resultados del proceso de importación

En este apartado mostraremos resultados comparativos de distintas importaciones, para poder extraer información sobre rendimientos en tiempo de las importaciones, tamaños de archivos OWL generados, comparados con los parámetros de cada uno de las importaciones (número de foros importados, cantidad de mensajes y profundidad de las estructuras en árbol).

Los datos comparativos para cada una de las experiencias educativas de las que se ha extraído información se muestran en la tabla 7.2.

ID comunidad	Nº Foros	Nº Mensajes	Profundidad	Tiempo exportación (ms)	Tamaño de ontología .owl (KBytes)	Tamaño comprimido (KBytes)
133	4	31	6	274	35	7
134	4	130	4	645	148	19
135	4	63	7	287	86	14
136	5	86	7	389	108	17
137	4	93	5	385	119	16
138	4	104	5	414	121	18
139	2	46	5	252	59	10
140	3	123	4	512	149	23
141	2	127	6	504	141	19
142	1	88	6	340	108	16
164	8	59	2	254	77	13
165	4	37	8	151	42	7
166	2	6	4	27	6	2
167	4	25	4	99	32	7
176	2	60	4	231	64	11
177	2	6	3	32	10	4
178	2	29	2	125	33	7
179	2	44	5	189	52	10
180	2	15	2	70	16	4
181	1	32	8	129	33	5
182	1	12	4	49	13	3
183	2	64	6	234	69	11
184	2	21	2	84	25	6
185	9	134	4	580	170	24
186	7	10	5	59	14	4
187	2	79	2	330	114	15
188	2	35	8	104	39	7
189	2	4	4	25	13	2
190	2	55	2	205	64	9
191	2	40	6	180	52	8
192	2	12	4	49	15	4

Tabla 7.2. Comparativa de las extracciones a .owl según comunidades

En la tabla 7.2 se muestra los valores de resultado para cada una de las 31 experiencias educativas con las que se ha tratado. Como cada experiencia se ha importado 8 veces, los valores temporales son la media de las 8 importaciones, para evitar errores puntuales.

En la tabla identificamos cada una de las comunidades por medio de su identificador de comunidad. Una comunidad se puede equiparar en este caso con una experiencia educativa, ya que contiene un conjunto de usuarios, unos contenidos estáticos, foros y tests. Se entiende como profundad del foro, a número de niveles de anidación de los mismo, de forma que un foro debe tener como mínimo una profundidad de 1 (nadie contesta a ningún mensaje de nadie). Para el cálculo de los tiempos de extracción (el tiempo que dedica el importador en generar el archivo OWL correspondiente) hemos eliminado el tiempo de arranque del extractor y mostramos la media a lo largo de las ocho importaciones realizadas. Como podemos apreciar, en las experiencias educativas existe una gran varianza respecto al número de foros, mensajes y profundad de foros, pese a tratar los mismos contenidos estáticos.

El primer valor que calculamos en las extracciones es el tiempo necesario para inicializar todas las clases que utilizamos para la importación. Medimos este tiempo por medio de marcas de tiempo en el código desde el inicio hasta que hemos generado en Jena el modelo, la ontología y hemos importado la TBox.

Descubrimos que este tiempo sólo es apreciable para la transformación en OWL de la primera comunidad para cada importación, siendo cero o menor de 20 ms para el resto de comunidades. Atribuimos este fenómeno a:

- **El tiempo realmente apreciable es el de inicialización de las clases de Jena**. Jena debe inicializar sus clases internas una vez y mantenerlas cargadas en la máquina virtual de Java todo el tiempo (por ejemplo, mediante clases *Singletone*). Muy probablemente cuando importemos una ontología en Jena (como en este caso con nuestra TBox), no realice ninguna comprobación sobre la misma para chequear consistencias, sino que añada la sintaxis correspondiente y nada más.

- El **tiempo de acceso al sistema de ficheros** debe ser más importante la primera vez que accedemos a un directorio desde una máquina virtual que las siguientes veces.

- **La apertura de socket** para la comunicación JDBC con el servidor de la base de datos es necesario sólo la primera vez que accedemos a la misma, ya que está implementado un sistema de pool de conexiones a la base de datos que permite la reutilización de las mismas conexiones). Aunque al principio no accedamos directamente a la base de datos en la inicialización, sí que instanciamos las clases *Manager* de la base de datos (*cfp.poseidon.mgr.**) que empiezan esta comunicación.

Los valores obtenidos de inicialización para las ocho importaciones nos dan un valor medio de unos 8,5 segundos y con una desviación típica de medio segundo, como comprobamos en la figura 7.17. Si consideramos que la desviación típica calculada es la real, obtenemos que la media de tiempos de inicialización se encuentra entre 8,12 y 8,81 segundos con un 95% de probabilidad. Comparando este valor temporal con los tiempos necesarios para la extracción de cada comunidad, vemos que los valores de los tiempos de inicialización son un orden de magnitud superior a los valores de los tiempos de extracción de una comunidad.

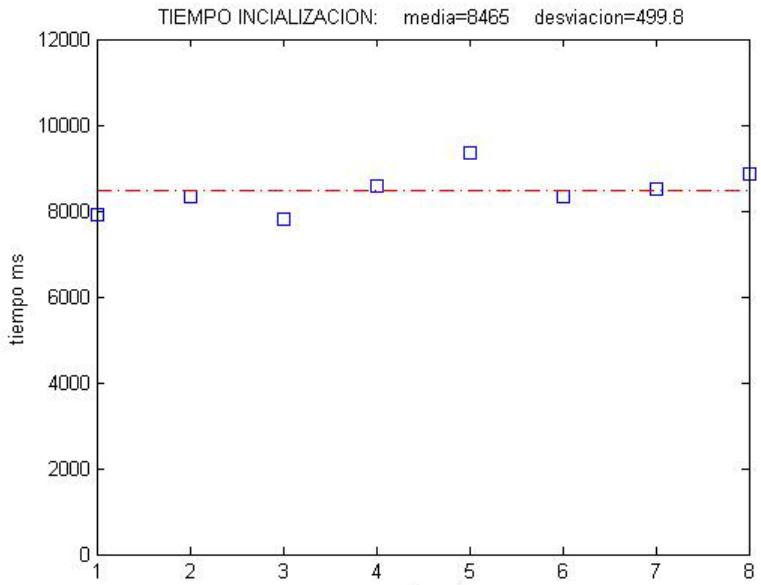
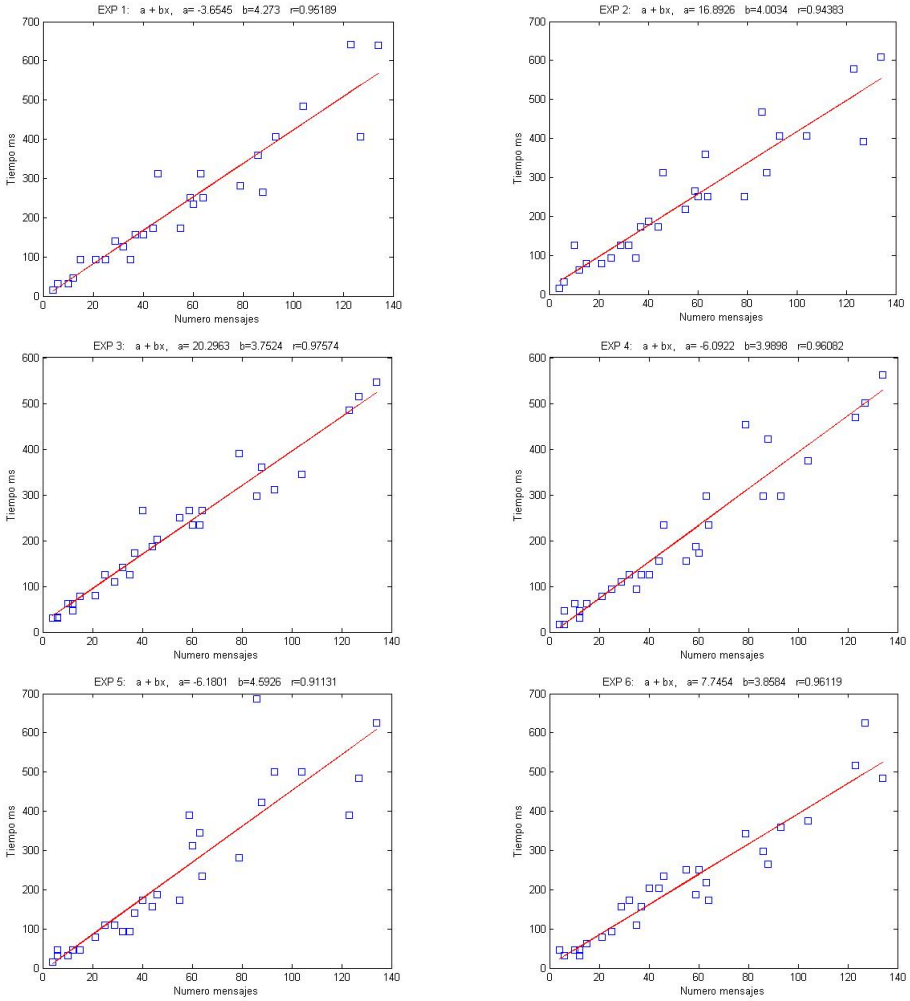


Fig. 7.17. Media del tiempo de inicialización para las 8 importaciones

Otro valor que sacamos es la relación entre el tiempo de creación de los archivos owl (quitada la inicialización) y la cantidad de mensajes de la comunidad. Se calcula las rectas de regresión para cada uno de los 8 experimentos (Fig. 7.18), calculando al final la curva de regresión de las medias de tiempos de los 8 experimentos (Fig. 7.19), para el que obtenemos una relación lineal. Por tanto podemos concluir que existe una relación lineal entre tiempos y número de mensajes a extraer.



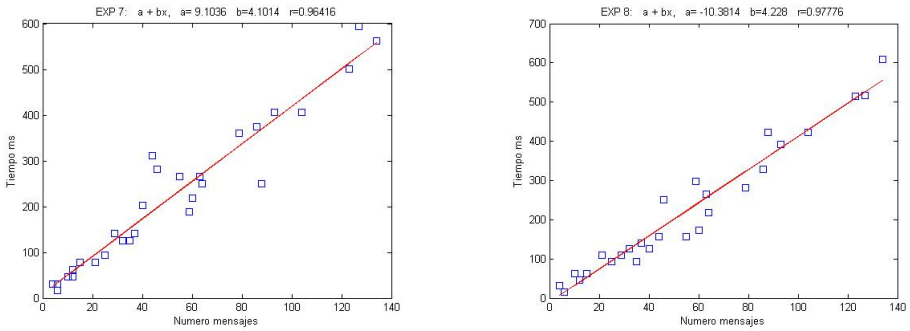


Fig. 7. 18. Rectas de regresión de las ocho importaciones

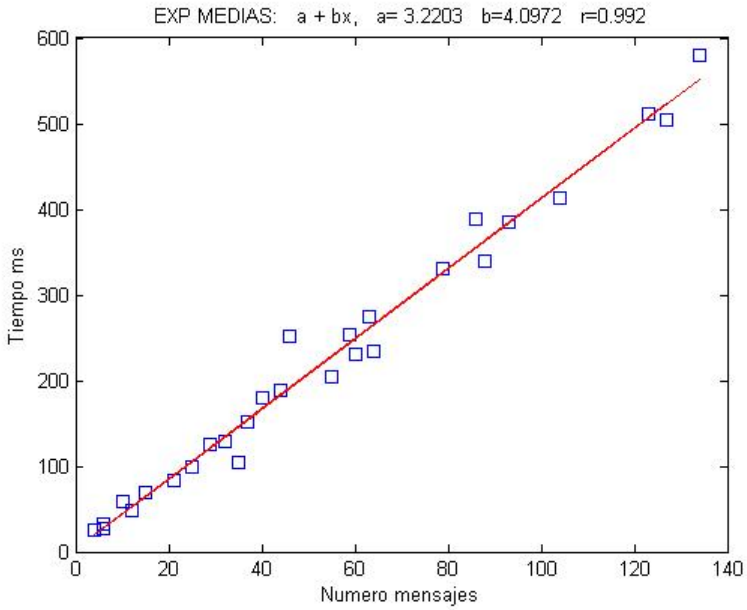


Fig 7.19. Recta de regresión de las medias

En la figura 7.19 podemos observar que si aplicamos la recta de regresión a la media de cada uno de los experimentos obtenemos una relación casi lineal, con un tiempo de 4,1 milisegundos por mensaje enviado al foro. A partir de esto podemos hacernos una idea del tiempo que le puede costar a un agente validar cada ABox que obtiene.

Realizamos un estudio también sobre la relación entre el número de mensajes y el tamaño del archivo owl, tanto su tamaño directo como comprimido. La compresión se realiza en formato .zip. Con este estudio pretendemos saber cómo dimensionar el almacenamiento, tanto para los servidores de contenidos semánticos como para los propios agentes.

Concluimos en abos casos que el crecimiento del tamaño es lineal con la cantidad de mensajes en la comunidad, siendo un poco menos claro para los archivos comprimidos (r=0,98) que en los no comprimidos, y teniendo los archivos comprimidos un tamaño inicial (independiente de los mensajes) mayor que el del archivo no comprimido. En la figura 7.20 podemos apreciar que el tamaño inicial de los archivos comprimidos es unos 2Kbytes, frente a los 500 bytes del archivo no comprimido, pero ya a partir de 2 mensajes de la extracción se ocupa menos espacio de disco con los archivos comprimidos, por lo que la compresión será recomendable en todo caso. En este caso no se hace la media para cada importación, ya que en cada importación se obtienen los mismos archivos.

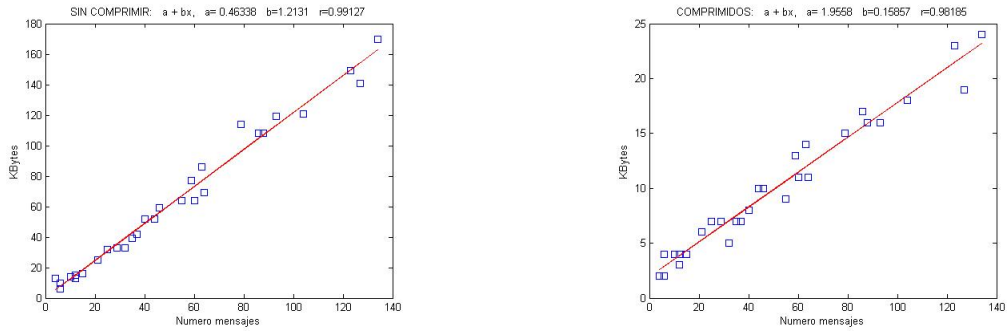


Fig. 7.20. Tamaños de los archivos sin comprimir y comprimidos

La mejora que introduce la compresión es reducir a un 13% el tamaño medio de ocupación por mensaje (de 1,2 KB a 0,16 KB), aunque de todas formas los tamaños de los archivos OWL son comparables al tamaño de cualquier imagen (normalmente .gif o .jpg) del contenido estático del curso (el mayor tamaño es de 170KB). Esto nos hace concluir que almacenar el archivo OWL no ha de ser ningún problema a nivel almacenamiento en los propios repositorios de contenidos.

7.4 Sistema de búsquedas e inferencias

En este apartado explicaremos el prototipo realizado para la realización de inferencias y búsquedas sobre la información OWL DL que hemos extraído de nuestros foros reales. Este apartado pretende comprobar el comportamiento de los posibles agentes de búsqueda de nuestra arquitectura. Dichos agentes obtendrán conocimientos de archivos .owl que les vaya interesando y sobre éstos conocimientos realizará las inferencias. En la figura 7.21 podemos ver la parte de nuestra arquitectura que vamos a

validar, es decir, el comportamiento del agente que obtiene información semántica, realiza búsquedas inteligentes y presenta resultados al usuario.

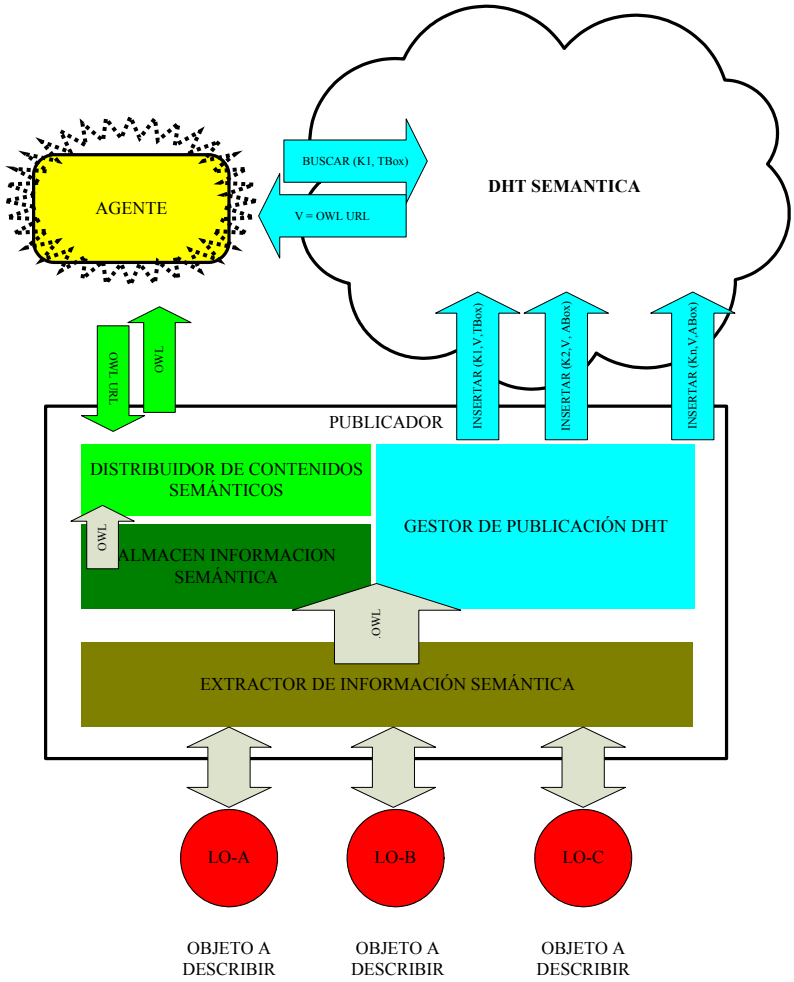


Fig. 7.21. Parte de la arquitectura simulada en este apartado

Para realizar este prototipo se planteó la posibilidad de utilizar alguna API que permita inferencias (como puede ser Jena 2 [DICK04] o la API de Protégé) o utilizar alguna de las herramientas de autor que permiten inferencias como Protégé o SWOOP.

Como la arquitectura que queremos plantear se basa en un agente inteligente que presenta al usuario un conjunto de posibilidades de búsqueda para que sea el usuario quien las formule, consideramos que una herramienta como Protégé será la más adecuada. Protégé permite realizar inferencias y encontrar las instancias que cumplen unas determinadas propiedades.

Contiene, además un plug-in de RDQL para realizar queries a nivel de RDF , que permitirá realizar búsquedas mixtas entre localización de palabras en los recursos e inferencias sobre los mismos (búsquedas RDF y OWL). También contiene una ampliación denominada *Queries* que permite hacer búsquedas directas en nuestra base de conocimiento.

Por tanto el esquema sería el mismo que para el chequeo de ontologías (figura 7.9), pero eligiendo un único razonador. En este punto tenemos que decidir cuál de los razonadores más utilizados para OWL DL (FaCT++, Racer, Pellet) es al más adecuado para las ontologías que utilizamos y el esquema propuesto. Por esta razón se describirán las tres posibilidades y se hará una pequeña prueba para evaluar rendimientos y comparar los razonadores. Por tanto a continuación describiremos los tres razonadores, realizaremos una comparación de uso, características y rendimientos y elegiremos el más adecuado para nuestros propósitos.

7.4.1 FaCT++ 1.1.3

FaCT++ [WWW71], desarrollado por la Universidad de Manchester, es la continuación del razonador opensource para TBoxes FaCT (Fast Classification of Terminologies), siendo la versión antigua desarrollada en Lisp y la versión actual desarrollada en C++ para acelerar los algoritmos.

La nueva versión permite ya razonar con ABoxes y en las últimas versiones ha mejorado bastante su interfaz con DIG. FaCT++ está optimizado para la lógica de OWL DL. Los algoritmos utilizados son optimizaciones del algoritmo Tableaux.

Su instalación sobre windows conlleva cierta dificultad al tener la necesidad de instalar la plataforma .NET 2.0, y no se ha encontrado información alguna sobre su instalación, manejo y mensajes de error mostrados por el razonador. Su desarrollo se lleva de forma particular entre dos personas (Ian Horrocks y Dmitry Tsarkov [TSA03],[TSA04],[GAR06])y no tiene el apoyo de ninguna comunidad de desarrolladores y beta-testers asociado, por lo que se encuentra aún en una fase algo inestable.

Aunque no disponga de información de uso y ayuda, al menos el servidor HTTP de DIG muestra por su salida estándar bastante información relativa a las acciones que realiza y errores que ocurren. Su interfaz DIG permite ejecutarlo como servidor HTTP. En la Fig. 7.22 podemos observar un ejemplo de utilización de FaCT con Protégé.

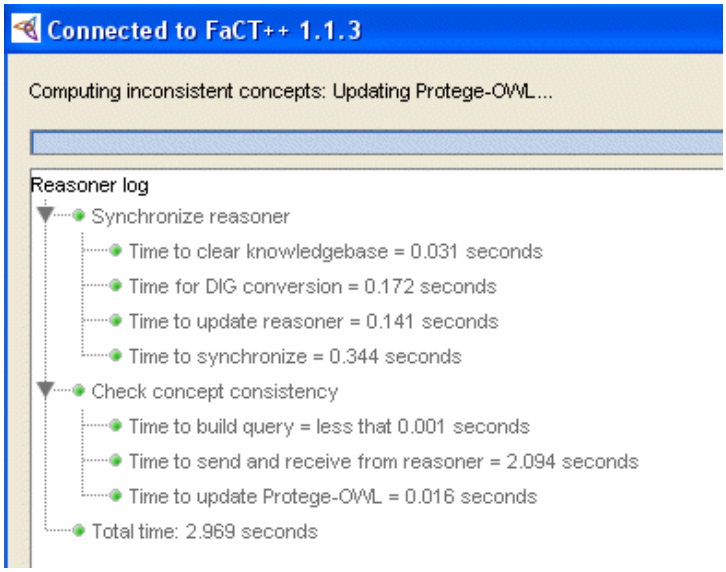


Fig. 7.22. Uso de FaCT++ con Protégé

7.4.2 RACER 1.9

Racer (Reasoner for Aboxes and Concept Expressions Renamed) [WWW13] fue uno de los primeros razonadores para ABoxes que apareció en el mercado (hay versiones desde el año 2002). Se desarrolló por las universidades alemanas de Hamburgo y actualmente es software propietario y de pago, aunque permite una descarga de prueba de tres meses.

Racer 1.9 soporta OWL DL excepto para los nominales (clases definidas por una enumeración de sus miembros, que implementa como definiciones parciales) y para tipos de datos no estándar (fuera de xsd:X). Al contrario que las primeras versiones, por defecto no asume UNA (Unique Name Assumption), para ser compatible con OWL. Se basa en el algoritmo Tableaux con optimizaciones y cacheos de inferencias obtenidas.

Racer es un razonador con más posibilidades que un simple razonador OWL, ya que incluye un cliente para hacer queries en OWL-QL (RacerPorter), su propio lenguaje de queries (nRQL, new Racerpro Query Language) e incluye una herramienta de cliente exclusiva (RICE, Racer Interactive Client Environment), y una API en Java para poder interactuar con él directamente (JRacer).

7.4.3 Pellet 1.3

Este razonador [WWW59] (actualmente versión 1.3-beta 2) es un razonador opensource desarrollado por Mindswap. Pellet es un razonador exclusivo

para OWL DL e implementado en java, que implementa el interface DIG y además RDQL para queries sobre la información en RDF. Además, dispone de una API para poder ser utilizado directamente desde Java.

Como los otros dos sistemas, se basa en el algoritmo Tableaux, como se indica en [PAN05]. De todos los razonadores probados, la implementación de DIG es la que presenta errores más aclaratorios para la corrección de ontologías y además es el razonador que más chequeos hace al principio, el que te permite corregir errores en la ontología directamente. Además, al recibir una ontología realiza optimizaciones internas, de forma que los siguientes queries sean más rápidos. Es el único que se presenta también como una web donde se puede enviar el archivo .owl para validarlo, como vimos en la figura 7.11.

7.4.4 Selección del razonador. Comparativa de uso

Para seleccionar el razonador primero compararemos sus características generales y por último realizaremos unas mediciones de rendimiento de los mismos.

Existen varias comparativas entre razonadores para OWL [PAN05], [WWW60], [GAR06], [HAA05], [RUT05] aunque las conclusiones finales son que no hay un razonador mejor que los demás, sino que depende de las ontologías que usemos en concreto para ver los rendimientos y de quién haga dicha comparativa. Lo que queda constatado en la mayoría de comparativas es que Pellet da una respuesta muy aceptable en todas las comparativas.

7.4.4.1 Comparativa de características generales

Como podemos ver en la tabla 7.3 resumen de la comparativa, todos implementan el interfaz DIG, por lo que podremos utilizar dicho interfaz a la hora de implementar un prototipo que pueda ir cambiando de razonador.

Característica	FaCT++	Racer 1.9	Pellet 1.3
Evaluación objetiva			
Multiplataforma	NO	NO	SI
Código abierto	SI	NO	SI
Freeware	SI	NO (3 meses de prueba)	SI
Implementación DIG	SI	SI	SI
API para utilizar	NO	SI	SI

Orientación a OWL	Total	Media	Total
Evaluación subjetiva			
Facilidad de instalación	Medio (.NET 2.0)	Alto	Alto
Información sobre errores en inferencias	Medio	Bajo	Alto
Comunidad de desarrolladores implicados	NO	NO	SI

Tabla 7.3. Comparativa de los razonadores

En la tabla 7.3 apreciamos que FaCT y Pellet son soluciones opensource y gratuitas, mientras Racer es una versión de pago (aunque se puede utilizar de forma indefinida con el interfaz DIG, por lo que se considerará como posibilidad factible para implementar el prototipo). Otra característica importante, para la generación de agentes inteligentes que tengan su propio razonador integrado, es la existencia de una API para poder atacar al razonador directamente. En este caso tanto Racer como Pellet ofrecen una API en java para realizar dicha integración. De todas formas, consideramos que la arquitectura que ataca al razonar por medio de HTTP por medio de DIG es una arquitectura similar a la de servicios Web que puede ser muy interesante para futuras implementaciones.

En la comparación subjetiva, vemos que FaCT, pese a ser opensource, no cuenta con mucha información de ayuda, ni comunidad grande de soporte, además de ofrecer poca información al usuario en caso de error (aunque un poco más de la que ofrece Racer en su interfaz DIG). Sólo Pellet cuenta con el apoyo de una comunidad de desarrolladores y beta testers que permite acelerar la madurez del producto.

Tras realizar esta comparativa general, veremos en el siguiente apartado una comparativa objetiva sobre rendimientos, haciendo que los tres razonadores trabajen con la ontología especificada en esta Tesis y con las instancias particulares provenientes de la importación de datos reales realizada.

7.4.4.2 comparativa de rendimientos

Para evaluar el rendimiento, utilizaremos el mismo esquema de trabajo que para el chequeo de las ontologías (ver Fig. 7.9), donde atacamos a los razonadores (en este caso tres, FaCT, Racer y Pellet) por medio de DIG

sobre HTTP, por lo que tendremos una única instancia de Protégé (a modo de interfaz de usuario) que irá atacando a los tres razonadores (cada uno trabajando sobre un puerto) y realizándoles varias pruebas de rendimiento, midiendo los tiempos de respuesta como variable de rendimiento.

Los seis tipos de prueba de rendimiento que vamos a utilizar son:

- **Chequeo de consistencia de nuestra TBox**. Es decir, comprobar que no hay clases inconsistentes (que nunca podrán tener instancias). Contiene 25 clases, 23 propiedades y 6 instancias (los Roles).

- **Chequeo de consistencia de una ABox importada**. Chequea la consistencia de nuestra TBox ampliada con una de nuestras ABoxes. En este caso se cogerán los datos del foro ID=134 (ver tabla 7.2), con 4 foros, 130 mensajes en una profundidad de hasta 4 niveles.

- **Clasificación de nuestra TBox**. Esto revisa las clases una por una para ver si existe algún tipo de herencia implícita en la declaración, de forma que podamos saber todas las clases de las que es especialización la clase inicial.

- **Clasificación de la ABox de la comunidad 134**, como se comenta en el punto anterior.

- **Cálculo de tipos inferidos en la ABox.** Esta petición consiste en solicitar al razonador que infiera a qué clases pertenece cada instancia, aunque no lo hayamos indicado explícitamente en el documento OWL.

- **Simulación de una inferencia**. En este caso, buscaremos todos los recursos que sean descendientes del comentario ID=1129. Para hacer esto definimos una clase descendiente de la clase *TreeComment*, que se caracteriza porque sus instancia son todas las que se relación por medio de la propiedad *isDescendentOf* con el comentario particular 1129 (TreeCommentID_1129).:

[Expr. 7.12]

$$Descendientes\_de\_1129 \subseteq TreeComment$$

$$Descendientes\_de\_1129 \equiv isDescendentOf \ni TreeCommentID\_1129$$

Se ha elegido esta inferencia porque su contestación necesita un agente que no busque únicamente por palabras o conjuntos de caracteres, sino que permita razonamientos y por tanto podamos definirle una propiedad *isChildOf* que sea subpropiedad de otra transitiva denominada *isDescendentOf*, y buscar por algún valor de esa propiedad transitiva.

Para implementar las búsquedas inteligentes en Protégé, se opta por definir las propiedades que buscamos en nuestras instancias como una clase a la que no le colocamos ninguna instancia directamente. Luego pedimos al razonador que realice el cálculo de tipos inferidos, acción que comprueba si las instancias que tenemos se pueden colocar como miembros de cualquier clase de nuestra ontología que no esté explícitamente en nuestra ontología inicial, por lo que contestará a nuestra inferencia rellenando la clase con las instancias que lo cumplan.

Se comprueba manualmente que en este caso hay dos instancias de la clase *TreeComment* que son descendientes de la instancia con identificador de comentario 1129, que son las instancias con indentificadores de comentario 1278 y el 1296 (Ver Fig 7. 23).

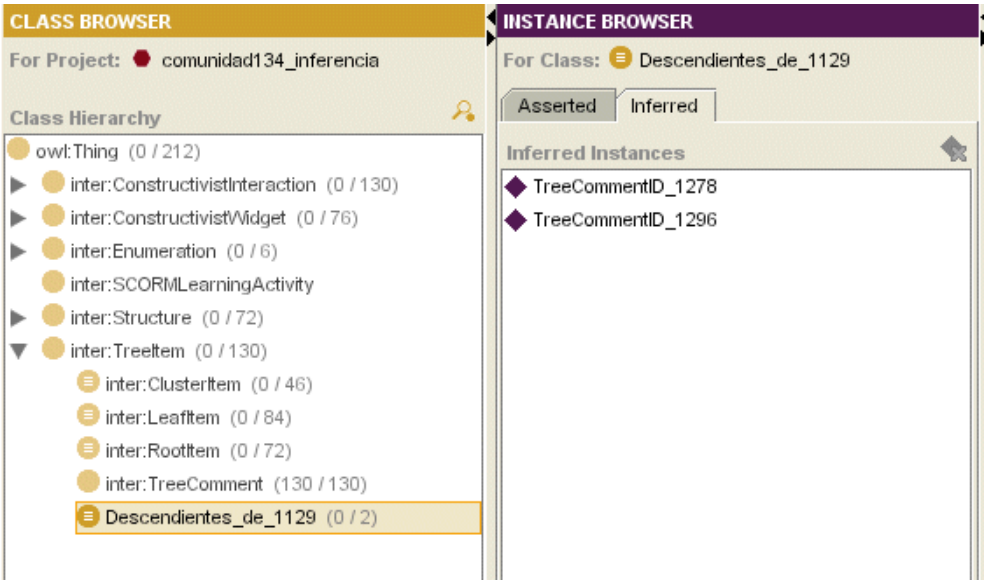


Fig. 7. 23. Resultados de la inferencia realizada con Protégé y Pellet

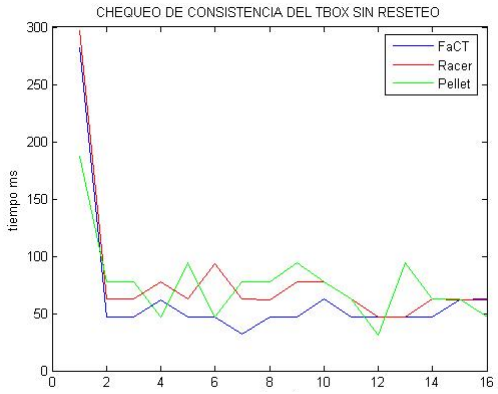
Por ejemplo, en la figura 7.23 vemos el resultado en Protégé tras definir la clase *Descendientes_de_1129* como se indica en la expresión 7.12 y

solicitar la inferencia de clases al razonador conectado vía DIG (en este caso Pellet). Vemos que ha inferido que las instancias *TreeCommentID_1278* y *TreeCommentID_1296* pertenecen a esa clase, por tanto son descendientes del comentario 1129. También apreciamos que no sólo ha inferido esta clase, sino que ha realizado todas las inferencias de pertenencia a clases de su base de conocimiento. Por ejemplo, pese a no indicar en nuestros archivos OWL ninguna instancia de la clase *LeafItem*, vemos que existen 84 instancias que pertenecen a esa clase.

7.4.4.3 Resultados de la comparación de rendimientos

Para comparar los rendimientos de los motores de inferencias, en una primera fase se hicieron las pruebas de forma que, por cada razonador, repetíamos 16 veces la misma prueba sin resetear el sistema cada vez (por ejemplo, clasificamos la TBox 16 veces seguidas con Racer).

En la figura 7.24 podemos ver comparados los rendimientos de los tres razonadores, en los que se ve una gran diferencia entre la primera ejecución del primer tipo de prueba y las siguientes 15 repeticiones del primer tipo de prueba. En cada gráfica se agrupan el mismo tipo de prueba, para los tres razonadores estudiados.



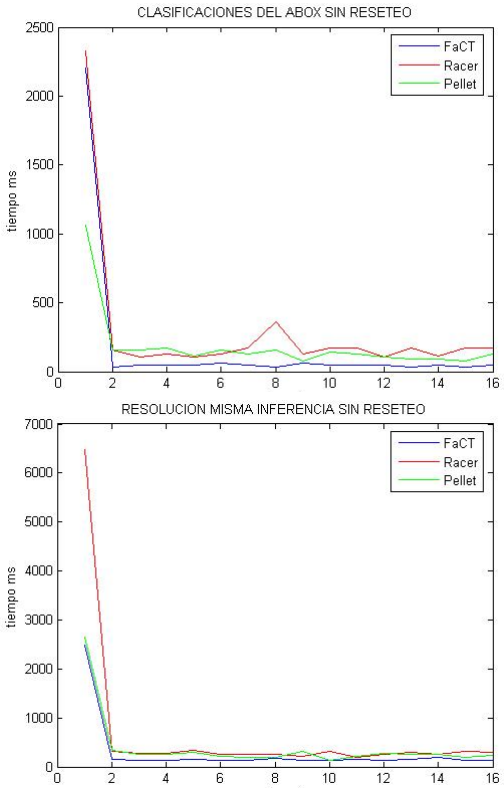
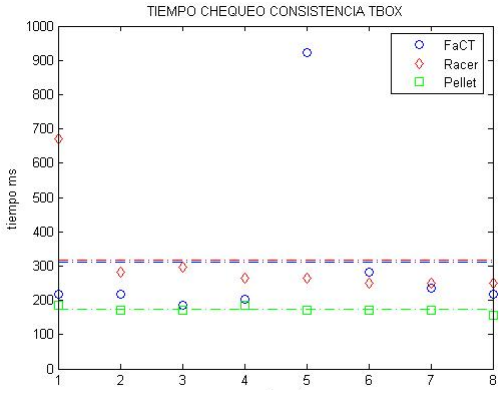


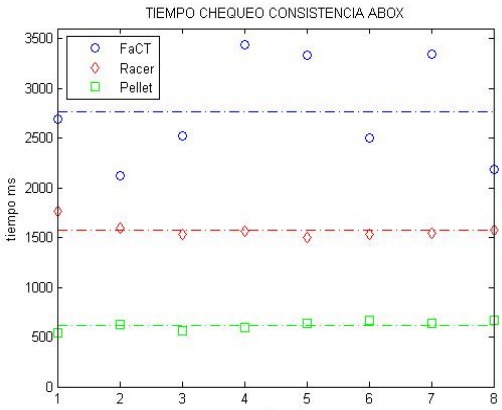
Fig.7.24. Diferencia entre el primer experimento y siguientes en tres tipos de pruebas

En los gráficos de la figura 7.24 se comprueba que existe una gran diferencia entre la primera vez que se realiza el experimento y las siguientes veces. Esto es debido a que los razonadores, para mejorar el rendimiento, cachean los resultados de los queries que les hacemos, como se indica en [GAR06], y además si no cambiamos la ontología cachean también las inferencias realizadas la primera vez que le pedimos al razonador que lo haga. Por tanto concluimos que para evaluar el rendimiento no podemos repetir la misma prueba sobre el mismo razonador sin haber reseteado antes el razonador, ya que al cachear los resultados falsea el tiempo de cálculo de los mismos.

Esto nos hace pasar a la segunda parte de las pruebas, donde nos interesa medir los valores de los tres razonadores de la forma más aleatoria posible y buscando que el razonador resetee su memoria entre experimento y experimento, para simular el rendimiento de nuestros agentes inteligentes cada vez que hay una ontología nueva o se añaden conocimientos a la misma.

Para cada tipo de prueba de rendimiento, para evitar las fluctuaciones aleatorias que puedan ser debidas a otras causas, realizaremos ocho pruebas iguales por cada razonador, atacando a los razonadores de forma alternada para que las posibles fluctuaciones aleatorias afecten por igual a los resultados de los tres razonadores. En las gráficas de las figuras 7.25 y 7 .26 veremos los resultados obtenidos. En cada gráfica representaremos las ocho repeticiones (con reseteo entre cada repetición) de un tipo de prueba para los tres razonadores (24 experimentos por gráfica), marcando con raya discontinua la media de cada razonador.





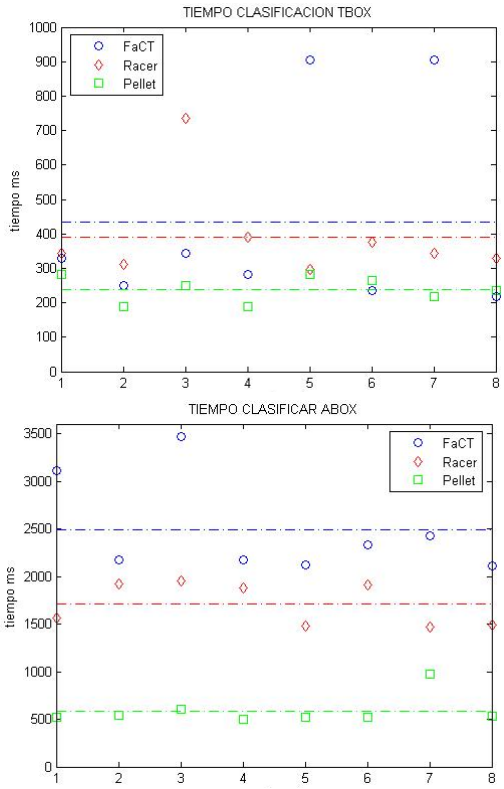
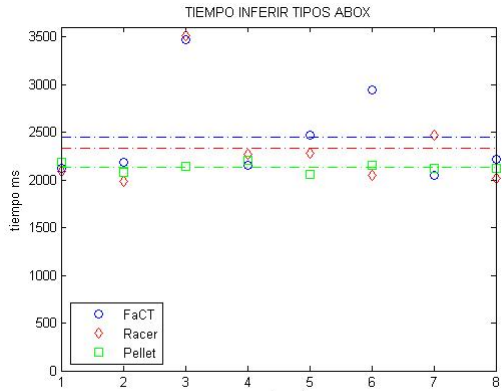


Fig. 7. 25. Comparativas de rendimiento con reseteo entre cada prueba

En la figura 7.25 tenemos las gráficas de los 4 primeros tipos de pruebas, es decir, los chequeos y las clasificaciones de la TBox y la ABox. Esta parte se puede considerar el rendimiento de un agente inteligente a la hora de ampliar su base de conocimiento con nuevas ontologías (TBox) o con instancias de las ontologías que ya tiene (ABox). Pellet es el razonador que de momento ofrece mejor resultado para todas las pruebas.



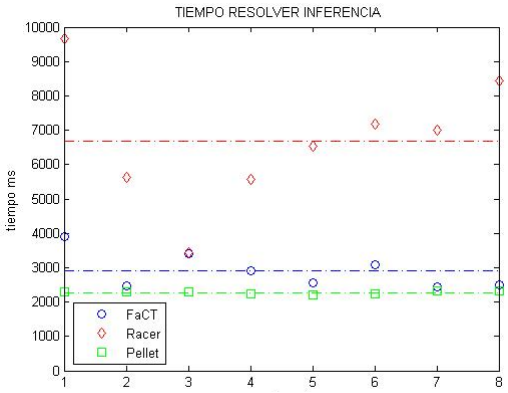


Fig. 7.26. Comparativas de rendimientos al inferir los tipos

En las gráficas de la figura 7.26 se indican los rendimientos a la hora de inferir tipos y resolver la inferencia del último tipo de prueba definido, que asimilamos con los rendimientos de razonamiento del agente inteligente que esté utilizando ese motor de inferencia. Como podemos ver en las gráficas, la media más baja en todos los casos es la que ofrece Pellet, siendo también el razonador con las varianzas más pequeñas comparando con el resto de razonadores. En la tabla 7.4 podemos ver los valores numéricos de las pruebas realizadas.

	valor (ms)	FaCT ++	Racer 1.9	Pellet 1.3
1) Consistencia TBox	media	**310**	**316**	**173**
	varianza	249	145	9,9
2) Consistencia ABox	media	**2765**	**1576**	**619**
	varianza	532	82	47
3) Clasificación TBox	media	**434**	**391**	**289**
	varianza	295	143	38
4) Clasificación ABox	media	**2488**	**1707**	**588**
	varianza	515	224	158
5) Tipos Inferidos ABox	media	**2450**	**2334**	**2134**
	varianza	501	504	49
6) Simulación de inferencia	media	**2908**	**6681**	**2277**
	varianza	527	1898	43

Tabla 7.4. Comparativa de rendimientos de los razonadores

En la tabla 7.4 corroboramos lo que se había visto en las figura 7.25 y 7.26, que los mejores rendimientos en las seis pruebas los ofrece el razonador Pellet, ofreciendo también los más estables, ya que la varianza es menor.

Esto no tiene por qué significar que un razonador es mejor que otro en todos los casos, pero sí que podemos concluir que, para el tipo de uso propuesto (con Protégé y DIG) , para TBoxes y ABoxes como los que utilizamos. Pellet es quien da mejores resultados y además el que ofrece valores más estables. Estos resultados de rendimiento vienen a sumarse a la comparativa general de razonadores de la tabla 7.3, donde el mejor valorado es también Pellet.

Se puede apreciar que Racer es el siguiente en mejores resultados, menos en la simulación de la inferencia (que es realmente para lo que queremos el razonador en el prototipo). Además, FaCT es el que ofrece valores más variados (mayor varianza calculada) excepto en la simulación de la inferencia.

Por estas razones utilizaremos Pellet como razonador en nuestro prototipo, al ser el mejor evaluado en la mayoría de los puntos estudiados. Es muy probable que la selección de un razonador u otro pueda ir variando según las versiones de los mismos se van desarrollando, por lo que nuestra recomendación de Pellet es para este prototipo y con las últimas versiones de razonadores que hemos podido obtener.

7.4.5 Descripción del sistema de búsquedas

Para comprobar el sistema de búsquedas del agente inteligente, partiremos de las siguientes premisas:

- Supondremos que el agente de búsqueda ha actualizado su base de conocimiento como se especifica en el capítulo de arquitectura propuesta. En nuestro prototipo simularemos dicha adquisición de información por medio de los imports de Protégé.

- Una vez conseguida la información necesaria, mostraremos cómo poder hacer búsquedas RDF (que no OWL) para buscar por medio de substrings.

- Realizada las búsquedas por substrings, mostraremos cómo realizar búsquedas en OWL, que necesiten del razonador escogido. Las inferencias se realizarán de igual forma que se han realizado en el

apartado de comparación de rendimientos. De esta forma realizaremos búsquedas mixtas, en las que se realizarán inferencias y se poblarán clases con búsquedas de palabras clave en RDF.

A continuación detallaremos cada uno de los tres pasos para esta validación.

7.4.6 Carga de conocimientos en el agente

Como ya dijimos anteriormente, utilizaremos Protégé como base para la simulación. Generaremos primero una ontología vacía, dentro del espacio de nombres que asignamos a nuestro agente:

[Expr. 7.13]

```
http://dcom.upv.es/owl/rorollo/agenteKB#
```

Luego realizaremos importaciones los documentos OWL que contienen la TBox y de las distintas ABox obtenidas en el proceso de importación de datos reales. Por ejemplo, supondremos un agente que ha obtenidos los datos de foros de cuatro comunidades (id=134, 140, 145 y 185).

Para realizar esto en Protégé, utilizamos la pestaña *Metadata* y añadimos las importaciones necesarias, indicando no sólo la URI importada, sino la ubicación física de esa URI en fichero, ya que como comentamos la URI es un identificador único y no tiene por qué ser un localizador uniforme URL (ver Fig. 7.27).

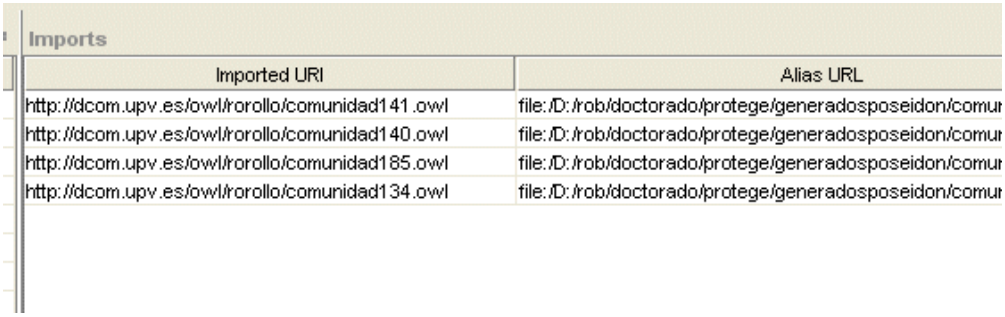


Fig. 7.27. Importaciones de ontologías de las comunidades

En la figura 7.27 vemos como importamos las ABox de los foros de comunidades que vamos a utilizar en la prueba. No indicamos la TBox, ya que cada una de las ABox indica que importa la TBox, por lo que el sistema buscará esa TBox directamente. Sí que es necesario indicar en otro lugar de

Protégé la ubicación física donde puede encontrar información relativa a la URI de la TBox, ya que tampoco es una URL disponible.

Les daremos un espacio de nombres distinto a cada una de las importaciones, de modo que podamos visualmente saber de qué comunidad se ha importado cada cosa. De esta forma nuestro agente dispone ahora de una base de conocimiento con más de 500 mensajes procedentes de 18 foros distintos de 4 imparticiones del mismo curso (ver Fig. 7.28).

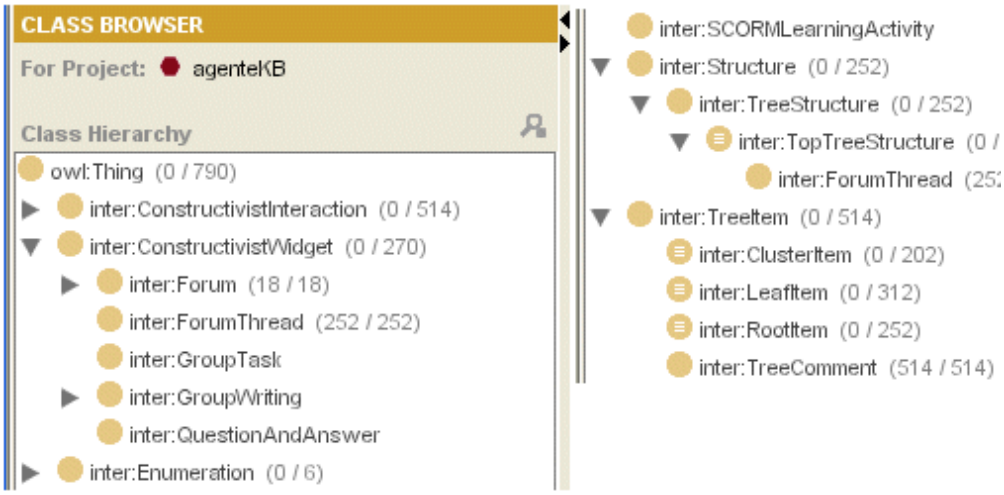


Fig. 7.28. Objetos en la base de conocimiento de nuestro agente

En la figura 7.28 podemos ver las instancias que ha cargado el agente, de forma que, por ejemplo, tiene 514 comentarios en su base de conocimiento.

Con Pellet realizamos un chequeo de consistencia que dura 4 segundos y una inferencia de tipos para clasificar las instancias que nos cuesta 30 segundos de media. Vemos que los tiempos son relativamente altos y esto es debido a que la complejidad de los algoritmos utilizados (Tableaux) tiene un crecimiento exponencial con el número de elementos incluidos en el razonamiento.

Una vez el agente ha adquirido la información necesaria, realizará búsquedas en su base de conocimiento. En este caso serán búsquedas mixtas RDF/OWL, que permite realizar inferencias y también realizar búsquedas por palabras clave, ya que lo que pretendemos es mejorar y ampliar las búsquedas por palabras clave, no sustituirlas completamente por búsquedas OWL.

7.4.7 Búsquedas por palabras clave

OWL no permite realizar búsquedas dentro de los valores de las funciones de tipos de datos (como puede ser buscar substrings dentro de una valor xsd:string), por lo que tenemos que recurrir a una capa más profunda, hasta RDF para poder realizar dichas búsquedas.

Existen múltiples lenguajes para indicar búsquedas en un conjunto de grafos RDF [WWW61]. Los podemos dividir en 3 grupos:

- **Queries basados en sintaxis SQL**, como RDQ, RDQL, SeRQL y SPARQL.

- **Queries que utilizan el propio RDF o XML** para definir las preguntas y las respuestas. En este grupo tenemos N3QL (que usa RDF pero en su versión abreviada Notation 3) y RDFQ.

- **Queries en otros lenguajes** o específicos de las aplicaciones, como el sistema de Queries de Protégé (Query Tab) y Versa.

W3C está adoptando SPARQL como candidato a recomendación desde Abril de 2006 [WWW62], y Protégé ha añadido un plug-in de RDQL en su última versión (3.2 beta).

Sin embargo, nosotros vamos a utilizar una versión más visual de los queries que es la ampliación *Query Tab* que proporciona Protégé. *Query Tab* permite realizar búsquedas de individuos que cumplen un cierto criterio sencillo. Además, la versión 3.2 beta de Protégé es, a fecha de la realización de este experimento, muy inestable y tiene graves problemas de transformación a DIG 1.1.

Como ejemplo, el agente buscará en su base de conocimiento los comentarios que contengan el texto “extintor” en su contenido (Ver Fig. 7.29). Nos devuelve los comentarios con identificadores 676, 817, 2695 y 2699.

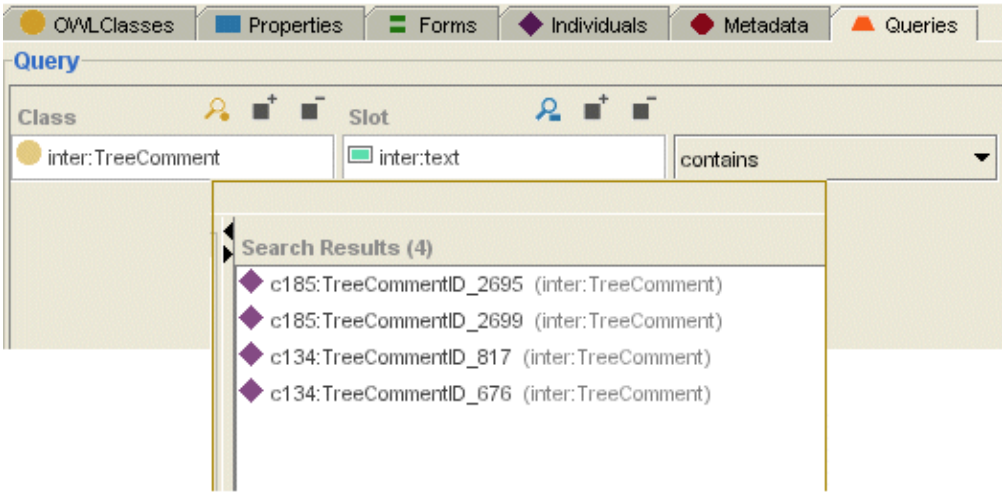


Fig. 7. 29. Búsqueda por substrings

En este punto, si fuera una búsqueda real, el usuario podría comprobar los textos de los comentarios y seleccionar los que más le convenga, o directamente solicitar al agente que le busque los nombre de los Threads distintos en los que se habla de extintores, o las comunidades en las que se ha hablado sobre extintores. Para resolver estas últimas consultas se necesitan razonadores, y es lo que se mostrará en el apartado siguiente.

7.4.8 Inferencias en OWL

Aunque hay algunas propuestas de lenguajes para realizar queries sobre OWL (como OWL-QL) y algunas extensiones para realizar inferencias sobre bases de conocimiento (como por ejemplo, nRQL para Racer), no están suficientemente desarrolladas o lo están sólo para algún tipo de razonador (nRQL). Por esta razón haremos las inferencias como hemos comentado anteriormente, es decir, por medio de la generación de clases definidas con nuestros parámetros de búsqueda y realizaremos una cálculo de tipos inferidos nos incluirá en la clase todos los individuos que cumplan las restricciones.

Siguiendo el ejemplo anterior, queremos saber los primeros elementos (instancia de la clase *RootItem*) de los threads donde se habla de extintores. Para realizar esto hacemos lo siguiente:

1. Generar una clase definida llamada *Comentarios_sobre_extintores* donde colocaremos las clases que nos ha devuelto las búsquedas por palabra clave. Esta acción se realizará manualmente, mientras que en

el caso de uso de agentes inteligentes tendría que ser el propio agente el que definiera estas clases enumeradas.

[Expr. 7.14]

$$\begin{aligned}&\text{Comentario\_sobre\_extintores} \equiv \{\\ &\text{c134} : \text{TreeCommentID\_817}\\ &\text{c134} : \text{TreeCommentID\_676}\\ &\text{c185} : \text{TreeCommentID\_2695}\\ &\text{c185} : \text{TreeCommentID\_2699}\}\end{aligned}$$

En la expresión 7.14 vemos la definición de la clase enumerada, donde c134 y c185 son los alias de los espacios de nombres correspondientes a la comunidades 134 y 185.

2. Mirar cuáles de los elementos de esta nueva clase son ya instancias de la clase *RootItem*. En nuestro ejemplo son los objetos de la clase *TreeComment* con ids 817 y 676.

3. Inferir qué instancias de la clase *RootItem* tienen algún elemento de esta nueva clase *Comentario_sobre_extintores* como descendiente:

[Expr. 7.15]

$$\begin{aligned}&\text{RootItem\_relacionado\_extintores} \equiv\\ &\text{RootItem} \cap \exists \text{hasDescendent.Comentario\_sobre\_extintores}\end{aligned}$$

En la expresión 7.15 vemos que hemos definido la clase que poblaremos con los resultados al realizar el cálculo de tipos inferidos.

El tiempo para inferir los individuos que pertenecen a esta nueva clase está sobre los 4 segundos (tiempo total 4,047 segundos, ver Fig. 7.30).

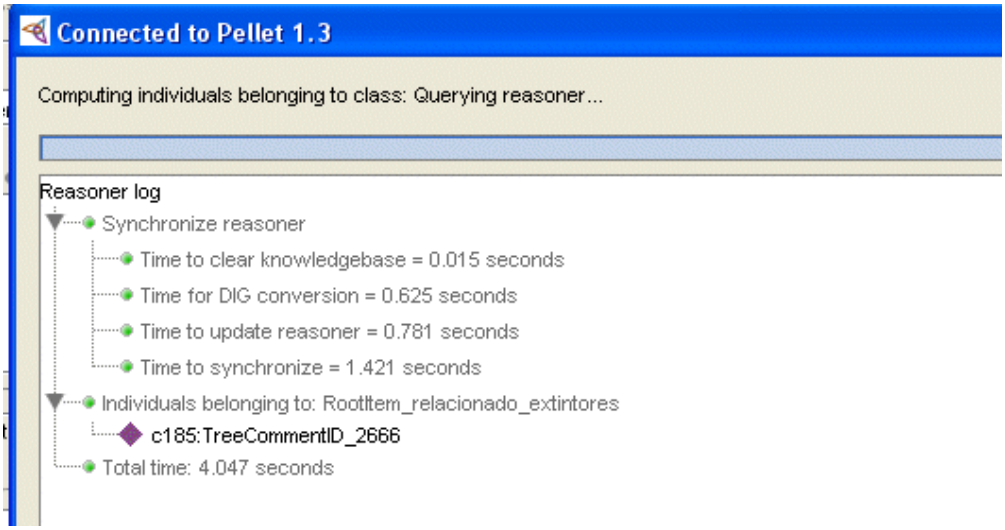


Fig. 7.30. Inferencia de los individuos que pertenecen a la nueva clase.

Como vemos en la figura 7.30, el motor de inferencia ha descubierto que en nuestro ejemplo, tanto el objeto *TreeComment* id=2695 como el objeto *TreeComment* con id=2699 están en el mismo thread cuyo elemento *RootItem* es la instancia de la clase *TreeComment* con id=2666.

7.5 Conclusiones de la validación del sistema

Como podemos ver el prototipo ha sido capaz de asistir al usuario en la búsqueda de comunidades (y por tanto, cursos) que estén relacionados con una temática determinada. También se podrían realizar otros tipos de inferencia, según se vaya ampliando la TBox con nuevos conceptos y definiciones relacionadas con los foros. Por tanto la principal conclusión de este capítulo radica en que la ontología especificada en esta Tesis es válida y útil para la realización de búsquedas inteligentes.

También validamos mediante esta prueba de concepto partes de la arquitectura propuesta en el capítulo 5, ya que hemos validado las partes relacionadas con OWL de la misma: la extracción de información semántica de experiencias educativa existentes y la realización de búsquedas inteligentes por parte de agentes. Se ha demostrado la posibilidad de extraer información semántica de los cursos (prototipo del extractor de información semántica), así como el uso de los agentes de la información semántica. La validación de la DHT semántica no es directa, sino que se basa en la validación de la DHT básica, bastante validada por sistemas como eMule, siendo lo novedoso la nueva utilidad que se le da y las claves de búsqueda que se utilizan. Por tanto, dado que las diferencias son principalmente el

añadido de la marca de ABox/TBox en la DHT, podemos considerar esta marca como parte de la clave utilizada por la DHT básica, por lo que proponemos sea un string ("ABox" o "TBox") añadido al nombre de recurso a buscar, de forma que al aplicar la función hash al nombre del recurso para obtener la clave, nos ubique la información de la TBox y la ABox de forma independiente (ver Fig. 7.31).

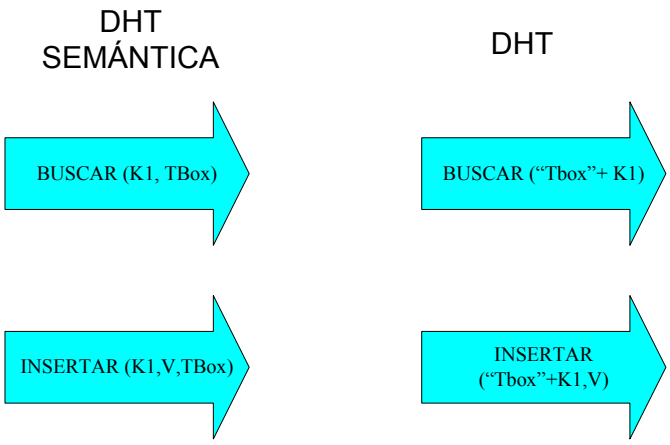


Fig. 7.31. Transformación de funciones de la DHT Semántica a DHT

Podemos observar en la figura 7.31 como se puede transformar una petición a la DHT Semántica por una petición a la DHT normal, por lo que sólo tenemos que utilizar la DHT actual y añadirle un interfaz que traduzca entre peticiones de la DHT Semántica a peticiones en DHT normal.

La parte de la arquitectura relativa a la distribución de contenidos educativos una vez localizados no se desarrollará más a fondo en la Tesis, ya que en este caso el interés se centra en las búsquedas inteligentes de recursos más que en el acceso a los mismos. De todas formas dicha distribución se propone que se base en los protocolos ya testeados por millones de usuarios: HTTP y BitTorrent.

Otra conclusión a la que llegamos es que tanto las herramientas como los estándares para OWL están aún en fase inicial de desarrollo. Si ahora se está empezando a estandarizar los queries sobre RDF (con SPARQL), podemos imaginarnos que queda aún tiempo para la estandarización de la siguiente capa OWL. Hasta el interfaz DIG 1.1 tiene indefiniciones que nos limitan la expresividad en OWL [RUT05]. Por otro lado, aunque Protégé es una herramienta muy testeada, incluso las últimas versiones (3.2 beta) tienen gran cantidad de errores y excepciones inesperadas. Respecto a los

razonadores, hemos tenido que trabajar con versiones beta debido al continuo desarrollo de los mismos. Incluso muchos no cubren OWL DL completamente (como es el caso de Racer).

Otra necesidad que detectamos para que las búsquedas semánticas sean más útiles, es la necesidad de filtrar elementos por capas inferiores a OWL. Es por esto, que en la validación de nuestro agente ya hemos introducido el concepto de búsquedas mixtas, en las que se puede buscar por palabras clave. Este proceso es similar a lo implementado en SPARQL para RDF, que permite definir filtros con queries de XQuery (por ejemplo, para buscar por palabras clave). De este modo debería desarrollarse un lenguaje de Queries en OWL que permita realizar filtros de capas inferiores, de forma que hiciera clases anónimas con los resultados obtenidos por ese filtrado, del mismo modo que hemos hecho nosotros en nuestro prototipo con los individuos con la palabra"extintor" en parte de su campo texto. Por tanto el modelo de filtrado mixto presentado en esta Tesis podría ser el ejemplo a seguir para la definición del lenguaje de consulta a OWL, así como la prueba realizada en este capítulo sería un ejemplo de funcionamiento por parte de los razonadores para resolver estas búsquedas mixtas.

Es importante destacar que OWL-DL también tiene la limitación a la hora de poder definir reglas, como por ejemplo propiedades que sean el encadenamiento de otras propiedades. Esto nos permitiría, por ejemplo, definir una propiedad que relacione un objeto *TreeComment* cualquiera con la instancia de la clase *LearningActivity* en la que está incrustado el foro al que pertenece:

[Expr. 7.16]

$$\forall X \in TreeComment, \forall Y \in RootItem, \forall Z \in ForumThread$$
$$hasDescendent(Y, X) \wedge isFirstItemOf(Y, Z) \Rightarrow belongToThread(X, Z)$$

$$\forall K \in Forum, \forall L \in SCORMLearningActivity$$
$$belongToThread(X, Z) \wedge isForumThreadOf(Z, K) \Rightarrow belongToForum(X, K)$$
$$belongToForum(X, K) \wedge relatedTo(K, L) \Rightarrow belongToSCORMLearningActivity(X, L)$$

Expresiones como la 7.16 nos permitiría encadenar búsquedas más complejas y por tanto con menos interacción por parte del usuario. Hemos de indicar que existe el lenguaje SWRL, que amplía OWL con reglas. SWRL está soportado también por Protégé pero no por la mayoría de los razonadores. La razón radica en que el uso de reglas implica algoritmos

distintos a los basado en Tableaux para probar teoremas. En estos casos es más acertado el uso de encadenamientos hacia delante (forward chaining o data driven) y hacia detrás (backward chaining o goal driven).

También hace falta la optimización de los algoritmos de los razonadores, ya que el crecimiento de la ontología genera un crecimiento exponencial de los recursos necesarios. Por ejemplo, la inferencia de tipos de la ontología con una comunidad (id=134) con 130 mensajes costaba de media 2,3 segundos, mientras que la clasificación de una otología que importe 4 comunidades (514 mensajes) nos da unos valores de rendimiento de unos 40 segundos para inferir los tipos.

La ontología propuesta abre las puertas para que en un futuro puedan ampliarse las aplicaciones de la misma. Por ejemplo, se podrían incluir conceptos que permitieran una clasificación basada en UDC [WWW64] o LCC [WWW65], lo que permitiría que cada foro o curso indicara sobre qué temáticas está trabajando, y de esta forma acotar nuestra búsqueda a las temáticas en que estamos interesados. Es este caso se podrían realizar dichas clasificaciones en una implementación OWL de la especificación del Simple Knowledge Organization System (SKOS) [MIL05], [DEH06], ya que es una especificación basada en RDF. De todas formas, al trabajar con suposición OWA, nuestra ontología es susceptible de ser ampliada en el futuro, siendo esta una de la mayores ventajas del uso de OWL.

Capítulo 8: Conclusiones y líneas de trabajo futuras

8.1 Introducción

En todo trabajo o estudio se necesita una contextualización donde se indiquen los fundamentos teóricos, las condiciones de trabajo y el estado del arte. También es necesario un conjunto de suposiciones y experimentos o pruebas de concepto que permitan validar las suposiciones. Además, todo trabajo se encuentra incompleto sin conclusiones. Las conclusiones son una de las partes esenciales del mismo, ya que permiten explicitar el conocimiento generado y sirven de punto de revisión. Será de este alto en el camino de donde partirán futuros trabajos. En este capítulo se indican a modo de resumen las conclusiones que han ido apareciendo en los distintos capítulos.

Se pretende que la Tesis Doctoral, que cierra un ciclo, abra otro ciclo, de modo que se convierta en un principo para futuras investigaciones y desarrollos, ya sea de forma individual o por medio de una colaboración entre miembros de la comunidad científica, que interactúen entre sí por medio de reuniones, artículos compartidos o participación en foros comunes. Por esta razón dedicaremos un apartado del capítulo a esbozar posibles lineas futuras.

Como apartado final, relacionaremos las conclusiones y resultados entre sí generando unas conclusiones globales, donde se corroborará la consecución de los objetivos planteados al principio de esta Tesis. Tras este apartado daremos por concluida la Tesis Doctoral.

8.2 Conclusiones

Las conclusiones principales se dividen en conclusiones respecto a las interacciones alumno-sistema (evaluaciones) y alumno-alumno (foros), conclusiones tras la propuesta de arquitectura que soporte un sistema para facilitar la interoperabilidad de objetos de aprendizaje, respecto a la especificación de la ontología propuesta y validación del conjunto.

Respecto a las evaluaciones en teleformación, podemos concluir que es posible transformar las especificaciones existentes y mayoritariamente utilizadas (IMS QTI) en una ontología basada en OWL DL, así como enriquecerla con conceptos de otras especificaciones (TeML) que enriquecen la información a procesar. También se demuestra la ventaja de incluir las interacciones constructivistas en la información a almacenar en los repositorios de contenidos, en lugar de dejar perder esa información en el LMS. Se ha comprobado, a su vez, la falta de ontologías que describan a los foros, cuando es un elemento en el que puede haber una gran cantidad de información semiestructurada, y además se ha demostrado que es una herramienta muy útil para la resolución de dudas puntuales en las grandes comunidades virtuales de aprendizaje.

A la hora de proponer la arquitectura, vemos como óptima una arquitectura P2P por razones de escalabilidad y resistencia a fallos. Al ser una arquitectura P2P, necesita nodos que estén interesados en formar parte de la red durante mucho tiempo. En nuestro caso han sido los propios ofertantes de Objetos Educativos los que realizan la tarea de mantener una estructura mínima. También nos damos cuenta que es muy importante atender a los derechos de autor de los contenidos educativos de la red. Es por esto por lo que se se han restringido los lugares de almacenaje del Objeto Educativo.

Concluimos a su vez que la arquitectura es una arquitectura para la web semántica de uso general, puesto que sirve para la interoperabilidad de cualquier ontología OWL DL que queramos, debido a que no se ha realizado ninguna restricción ni suposición sobre las clases descritas en la ontología..

Respecto a la especificación de la ontología hemos visto que el nivel de lenguaje OWL DL optimiza, basándonos en las capacidades de cálculo y algoritmos actuales, la relación entre la riqueza descriptiva y la posibilidad de resolver las inferencias presentadas. Además, se ha desarrollado una ontología que abarca los tres principales modos de aprendizaje, como vemos en la figura 8.1, el enfoque conductista, el enfoque constructivista y podemos considerar que el enfoque cognoscitivista se ofrece por medio de los contenidos estáticos.

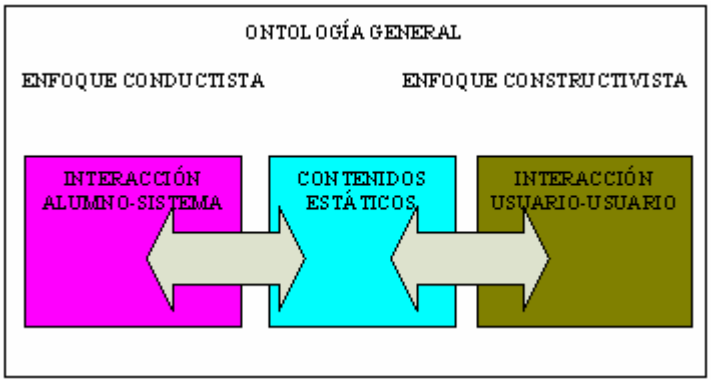


Fig. 8.1. Inclusión de los distintas teorías de aprendizaje en la ontología

Por último, en la validación hemos podido comprobar que las herramientas para generar y trabajar con OWL son aún inestables , ya que la temática es bastante novedosa. Ocurre esto también en los razonadores, donde parece que actualmente Pellet es la mejor opción para un sistema como el nuestro.

8.3 Líneas de trabajo futuras

Las líneas futuras propuestas se basan en los dos apartados principales de la Tesis, que son la ontología especificada para teleeducación y la arquitectura propuesta para la web semántica, y que nos permite explotar la ontología especificada.

Respecto a la ontología, aunque se ha descrito en OWL DL, hemos visto como la introducción de reglas en SWRL mejora la expresividad del lenguaje, por lo que sería interesante profundizar esta vía ampliando la

ontología especificada con reglas, de modo que el sistema fuera más inteligente. Tendremos que esperar a razonadores optimizados para SWRL para que sea interesante abarcar esta tarea.

También se propone realizar en un futuro ampliaciones puntuales de la ontología especificada, como puede ser el desarrollo en profundidad de otras interacciones constructivistas, como son los trabajos o tareas en grupo, los trabajos con documentos compartidos y las faqs. Incluso podrían ampliarse los distintos conceptos con la aparición de nuevos tipos de preguntas en las evaluaciones. Es realmente en los campos de las interacciones de los distintos actores de una experiencia educativa donde hay espacio de desarrollo, ya que sobre los contenidos estáticos se ha trabajado mucho en los últimos años y ya existen especificaciones aceptadas.

Otra oportunidad de investigación que surge de esta Tesis es el estudio del uso de la ontología propuesta no sólo para la interoperabilidad y búsquedas, sino para que la entienda el propio LMS y pueda publicar directamente una experiencia educativa, al estilo de lo que puede hacer un LMS con la información del manifiesto.

En la propuesta de arquitectura también hay líneas de trabajo interesantes, como son el desarrollo de un publicador de contenidos completo en lugar de realizar una prueba de concepto como ha ocurrido en capítulos anteriores. Además, se propone en un futuro implementar la DHT Semántica por medio de una capa sobre una DHT clásica en la que se realicen las transformaciones entre las arquitecturas, y evaluar parámetros de rendimiento de esta estructura, por medio de simuladores de varios cientos de nodos.

Incluso existe campo de desarrollo en la definción del proceso de obtención y ensamblaje de los objetos educativos elegidos, ya que se propone una metodología variable en función del tamaño del recurso. También sería interesante probar la arquitectura con otras ontologías OWL-DL para comprobar su eficacia como arquitectura de web semántica en general.

Otra posible línea sería trabajar con los futuros motores de inferencias, y modificar los interfaces, de forma que no sea necesario utilizar DIG como método de comunicación. Se podría implementar distintos agentes inteligentes que ya solicitaran directamente al usuario información y le presentaran un interfaz amigable para que el usuario pudiera generar la consulta basándose en los conceptos definidos en la TBox de la ontología.

Será a partir de la mejora de algoritmos de inferencia y de mejoras en la computación de los motores cuando se produzca un verdadero despegue de las tecnología asociadas a la web semántica, como ocurrió en su día con las bases de datos relacionales.

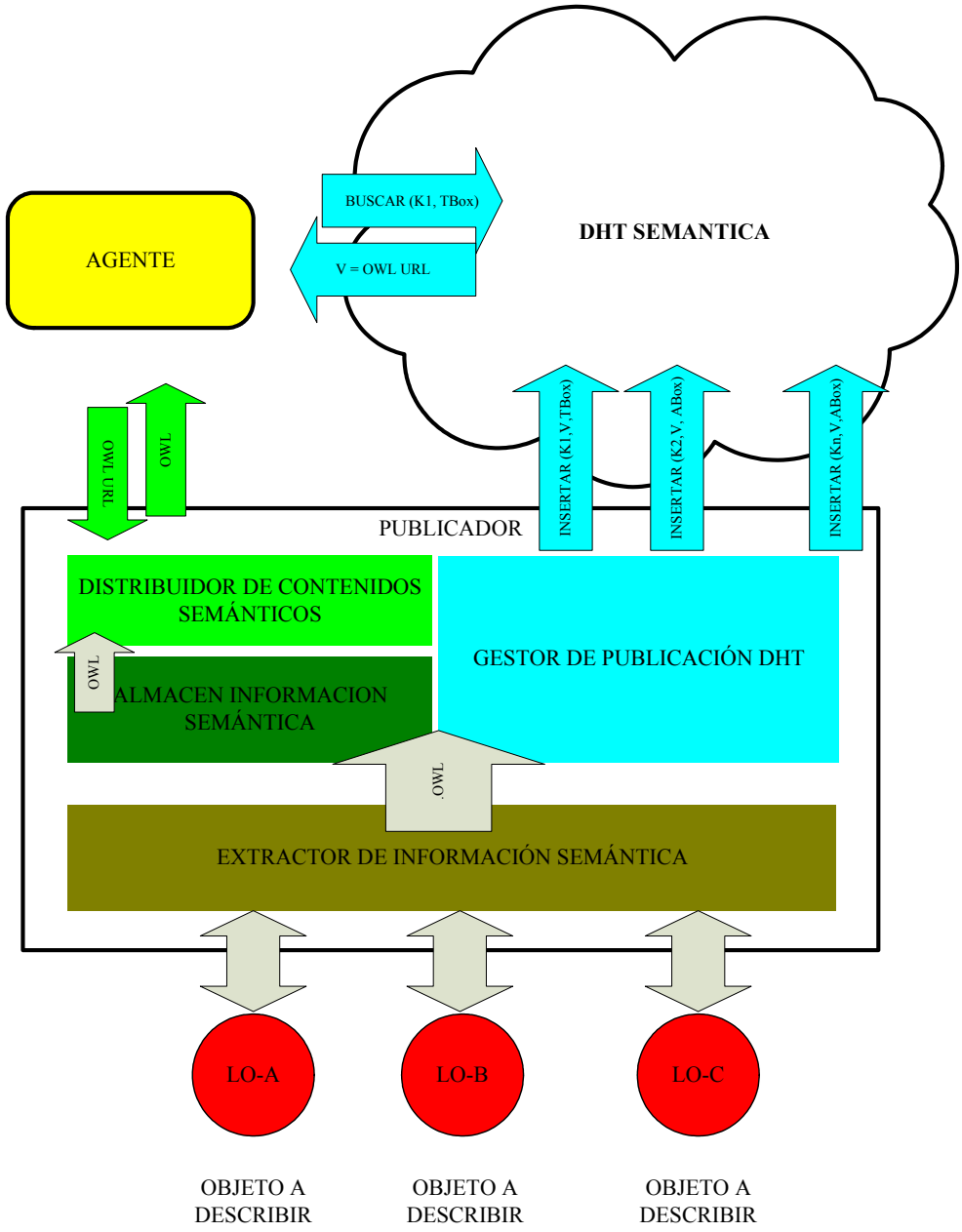


Fig. 8.2. DHT Semántica

8.4 Conclusiones finales

Como conclusión final de esta Tesis vemos que hemos conseguido los objetivos planteados al principio de la misma. Por una parte se han estudiado y aplicado las últimas tendencias en interoperabilidad de recursos educativos por medio de lenguajes ontológicos, incluyendo a su vez información respecto a las interacciones de los usuarios por medio de los foros y por medio de las evaluaciones. Esta información se almacena para su uso posterior por parte de los agentes inteligentes.

Para alcanzar estos objetivos primero se han investigado el estado actual de la web, desde la web sintáctica a la web semántica, analizando la estructura de capas existente para poder luego desarrollar este trabajo sobre una base de conocimiento firme. También se ha analizado el estado del arte en la Teleeducación desde el prisma de la web semántica, estudiando las ontologías educativas extraídas de los estándares y modelos de referencia que tienen más importancia en la actualidad. Además, se ha profundizado en los fundamentos lógicos que nos han permitido especificar la ontología propuesta y fundamentar la razón por la cual se ha escogido OWL como lenguaje ontológico en la Tesis.

Gracias al análisis realizado sobre las ontologías educativas más importantes, hemos extraído las relaciones implícitas de las mismas, que nos han servido para ser explicitadas en nuestra ontología. De hecho, la ontología especificada en esta Tesis reutiliza los conceptos ya aceptados y comprobados provenientes de otras ontologías, escogiendo los más adecuados para nuestros propósitos. Como ejemplo destacable, respecto a las interacciones alumnos-sistema nos hemos basado en conceptos de SCORM (relación entre los resultados de evaluaciones y los objetivos conseguidos), IMS QTI (conceptualización de preguntas, proceso de obtención de resultado y tipos de preguntas), TeML (sistema de feedbacks, tipos de preguntas y relación de preguntas con posibles respuestas, así como dificultades de las evaluaciones) y IEEE 1484 (tipos de preguntas en las evaluaciones y campos relacionados). En los casos que se ha constatado que no existía ninguna ontología reaprovechable para nuestros objetivos, ha sido necesario construir nuevos conceptos y relaciones, que además de ser utilizados en esta Tesis podrían servir también para estudios posteriores. La reutilización de conceptos ya probados ha resultado de gran utilidad para nuestros fines, ya que facilita la importación de información a nuestro sistema puesto que existen grandes similitudes conceptuales con la mayoría de sistemas basados en estos estándares.

De esta forma, tras enumerar las distintas teorías de aprendizaje que queremos incluir en nuestro sistema (conductismo, cognoscitivismo y constructivismo), se han tenido que describir las interacciones de los usuarios más importantes que se producen en un proceso formativo por internet, es decir las interacciones alumno-sistema y las interacciones entre usuarios, que no figuraban hasta el momento en ninguna de las ontologías analizadas con el objetivo de permitir búsquedas para la reutilización. Dicha información en nuestro sistema es accesible para los agentes inteligentes con el objetivo de realizar búsquedas tanto por palabras clave como utilizando motores de inferencia, ya que se ha definido, desarrollado y evaluado una ontología expresada en OWL DL con esas características. A raíz de esto se ha presentado también un ejemplo de búsquedas mixtas RDF/OWL, lo que amplía aún más el campo de acción de los agentes inteligentes. Esta definición de búsqueda mixta ha resultado muy positiva a la hora de facilitar las tareas de localización, ampliando enormemente las posibilidades de los agentes.

Asimismo, se ha propuesto y definido una arquitectura de explotación, basada en el nuevo concepto de web semántica, de forma que pretendemos que esta arquitectura impulse su uso en la teleeducación. Hemos validado también la ontología especificada de forma que se han presentado distintas posibilidades de uso de agentes inteligentes en la teleeducación, proponiendo un sistema de localización donde el agente presenta al usuario posibilidades de búsqueda, el usuario define la búsqueda utilizando los términos ofrecidos por el agente y por último el agente presenta los posibles resultados al usuario. De esta forma se mejora considerablemente el resultado obtenido respecto a una simple búsqueda por palabras clave.

Esta Tesis sienta las bases de una arquitectura que facilita la localización en la web semántica de recursos educativos reutilizables para impulsar la teleeducación en todos sus ámbitos. Para ello se basa en el análisis riguroso de las necesidades y estándares actuales, realizando una especificación que se ha validado para demostrar su utilidad en el campo de la interoperabilidad de recursos educativos. Por todo esto, esperamos que esta Tesis sirva también para desarrollos futuros en los que se implementen agentes inteligentes con motores de inferencia más avanzados y se pueda llevar a cabo una arquitectura similar a la propuesta, en la que se facilite el acceso a contenidos semánticos y se permitan grandes variaciones de carga. También esperamos que se desarrollen ontologías nuevas basadas en la ontología especificada en esta Tesis, de forma que se enriquezca el conocimiento de la teleformación por parte de los agentes inteligentes, revertiendo a su vez en

un mejor servicio para los usuarios que pretendan reutilizar contenidos educativos.

Anexo 1. Glosario

Términos y Acrónimos

ABox. parte de una base de conocimiento en lógica descriptiva relativa a las instancias de clases.

Agente Inteligente. Toda entidad que percibe y actúa sobre su entorno, desarrollando cierto grado de inteligencia . En esta Tesis nos referiremos a agente inteligente como las entidades no humanas que nos facilitarán la búsqueda de recursos educativos, por medio de la realización de inferencias sobre la base de conocimiento que han adquirido en la web semántica.

AHA. Adaptative Hypermedia Architecture. Sistema que utiliza la web para generar y mostrar redes semánticas.

AICC. Aviation Industry Computer-Based-Training Committee. Organismo encargado de estandarizar la formación basada en computadores de la industria aérea.

AIMS. Agent-based Information Management System. Plataforma de teleformación basada en redes semánticas.

Ajax. Asynchronous JavaScript and XML. Técnica de programación de alto nivel que utiliza componentes que se ejecutan en la web por medio de Javascript en el cliente y comunicación con el servidor vía XML.

CDN. Content Distribution Network. Red de ordenadores en internet que colaboran para distribuir contenidos de forma transparente al usuario.

Clasificación de una ontología. Proceso de inferencias por el que se explicitan todas las relaciones de herencia entre clases de una ontología.

Consistencia de una ontología. Proceso de inferencias por el que se confirma que no existe ninguna incongruencia en la ontología, es decir, que todas las clases de la misma pueden tener instancias.

DIG. Description Logic Implementation Group Interface. Interfaz que define un protocolo de intercabio de ontologías descritas en lógica descriptiva, que suelen aceptar la mayoría de razonadores lógicos.

DL. Lógica descriptiva. Subtipo de lógica de primer orden donde se separa completamente entre clases, propiedades e individuos, y las relaciones tienen un único sujeto y un único predicado.

DTD. Document Type Definition. Lenguaje para definir la estructura de un documento XML.

EGA. Extreme Groups Approach. Sistema de obtención de resultados estadísticos basado en el estudio de los valores extremos de una muestra.

FOL.First Order Logic. Lógica de primer orden. Lógica predicativa que permite la relación entre conteptos por medio de propiedades.

GPL. GNU General Public License. Licencia de software libre.

HTTP/GET. Método de solicitud de un recurso en el protocolo HTTP en el que el propio mensaje de petición incial indica la solicitud por completo.

HTTP/POST. Método de solicitud de un recurso en el protocolo HTTP en el que es necesario un segundo envío con más información por parte del solicitante para definir completamente la petición solicitada solicitado.

IEEE. Institute of Electrical and Electronics Engineers. Organización internacional sin ánimo de lucro que busca el desarrollo de la tecnología. IEEE prepara y publica estándars en los distintos campos, como la educación.

IEEE LTSC. IEEE Learning Technology Standards Committee. Comité encargado del estudio, desarrollo y publicación de los estándares de IEEE relativos al aprendizaje.

IMS o IMS Global. Instructional Managed Systems.Organización sin ánimo de lucro dedicada a presentar especificaciones que faciliten la interoperabilidad de los sistemas de aprendizaje y los contenidos de aprendizaje.

IMS CMI. Utilización dentro de IMS de la especificación de AICC denominada Content Management Interface. Esta especificación es relativa a las comunicaciones entre el aprendiz y el LMS.

IMS CP. IMS Content Packaging. Especificación sobre el empaquetamiento de contenidos de aprendizaje

IMS LD. IMS Learning Design. Especificación para ladefinición de unidades de aprendizaje y para definir reglas de paso de objetivos.

IMS QTI. IMS Question & Test Interoperability. Especificación relativa a las distintas evaluaciones en un curso on-lien.

IMS SS. IMS Simple Sequencing. Especificación que trata de definir las reglas de navegación en una experiencia formativa.

IPSec. Internet Protocol security. Extensión del protocolo IP con cifrado para permitir servicios de autenticación.

JSF. JavaServer Faces Technology. Tecnología de programación Java para servidores web basado en el paradigma modelo vista. Se construye sobre JSP.

JSP JavaServer Pages Technology. Tecnología de programación Java para servidores web que permite generar contenido dinámico para la web.

LMS. Learning Management System. Software ejecutado en uno o varios servidores web que permite la gestión de un curso on-line.

LOM. Learning Object Metadata. Modelo de información que define los metadatos de los distintos objetos educativos. El estádard más aceptado es IEEE 1484.12.1 – 2002 (IEEE LOM).

LOWID. Identificador de nodo en una red P2P como eMule que no puede aceptar conexiones inesperadas desde la web, normalmente por estar detrás de un firewall. Los nodos con LOWID son penalizados en los sistemas P2P ya que no permiten utilizar sus recursos de forma inesperada.

LP. Lógica proposicional. Lógica en que los elementos relacionados por operadores lógicos son indivisibles, siendo la lógica más sencilla tratada en esta Tesis.

MATHML. Mathematical Markup Language. Lenguaje basado en XML para describir conceptos y operaciones matemáticas.

NAT. Network address translation. Traductor de direcciones de red utilizado normalmente para permitir a un conjunto de ordenadores de un red privada acceder a internet utilizando una única dirección IP para todos..

OWA. Open World Assumption. Suposición en las inferencias lógicas que se basa en el hecho que si no puedo demostrar que algon no es verdadero no significa que es falso. Esta suposición es necesaria cuando se hacen inferencias sobre bases de conocimiento que se van ampliando con nueva información.

OWL. Web Ontology Language. Especificación de W3C para la inserción de ontologías en la web. Lenguaje web basado en RDF para la definición de ontologías.

OWL-S. Ontología para servicios web basada en OWL.

P2P. Peer to Peer. Atributo de una red que indica que todos los elementos pueden compartir una o todas las funciones del sistema, evitando centralizaciones.

RDF. Resource Description Framework. Especificación de W3C para la realización de aserciones en la web. Lenguaje que permite relaciones un recurso sujeto con un recurso objeto por medio de un tercer recurso denominado predicado.

RDF-QEL. RDF-based Query Exchange Language. Lenguaje para solicitar búsquedas en documentos RDF

RFC. Request For Comments. en W3C son documentos que realizan nuevas propuestas, protocolos o definiciones. Internet Engineering Task Force (IETF) adopta algunos de estos documentos como estándars para Internet. **RIA**. Rich Internet Aplication. Tecnologías que se basan en simular la mayoría de funcionalidades de aplicaciones locales que ejecutaríamos normalmente desde nuestro ordenador en

una aplicación web, normalmente por medio de técnicas de programación como Ajax.

RQL. RDF Query Language. Lenguaje para solicitar búsquedas en documentos RDF

SCO. Sharable Content Object. En el modelo SCORM se refiere a cualquier objeto educativo que sea lo suficiente independiente para se compartido.

SCORM. Shareable Content Object Reference Model. Modelo que reutiliza un conjunto de estándars y especificaciones para la teleformación.

SGML. Standard Generalized Markup Language. Lenguaje generalizado para generación de otros sublenguajes de marcas, como XML.

SKOS. Simple Knowledge Organisation System. Lenguajes diseñados para la representación de tesaurios y clasificaciones, basado en RDF y RDFS.

SSL. Secure Sockets Layer, actualmente evolucionada a Transport Layer Security (TLS). Protocolo que permite la comunicación segura en internet.

TLS, ver SSL

SWRL. Semantic Web Rule Language. Lenguaje fruto de la unión de OWL y RULEML para la generación de ontologías utilizando lógica descriptiva y reglas.

TBox. Parte de una base de conocimiento en lógica descriptiva relativa a las clases y su estructura, a modo de taxonomía.

TTL. Time To Live. Parámetro de los paquetes de información en una red que indica normalmente el número de saltos que restan para desaparecer si no ha alcanzado su destino.

UML. Unified Modeling Language. Es un lenguaje de especificación para modelación de objetos. Se utiliza principalente en el campo de programación basada en objetos.

UNA. Unique Name Assumption. Suposición en las inferencias lógicas que se basa en que cada recurso tiene un único URI que lo identifica, por lo que podemos concluir que dos objetos representados por distintas URIs no pueden ser el mismo objeto.

URI. Uniform Resource Identifier. Secuencia de caracteres que identifica un recurso abstracto o físico.

URL. Uniform Resource Locator. URI que permite además localizar de forma automática el recurso.

W3C World Wide Web Consortium. Es la principal organización internacional que define los estándars para la web.

XHTML. Extensible HyperText Markup Language. Lenguage de marcas basado en XML, con la misma utilidad que HTML pero más extricto en la sintaxis.

XML. Lenguaje de marcas recomendado por W3C que se utiliza generalmente para la transmisión de datos por la web

XMLS. XML Schema. Lenguaje para definir la estructura de un documento XML, expresado a su vez en XML, en contraste con DTD que tiene una sintaxis propia.

Anexo 2. Referencias

Artículos e Informes

[ADL04a] SCORM. Advanced Distributed Learning (ADL) Sharable Content Object Reference Model (SCORM®). Content Aggregation Model (CAM) Version 1.3.1. Julio 2004.

[ADL04b] SCORM. Advanced Distributed Learning (ADL) Sharable Content Object Reference Model (SCORM®). Run-Time Environment (RTE) Version 1.3.1. Julio 2004.

[ADL04c] SCORM. Advanced Distributed Learning (ADL) Sharable Content Object Reference Model (SCORM®). Sequencing and Navigation (SN) Version 1.3.1. Julio 2004.

[AICC04] AICC CMI001, CMI Guidelines for Interoperability, Version 4. Agosto 2004.

[AKE04] A. Akella, J. Pang, B. Maggs, S. Seshan, A. Shaikh. "A Comparison of Overlay Routing and Multihoming Route Control". ACM Sigcomm. Septiembre 2004.

[ANA05] R. Anane et al. "A Web Services Approach to Learning Path Composition". Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies (ICALT'05). Taiwan. Julio 2005.

[ANI04] L. Anido, J. Rodríguez, M. Caeiro, J. Santos. "A Focal Access Point to Follow the Standardization of Learning Technologies". Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT'04). Finlandia. Agosto 2004.

[ANK04] A. Ankilenkar, M. Paolucci et al "OWL-S: Semantic Markup for Web Services". White paper for OWL. 2004.

[ALO05] F.Alonso, G. López, D. Manrique, B. Ruiz. "XML repositories definition for adaptive e-learning courses". 3rd International Conference on Multimedia and Information. Junio 2005.

[ALO06] J.A. Alonso, M.J. Hidalgo, F.J. Martin, J.L. Ruiz. "Formalización de la lógica descriptiva ALC en PVS". Grupo de Lógica Computacional. Universidad de Sevilla. Marzo 2006.

[ANS03] E. Ansell, J. Park. "Tracking Tech Trends". Education Week, 22(5). Pp. 43-44, 48. Julio 2003.

[ARE01] C. Areces, M. de Rijke. "From Description to Hybrid Logics, and Back". In Wolter, F., Wansing, H., de Rijke, M., and Zakharyaschev, M., editors, Advances in Modal Logic, CSLI Publications. Pp. 17–36. 2001.

[ARO03] L. Aroyo, S. Pokraev, R. Brussee. "Preparing SCORM for the Semantic Web. Paper presented at the International Conference on Ontologies", Databases and Applications of Semantics (ODBASE'03). Italia. Noviembre 2003.

[ARO04] L. Aroyo, D. Dicheva. "The New Challenges for E-learning: The Educational Semantic Web". Educational Technology & Society, 7 (4), Pp: 59-69. 2004

[BAL04] M. Baldoni, C. Baroglio, V. Patti, and L. Torasso. "Reasoning about learning object metadata for adapting SCORM courseware". EAW'04: Methods and Technologies for personalization and Adaptation in the Semantic Web. Pp 4–13. Holanda. 2004.

[BAL96] S. Balli, L. Diggs, "Learning to Teach with Technology: A Pilot Project with Preservice Teachers" Educational Technology, 36, 1, Pp 56-61. Febrero 1996.

[BAR04] J.P. Baraldi, D.O. Romero, C.M. Rinaldi. "Gateway para el Reciclaje de Sistemas E-learning que no cumplen con SCORM". LatinEduca 2004.

[BECH03] S. Bechhofer. "The DIG Description Logic Interface: DIG/1.1". Febrero 2003.

[BECH03] S. Bechhofer, I. Horrocks, P. Patel-Schneider. "Tutorial on OWL". 2nd International Semantic Web Conference. EEUU. Octubre 2003.

[BEN03] D. Benz. "Description Logics, the logical foundation of the Semantic Web". Seminar on Semantic Web Descrition Logics. Noviembre 2003.

[BES05] P. Besana, D. Robertson, M. Rovatsos."Exploiting interaction contexts in P2P ontology mapping". 2nd. International Workshop on Peer to Peer Knowledge Management. EEUU. Julio 2005.

[BHA05] K. Bhatia. "Peer-To-Peer Requirements On The Open Grid Services Architecture Framework". OGSAP2P Research Group. Julio 2005.

[BID04] M. Biddulph. "Crawling the Semantic Web". Journal of Web Semantics. 2004.

[BRA85] R.J. Brachman, J. Schmolze. "An Overview of the KL-ONE Knowledge Representation System", Cognitive Science 9. 1995.

[BUY02] R. Buyya. "Convergence Characteristics for Clusters, Grids, and P2P networks". Panel at the P2P conference. Suecia.

[CAE04] M. Caeiro, L. Anido, M. Llamas. "Towards IMS-LD Extensions to Actually Support Heterogeneous Learning Designs. A Pattern-based Approach". Proceedings 4th IEEE International Conference on Advanced Learning Technologies (ICALT'04). Finlandia. Agosto 2005.

[CAE05] M. Caeiro, L. Anido, M. Llamas. "A Perspective and Pattern-based Evaluation Framework for EMLs' Expressiveness for Collaborative Learning: Application to IMS LD". Proceedings 5th IEEE International Conference on Advanced Learning Technologies (ICALT'05). Taiwan. Julio 2005.

[CAP04] O. Caprotti, M. Dewar, D. Turi. "Mathematical Service Matching Using Description Logic and OWL". Mathematical Knowledge Management: Third International Conference, Proceedings. Polonia. Septiembre 2004.

[CHA04] W. Chang, H. Hsu, T.K. Smith, C. Wang. "Enhancing SCORM metadata for assessment authoring in e-Learning". Journal of Computer Assisted Learning 20. Pp 305-316. Agosto 2004.

[CHA98]V.K. Chaudhri, A. Farquhar. "Especificación Open Knowledge Base Connectivity 2.0.3". Abril 1998.

[CON01] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P.F. Patel-Schneider, L. A. Stein. "DAML+OIL Reference Description" Marzo 2001.

[COO02] B.F. Cooper, H. Garcia-Molina. "Peer-to-peer resource trading in a reliable distributed system". 1st International Workshop on Peer-to-Peer Systems. EEUU. Marzo 2002

[CRE05]A. Cregan, M. Mochol, D. Vrandecic, S. Bechhofer. "Pushing the limits of OWL, Rules and Protege. A simple example". OWL Workshop on Experiences & Directions. Irlanda. Noviembre 2005.

[DAM05] V. Damjanovic, M. Kravcik, V. Devedzik. "eQ: An Adaptive Educational Hypermedia-based BDI Agent System for the Semantic Web". Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05). Taiwan. Julio 2005.

[DEH06] S. Dehors, C. F. Zucker, R. D. Kuntz "Reusing Learning Resources based on Semantic Web Technologies". Proceedings 6th International Conference on Advanced Learning Technologies. ICALT. ISBN 978-0-7695-2632-4. EEUU. 2006.

[DICK04] I. Dickinson. "Implementation experience with the DIG 1.1 specification". HP Technical Report. Mayo 2004.

[DOD01] P. Dodds (editor). "The SCORM overview. ADL Sharable Content Object Reference Model". 2001.

[DRI04] L.P. Dringus, T. Ellis. "Using data mining as a strategic for assessing asynchronous discussion forums". Computers & Educations 45. Pp: 141-160. Mayo 2004

[EDE06] A.H. Eden, Y. Hirshfeld, R. Kafman. "Abstraction Classes in Software Design". IEE Software Vol. 153, No. 4. Pp: 163–182. Agosto 2006.

[EDT] Edutools: iniciativa de WCET – the Western Cooperative for Educational Telecommunications para realizar comparaciones en la comunidad de e-Learning. http://www.edutools.info.

[ELL05] H.Ellis. "A Collaborative Approach to a Course on the Semantic Web". 35th ASEE/IEEE Frontiers in Education Conference. EEUU. Octubre 2005.

[FAR06] R.Farré et al. Notas de clases para Introducción a la Lógica. Universidad Politécnica de Cataluña. Diciembre 2006.

[FED05] Fedora Open Source Repository Software:White Paper. October 2005. http://www.fedora.info

[FEL02] P.A. Felber, E.W. Biersack, L. Garcés-Erice, K.W. Ross, G. Urvoy-KellerData "Indexing and Querying in DHT Peer-to-Peer Networks". 2002.

[FIE00] Fielding, Roy Thomas. "Architectural Styles and the Design of Network-based Software Architectures". Doctoral dissertation, Capítulo V. University of California. EEUU 2000.

[FOS03] I. Foster, A. Tamnitchi. "On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing". 2003.

[FOWL05] C. Fowler. “Revisiting the Delta ontology”. Disponible en http://www.essex.ac.uk/chimera/

[FRA01] E. Franconi. “Description logics lecture notes”. Pres. slides, University of Manchester, Marzo 2002.

[GAM01] C. Qu, J. Gamper, W. Nejdl, ”A Collaborative Courseware Generating System Based on WebDAV, XML, and JSP”, IEEE International Conference in Advanced Learning Technologies 2001 (ICALT 2001), Pp. 441-442. USA. Agosto 2001.

[GAR03] D. Garlan, M. Shaw. “An Introduction to Software Architecture,” Advances in Software Engineering and Knowledge Engineering, Volumen I. World Scientific. EEUU. 1993.

[GAR04] A. Garcia Jiménez. “Instrumentos de representación del conocimiento: Tesauros frente a Ontologías”. Anales de Documentación, número 007. Pp: 79-95. Universidad de Murcia.

[GAR06] T. Gardiner, I. Horrocks, D. Tsarkov. “Automated Benchmarking of Description Logic Reasoners”. Mayo 2006.

[GER03] J. Gerke, D. Hausheer, J. Mischke, B. Stiller. “An Architecture for a Service Oriented Peer-to-Peer System (SOPPS)” Praxis der Informationsverarbeitung und Kommunikation. ETH. Pp: 90-95. Suiza. 2003.

[GOL96] M. Goldberg, S. Salari, P. Swoboda. “World Wide Web Course Tool: An Environment for Building WWW-Based Courses”, Computer Networks and ISDN Systems, 28. Pp: 1219–1231. Paises Bajos. 1996.

[GOR93] M.J.C. Gordon, T. F. Melham. “Introduction to HOL: a theorem proving environment for hier order logic”. Cambridge University Press. 1993.

[GRUB93] T. R. Gruber. “Towards Principles for the Design of Ontologies Used for Knowledge Sharing". In Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers. 1994.

[GUE05] A. Guerrero, V. A. Villagrá, J.E. López. “Definición del comportamiento de gestión de red con reglas SWRL en un marco de gestión basado en ontologías OWL”. Irlanda. Octubre 2006.

[GUO06] W. Guo, D. Chen. “Semantic Approach for e-learning System”. Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences. China. Junio 2006.

[GUP01] G. Gupta, E.Pontelli, K.Ali, M. Carlsson, M. Hermenegido. “Parallel Execution of Prolog Programs: A Survey”. ACM Transanctions on Programming Languages and Systems, Vol 23, Num 4, Pp: 472, 602. ACM Press, Julio 2001.

[HAA04] P. Haase, et al. ”Bibster — A Semantics-Based Bibliographic Peer-to-Peer System”. 3rd International Semantic Web Conference (ISWC 2004). Pp. 122–136. Japón. Noviembre 2004.

[HAA05] V. Haarslev, R. Möller, M. Wessel. “Description Logic Inference Technology: Lessons Learned in the Trenches”. Proc. International Workshop on Description Logics. 2005.

[HAR03] Harth, Andreas. “An Integration Site for Semantic Web Metadata”. 2003.

[HAT02] M. Hatala, G. Richards. “Learning Object Repository Technologies for TeleLearning: The Evolution of POOL and CanCore”. Informing Sciencia + IT Education Conference. Irlanda. Junio 2002.

[HAY01] Patrick Hayes, Christopher Menzel “A Semantics for the Knowledge Interchange Format”. 2001.

[HAY04] M. Haynos. “Perspectives on grid: Grid computing - next-generation distributed computing. What's different between grid computing and P2P, CORBA, cluster computing, and DCE?”. Enero 2004. http://www-128.ibm.com/developerworks/grid/library/gr-heritage/

[HAR03] A. Hart. “An Integration Site for Semantic Web Metadata”. 2003.

[HAT02] M. Hatala, G. Richards. “Making a Splash: a Heteregeneous Peer-To-Peer Learning Object Repository Technology”. 2002.

[HAY01] P. Hayes, C. Menzel “A Semantics for the Knowledge Interchange Format”. 2001.

[HAY04] M. Haynos. “Perspectives on grid: Grid computing -- next-generation distributed computing. What's different between grid computing and P2P, CORBA, cluster computing, and DCE?”. Enero 2004. http://www-128.ibm.com/developerworks/grid/library/gr-heritage .

[HAR03] R. Hartwig, M. Herczeg. “A Process Repository for the Development of E-Learning Applications” Proceedings of the The 3rd IEEE International Conference on Advanced Learning Technologies (ICALT’03). Grecia. Junio 2003.

[HEF99] J. Heflin, J. Hendler, S. Luke. “SHOE: A Knowledge Representation Language for Internet Applications”. Technical Report, University of Maryland, 1999.

[HEN00] J. Hendler, D. L. McGuinness. “The DARPA Agent Markup Language”. IEEE Intelligent Systems, 15. 2000.

[HOR03] I. Horrocks, P.F. Patel-Schneider, F. van Harmelen. “From SHIQ and RDF to OWL: The Making of a Web Ontology Language”. 2003.

[HOR04] M. Horridge, H. Knublauch et al. “A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools”. The University Of Manchester. Reino Unido. Agosto 2004.

[HOW04] S.L. Howell. “E-Learning and Paper Testing: Why the Gap?”. EDUCASE Quaterly, 4 Pp: 8-10. 2004.

[HUE97] J. L. Hueso, A. Martínez, R. Romero, J. R. Torregrosa . “Web y Multimedia al servicio de la enseñanza de las Matemáticas”. Edutec’97. 2007. http://www.ieev.uma.es/edutec97/edu97_c2/2-2-16.htm.

[IAN02] I. Horrocks, U. Sattler. “Logical Foundations for the Semantic Web”. University of Glasgow. 2002.

[IAN03] I. Horrocks and P. F. Patel-Schneider. “Three theses of representation in the semantic web”. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003). 2003

[IEEECMI] IEEE 1484.11.1, Standard for Learning Technology. Data Model for Content Object Communication 2005. 2005. http://ltsc.ieee.org/

[IEEEECMA] IEEE 1484.11.2 Standard for Learning Technology – ECMAScript Application Programming Interface for Content to Runtime Services Communication. Noviembre 2003. http://ltsc.ieee.org/

[IEEE1471] IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems .2000.

[IMS03a] IMS Abstract Framework: White Paper. Version 1.0. Julio 2003.

[IMS03b] IMS Abstract Framework: Applications, Services and Components. Version 1.0. Julio 2003.

[IMS03c] IMS Abstract Framework: Glosary. Version 1.0. Julio 2003.

[IMSCP04] IMS Content Packaging Information Model, Version 1.1.4. Final Spec. Octubre 2004. http://www.imsglobal.org/content/packaging/index.html.

[IMSRDCEO] IMS Reusable Definition of Competency or Educational Objective - Information Model Version 1.0 Final Specification. Octubre 2002. www.imsglobal.org.

[IVO06] C. Iborra. "Lógica y Teoría de Conjuntos". Apuntes de la Universidad de Valencia. Octubre 2006.

[JSR140] Java Specification Request 170. Content Repository APIfor Java™ Technology. version 1.0.1. Marzo 2006.

[JUL00] V. Julian, V. Botti. "Agentes Inteligentes: el siguiente paso en la Inteligencia Artificial". NOVATICA . Junio 2000.

[JXTA04] JXTATM Technology: Creating Connected Communities. Enero 2004. www.jxta.org/

[JXTA05]JXTA v2.3.x: Java™ Programmer's Guide. Abril 2005. www.jxta.org/.

[KAL04] A. Kalyanpur, E. Sirin et al. "Hypermedia Inspired Ontology Engineering Environment: SWOOP". International Semantic Web Conference. Japón. 2004.

[KAL05] A. Kalyanpur, E. Sirin et al. "Swoop: A 'Web' Ontology Editing Browser". Elsevier's Journal Of Web Semantics (JWS), Vol. 4 Issue 1. 2005.

[KEL39] T.L. Kelly. "The selection of upper and lower groups for the validation of test items". Journal of Educational Psychology. Pp: 30, 17-24. 1939.

[KER04] R. Keralapura, N. Taft, C.Chuah, G. Iannaccone. "Can ISPs Take the Heat from Overlay Networks?". ACM HotNets Workshop. EEUU. Noviembre 2004.

[KIF95] M. Kiefer, G. Lausen, James Wu. "Logical Foudations of Object-Oriented and Frame-Based Languages" Journal of ACM 1995, vol 42. Pp: 741-842. 1995

[KIM05] T.Kim, M. Kim, G. Park. "On Employing Ontology to e-Learning". Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05). 2005.

[KLEE98] J. Kleeman, "Now is the Time to Computerize Pen and Paper Tests. A White Paper". 1998. http://www.questionmark.com/

[KNU04]H. Knublauch, M.A. Musen. "Editing Description Logic Ontologies with the Protege OWL Plugin". Proceedings of the 2004 International

Workshop on Description Logics, Whistler, British Columbia. Canada. 2004.

[KRD04] N. Krdzavak. "Description Logics Reasoning in Web-Bases Education Environments". Adaptive Hypermedia and Collaborative Web-based Systems (AHCW'04). Alemania. Junio 2004.

[LAN02] S. Langridge. "Pingback 1.0 specification". 2002. http://www.hixie.ch/specs/pingback/pingback-1.0

[LAN96] J. Laney, "Going the Distance: Effective Instruction Using Distance Learning". Educational Technology. Pp: 36, 2, 51-54. 1996.

[LLO96] M.J.G. Lloyd, K. McCottery, "The Introduction of Computer Based Testing on an Engineering Technology Course," Assessment and Evaluation in Higher Education. Pp: 21, 1, 83-90. 1996.

[LSSI02] LEY 34/2002, de 11 de julio, de Servicios de la Sociedad de la Información y de comercio electrónico. Art 12.

[LTSC02] IEEE Standard for Learning Object Metadata. 2002.

[LUG02] George F. Luger. "Artificial Intelligence, Structures and Strategies for Complex Problem Solving". 4ª edición. Ed. Pearson Education Limited, 2002.

[MAC98] D. Mackenzie, "TRIADS System,". 1998. http://www.derby.ac.uk/assess

[MAK02] "Making Sense of Learning Specifications & Standards: A decision Maker's Guide to their Adoption" The Masie Center. Marzo 2002.

[MAN02] V. Manso, "Herramienta de Teleeducación para Evaluación basado en XML". Trabajo de Final de Carrera codirigido por R. Romero y C. Palau. 2002.

[MAN02B] V. Manso, J.M. Raga, R. Romero, C. E. Palau et al, "An XML Approach for Assessment in Education". IASTED International Conference on Applied Informatics. Austria. 2002.

[MAO05] Y. Mao et al. "Sub-Ontology Evolution for Service Composition with Application to Distributed e-Learning". Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05). China. Noviembre 2005.

[MAR02] Philippe Martin. "Knowledge Representation, Sharing and Retrieval on the Web". Distributed System Technology Centre. Australia. 2002.

[MAU96] M. Mauldin. "The Formative Evaluation of Computer-Based Multimedia Programs," Educational Technology. Pp: 36, 2, 36-40. 1996.

[MAY02] P. Maymounkov, D. Mazieres. "Kademlia: A Peer-to-peer Information System. Based on the XOR Metric". 2002.

[MCC05] P. McCarthy. "Search RDF data with SPARQL". Mayo 2005. http://www-128.ibm.com/developerworks/xml/library/j-sparql/.

[MIL05] A. Miles, D. Brickley. "SKOS Core Guide". W3C Working Draft 2. Noviembre 2005.

[MODX06] BBphp Team. "MODX: XML Modification Description Format". 2006.

[MON02] Monforte C., Vierlinger U. "The Role of the Facilitator in eLearning ". Procs. SEFI Annual conference. Italia. 2002.

[MOO89] Moore, M. G. “Three types of interaction”. The American Journal of Distance Education . Pp: 1-6. 1989.

[MOS97] M. Moser, et al. “SETHEO and e-SETHEO – the CADE-13 systems. Journal of Automated Reasoning”,18(2). Pp:237–246. 1997.

[NEI02] W. Nejdl, B. Wolf et al. “EDUTELLA: A P2P Networking Infrastructure Based on RDF”. World Wide Web Conference WWW2002. Hawai 2002.

[NEW96] T. Newby, D. Stepich, J. Lehman, J. Russell. "Instructional Technology for Teaching and Learning: Designing Instruction, Integrating Computers, and Using Media". Prentice-Hall, Inc., Englewood Cliffs, EEUU. 1996.

[NON95] I. Nonaka, H. Takeuchi. “The knowledge creating company. How japanese companies create the dynamics of innovation”. Oxford University Press. 1995.

[NOY00] N. F. Noy, R. W. Fergerson, M. A. Musen. “The knowledge model of Protege-2000: Combining interoperability and flexibility”. 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000). France. 2000.

[NOY01] N. F. Noy, D. L. McGuinness. “Ontology Development 101”. Standford University. 2001.

[ORE05] T. O’Reilly. “What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software”. Septiembre 2005.

[OWL04a] J. Heflin. “OWL Web Ontology Language Use Cases and Requirements”. W3C Recommendation. Febrero 2004.

[OWL04b] M. Dean, G. Schreiber. (eds). “OWL Web Ontology Language Reference”. W3C Recommendation. Febrero 2004. http://www.w3.org/TR/owl-ref/.

[PAL03] C.E. Palau, V. Manso et al. “An XML Approach for Assessment in Education”. International Journal of Computers & Applications. ISSN 1206-212X. Acta Press. Pp: 24-37. 2003.

[PAL04] C.E. Palau , R. Romero, V. Manso, J.C. Guerri, M. Esteve. “Semantic Web for Re-usability in Open and Distance Learning”. Web Based Education 2004. Alemania. 2004.

[PAL06] R. Palma, P. Haase. “Oyster - Sharing and Re-using Ontologies in a Peer-to-Peer Community”. Proceedings of the 15th International Conference on World Wide Web. Escocia. Mayo 2006.

[PAL99] C. Palau, M. Esteve, J. C. Guerri, “Educational Uses of the WWW: An Evaluation Tool”, Word Wide Web, vol. 2, n 4, Pp: 231-249. Diciembre 1999.

[PAN02] J. Z. Pan, I. Horrocks. “ Reasoning in the SHOQ(Dn) Description Logic”. 2002.

[PAN05] Z. Pan. “Benchmarking DL Reasoners Using Realistic Ontologies”. OWL Workshop on Experiences & Directions. Irlanda. 2005.

[PAO03] M. Paolucci, K. Sycara. “Autonomous Semantic Web Services”. IEEE Computer Society. Octubre 2003.

[PAU94] L.C. Paulson. “Isabelle: A Generic Theorem Prover”. 1994.

[PER05] M. Perry et al. “Peer-to-Peer Discovery of Semantic Associations”. 2nd. International Workshop on Peer to Peer Knowledge Management. EEUU. Julio 2005.

[QMK06] Question Mark Computing Ltd., “Question Mark Web,”. 2006. http://www.questionmark.com/.

[QTI06] IMS Question and Test Interoperability. Version 2.1 Public Draft Specification. 2006. http://www.imsglobal.org/question/.

[QTI06b] IMS Question and Test Interoperability. XML Binding. 2006. http://www.imsglobal.org/question/.

[QUIT] http://chemware.co.nz/quizit.htm.

[RACER05] RacerPro reference manual. Version 1.9. Racer Systems Gmbh & Co. Diciembre 2005.

[RACER05B] RacerPro User’s Guide. Version 1.8. Racer Systems Gmbh & Co. Abril 2005.

[RAG02] J.M. Raga. “Herramienta Cooperativa para Trabajo en Grupo Desarrollada en Java Basada en XML”. Trabajo de Final de Carrera codirigido por R. Romero y C. Palau. 2002.

[REB04] “El estándar SCORM para EaD (Educación a Distancia)”. Tesina del Máster en Enseñanza y Aprendizaje Abiertos y a Distancia. Universidad Nacional de Educación a Distancia. Diciembre 2004.

[REC05] A. Rector et al. “Ontology Design Patterns and Problems: Practical Ontology Engineering using Protégé-OWL”. 4th International Semantic Web Conference (ISWC). Irlanda. Noviembre 2005.

[REY06] M. Rey, A. Fernández, R. Diaz, J. J. Pazos. “Extending SCORM to Create Adaptive Courses”. First European Conference on Technology Enhanced Learning (EC-TEL 2006). Grecia. Octubre 2006.

[RFC1855] Netiquette Guidelines. http://www.ietf.org/rfc/rfc1855.txt

[RFC2440] OpenPGP Message Format. http://www.ietf.org/rfc/rfc2440.txt

[RFC2718] Guidelines for new URL Schemes. http://www.ietf.org/rfc/rfc2718.txt

[RFC3023] XML Media Types. http://www.ietf.org/rfc/rfc3023.txt

[RFC3275] (Extensible Markup Language) XML-Signature Syntax and Processing. http://www.ietf.org/rfc/rfc3275.txt

[RFC3986] Uniform Resource Identifier (URI): Generic Syntax. http://www.ietf.org/rfc/rfc3986.txt.

[RIA02] A. Riazanov, A. Voronkov. “The Design and Implementation of Vampire”. AI Communications,15. 2002.

[ROM02] R. Romero. Aplicaciones Cooperativas para Teleeducación basadas en el modo de comunicación Multicast. Trabajo de Investigación del programa de Doctorado de Ingeniería de Telecomunicación. 2002.

[ROM06] R. Romero, V. Manso, C .Palau. “Preparing Proprietary Systems for Continuous Education e-Learning for inter-operatibility: exporting to SCORM”. Proceedings 6th International Conference on Advanced Learning Technologies. ICALT 2006. Holanda. 2006.

[ROM99] R. Romero. "Implementación y desarrollo de un sistema de simulación en entorno Web". Trabajo Final de Carrera de Ingeniería de Telecomunicación. 1999.

[ROU03] M. Roussopoulos et al. 2 "P2P or not 2 P2P?". 2003.

[ROU05] D. Roure, N.R. Jennings, N.R. Shadbolt "The Semantic Grid: Past, Present, and Future". Proceedings of the IEEE, Vol 93, número 3. Marzo 2005.

[RUS96] S. Russell. "Inteligencia Artificial: un enfoque moderno". Prentice - Hall. México. 1996.

[RUT05] A. Ruttenberg, J. A. Rees, J. S. Luciano. "Experience Using OWL DL for the Exchange of Biological Pathway Information". OWL Workshop on Experiences & Directions. Irlanda. Noviembre 2005.

[SAN05] J.M. Santos, L. Anido, M. Llamas. "Design of a Semantic Web-based Brokerage Architecture for the E-learning Domain. A Proposal for a Suitable Ontology". 35th ASEE/IEEE Frontiers in Education Conference. EEUU. Octubre 2005.

[SAMP02] D. Sampson, C. Karagiannidis, F. Cardinale. "An Architecture for Web-based e-Learning Promoting Re-usable Adaptive Educational e-Content". 2002.

[SCHO89] U. Schoening. "Logic for Computer Scientists". Birkhauser. EEUU. 1989.

[SER02] S. Krivov, "Semantic Web and its Logical Foundations". Ecoinformatics Collaboratory. Gund Institute. 2002.

[SEC04] G. Secundo, A. Corallo, G. Elia, G. Passiante. "An e-Learning System Based on Semantic Web Supporting a Learning in Doing Evironment". International Conference on Information Technology Based Higher Education and Training. Turquía. Junio 2004.

[SIE05] R.M. Siebes, "pNear: combining Content Clustering and Distributed Hash Tables".2nd. International Workshop on Peer to Peer Knowledge Management. EEUU. Julio 2005.

[SIE06] R.M. Siebes, "Semantic Routing in Peer-to-Peer Systems". Tesis doctoral. Holanda. 2006.

[SIM06] S. Sim, R.I. Saiah. "Relationship Between Item Difficulty and Discrimination Indices in True/False-Type Multiple Choice Questions of a Para-clinical Multidisciplinary Paper". Annals Academy of Medicine. Febrero 2006.

[SOE96] A. Soeiro. "The teaching Challenge of Open and Distance Learning". EUCEN Journal Archive, 1996.

[STE04] K. Stege, T. Kelder, G. Huisman. "Tutorial for Creating a BioPAX Pathway with Jena". Technical University of Eindhoven. 2004.

[SWAP04] SWAP Final Report. Proyecto SWAP EU IST-2001-34103. Noviembre 2004.

[SWRL04] I. Horrocks et al. "SWRL:A Semantic Web Rule Language Combining OWL and RuleML". W3C Member Submission. Mayo 2004.

[TAL03] D. Talia, P. Trunfio. “Towards a Synergy Between P2P and Grids”. IEEE Internet Computing, 2003 2003. 7(4). Pp: 94-96. 2003.

[TAL04] D. Talia, P. Trunfio. “A P2P Grid Services-Based Protocol: Design and Evaluation”. EuroPar. 2004.

[TBL1998] Tim Berners-Lee. “Semantic Web Road Map”. W3C Design Issues. Octubre 1998. URLhttp://www.w3. org/DesignIssues/Semantic.html.

[TBL2001] Berners-Lee, Hendler, Lassila. “The Semantic Web”. Scientific American. Mayo 2001.

[TBL2001] T. Berners-Lee, J. Hendler, O. Lassila. “The Semantic Web”. Scientific American. 2001.

[TELE01] Teleologic Learning Company. “Objects, Vectors and Learning: implications for SCORM”. 2001.

[TIN97] L. Tinoco, E. Fox and N. Barnett, “Online Evaluation in WWW-based Courseware, The QUIZIT System”, Proceedings of the 28th Annual Symposium for Computer Based Education. Pp. 194-198. 1997.

[TOU05] R. Tous, J. Delgado. “Using OWL for Querying an XML/RDF Syntax”. WWW2005. Japón. Mayo 2005.

[TRA03] B. Traversat, M. Abdelaziz, E. Pouyoul. “Project JXTA: A Loosely-Consistent DHT Rendezvous Walker”. 2003.

[TRA03B] B. Traversat, M. Abdelaziz, E. Pouyoul. “Project JXTA 2.0 Super-Peer Virtual Network”. 2003.

[TRO04] M. Trott, B. Trott. “TrackBack 1.2 Technical Especificacion”. 2004. http://www.sixapart.com/pronet/docs/trackback_spec.

[TSA03] D. Tsarkov, I. Horrocks. “DL Reasoner vs. First-Order Prover”. University of Manchester. 2003.

[TSA04] D. Tsarkov, I. Horrocks. “Efficient reasoning with range and domain constraints”. 2004 Description Logic Workshop (DL 2004), vol 104. Pp: 41-50. 2004.

[TUR05] Z. Turk et al. “Semantic Grid - Interoperability Solution for Construction VO?”. Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC’05). EEUU. Abril 2005.

[ULL04] C. Ullrich. “Description of an instructional ontology and its application in web services for education” SWEL Workshop 2004. Pp: 17-23. Japón. 2004.

[VER04] D.C. Verma. “Legitimate Applications of Peer to Peer Networks”. John Wiley & Sons, Inc. 2004.

[WAG03] Gerd Wagner, Said Tabet, Harold Boley “MOF-RuleML: The Abstract Sytax of RuleML as a MOF Model”. OMG Meeting. EEUU. Octubre 2003.

[WIL03] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds. “Efficient RDF Storage and Retrieval in Jena2” HP Techical Report. Diciembre 2003.

[WIL03] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds. “Efficient Storage and Retrieval of RDF Graphs in Jena2”. First International Workshop on Semantic Web and Databases. Alemania. Septiembre 2003.

[XIE03] M. Xie, "P2P Systems Based on Distributed Hash Table". Septiembre 2003.

[YAN05] J. Yang, Q. Li, Y. Zhuang "Multi-modal Information Retrieval with a Semantic View Mechanism". Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05). China. Marzo 2005.

[ZHDA04] A.V. Zhdanova. "Jena Tutorial". 2004.

[ZHO06] J. Zhou, J. Koivisto, E. Niemela. "A Survey on Semantic Web Services and a Case Study". Proceedings of the 10th International Conference on Computer Supported Cooperative Work in Design. China. Mayo 2006.

[ZOU03] L. Zou. "State and File Sharing in Peer to Peer Systems". Tesis doctoral. Georgia Institue of Technology. Diciembre 2003.

Recursos en Internet

[WWW1] Página principal de la web semántica en W3C. http://www.w3.org/2001/sw/

[WWW2] Página principal de RDF en W3c. http://www.w3c.org/rdf

[WWW3] http://www.ontoweb.org

[WWW4] http://www.cogsci.princetown.edu/~wn/

[WWW5] http://www-ksl-svc.standford.edu:5915

[WWW6] http://www.cyc.com

[WWW7] http://www.topicmaps.org

[WWW8] http://www.uml.org

[WWW9] http://citeseer.ist.psu.edu/

[WWW10] http://www.topicmaps.org/xtm

[WWW11] http://www.isotopicmaps.org

[WWW12] http://www.ruleml.org

[WWW13] http://www.sts.tu-harburg.de/~r.f.moeller/racer

[WWW14] http://www.daml.org/services/

[WWW15] IMS web page. http://www.imsglobal.org

[WWW16] http://web.mit.edu/oki

[WWW17] http://www.adlnet.org/

[WWW18] http://www.openuss.org

[WWW19] http://www.ist-universal.org

[WWW20] http://www.govtalk.gov.uk/schemasstandards/egif.asp

[WWW21] http://www.ariadne-eu.org

[WWW22] http://dublincore.org

[WWW23] http://ltsc.ieee.org/wg12/

[WWW30] Comparativa de distintas de finiciones de P2P. http://www.anu.edu.au/people/Roger.Clarke/EC/P2POview.html#Defns

[WWW31] http://www.cachelogic.com

[WWW32] http://www.kazaa.com

[WWW33] http://www.coralcdn.org/

[WWW34] http://fedsearch.merlot.org

[WWW35] http://www.exlibrisgroup.com/
[WWW36] http://www.edna.edu.au/
[WWW37] http://www.merlot.org
[WWW38] http://edutella.jxta.org/
[WWW39] http://www.edusplash.net/
[WWW40] http://cordra.net
[WWW41] http://www.ubbcentral.com/
[WWW42] http://www.phpbb.com/
[WWW43] http://www.blabla4u.com/
[WWW44] http://www.fusetalk.com/
[WWW45] Comparativa de software para foros: http://en.wikipedia.org/wiki/Comparison_of_Internet_forum_software
[WWW46] http://java.sun.com
[WWW47] http://www.programacion.com/php/foros/
[WWW48] http://2ch.net/
[WWW49] Comparativa entre Pingback y Trackback. 2002, por el autor de Pingback. http://ln.hixie.ch/?start=1033171507&count=1
[WWW50] SAKAI : Collaboration and Learning Environment for Education. http://sakaiproject.org/.
[WWW51] Descripción de Wiki. http://es.wikipedia.org/wiki/Wiki
[WWW52] Altova Semantic Works page. http://www.altova.com/products/semanticworks/semantic_web_rdf_owl_editor.html
[WWW53] SWOOP page. http://www.mindswap.org/2004/SWOOP/
[WWW54] OWL consistency chequer, basado en Pellet. http://www.mindswap.org/2003/pellet/demo.shtml.
[WWW55] Jena 2 Ontology API. http://jena.sourceforge.net/ontology/
[WWW56] B. McBride. Jena Tutorial. http://jena.sourceforge.net/tutorial/index.html
[WWW57] Página principal de Jena. http://jena.sourceforge.net/
[WWW58] Información del proyecto wonderweb IST-2001-33052. http://wonderweb.semanticweb.org/
[WWW59] Página principal de Pellet. http://www.mindswap.org/2003/pellet/index.shtml
[WWW60] Informe de rendimientos de Pellet. http://www.mindswap.org/2003/pellet/performance.shtml
[WWW61] RDF Query Survey. http://www.w3.org/2001/11/13-RDF-Query-Rules/
[WWW62] SPARQL Query Language for RDF. W3C Recommendation Candidate. http://www.w3.org/TR/rdf-sparql-query/
[WWW63] Proyecto OWL de la universidad de Standford (Standford Knowledge Systems Laboratory). http://ksl.stanford.edu/projects/owl-ql/
[WWW64] Universal Decimal Classification. http://www.udcc.org/
[WWW65] Library of Congress Cataloging. http://www.loc.gov/catdir/
[WWW66] Página principal de Google. http://www.google.com

[WWW67] Página principal de Wikipedia. http:// www.wikipedia.org
[WWW68] W3C XML Signature Recommendation. http://www.w3.org/TR/xmldsig-core/
[WWW69] Observatorio sobre estándares de tecnologías de la educación. http://www.cen-ltso.net/
[WWW70] Ejemplo de crítica a Web 2.0. http://www.point-studios.com/archives/2005/10/web_20_buzzword.htm
[WWW71] Web de FaCT++. http://owl.man.ac.uk/factplusplus/
[WWW72] Página principal de blackboard. http://www.blackboard.com.
[WWW73] Foro de discusión de Prótégé. protege-discussion@SMI.Stanford.EDU
[WWW74] Foro de discusión de la web semántica. semanticweb@yahoogroups.com
[WWW75] Página principal de WebCT. http://www .webct.com
[WWW76] Página de W3C sobre XML. http://www.w3.org/XML/
[WWW77] Microsoft's Repository Object Architecture. http://msdn2.microsoft.com/en-us/library/aa179073(SQL.80).aspx
[WWW78] Página principal del proyecto de desarrollo Protégé. http://protege.stanford.edu.