Proyecto fin de carrera

Título:

XML y web semántica: Bases de datos en el contexto de la web semántica.

MEMORIA

Autor: Joaquín Rincón Cinca. Abril – 2012

Índice

Tabla de contenido

Acrónimos

Acrónimo	Significado
DW	Data Warehouse (almacén de datos)
ESB	Enterprise Service Bus
ETL	Extract, Transform, Load (extraer, transformar, cargar)
FOAF	Friend of a Friend (amigo de un amigo)
Modelo E/R	Modelo Entidad-Relación
ORDI	Ontology Representation and Data Integration (representación de ontologías e integración de datos)
OWL	Web Ontology Language (lenguaje de ontologías web)
RDF	Resource Description Framework (marco de descripción de recursos)
SAIL	Storage And Inference Layer (Capa de almacenamiento e inferencia)
SGBD	Sistema de Gestión de bases de datos
SPARQL	Hay dos posibles significados: Simple Protocol and RDF Query Language (protocolo simple y lenguaje de consulta RDF) SPARQL Protocol and RDF Query Language (protocolo SPARQL y lenguaje de consulta RDF)
TRREE	Triple Reasoning and Rule Entailment Engine (Razonamiento de tripletes y motor de vinculación de reglas)
URL	Uniform Resource Locator (localizador uniforme de recursos)
URI	Uniform Resource Identifier (identificador uniforme de recursos)
W3C	World Wide Web Consortium
XML	eXtensible Mark-up Language (Lenguaje extensible de marcas)
XSD	XML Schema Definition (Definición de esquemas XML)

Resumen Ejecutivo

En este proyecto fin de carrera analizaremos el estado actual de los gestores de bases de datos semánticas. Estos gestores ofrecen funcionalidades especiales comparados con otros sistemas gestores de bases de datos (SGBD) tradicionales; fundamentalmente la capacidad de añadir contenido semántico a las relaciones entre los datos que se guardan en el gestor.

La idea de añadir contenido semántico a los modelos de datos es bastante antigua, pero no ha habido productos comerciales que ofreciesen ese tipo de funcionalidad hasta hace poco. El avance de la tecnología, que ahora permite la creación de SGBD semánticos, unido al auge de la "web semántica" están dando protagonismo a esta forma de gestionar los almacenes de información.

Este trabajo definirá que es una base de datos semántica, que ventajas ofrecen, como se utilizan y en que tipo de proyectos o sistemas tiene sentido usarlas. Además estudiaremos en detalle una de ellas, OWLIM[1], de la empresa Ontotext para evaluar la dificultad de usarla, su rendimiento y sus capacidades específicas.

[1] http://www.ontotext.com/owlim

1. Descripción del trabajo

El objetivo de este trabajo es estudiar los SGBD utilizados en el contexto de la web semántica en general y analizar uno en particular.

Haré un análisis genérico de las bases de dates semánticas comparadas con los SGBD relacionales. A continuación veremos las principales características de los distintos SGBD semánticos y veremos en que se diferencian los unos de los otros y que clase de funcionalidad semántica aportan.

Por último, hare un estudio más en detalle del producto OWLIM.

a. Objetivos

Proporcionar una visión de las bases de datos semánticas:

- Como se comparan con las relacionales
- En que caso deben usarse
- Como se usan. Modelos
- Que características y nuevas funcionalidades proporcionan
- Limitaciones
- Ventajas

Hacer un estudio detallado de OWLIM:

- Funcionalidad
- Versiones
- Como se instala y se usa
- Consultas
- Escenarios de uso
- Rendimiento

2. Bases de datos semánticas

a. Definiciones para entender el contexto

Modelo de datos

Un modelo de datos es un modelo abstracto que recoge información sobre los datos que va a utilizar un sistema.

El modelo de datos sirve para la transmisión de información entre los miembros de un proyecto y para poner en común una representación del "universo del discurso" que queremos modelar en nuestro sistema. [i]

Existen muchas formas de hacer esta representación:

- La representación puede ser a distinto nivel conceptual: modelo conceptual, lógico o físico
- Puede usar una representación gráfica de conceptos distintos: UML, modelo Entidad-Interrelación, modelo relacional, etc.

En general, en el proceso de creación de un sistema, se empieza creando modelos conceptuales con una cierta capacidad de guardar información semántica y luego se van creando modelos de más bajo nivel, basados en los anteriores, pero más cercanos al modelo final que se creara en el SGBD. Normalmente en este proceso se pierde información semántica y se gana detalle de implementación.

Modelos conceptuales o semánticos

Estos son un tipo específico de modelo de datos que son capaces de guardar información semántica. El caso más normal es que tengamos una serie de entidades, que tenemos que modelar y que forman parte del universo del discurso de nuestro sistema, y que podamos establecer relaciones entre esas entidades[ii].

Las relaciones tendrán una serie de atributos comunes a todas ellas (como las cardinalidades) y además se pueden definir atributos específicos de cada relación. Igualmente las entidades tendrán una serie de atributos que serán comunes a todos los elementos que pertenezcan a la misma entidad.

El modelo Entidad-Relación (también llamado Entidad-Interrelación, pero normalmente representado como modelo E/R) fue creado por Peter Chen en 1976 y es uno de los más utilizados[iii].

Aquí abajo tenemos un ejemplo de un modelo entidad relación sacado de la Wikipedia en castellano[iv]:

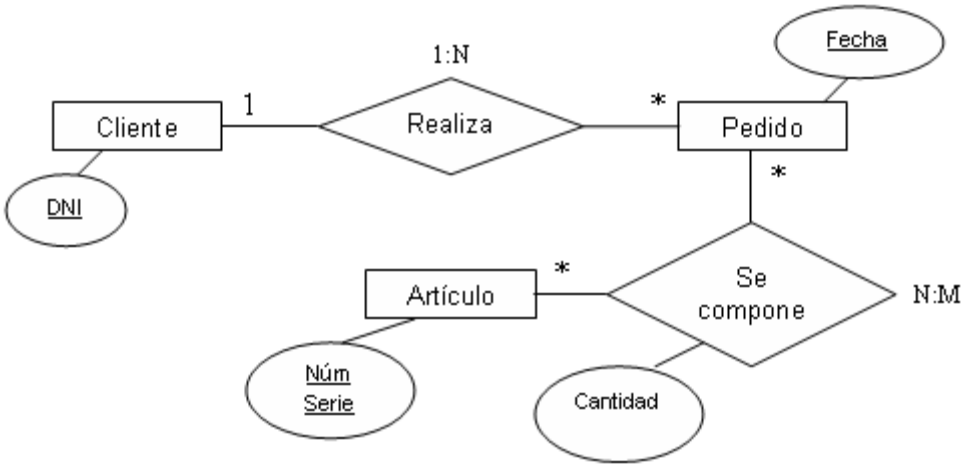


Figura 1 - Modelo E/R

Se pueden ver tres entidades (Cliente, Pedido y Artículo) y dos relaciones (Realiza y Se compone), alguna de las relaciones tienes atributos propios además de la información de cardinalidad.

Los modelos semánticos, al tener capacidades de modelado avanzadas y parecerse al mundo real son mejores a la hora de tratar con las personas que encargan el desarrollo del sistema. Normalmente estas personas no tienen formación informática técnica y otros modelos serían más difíciles de entender para ellos.

Modelos lógicos

El modelo lógico se suele considerar un nivel intermedio entre un modelo conceptual y un modelo físico real que tiene detalles específicos de implementación.

Este modelo dependerá de la tecnología específica del SGBD, pero suele representarse como tablas y columnas o etiquetas XML o como una jerarquía de clases[v].

El modelo relacional y sus limitaciones

El modelo relacional es un modelo lógico que se representa mediante **tablas** (a veces llamadas relaciones, lo cual puede llevar a confusiones). Cada una de estas tablas tiene una serie de **filas** que comparten los mismos **atributos**, representados en columnas. Para que el modelo sea correcto debe haber un atributo o conjunto de atributos que identifican una fila de la tabla de forma unívoca; por ejemplo el DNI en una tabla de personas, ya que no puede haber dos personas con el mismo DNI. A ese atributo o conjunto de atributos se le llama **Clave Primaria**.

Edgar F. Codd[2] creó el modelo relacional en 1969 y ya desde el primer artículo que lo describía hizo mucho hincapié en separar el modelo lógico de tablas de la posible implementación a nivel físico[vi].

[2] http://en.wikipedia.org/wiki/Edgar_F._Codd

Hoy día hay muchos SGBD que se llaman relacionales por el hecho de implementar el modelo relacional, con lo cual muchas veces al hablar de modelo relacional se confunden los niveles físico y lógico, pero el modelo relacional es principalmente un modelo lógico y, de hecho, existió como modelo teórico algo más de 10 años hasta que en 1980 empezaron a aparecer los primeros SGBD comerciales que daban soporte a este modelo, como Oracle.

El modelo relacional tiene una semántica bastante limitada, existen las relaciones entre tablas, pero esa relación sólo es de un tipo llamado "*clave externa*". Una tabla B tiene una relación de clave externa hacia otra tabla A cuando un atributo o conjunto de atributos de B representa la clave primaria de A.

Por ejemplo, en un modelo relacional podemos tener una tabla de *clientes*, que tiene el `código_de_cliente` como clave primaria y una tabla de *pedidos* que tendrá el `código_de_cliente` como uno de sus atributos para identificar de qué cliente es el pedido. El atributo `código_de_cliente` dentro de la tabla de pedidos es una clave externa que relaciona la tabla de *pedidos* con la de *clientes*.

Si además de esta tabla de pedidos tenemos una tabla de facturas, será normal que esta tabla también tenga una clave externa hacia los clientes. Estas dos relaciones son exactamente iguales, no hay nada que las diferencie y el modelo relacional no es capaz de aportar una semántica que lo permita.

A cualquier otra tabla que tenga una relación de clave externa con la tabla de clientes le pasara lo mismo, el modelo solo nos dice que existe una relación, pero no sabemos por que o que semántica implementa esa relación.

El modelo relacional tienes otras muchas limitaciones si lo comparamos con el modelo E/R, por ejemplo no admite las relaciones con cardinalidad n:m. Las relaciones de clave externa sólo tienen la cardinalidad 1:n. Para modelar una relación n:m en el modelo relaciones necesitamos una tabla intermedia.

Por ejemplo: supongamos que tenemos un modelo entidad relación con una entidad de alumnos y otra de asignaturas. Tenemos además una relación entre ambas entidades con cardinalidad n:m, lo que indica que un alumno puede estar matriculado de varias asignaturas y en una asignatura estarán matriculados varios alumnos.

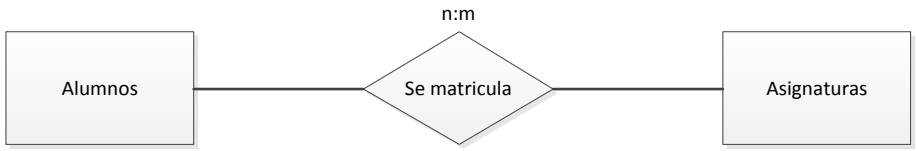


Figura 2 - Modelo E/R con una relación n:m

Si queremos pasar este sencillo modelo a un modelo relacional vemos que no se puede hacer directamente, ya que las relaciones n:m no existen. Tenemos que modelarlo mediante el uso de tres tablas, una de asignaturas, una de alumnos y otra de relación entre ellos.

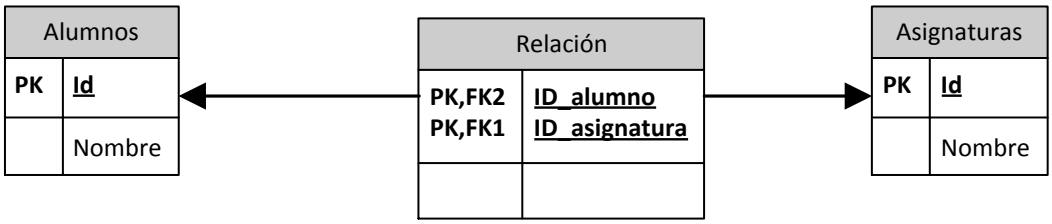


Figura 3 - Modelo relacional que modela una relación n:m

Como vemos, lo que en el modelo E/R eran dos entidades y una relación se convierte en el modelo relacional en tres tablas y dos relaciones.

Como ya he comentado el modelo relacional tiene muchas limitaciones, he mostrado esta sólo a modo de ejemplo, pero sin duda la mayor limitación es la falta de información semántica al ser todas la relaciones del mismo tipo.

La semántica en el caso de un modelo relacional no está en el modelo de por si, sino en la capa de aplicación que existirá por encima. Esto hace que la integración de los datos sea difícil como veremos más en detalle más adelante en el trabajo.

b. Las bases de datos semánticas

Los SGBD semánticos son sistemas que son capaces de gestionar los datos y guardar la semántica de sus relaciones.

La idea principal es ser capaz de modelar el mundo real, con sus relaciones y que esa información forme parte de la información que guarda el SGBD.

Con las bases de datos semánticas está pasando algo parecido a lo que pasó con el modelo relacional. Este modelo apareció en 1969, pero hasta 1980 no hubo un producto comercial que fuese capaz de implementarlo. Los modelos semánticos se conocen desde la década de los 70 (siendo el 76 con la publicación del modelo E/R por parte de Chen un momento sumamente importante) pero hasta hace poco no ha habido productos comerciales que pudieran implementar este tipo de modelo sin perder información o con un rendimiento admisible.

Estamos ahora en ese punto en el que los primeros productos comerciales están disponibles y su rendimiento empieza a ser razonable como para usar estas herramientas en desarrollos de sistemas importantes. Veremos si este modelo alcanza el éxito que tuvo el modelo relacional o si se queda para un nicho pequeño de aplicaciones.

Aunque finalmente el uso de este modelo no se generalice, lo que está claro es que para algunos problemas es muy superior a los modelos existentes. Veremos esos casos en detalle a lo largo de este trabajo.

Veamos un ejemplo de algunas de las limitaciones que tiene el modelo relacional y la idea germinal que da lugar a las bases de datos semánticas: Supongamos que tenemos una aplicación para gestionar requisitos de un proyecto o sistema. Esta aplicación permite gestionar distintos tipos de requisitos (de usuario, de la solución, de sistema,…) todos los requisitos tienen una serie de atributos comunes, como un

código, un tipo, un nombre, una descripción detallada, etc., pero además según el tipo de requisito pueden tener atributos específicos. Lo que tenemos aquí es una relación de herencia, tenemos una clase que llamaremos Requisitos y varias clases hijas que extienden los atributos de esa clase con atributos específicos.

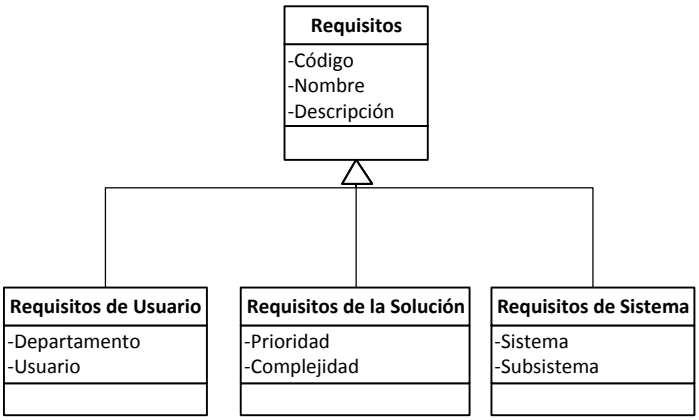


Figura 4 – Jerarquía de clases que representa tipos de requisitos

Modelar algo como esto en una base de datos relacional es complicado y cualquier solución tiene problemas. Si optamos por crear cuatro tablas, donde los tipos de requisitos tienen una clave externa hacia el tipo padre, necesitamos tres consultas distintas según queramos sacar datos de los distintos tipos. Si quisiéramos que al movernos por la tabla de Requisitos pudiésemos mostrar los datos completos de un requisito, o bien creamos tres claves externas desde esta tabla hacia sus "hijas" (donde solo uno de esos tres campos tendrá un valor) o bien hacemos tres búsquedas hasta encontrar de que tipo es el requisito.

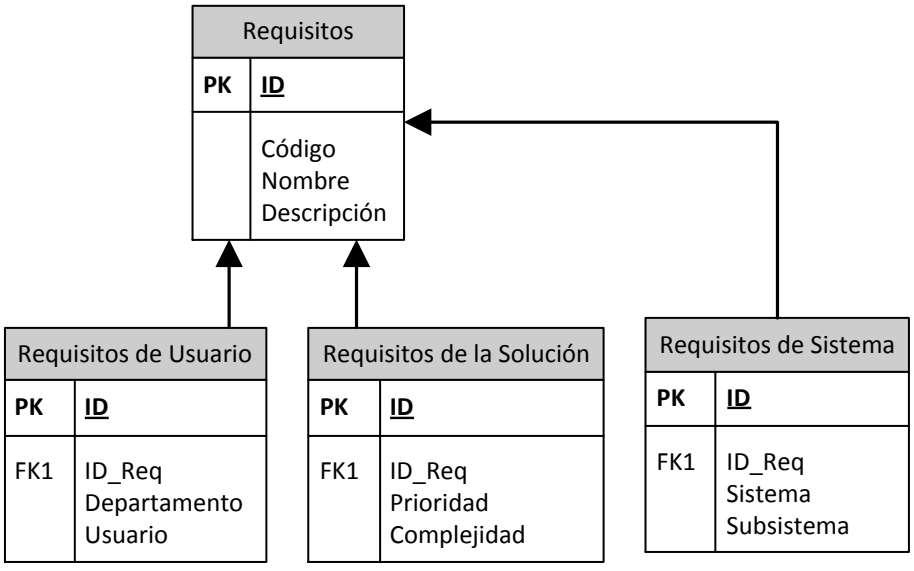


Figura 5 – Un posible modelo relacional que implementa la jerarquía anterior.

Otra posible solución es crear tres tablas independientes repitiendo los campos comunes en cada tabla. Ese modelo también tiene el problema de que no sabemos en que tabla está un determinado requisito, para mostrar un listado de requisitos tenemos que leer información de tres tablas, etc.

La evolución de este modelo también es compleja. Imaginemos ahora que queremos poder dividir los requisitos en Funcionales y No Funcionales. Son dos nuevos tipos de hijos del tipo requisito, pero ahora la cosa se vuelve muy compleja, por que un Requisito de Usuario (por ejemplo) puede ser Funcional o No Funcional, tenemos herencia múltiple.

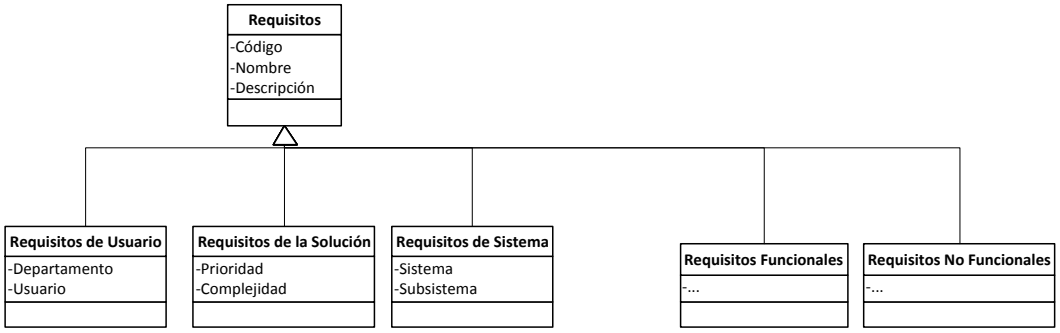


Figura 6 – Evolución de nuestro modelo de clases

Crear ahora un esquema relacional que capture esta semántica se vuelve muy complejo. Un mismo requisito puede tener valores de atributos en tres tablas. Para mostrar los datos completos de un requisito necesitamos buscar en seis tablas o bien crear un número elevado de claves externas de una tablas hacia otras (muchas de esas claves luego estarán vacías).

Pero existe una solución de otro tipo. Si volvemos al caso original en el que tenemos Requisitos y sus tres posibles tipos hijos, podríamos crear un modelo relacional de este tipo:

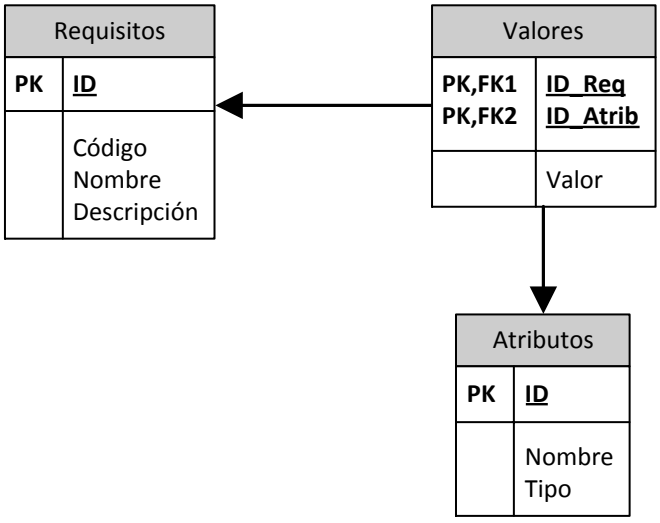


Figura 7 – Nuevo modelo relacional

Dentro de la tabla de atributos tendríamos los siguientes registros:

ID	Nombre	Tipo
1	Departamento	Texto
2	Usuario	Texto
3	Prioridad	Alta, Media, Baja

4	Complejidad	Alta, Media, Baja
5	Sistema	Texto
6	Subsistema	Texto
7	Tipo de Requisito	Usuario, Funcional, Sistema

Para dar valor a los atributos de un requisito creamos registros en la tabla valores que hacen referencia al atributo al que dan valor y guardan ese valor.

Ahora buscar todos los valores de todos los atributos que tenga un requisito es una consulta sencilla que une estas tres tablas.

Si ahora queremos añadir la casuística de los Requisitos Funcionales y los No Funcionales, sólo tenemos que añadir nuevos registros a la tabla de atributos, no hace falta cambiar el modelo.

Modelar de esta forma no se considera una buena práctica en el mundo relacional por que la información no está normalizada (algo muy importante para que el rendimiento de los modelos relacionales sea razonable), pero como hemos visto, puede solucionar algunos problemas complejos.

Ahora podemos ir un paso más allá. Los atributos que tiene la tabla Requisitos también podrían modelarse de esta forma y tendríamos un modelo como este:

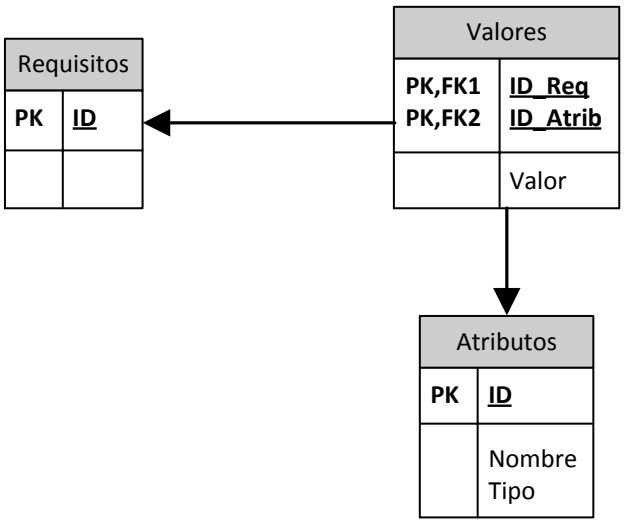


Figura 8 – Modelo relacional con los requisitos completamente parametrizados

ID	Nombre	Tipo
1	Departamento	Texto
2	Usuario	Texto
3	Prioridad	Alta, Media, Baja
4	Complejidad	Alta, Media, Baja
5	Sistema	Texto
6	Subsistema	Texto
7	Tipo de Requisito	Usuario, Funcional, Sistema
8	Código	Texto
9	Nombre	Texto
10	Descripción	Texto Largo

Bien, pues, simplificando bastante, esta es la idea central de las bases de datos relacionales. Bases de datos totalmente desnormalizadas en las que se describen "atributos" de lo elementos que pertenecen a sus entidades de una forma totalmente uniforme. Además, aunque el mecanismo parezca muy simple (realmente lo es), hemos ganado en capacidad de expresar contenido semántico.

En el modelo inicial, las relaciones "padre-hijo" estaban implementadas mediante la única forma de relación en el modelo relacional: la clave externa. Tendrá que ser nuestra explicación, nuestro conocimiento o la lógica de negocio de la aplicación que use estos datos lo que de el contenido semántico de la relación de herencia.

Ahora el Tipo de un requisito es un atributo explicito. El modelo es tan simple que es muy fácil de entender y extenderlo es tan sencillo como añadir datos a las tablas, sin tener que cambiarlas.

Iremos viendo como añadir capacidades a este modelo hasta llegar a bases de datos realmente funcionales y con capacidad semántica completa.

Vamos ahora a definir algunos conceptos que se han vuelvo comunes en el área de las bases de datos semánticas y la web semántica.

Web Semántica[vii]

La web semántica, usualmente también llamada Web 3.0, es una evolución de la World Wide Web actual en la que se añade contenido semántico a los documentos que la forman y de esa forma puede ser mejor, más potente que la actual y su consulta y navegación puede ser automatizada y realizada por máquinas, que descubren información por nosotros[viii].

A día de hoy es muy difícil organizar la información de la web, Google lo consigue a base de fuerza bruta y, aunque sus resultados son impresionantes, muchas veces no es suficiente. El ejemplo típico es que si yo busco la palabra "gato" en Google veré muchas páginas y fotos de felinos, pero quizás yo estaba buscando información sobre la herramienta que se usa para elevar coches. En una web semántica ese tipo de confusiones no existirán por que más allá del hecho básico de que la palabra "gato" aparezca en ambos tipos de documentos, la información semántica añadida permitirá catalogarlos de forma totalmente distinta, unos en relación con la zoología y las mascotas y otros en relación con la mecánica.

El mecanismo fundamental por el que se añade información semántica a la web actual es mediante el uso de RDF.

RDF

RDF (Resource Description Framework) nació como un lenguaje para representar información de recursos web.

La estructura de una colección RDF será un conjunto de tripletes que tendrán los siguientes componentes[ix]:

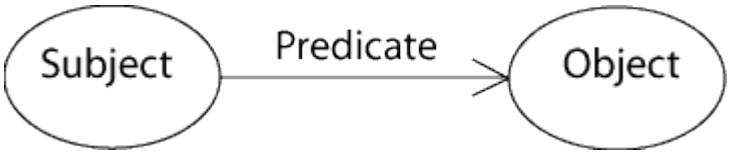


Figura 9 – Triplete RDF

La existencia de un determinado triplete quiere decir que existe una relación entre el objeto y el sujeto indicada por el predicado. La dirección de la relación es importante.

Veamos un ejemplo: Si queremos expresar que una página web habla de mascotas, tendríamos una construcción del estilo: (URL_página, "Trata el tema", "Mascotas).

RDF se ha extendido a otros entornos, más allá de proporcionar información añadida a la web, por que se ha mostrado como una forma eficiente y sencilla de representar conocimiento.

Si volvemos al ejemplo de los requisitos y al último modelo propuesto, podemos ver el paralelismo con RDF. La información de un determinado requisito se puede almacenar como una serie de tripletes:

```
(ID_Req, "Código", "X-0001");
(ID_Req, "Nombre", "Requisito 1");
(ID_Req, "Descripción", "Requisito número 1 del ejemplo de la memoria
del proyecto sobre web semántica");
…
```

Los distintos atributos de un determinado requisito funcionan como Predicados en los tripletes y los valores de esos atributos son los Objetos que se asignan. La base de datos relacional es tan simple que puede ser sustituida sin perdida de información por un conjunto de tripletes como estos. Una vez más, esta es la idea principal de las bases de datos semánticas, poder guardar la información de esta manera, pero teniendo unos tiempos razonables de recuperación cuando realizamos consultas.

Los tripletes pueden conectar dos entidades. Por ejemplo, si tenemos en nuestro modelo de los requisitos que uno de los atributos de un requisito es quien lo ha escrito, podemos tener unos tripletes con la siguiente información:

```
(ID_Req, "Escrito por", User_1);
(User_1, "Nombre", "Joaquín");
(User_1, "ID Usuario", "jrincon");
```

Podríamos haber puesto un triplete como este: `(ID_Req, "Escrito por", "Joaquín")`; si no tenemos más información del usuario, pero si los usuarios existen como entidades que tienen varios atributos a su vez, lo que podemos hace es conectar dos identificadores mediante una relación.

Es importante darse cuenta que el predicado "Nombre" es el mismo para dar nombre a un usuario que el que hemos usado para dar nombre a un requisito, esto tiene sentido siempre que el uso sea consistente.

Podemos representar un conjunto de tripletes RDF como un grafo que conecta entidades con sus propiedades y entidades entre si. En el caso de los dos ejemplos anteriores, si juntamos los seis tripletes en un único grafo, tendríamos algo así:

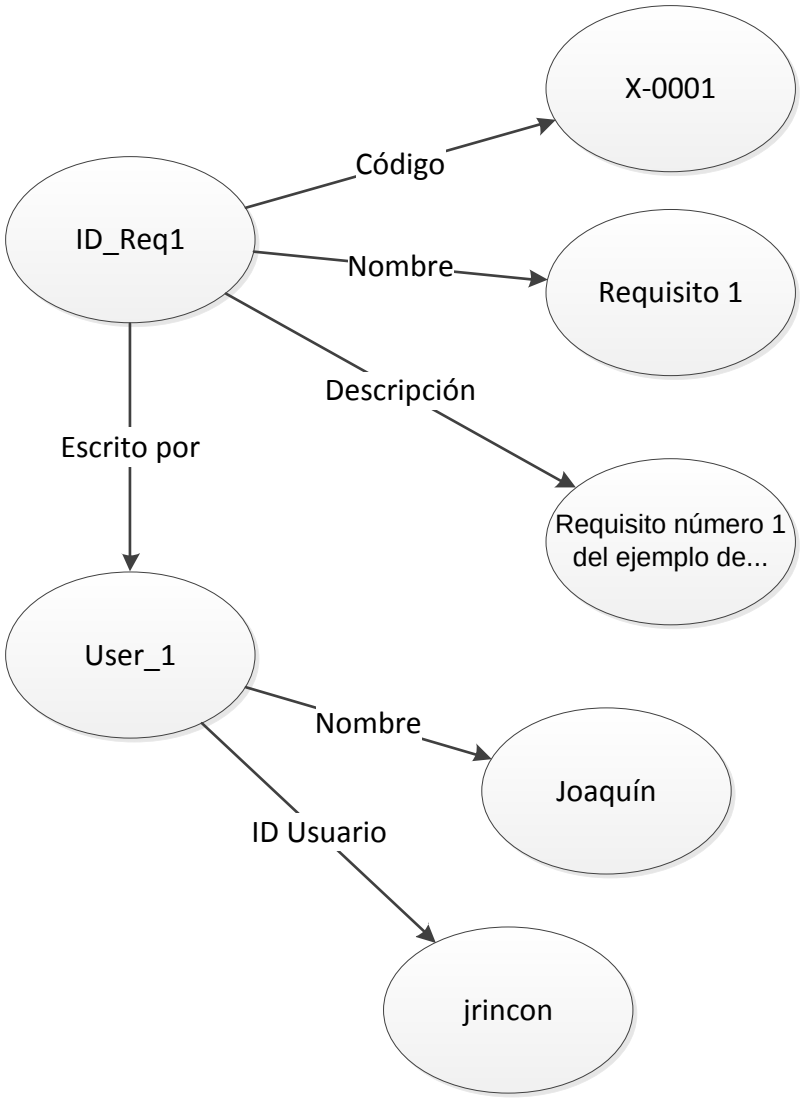


Figura 10 – Conjunto de tripletes RDF representados como un grafo

Si tenemos varios grafos que usan un conjunto consistente de objetos y sujetos, se pueden unir sin mayor esfuerzo. De igual forma, si tenemos varios conjuntos de tripletes, la unión es simplemente el conjunto final de poner todos los tripletes juntos.

URIs

Para que se pueda hacer un uso formal de RDF es importante saber que tenemos un conjunto consistente de objetos, sujetos y predicados, tal y como acabamos de comentar. La forma de poder estar seguros de esa consistencia es poder asegurar si estamos hablando de la misma cosa o no, necesitamos identificadores únicos a nivel universal. Es lo que se denomina en ingles “Strong Identifiers” (Identificadores fuertes).

RDF propone que se utilicen URIs para identificar los distintos elementos de un triplete, una URI (Universal Resource Identifier) es un identificador único de un

recurso. Si dos URIs son iguales es por que identifican exactamente el mismo recurso. **La propuesta de RDF es que cualquier elemento del universo del discurso se conceptualiza como un recurso y consecuentemente se le puede asignar una URI única.**

Así, los sujetos, los objetos y los predicados son todos recursos y todos tienen su URI correspondiente que es lo que usamos en los tripletes

De esta forma, si encontramos un triplete como este `(URI_1, Predicado X, Objeto X)` y en cualquier otro lugar (base de datos, página web, etc.) encontramos otro triplete `(URI_1, Predicado Y, Objeto Y)` sabremos con absoluta seguridad que en ambos casos estamos hablando del mismo sujeto, por que las URIs son independientes del contexto y no hay ambigüedad sobre lo que representan[x].

Todos nosotros estamos muy acostumbrados a utilizar URLs (Universal Resource Locator), ya que se usan para identificar de forma unívoca recursos de internet (páginas, imágenes, etc.). Las URLs son un subconjunto de las URIs, que se pueden utilizar para designar recursos de un universo mucho más amplio que Internet y la web.

Pueden darse casos en los que una aplicación no conoce la URI de un recurso, pero tenemos información sobre ese recurso que queremos guardar. RDF proporciona un mecanismo para ello llamado “blank node” (nodo vacío o también nodo anónimo) que permite a una aplicación generar un ID local para ese nodo y de esa forma poder usarlo, aunque como no es una URI, no tendrá validez más allá del entorno de la aplicación o el sistema específico que lo use.

RDF/XML

La recomendación W3C de RDF (mirar la nota número viii) cubre la definición de RDF como un modelo de datos y explica como usar XML para “escribirlo”, debido a que se definieron a la vez en la misma recomendación mucha gente habla de RDF como RDF/XML, pero no son lo mismo.

Es importante saber que existen otras formas de “serializar”[3] RDF, entre las cuales podemos destacar: N-Triples[xi], N3[xii] y RDFa[xiii]. No vamos a entrar en detalle en ellas aunque es interesante destacar que RDFa es especial, ya que es realmente una forma de introducir información semántica en otros ficheros o documentos ya existentes. Por ejemplo, si tenemos una página en formato XHTML, RDFa se puede utilizar para añadir información sobre lo que aparezca en la página sin que cambie la presentación de esa página.

En general, el problema de serializar un conjunto de tripletes RDF es como compactar la información para reducir el espacio sin perder demasiada legibilidad, para ello se suelen implementar los siguientes mecanismos:

[3] podríamos entender el concepto de serializar la información como el proceso por el cual la podemos poner en un fichero simple de texto, enviarla a algún receptor y poder procesarla para recuperar la información original en el lugar de destino

- Prefijos: Normalmente las URIs son cadenas largas, pero muchos recursos comparten una buena parte de la URI común, especialmente cuando son recursos relacionados de alguna forma o almacenados en el mismo lugar. Para evitar tener que poner las URIs completas en los tripletes se puede definir un prefijo y luego en cada URI solo hay que poner el prefijo y la parte única de esa URI. Por ejemplo, tendremos un mecanismo de definición parecido a este:

  ```
  @prefijo elrincon: <http://www.elrincondejoaquin.com/>
  ```

  De esta forma, las páginas `http://www.elrincondejoaquin.com/recetario.htm` y `http://elrincondejoaquin.com/restaurantes.htm` se pueden expresar como `elrincon: recetario.htm` y `elrincon: restaurantes.htm`

- Múltiples predicados y objetos para un mismo sujeto. Es normal que un sujeto se repita varias veces en varios tripletes. Al serializar existe una construcción sintáctica que permite poner el sujeto una sola vez y expresar múltiples predicados y objetos para ese mismo sujeto sin tener que repetir su URI.
- Algunas construcciones son muy comunes y el estándar proporciona una abreviatura especial para ellas

Valores en RDF

En RDF se pueden usar valores literales, como en el caso de nuestro ejemplo cuando decíamos que el código de nuestro requisito es `X-0001` o que el nombre de nuestro usuario es `Joaquín`.

Es normal identificar el tipo de esos valores literales y normalmente también el idioma al que pertenecen en caso de tratarse de texto. La lista de tipos que se suele utilizar es la definida por el estándar XSD (XML Schema Definition, http://www.w3.org/2001/XMLSchema, usualmente abreviado como xsd:<tipo>) y el país se identifica con el código de la tabla de códigos ISO 639.

Vocabularios (Ontologías)

Como venimos diciendo, todos los elementos del universo del discurso se pueden considerar recursos y por lo tanto tener una URI, de esa forma podemos usar las URIs en nuestros tripletes RDF.

Si vamos a almacenar determinada información en formato RDF, por ejemplo información para gestionar requisitos de sistemas, debemos definir las entidades (sujetos), las propiedades de esas entidades, el tipo de las propiedades, las posibles relaciones entre entidades, etc. Para determinadas áreas del conocimiento ya se han juntado una serie de expertos y han definido estas entidades y relaciones. Han modelado como se debe guardar este tipo de información, que reglas cumplen las relaciones, que tiene sentido y que no y han definido lo que se llama un vocabulario.

Lo más sencillo si vamos a manejar información de alguno de estos dominios bien definidos es utilizar las URIs que suelen estar disponibles y que definen los predicados y algunas de las propiedades y no reinventar la rueda. Siempre podemos extender el

vocabulario con aquello que le falte si es que necesitamos alguna cosa que no está establecida.

Un ejemplo bien conocido de un vocabulario disponible en Internet (hay muchos y crecen día a día) es FOAF[xiv]: Friend Of A Friend (Amigo de un amigo) que es un vocabulario que permite definir el tipo de información que gestionan las redes sociales: datos de una persona (nombre, dirección, fotos en las que aparece, trabajo, etc.), de sus contactos, de sus amigos, etc.

Estos vocabularios son también llamados Ontologías. Podemos decir que una ontología es un conjunto de conceptos y reglas sobre como se relacionan esos conceptos de un determinado dominio de conocimiento. Como dice la definición de la Wikipedia[xv], "sirve para tener una especificación explicita del conocimiento común de un área determinada". En ese artículo de la Wikipedia vienen muchos enlaces a ontologías que están disponibles en Internet.

Es habitual ver en ingles las palabras Ontology, Vocabulary and Namespace (Espacio de nombres) de forma casi sinónima.

OWL

OWL[xvi]: Web Ontology Language (Lenguaje de ontologías web) es el estándar que define el lenguaje que se utiliza para definir las ontologías que hemos comentado en el apartado anterior.

Es decir, antes hemos hablado de que una ontología define un área de conocimiento, pero la forma en que se crea esa definición es siguiendo el estándar definido por OWL.

Es muy interesante explicar que el estándar OWL se define a su vez mediante RDF, es decir, el estándar de como se deben modelar las Ontologías que se usarán al crear almacenes de conocimiento con RDF está especificado mediante RDF, generándose una auto-referencia muy curiosa. Podemos decir que los **modelos son datos** ya que las ontologías son colecciones RDF.

El vocabulario completo de OWL usa partes de los vocabularios RDF, RDFS y XSD, además los elementos OWL propios.

Se trata de un estándar bastante complejo por ello y está dividido en tres niveles: OWL-lite, OWL-DL y OWL Full. Al ser complejo, existen herramientas que ayudan en la creación de ontologías, posiblemente *protégé*[4] es el más famoso de los editores de ontologías, siendo además gratuito y de código abierto (Open Source).

OWL Lite[xvii] se utiliza cuando las necesidades son limitadas, principalmente clasificaciones jerárquicas que tienen algunas restricciones. Se puede establecer la cardinalidad de las relaciones, pero solo si es 0 o 1. La idea es que sea sencillo construir software que use este nivel de expresividad y que la complejidad computacional se mantenga baja.

[4] http://protege.stanford.edu/

OWL DL ofrece la máxima expresividad mientras se asegura que sigue siendo "computable". Se asegura que no existe el "problema de parada"[xviii] y que toda computación termina en un tiempo finito.

OWL Full ofrece el máximo nivel posible de expresión, pero ya no ofrece garantías de computabilidad. Por ejemplo, una clase puede considerarse como un conjunto de individuos, pero también como un individuo y algunas otras características difíciles de manejar de forma automática.

En OWL todo individuo pertenece a la clase `owl:thing` y todas las clases son subclases de `owl:thing` aunque no se especifique explícitamente.

El predicado `rdfs:subClassOf` se utiliza para establecer jerarquías de clases y `rdf:type` para asignar la clase a la que pertenece a un individuo. Es interesante ver que tal y como se mencionó anteriormente, OWL usa predicados de otros vocabularios como RDFS y RDF en estos casos.

A continuación se pueden definir propiedades de las clases, tipos de datos de esas propiedades, cardinalidades, etc.

OWL permite establecer equivalencias entre clases y propiedades y proporciona el predicado `owl:sameAs` que es usado ampliamente para establece que dos individuos son el mismo (que dos URIs distintas son en realidad el mismo objeto).

SPARQL

Aquí tenemos un nuevo acrónimo, que en este caso tiene dos posibles interpretaciones:

SPARQL[xix]: Simple Protocol and RDF Query Language

SPARQL: SPARQL Protocol and RDF Query Language, lo que lo convierte en un acrónimo recursivo

Independientemente de que definición del acrónimo usemos, lo importante es entender que igual que existe el lenguaje SQL para hacer consultas en las bases de datos relaciones, tenemos SPARQL para hacer consultas en los datos guardados usando RDF.

Igual que en SQL tenemos una serie de posibles sentencias (Select, Delete, Update, …) en SPARQL también las tenemos. Vamos a verlas las principales por encima:

SELECT: Es parecida al Select de SQL, retorna una serie de valores en función de una selección. Esta sentencia tiene dos partes. Una donde se da una lista de los resultados que se desean y otra donde se describe un patrón del gráfico que estamos buscando y que sirve para mostrar solo las partes del gráfico que cumplen ese patrón.

```
SELECT var1, var2, …
```

```
WHERE { graph_pattern }
```

CONSTRUCT: La salida de esta instrucción será un conjunto de tripletes. Podemos decir que SELECT extrae valores y CONSTRUCT extrae tripletes válidos.

```
CONSTRUCT graph_pattern

WHERE { graph_pattern }
```

ASK: Solo devuelve si un determinado patrón existe o no

```
ASK graph_pattern
```

DESCRIBE: Esta instrucción puede comportarse de forma distinta según la herramienta, ya que el estándar permite que sea el implementador el que decida que información se considera útil y mostrarla.

```
DESCRIBE var1, var2, …

WHERE { graph_pattern }
```

Todas estas sentencias admiten diversos modificadores para mostrar los datos ordenados (ORDER BY), para hacer agrupaciones, etc.

Veremos algunos ejemplos más detallados de consultas SPARQL al hablar de OWLIM.

En el estándar SPARQL 1.1 hay todo un grupo de sentencias para modificar datos[xx] ("update"), con instrucciones como INSERT DATA, DELETE DATA, DROP, CLEAR, …

Juntándolo todo

Para juntar en una explicación de 30 segundos, lo que los anglosajones llaman "elevator speech" (discurso de ascensor), todo lo que hemos comentado sobre bases de datos semánticas podríamos decir:

Las bases de datos semánticas gestionan información almacenada mediante tripletes RDF que tienen un sujeto, un predicado y un objeto. Estos tres elementos de un triplete normalmente representan un recurso del universo mediante una URI. Existen vocabularios disponibles que se pueden usar sencillamente usando las URIs correspondientes que proporcionan. Esos vocabularios actúan como un modelo de datos disponible de una determinada área de conocimiento. Esos modelos se llaman ontologías y son, a su vez, un conjunto de tripletes RDF. El lenguaje para definir ontologías se llama OWL y el lenguaje para hacer consultas en estas bases de datos se llama SPARQL.

c. Ventajas de las bases de datos semánticas

Muy bien, ¿pero que ventajas aporta el uso de bases de datos semánticas?

Hasta ahora, las aplicaciones han estado íntimamente ligadas con sus datos. Cambios en la aplicación terminan requiriendo cambios en el modelo de datos, si aparecen nuevos datos es más que probable que tendremos que modificar la aplicación. Como este es un proceso costoso, la realidad es que esa evolución no se produce de forma fluida. Una de las metas principales de las tecnologías semánticas es la **separación entre datos y aplicación** mediante la mejora de los sistemas de representación del conocimiento, que de esa forma podrán evolucionar por separado[xxi].

El diseño de nuevas aplicaciones que funcionan de esta forma facilita la portabilidad de los datos. No tenemos datos para una aplicación específica, sino que tenemos aplicaciones capaces de entender un determinado modelo y cualquier fuente de datos que utilice ese mismo modelo podrá ser consumida por la misma aplicación sin tener que cambiar ni una coma.

En el desarrollo tradicional, la integración de datos suele ser costosa y difícil. Existen multitud de herramientas, mecanismos y tecnologías creadas únicamente para dar soporte a este tipo de tareas, como las herramientas ETL (Extract, Transform, Load), muy usadas para el movimiento de datos a almacenes estáticos de tipo Data Warehouse (DW), los ESB (Enterprise Service Bus) que son sistemas para soportar la mensajería entre aplicaciones corporativas, los servicios web para facilitar la interoperabilidad independientemente del lenguaje de programación, etc.

La integración de datos y sistemas, especialmente en las grandes corporaciones, es extremadamente importante. No tiene sentido que las grandes aplicaciones corporativas funcionen de forma aislada, la única forma de rentabilizar sus altos costes es mediante la integración y la creación de una capacidad de análisis integrado de los procesos de negocio a alto nivel.

Por ejemplo, las grandes compañías gastan millones en la creación de enormes almacenes de datos que replican la información ya existentes en sus sistemas transaccionales solo para poder tener una visión integrada de lo que ocurre.

Sin ninguna duda, una de las promesas con mayor potencial comercial de las bases de datos semánticas es la **facilidad de integración de los datos**. Como ya hemos visto, integrar dos grafos RDF es tan sencillo como concatenar uno a continuación del otro.

En el modelo relacional, para integrar dos bases de datos con el mismo modelo, tenemos que ir cogiendo tabla por tabla e incluir en cada una los datos de la otra que no se repiten. El orden en el que se realiza esta operación es importante, por que puede haber reglas de integridad referencial que requieran que algún dato maestro ya esté en su tabla antes de podamos introducir datos de detalle. Incluso para bases de datos con el mismo modelo (en cuanto no estemos hablando de un modelo trivial) este proceso es costoso y es fácil cometer errores.

Otra gran ventaja es la **flexibilidad del modelo**. Con flexibilidad me refiero a la capacidad de hacer cambios en unas partes de la solución sin que impacten a toda la solución. Por ejemplo, si tenemos una base de datos que usa una determinada ontología, muchos cambios en la ontología no requieren ningún cambio en la base de datos. Por ejemplo si la ontología ahora me permite una nueva relación entre

entidades, o una nueva propiedad, añadir esa información en la base de datos sólo requiere añadir nuevos tripletes RDF, no hay que hacer ningún cambio del modelo.

Sin duda esto tiene que verse como una gran ventaja para todos aquellos acostumbrados a trabajar con el modelo relacional. Pequeños cambios en el modelo, como el añadir un atributo a una tabla, requiere cambios en la base de datos, migración de los datos del modelo anterior al modelo nuevo, cambios en la aplicación, en las consultas de la base de datos, etc.

Como ya hemos comentado, el hecho de que la evolución de estos modelos sea tan costosa es una fuente de problemas. Muchas aplicaciones requieren cambios de base de datos cuando se pasa de una versión a otra y muchas empresas deciden no adoptar nuevas versiones que aportan nueva funcionalidad solo para evitar la migración de los datos.

Ya hemos visto que añadir nuevos elementos a una ontología no implica ningún cambio, pero, ¿que pasa si quitamos o cambiamos algo? Es posible que tengamos que buscar algunos tripletes y borrarlos o cambiarlos, pero, una vez más, es un cambio a nivel de datos, el modelo no cambia y la aplicación que lo usa tampoco tiene por que hacerlo.

Lo normal es que una aplicación o sistema que usa una ontología no tenga la lógica de esa ontología en el código, sino que lea la definición de esa ontología, que, como ya hemos dicho, será a su vez un conjunto de tripletes. Eso significa que los cambios en la ontología de los que hemos hablado también son a su vez cambios en los datos, luego una misma aplicación será capaz de aplicar unas reglas y si la ontología cambia, sin necesidad de cambiar la aplicación, será capaz de aplicar otras.

Como ya hemos dicho: los modelos son datos y eso hace que las aplicaciones están prácticamente blindadas frente a cambios en el modelo, por que no se trata más que un cambio en los datos: habrá más o menos registros.

Como este punto es importantísimo, voy a hacer un poco más de hincapié poniendo un ejemplo. Si volvemos a pensar en nuestro ejemplo de la aplicación que gestiona requisitos y que está basada en un modelo relacional, si tenemos una tabla de `Tipos de Requisitos` y la tabla `Requisitos` tiene una clave externa hacia esa primera tabla, gestionar un tipo nuevo de requisito solo implica añadir un registro en la tabla de tipos y no habrá que hacer nada en la aplicación.

Ahora imaginemos que la tabla de `Tipos de Requisitos` no existía. Si ahora esa información fuese necesaria tendríamos que crear la tabla y cambiar el modelo de datos. Hay que añadir el campo con la clave externa en la tabla `Requisitos`, hay que cambiar la aplicación para que al introducir o modificar un requisito se pueda elegir su tipo, hay que poder gestionar tipos: crearlos, modificarlos, borrarlos, etc. Claramente es un cambio de un gran impacto.

Lo que estamos diciendo es que un cambio como este último, en el caso de un modelo semántico, solo requiere la creación de unos nuevos tripletes que definan la entidad `Tipo de Requisito` y su relación con la entidad `Requisitos` y nada más. Es un

cambio tan sencillo como el que explicábamos al principio donde la tabla existía. Es como si nuestro modelo ya recogiese todas las posibilidades de posible evolución.

d. Bases de datos semánticas disponibles

Cada vez hay más herramientas en el mercado que permiten almacenar información usando RDF tal y como hemos contado hasta ahora.

Aquí pongo una lista con las que he encontrado y un link o bien directamente a la página de la herramienta o a la página de la wiki de W3C en la que se habla de esa herramienta[5]. Esta información era válida a fecha de 20 de mayo de 2012.

- 3Store (triple store).
- 4store (triple store).
- AllegroGraph RDF Store (triple store, programming environment, reasoner, development environment, rdfs reasoner). Directly usable from Java, LISP, Python, Prolog, C, Ruby, Perl
- ARC RDF Store (triple store). Directly usable from PHP
- Bigdata® (triple store, reasoner, rdfs reasoner).
- ClioPatria (triple store, programming environment, reasoner, rule reasoner). Directly usable from Prolog, C
- CumulusRDF, a Linked Data server (triple store). Directly usable from Java
- djubby (triple store, development environment). Directly usable from Python
- Dojo.data (triple store, programming environment). Directly usable from Javascript
- Dydra (sparql endpoint, triple store). Directly usable from Ruby
- Euler (triple store). Directly usable from Java, C-sharp, Python, Javascript, Prolog
- IBM SLRP (development environment, triple store).
- Jena, a Java RDF API and toolkit (triple store, programming environment, reasoner, rule reasoner, owl reasoner, rdfs reasoner). Directly usable from Java
- Mulgara Semantic Store (triple store). Directly usable from Java
- Open Anzo (development environment, reasoner, triple store, programming environment, owl reasoner, rdfs reasoner). Se puede usar directamente con Javascript, Java, .Net
- OpenLink Virtuoso (triple store, reasoner, rdf generator, sparql endpoint, owl reasoner, rdfs reasoner, rdb2rdf). Se puede usar directamente con C, C++, Python, PHP, Java, Javascript, ActionScript, Tcl, Perl, Ruby, Obj-C
- Oracle Spatial 11g (triple store, reasoner, owl reasoner). Se puede usar directamente con Java
- OWLIM (triple store)

[5] Casi toda la información de esta tabla viene de la que aparece aquí: http://www.w3.org/2001/sw/wiki/index.php?title=Special:Ask&offset=0&limit=500&q=%5B%5BCategory%3ATool%5D%5D+%5B%5BSW+Technology%3A%3ARDF%5D%5D&p=format%3Dul%2Ftemplate%3DToolDisplayLinkWithNameWithcategoryAndLanguage%2Flink%3Dnone%2Fcolumns%3D1&po=%3FTool+Name%3D%0A%3FCategory%3D%0A%3FProgramming+language%3D%0A

- Parliament (triple store, reasoner, owl reasoner, rdfs reasoner, rule reasoner). Se puede usar directamente con Java, C
- Racer Pubby (triple store, development environment, rdf or owl browser). Se puede usar directamente con Java
- RAP NetAPI (triple store, programming environment). Se puede usar directamente con PHP
- RDF Entity Manager (programming environment, triple store). Se puede usar directamente con Java
- RDFLib (triple store, programming environment). Se puede usar directamente con Python
- RDFStore (triple store, programming environment). Se puede usar directamente con Perl, C
- rdfstore-js (triple store, programming environment). Se puede usar directamente con Javascript
- RedStore (triple store, sparql endpoint).
- S3DB (triple store)
- Semantic Server (triple store).
- Semantics.Server (reasoner, triple store, owl reasoner, rdfs reasoner).
- SemWeb (triple store, programming environment). Se puede usar directamente con C-sharp
- SeRQL: SWI-Prolog Semantic Web Server (triple store, programming environment). Se puede usar directamente con Prolog
- Sesame (triple store, programming environment, reasoner, rdfs reasoner). Se puede usar directamente con Java, Python, PHP
- Stardog (sparql endpoint, triple store, development environment, reasoner, owl reasoner). Se puede usar directamente con Java
- Strabon (triple store, registry, sparql endpoint, visualizer). Se puede usar directamente con Java
- StrixDB (triple store, programming environment, reasoner, rdfs reasoner). Se puede usar directamente con Lua, C, C++
- StrixStore (triple store, sparql endpoint, reasoner, rule reasoner). Se puede usar directamente con C, C++
- Talis Platform (triple store, sparql endpoint).
- USeekM: GeoSparql for Sesame Triplestores (triple store, development environment). Se puede usar directamente con Java
- YARS (Yet Another RDF Store) (triple store, programming environment). Se puede usar directamente con Java

Esta lista es bastante exhaustiva, ya que viene de una página de W3C especialmente dedicada a la web semántica y a RDF. Cualquier persona que trabaje en esta área conocerá esta web y pedirá ser incluido en esta lista, aunque me resulta curioso que OWLIM no esté (lo he incluido yo en la lista). Sin embargo si aparece en la lista que hay en esta otra página: http://www.w3.org/2001/sw/wiki/Category:Triple_Store, que tiene ligeras discrepancias con la lista anterior.

Ahora vamos a pedir un dictamen al todopoderoso buscador de Google. La idea es buscar en Google, usando principalmente los términos “semantic database”, y ver que

aparece. El hecho de que unas u otras herramientas aparezcan en Google más arriba o más abajo, o que no aparezcan en absoluto, no determina nada con respecto a su calidad, pero si da una determinada idea de la actividad que hay en la web alrededor de esas herramientas.

Al iniciar la búsqueda, es llamativa la falta de herramientas comerciales en los links que aparecen. Hay mucho link a la Wikipedia, a algunos trabajos académicos, pero en general es difícil encontrar links directos a páginas web de productos.

Hay que decir que Ontotext, la empresa que hace OWLIM, son los primeros que aparecen, aunque es a través de una presentación que tienen en Slideshare.com

Cambiando la búsqueda y añadiendo RDF al final o directamente "RDF database" encuentro referencias a S3DB, AllegroGraph, Sesame, Stardog y Mulgara (los links a los sitios de estas herramientas están disponibles en el listado anterior), pero poco más. Estos links aparecen mezclados con artículos.

Un artículo[xxii] nombra las siguientes empresas y/o productos:

- Oracle
- Virtuoso
- AllegroGraph
- BigData
- OWLIM
- 5Store
- Talis Platform

Como aquellos que tienen un interés comercial. Ya sea porque hay que pagar por el producto o por que, aunque el producto sea gratuito, hay una empresa detrás con ánimo de lucro (normalmente basado en ofrecer servicios alrededor de la base de datos, aunque las licencias sean gratis).

Vemos que desde luego esta no es un área muy madura o muy concurrida por fuertes empresas comerciales. No vemos un producto de Microsoft, Google, SAP, o CA (Computer Associates), por nombrar solo algunos.

Oracle[6] e IBM son las excepciones, son las únicas empresas grandes y ampliamente conocidas que tiene un producto en esta área. En el caso de Oracle parece que no es un producto específico, sino una especie de añadido sobre su base de datos tradicional en su versión actual, que es la 11g.

Existen varios frameworks que permiten conectar una capa de persistencia independiente por debajo de la funcionalidad que ofrecen para hacer razonamiento, consultas SPARQL, etc. Parece que Oracle ha desarrollado unos adaptadores para dos de esos frameworks: Jena y Sesame (los links aparecen en la lista anterior) de forma que en lugar de usar su propio sistema de almacenamiento usan Oracle, pero parece que se trata de una versión comercial de Oracle 11g, no es un producto especial. Eso quiere decir que esos adaptadores sirven para que la información se

[6] http://www.oracle.com/technetwork/database/options/semantic-tech/index.html

almacene en una base de datos fundamentalmente relacional (Oracle dice que su base de datos es objeto-relacional), aunque parece que hay que instalar lo que llaman "latest Semantic Technologies components".

Vamos a intentar otro tipo de búsqueda, una que nos de resultados mas enfocados en lo que queremos. Una forma de conseguirlo es haciendo una búsqueda en un sitio de favoritos como Delicious[7]. En Delicious tenemos varios millones de favoritos que la gente ha guardado en la nube para tenerlos disponibles desde cualquier sitio y que no estén almacenados en su PC. Al guardar estos enlaces los usuarios ponen unas etiquetas para indicar de qué tema trata una determinada página web.

Al hacer una búsqueda en Delicious, buscamos en las etiquetas que la genta ha utilizado. Si buscamos por "semantic database" y encontramos algo, eso significa que alguien ha guardado un enlace y al calificarlo le ha puesto las etiquetas "semantic" y "database", luego es mucho más probable que lo que encontremos esté relacionado con el tema que nos interesa. Google proporciona muchos más resultados y extraídos de muchos más sitios, pero este tipo de búsqueda proporciona resultados más relevantes.

Al buscar en Delicious aparecen muchos enlaces a sitios que no proporcionan herramientas, sino que son almacenes de datos en formato RDF, como DBPedia[8] o Freebase[9], pero también aparecen enlaces como los que estamos buscando, a herramientas como 4store, AllegroGraph, OpenRDF (Sesame), Mulgara y algunos más (los links están disponibles en la lista completa más arriba).

Basado en todo lo anterior, vamos a poner en una tabla las herramientas que parecen estar más maduras a día de hoy con algunas de sus características.

Nombre	Lenguajes de programación	Soporte SPARQL	Tamaño (*10^6)	Comentario
4Store	PHP, Ruby, Python, Java[10]	Si	15.000	
AllegroGraph	Java, LISP, Python, Prolog, C, Ruby, Perl	Si	1.009.690	
BigData	Java	Si	12.700	
Jena	Java	Si	Versión SDB 650 Versión TDB 1.700	Es un framework muy utilizado aunque el almacén de datos sea otra herramienta
OpenLink Virtuoso	C, C++, Python, PHP, Java, Javascript, ActionScript, Tcl, Perl, Ruby, Obj-C	Si	15.400	
OWLIM	Java[11]	Si	12.000	

[7] http://delicious.com
[8] http://dbpedia.org/About
[9] http://www.freebase.com/
[10] Librerías de acceso proporcionadas por terceros
[11] Mas todos los que puedan proporcionar Sesame y Jena, que pueden usarse como frameworks junto con OWLIM como almacén de datos

Sesame (OpenRDF)	Java, Python, PHP	Si	70	Es un framework muy utilizado aunque el almacén de datos sea otra herramienta

Vamos a ver con algo más de detalle algunas de las herramientas de la lista anterior, principalmente las comerciales, que suelen tener más y mejor información en la web y unos instaladores más sofisticados.

AllegroGraph

AllegroGraph es la base de datos semántica de la empresa Franz, Inc. Forma parte de un completo conjunto de herramientas para hacer minería de datos y dar soporte a tecnologías semánticas. La hemos elegido por ser de los productos más maduros, cosa normal al tratarse de un producto comercial.

Según los datos que he podido consultar, a día de hoy tiene el record de la base de datos semántica más grande, con algo más de un billón (hablamos de un billón español, lo que los americanos llaman un "trillion") de tripletes[12]; 1,009,690,381,946 tripletes para ser exactos.

AllegroGraph ofrece casi todas las características que se pueden esperar de un producto comercial de bases de datos, como la posibilidad de hacer back-ups online (sin necesidad de parar la base de datos), concurrencia de conexiones para leer y escribir, cien por cien ACID[xxiii] para la gestión de transacciones, etc.

Arquitectura de AllegroGraph:

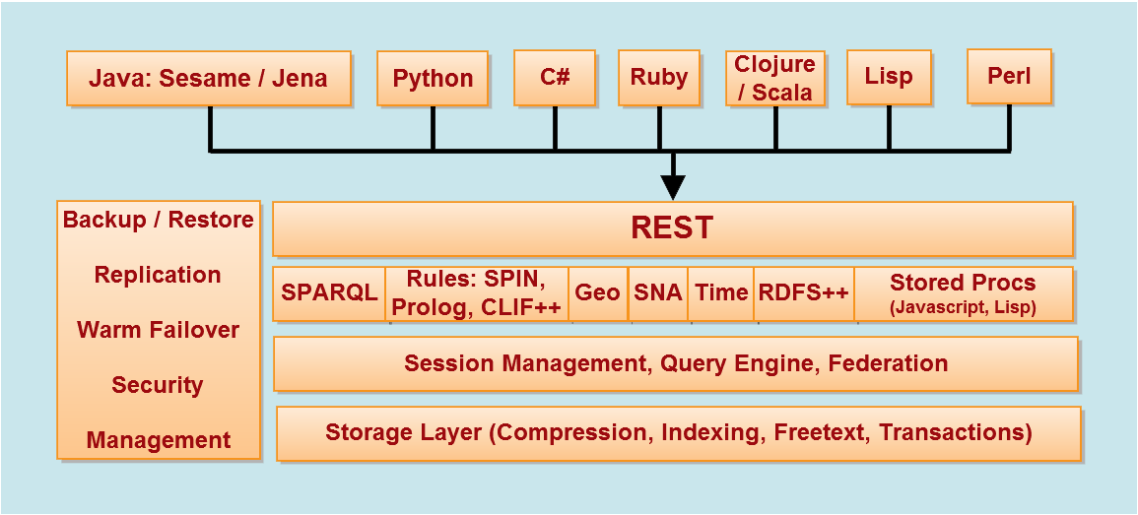


Figura 11 – Arquitectura de AllegroGraph

Como vemos en la arquitectura, el interface hacia el exterior es básicamente un conjunto de servicios web de tipo REST que pueden ser llamados desde cualquier lenguaje de programación.

[12] http://semanticweb.com/franz%E2%80%99s-nosql-database-successfully-loads-1-trillion-rdf-triples_b22391

Este producto sólo puede instalarse en máquinas Linux. Proporcionan una instalación estándar de EC2 (Elastic Cloud Computing, el servicio de infraestructura como servicio de la empresa Amazon) para crear directamente máquinas en Amazon con todo instalado y listas para empezar a funcionar.

Para ejecutar en Windows recomiendan usar una máquina virtual (por supuesto el rendimiento no será representativo) y ofrecen alguna ya disponible, son máquinas de tipo VMWare.

Entre el software disponible ofrecen librerías para clientes en varios lenguajes, siendo especialmente interesante el soporte que dan a Lisp, ya que la empresa Franz ha estado muy ligada a Lisp desde sus comienzos.

Existe una licencia gratuita que permite almacenar hasta 5 millones de tripletes.

Virtuoso[13]

Vamos a ver con un poco más en detalle el producto OpenLink Virtuoso. Lo hemos elegido por estar entre los propuestos por el departamento para hacer el trabajo fin de carrera sobre él, por tratarse de uno de los productos comerciales más maduros, por tener en producción algunas de las bases de datos semánticas más grandes que existen actualmente, ofrecer soporte para SPARQL y por poderse usar con muchos lenguajes de programación.

Este es uno de los productos comerciales y por lo tanto tiene un aspecto más cuidado, tanto la web como el instalador de la herramienta (la mayoría de las herramientas no tienen instalador).

El producto se llama Virtuoso Universal Server y pretende ser una solución única para diversas clases de almacenamiento, desde relacional tradicional hasta gestión documental con indexación de texto completo pasando por RDF y XML. A esto le añade acceso mediante servicios web y su propio servidor web.

En el siguiente link http://virtuoso.openlinksw.com/features-comparison-matrix/ aparece una lista de funcionalidades que tiene la herramienta según versiones y sirve para tener una idea de todo lo que puede hacer.

En la siguiente imagen podemos ver la arquitectura de la aplicación:

[13] http://virtuoso.openlinksw.com/

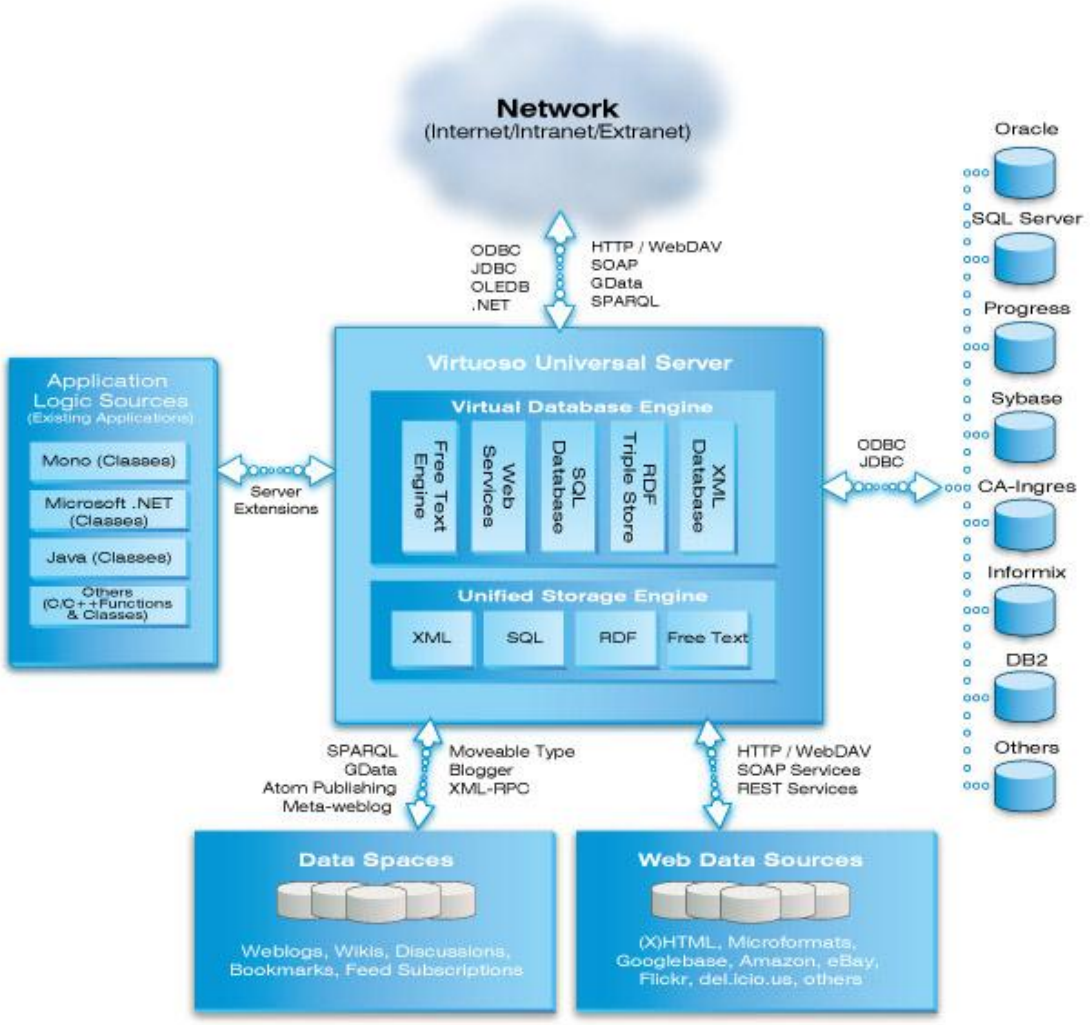
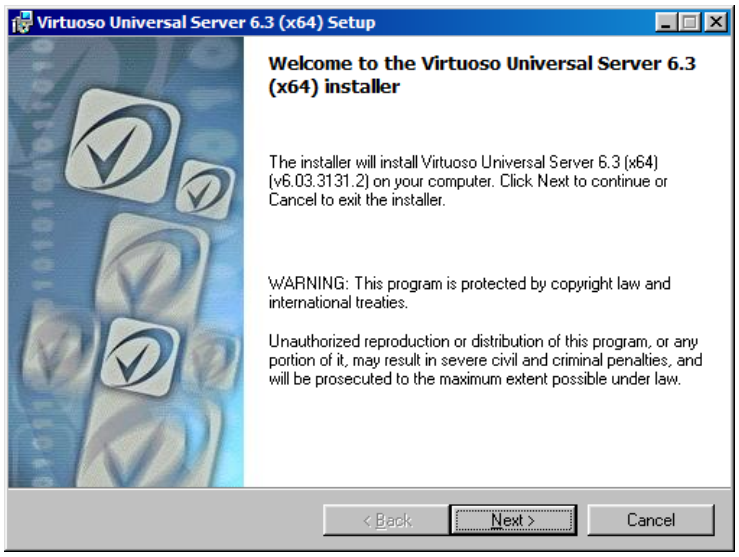


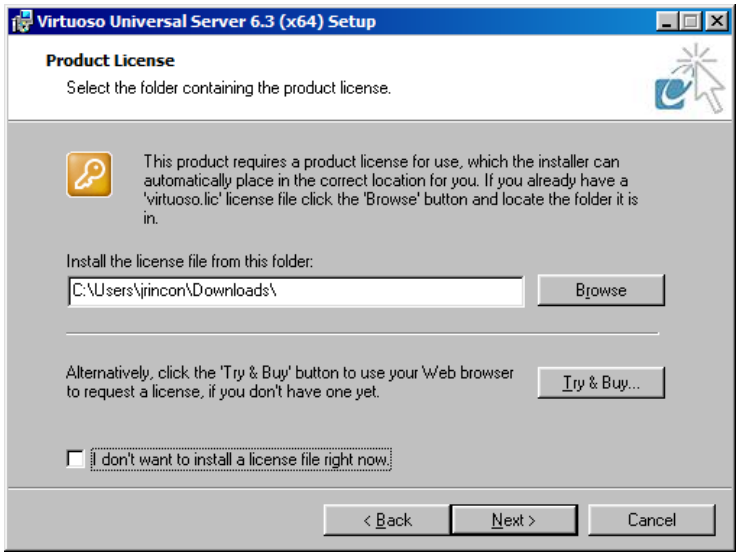
Figura 12 – Arquitectura de OpenLink Virtuoso Universal Server[14]

Vemos que se puede usar su propio sistema de almacenamiento o el de alguna otra base de datos que se pueda acceder mediante ODBC o JDBC, así que también ofrece la posibilidad de ser un motor de inferencia semántica sobre sistemas relacionales tradicionales.

Virtuoso tiene un instalador bastante sencillo de seguir. A continuación pongo algunos de los pasos de la instalación:

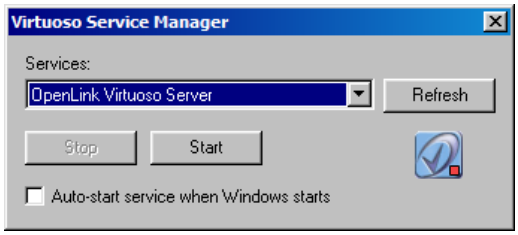
[14] http://virtuoso.openlinksw.com/vdb-conceptual-architecture/



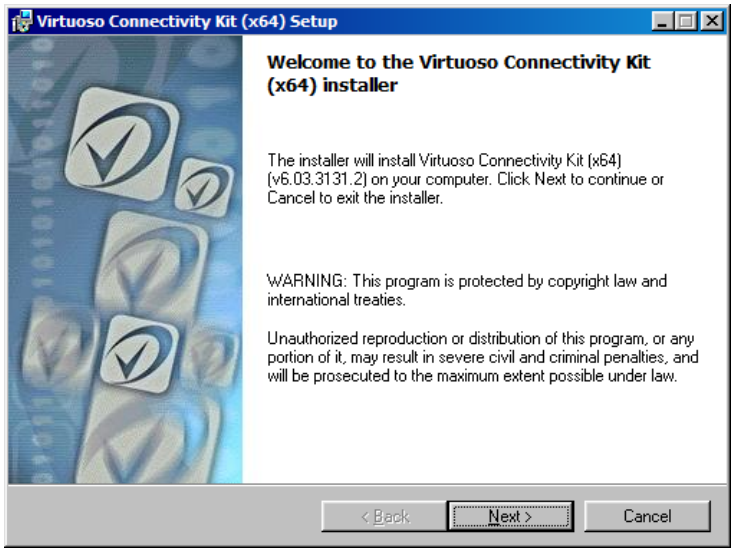


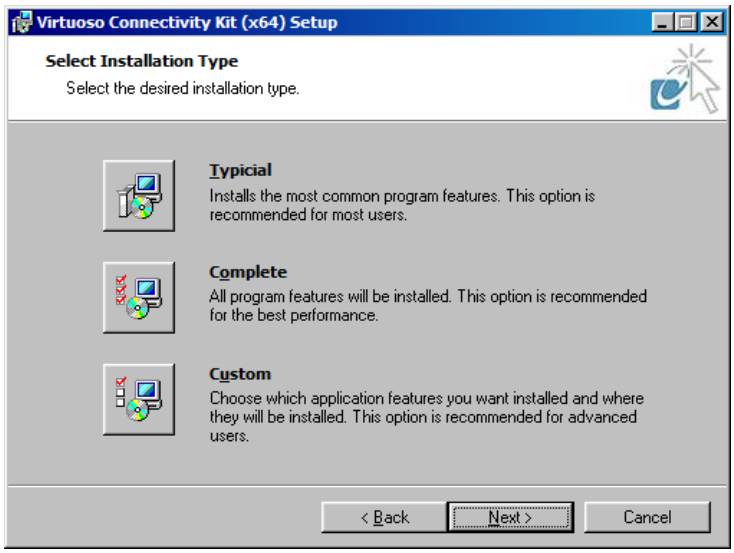
La licencia de evaluación solo proporciona 15 días, que es poco para un producto complejo y nuevo, en mi opinión. Además casi todas las herramientas ofrecen un mes como periodo de evaluación.

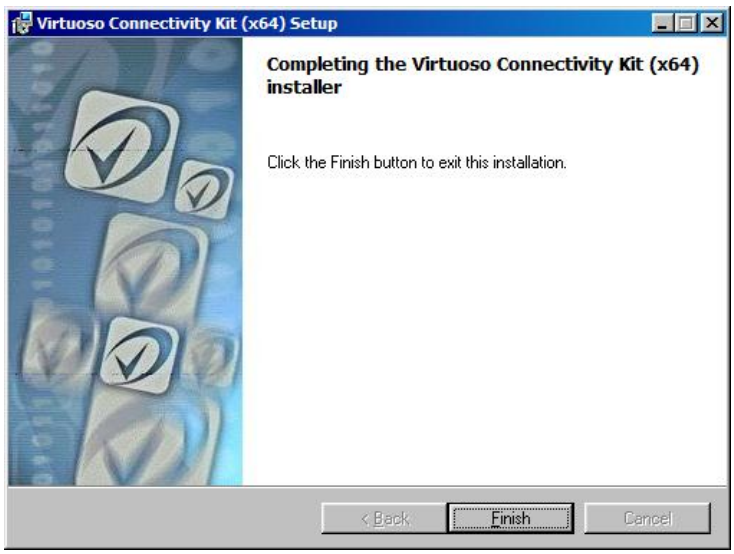
Una vez instalado, hay que arrancar el Service Manager y desde esta pequeña aplicación podemos poner en marcha el Virtuoso Universal Server o pararlo.



Igualmente se puede instalar un kit de conectividad para poder acceder al servidor usando diferentes tecnologías. La instalación también es sencilla.







Mi sensación con el producto es que intenta ser una solución a todo, de ahí su concepto de “servidor universal” y se desdibuja un poco. No es extraño que no aparezca entre los puestos destacados cuando se busca una base de datos semántica.

Personalmente opino que el mercado tradicional de bases de datos ya está saturado por productos perfectamente conocidos como Oracle, SQL Server de Microsoft, DB2 de IBM y mucho más. Además existen productos muy buenos y de altas prestaciones gratuitos, como MySQL. Una base de datos relacional más lo tienen muy difícil para aportar algo nuevo.

Entiendo lo que intenta la empresa con su concepto de servidor universal, pero creo que lo tendría mejor para posicionar su producto si se enfocase en el área semántica.

Por otra parte solo dan 15 días para evaluar, pero una vez instalado el servidor no está claro como seguir. No hay un punto claro para comenzar (“Getting started”) y, aunque hay tutoriales, mi impresión es que saltan a soluciones que no está claro como ejecutar.

En esta web: http://lod.openlinksw.com/ se puede trabajar con un almacén de datos en Virtuoso que a mayo de 2010 tenía más de 15.400.000.000 tripletes[xxiv].

Hay mucha información interesante en la web de OpenLink Virtuoso, incluyendo bastantes “White papers” que explican la mejor forma de sacar partido de la herramienta.

Por otra parte, Virtuoso también parece ser de las mejores soluciones en cuanto a rendimiento, comparada con algunas otras herramientas[xxv], aunque esta información puede variar muy rápidamente conforme aparecen nuevas herramientas en el mercado o se lanzan nuevas versiones de las ya existentes.

3. OWLIM

OWLIM[15] es un producto de la empresa Ontotext[16]. Se trata de un producto comercial (aunque tiene una versión gratuita como veremos más adelante) y consecuentemente tiene un grado de madurez mayor que la mayoría de las herramientas que hemos venido viendo.

Ontotext es una empresa Búlgara creada en el año 2000 que se define como una experta en el mundo de las tecnologías semánticas con productos propios. Han abierto una oficina en estados unidos y pertenecen a un grupo mayor, llamado Grupo Sirma[17], que es una empresa Búlgaro-Canadiense que se creó en el año 1992

Ontotext tiene ahora mismo más de 55 empleados y participa activamente en algunos productos semánticos open source como Sesame.

OWLIM está hecho en Java y permite el acceso mediante Sesame o Jena (sustituyendo el almacén de datos que proporcionan estos dos frameworks por defecto).

Aquí vemos un ejemplo de una arquitectura con OWLIM como sistema de almacenamiento:

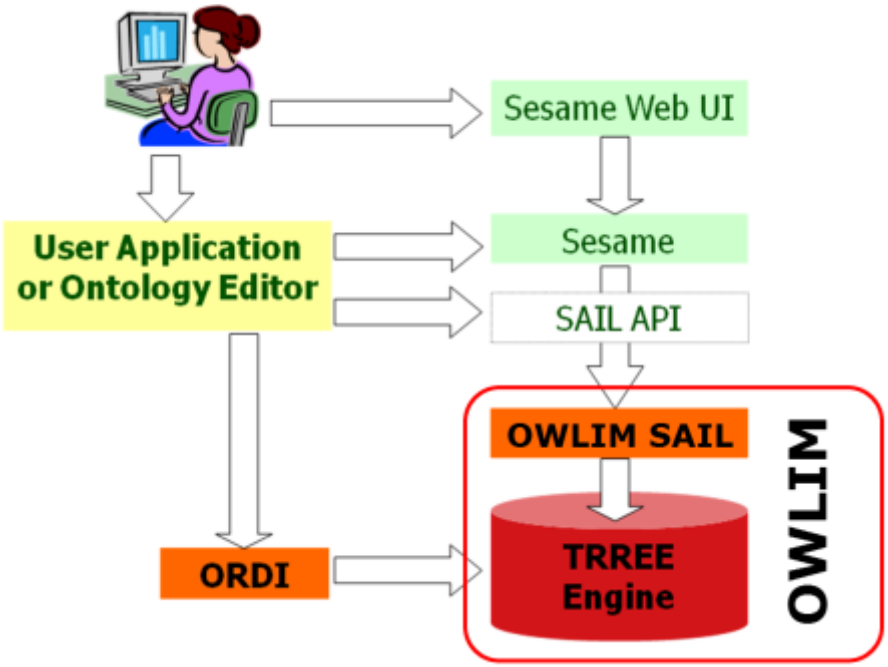


Figura 13 – Arquitectura de OWLIM[18]

ORDI significa Ontology Representation and Data Integration y es una librería Java para acceder a los datos.

[15] http://www.ontotext.com/owlim
[16] http://www.ontotext.com/
[17] http://www.sirma.com/
[18] http://owlim.ontotext.com/display/OWLIMv42/Primer+Introduction+to+OWLIM

El componente SAIL (Storage and Inference Layer) es una capa estándar de acceso al motor de inferencia y al almacenamiento. OWLIM está implementado como un conector SAIL en el framework Sesame

OWLIM se puede usar de las siguientes formas:

- Como servidor aislado al que se conectan las aplicaciones para hacer consultas de datos, al estilo de como funcionan los sistemas relaciones más famosos como Oracle o SQL Server.
- Como una librería Java que se puede usar desde una aplicación.
- A través de Sesame, que es el método preferido y recomendado por la empresa. Las aplicaciones usan la librería Sesame con todas sus funcionalidades (soporte de SPARQL 1.1, N3, N-Triples, RDF/XML, …) y OWLIM proporciona la capa de persistencia.
  En este caso, podemos acceder a los datos de OWLIM mediante todos los mecanismos que proporciona Sesame, que además de un API también puede ejecutarse como un conjunto de servicios web de tipo REST[xxvi].
- A través del GUI de Sesame como punto de acceso SPARQL
- A través de Jena, es igual que a través de Sesame, pero Ontotext afirma que el rendimiento será mejor con Sesame.

a. Versiones

Existen tres versiones de OWLIM

OWLIM-Lite

Se llamaba anteriormente SwiftOWLIM.

Ofrece una solución básica, que permite las mismas reglas de inferencia y capacidad semántica que las versiones superiores, pero que está limitada para que en la práctica no se pueda usar en entornos reales de trabajo. Está basada en una versión distinta del TRREE Engine (Triple Reasoning and Rule Entailment Engine, que también es un producto de Ontotext) que sus hermanas mayores.

OWLIM-SE

Se llamaba anteriormente BigOWLIM

Ofrece las mismas capacidades que la versión Lite, pero además mejora el almacenamiento de la información, la capacidad de optimización de las consultas, etc.

En la siguiente tabla se puede ver las diferencias principales entre las dos versiones

Parámetro	OWLIM-Lite	OWLIM-SE
Optimización de consultas	No	Si
Persistencia	Back-up como N-Triples	Ficheros binarios de datos e

		índices
owl:sameAs eficiente	No	Si
Capacidades avanzadas	Ninguna	RDF Rank Búsqueda de texto completo Extensión geo-espacial Optimización owl:sameAs
Soporte Sesame 2.4.x	Si	Si
Soporte Jena 2.6.x	No	Si

Además, la versión SE tiene mejor rendimiento al cargar información, al extraerla y al ejecutar consultas por usar la versión más avanzada disponible del motor TRREE.

OWLIM-Enterprise[19]

Se llamaba anteriormente BigOWLIM Replication Cluster.

Es igual que la versión SE pero permitiendo que se pueda desplegar en varios servidores a la vez de forma que permite ejecución en paralelo de consultas, tolerancia a fallo (por redundancia de servidores) y escalabilidad horizontal (si queremos más rendimiento o tenemos más datos sólo hay que añadir más servidores).

Esta imagen muestra un posible despliegue en el que se replica el nodo maestro (que es el que reparte el trabajo entre los otros nodos) para protegerse ante caídas de ese nodo y hay tres nodos de trabajo:

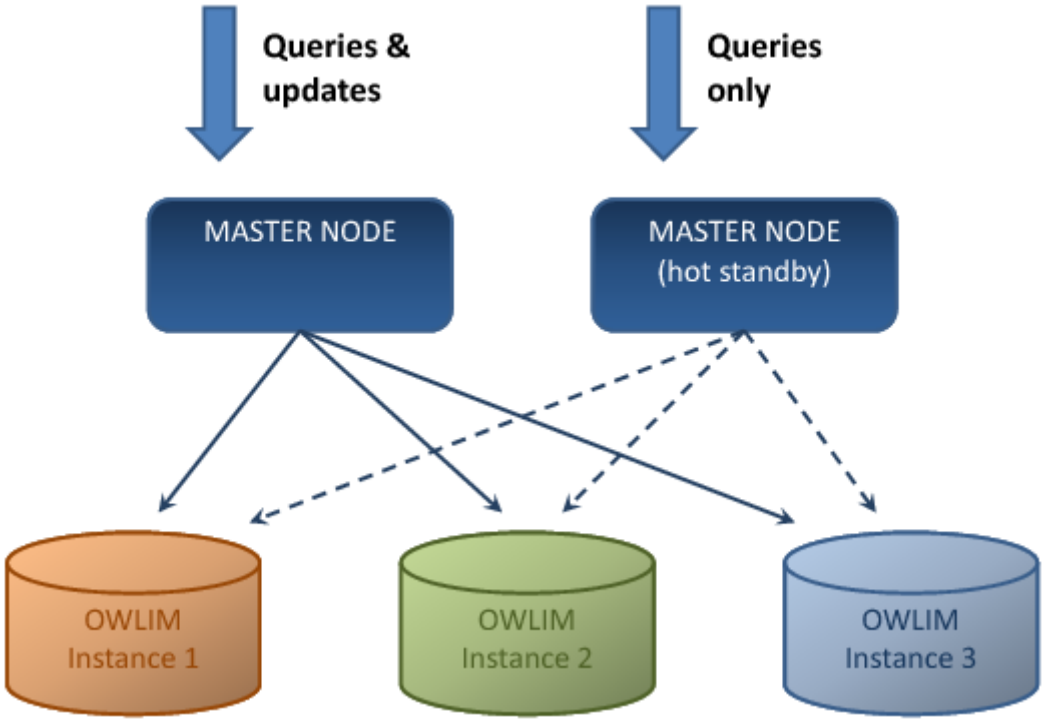


Figura 14 – OWLIM Replication Cluster

[19] http://www.ontotext.com/owlim/replication-cluster

b. Instalación

Vamos a describir el proceso de instalación en Windows.

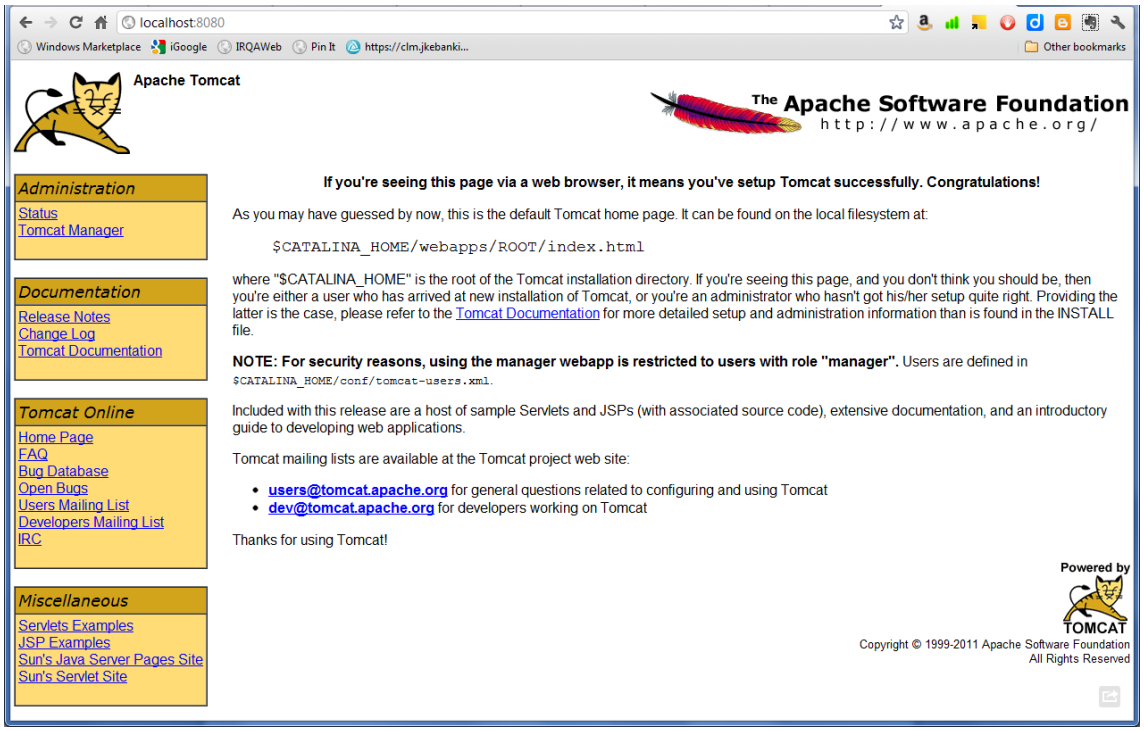
La instalación de OWLIM no es sencilla. Tampoco es excesivamente complicada, pero está muy lejos de la idea del famoso “next, next, next” o “continuar, continuar, continuar”.

Para empezar no hay un instalador y se supone que Tomcat[20] (versión 6, aunque no está especificado claramente y hay que investigarlo un poco para encontrarlo) está instalado. Tomcat es un servidor de aplicaciones Java con capacidad para ejecutar JSP y Servlets.

Instalar Tomcat 6 tampoco es tan sencillo como seguir un Wizard, aunque no es muy complicado. Hay que bajarse un fichero comprimido tipo zip donde está el producto, descomprimirlo en el directorio que queramos usar (en mi caso `C:\Apache-Tomcat-6`) y si tenemos Java ya instalado, como es mi caso, simplemente tenemos que arrancarlo con un fichero llamado `startup.bat`, que esta en el directorio `\bin` de la instalación de Tomcat.

Al arrancar aparece una pantalla de consola en la que se va viendo la actividad del servidor. Si todo va bien, en algún momento nos dice que Tomcat está arrancado. Para comprobar que en verdad está funcionando, podemos ir a la página principal del servidor de aplicaciones, que si no se han cambiado los parámetros de instalación, será http://localhost:8080, ya que Tomcat utiliza el puerto 8080 por defecto.

La página que aparece es la siguiente:



Pantalla Inicial de Tomcat.

[20] http://tomcat.apache.org/

Si vemos esta pantalla es que la instalación de Tomcat ha ido bien y el servidor de aplicaciones está funcionando como debe.

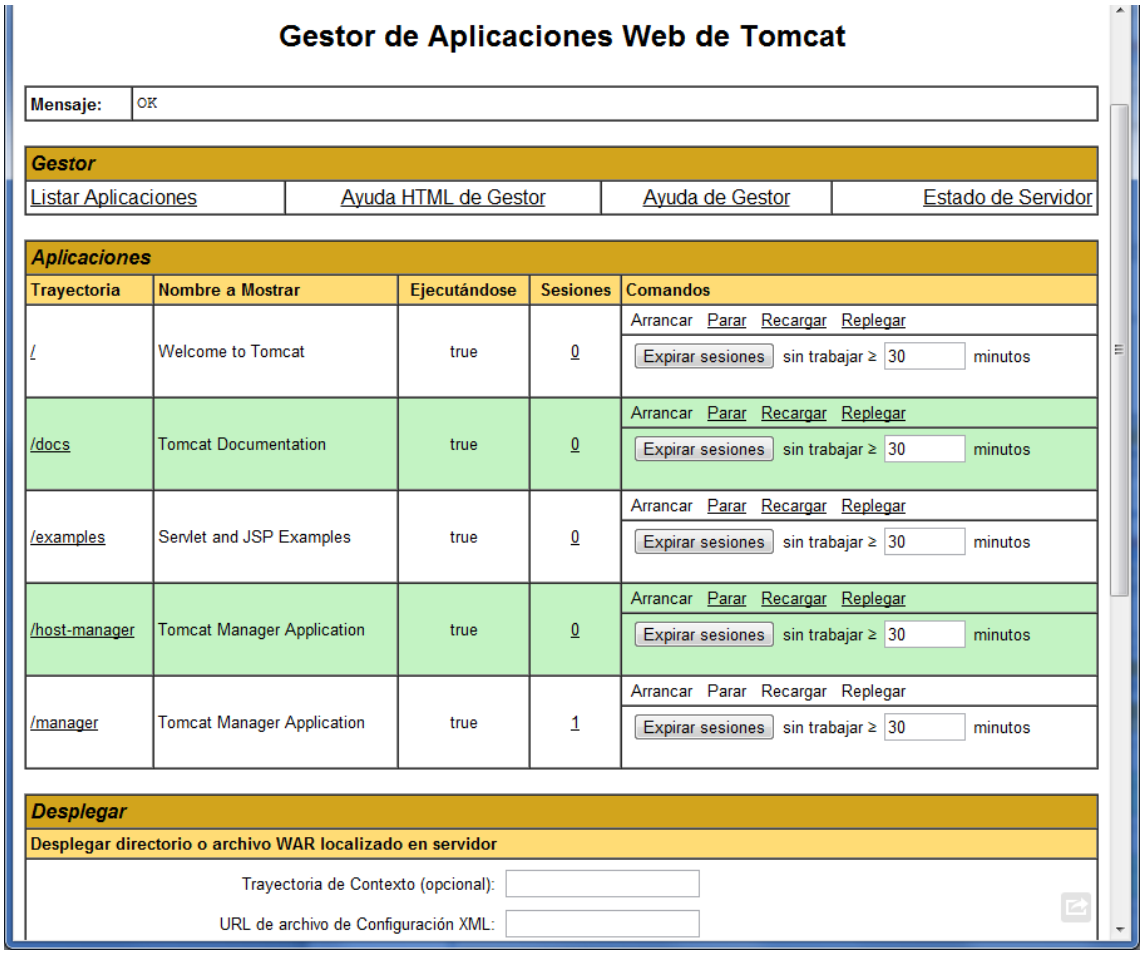
Si ahora pulsamos la opción de menú “Tomcat Manager”, nos pedirá un usuario y una contraseña. A estas alturas no debe haber usuarios definidos, luego hay que crear uno y además darle permisos para que pueda entrar en el “Tomcat Manager”.

Hay que editar el fichero `tomcat-users.xml` que está en el directorio `\conf` de la instalación de Tomcat. Hay que añadir una línea como esta:

```
<user username="tomcat" password="tomcat" roles="manager-gui"/>
```

Lo que crea un usuario con ID *tomcat*, palabra clave *tomcat* y con permisos de manager en el interfaz gráfico proporcionado.

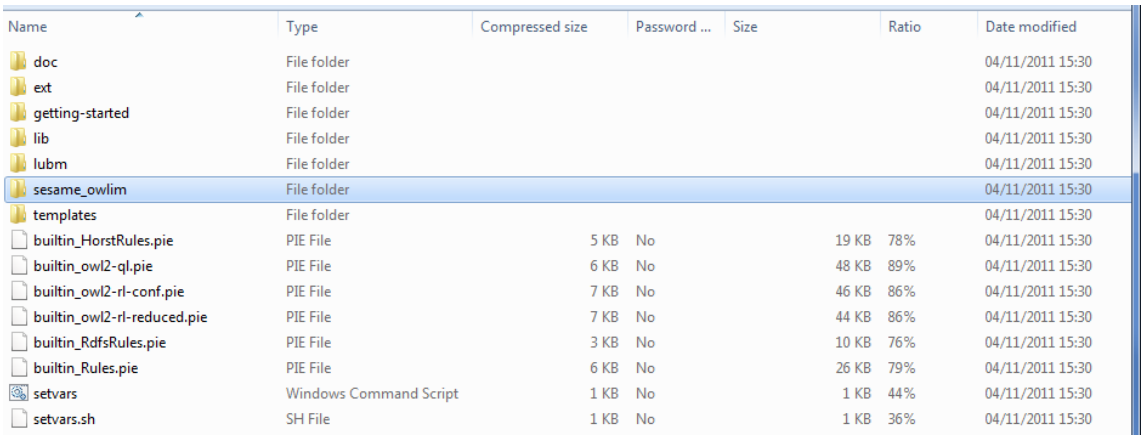
Paramos Tomcat y lo volvemos a arrancar y ahora al pulsar en “Tomcat Manager” ponemos el usuario y la palabra clave y podemos ver la lista de aplicaciones instalada, tal y como aparece en la imagen siguiente:



Pantalla del Tomcat Manager

Aquí podemos ver las aplicaciones de ejemplo que vienen con la instalación inicial de Tomcat.

Ahora podemos pasar a la instalación de OWLIM. En este caso también se trata de un ZIP que trae una estructura de directorios y distinta información. Esta es la estructura de directorios de primer nivel del ZIP:



Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
doc	File folder					04/11/2011 15:30
ext	File folder					04/11/2011 15:30
getting-started	File folder					04/11/2011 15:30
lib	File folder					04/11/2011 15:30
lubm	File folder					04/11/2011 15:30
sesame_owlim	File folder					04/11/2011 15:30
templates	File folder					04/11/2011 15:30
builtin_HorstRules.pie	PIE File	5 KB	No	19 KB	78%	04/11/2011 15:30
builtin_owl2-ql.pie	PIE File	6 KB	No	48 KB	89%	04/11/2011 15:30
builtin_owl2-rl-conf.pie	PIE File	7 KB	No	46 KB	86%	04/11/2011 15:30
builtin_owl2-rl-reduced.pie	PIE File	7 KB	No	44 KB	86%	04/11/2011 15:30
builtin_RdfsRules.pie	PIE File	3 KB	No	10 KB	76%	04/11/2011 15:30
builtin_Rules.pie	PIE File	6 KB	No	26 KB	79%	04/11/2011 15:30
setvars	Windows Command Script	1 KB	No	1 KB	44%	04/11/2011 15:30
setvars.sh	SH File	1 KB	No	1 KB	36%	04/11/2011 15:30

Estructura del paquete de instalación de OWLIM

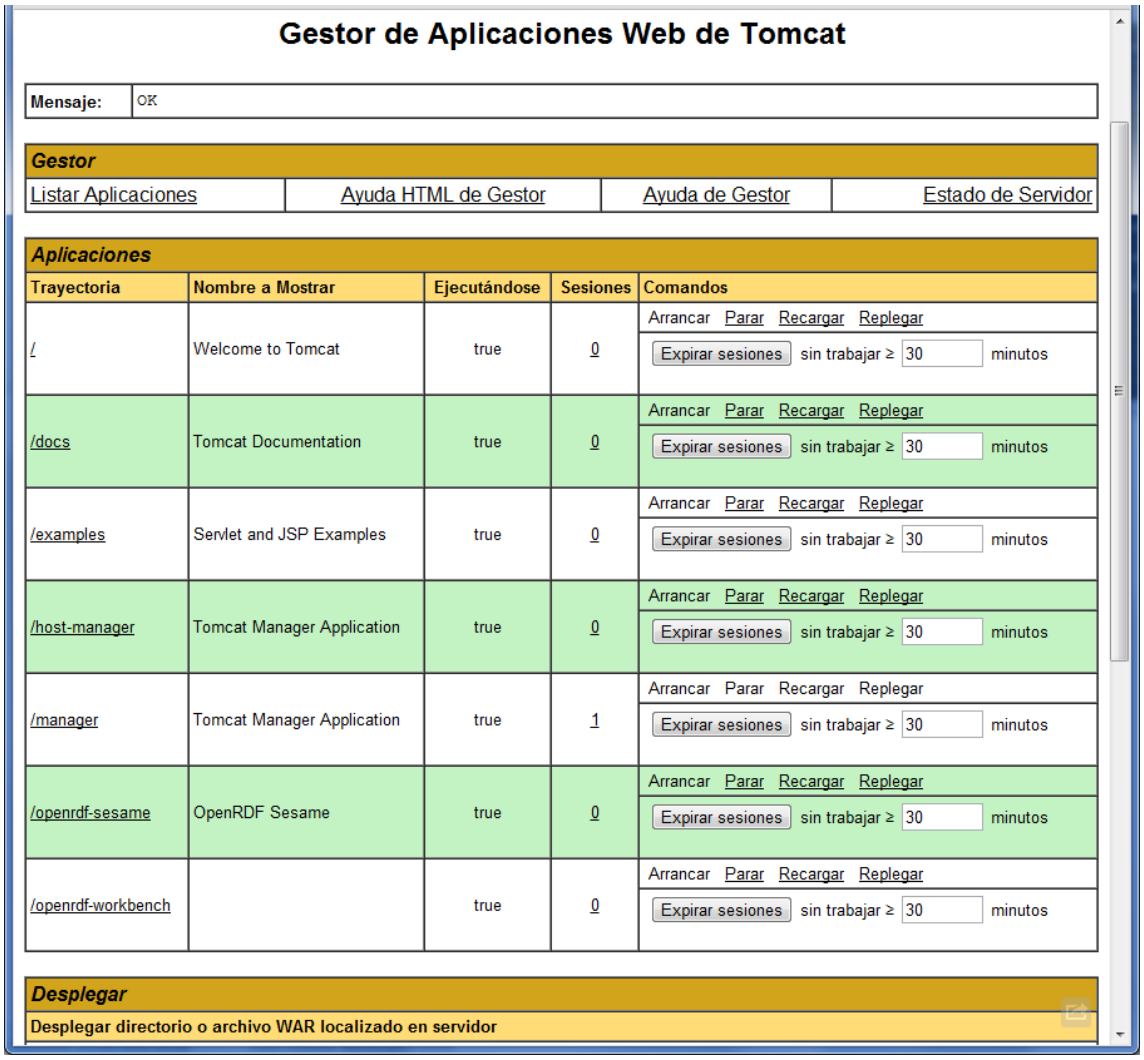
Tenemos varias formas de poder usar e instalar OWLIM, nosotros vamos a hacerlo junto con Sesame, que es un framework muy completo para gestión semántica y que ya hemos comentado anteriormente.

Lo que vamos a hacer es instalar Sesame de tal forma que utilice OWLIM para el almacenamiento de la información en lugar de usar su propio almacén RDF.

En el ZIP de OWLIM ya vienen dos paquetes .WAR (fichero de instalación de aplicaciones web para Tomcat) que instalan Sesame con OWLIM como almacén y el interfaz gráfico de Sesame para la gestión de bases de datos.

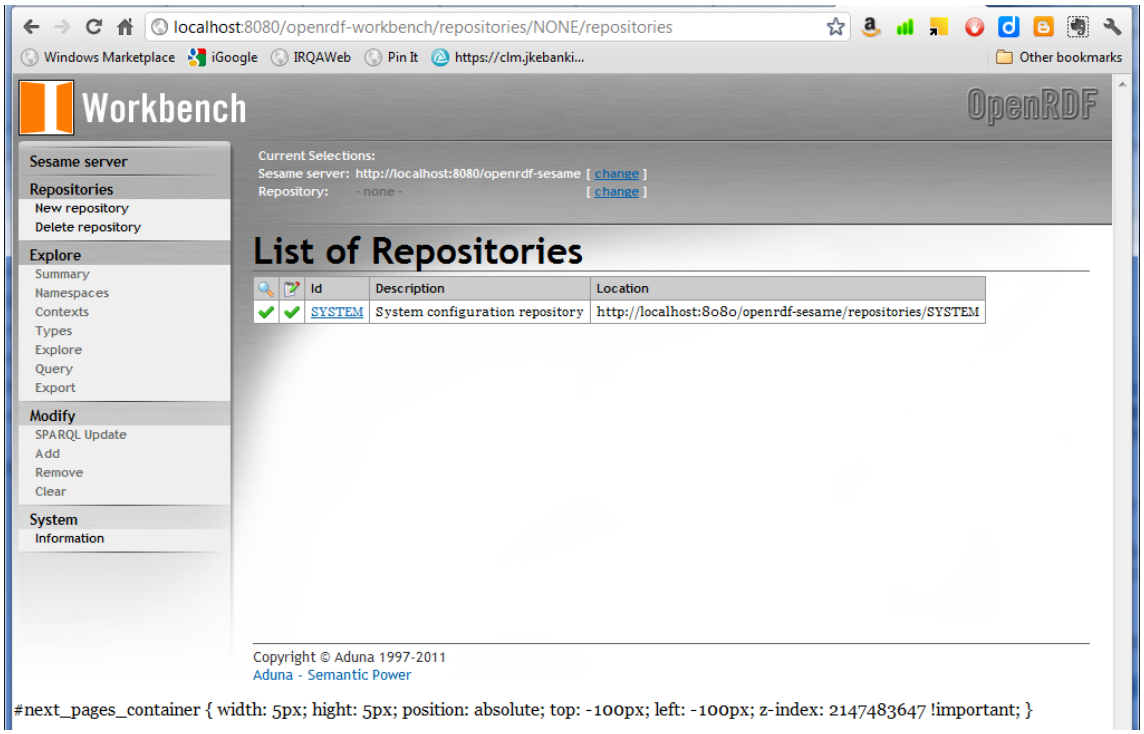
Estos dos ficheros se llaman `openrdf-sesame.war` y `openrdf-workbench.war` y se encuentran en el directorio `sesame_owlim` que aparece marcado en la imagen anterior.

Solo hay que copiar estos ficheros en el directorio `\weapps` de Tomcat y se produce el despliegue de la aplicación inmediatamente. Si todo ha ido bien, ya tenemos Sesame y OWLIM instalado y ahora el listado de aplicaciones del Tomcat Manager tiene estas dos nuevas entradas:



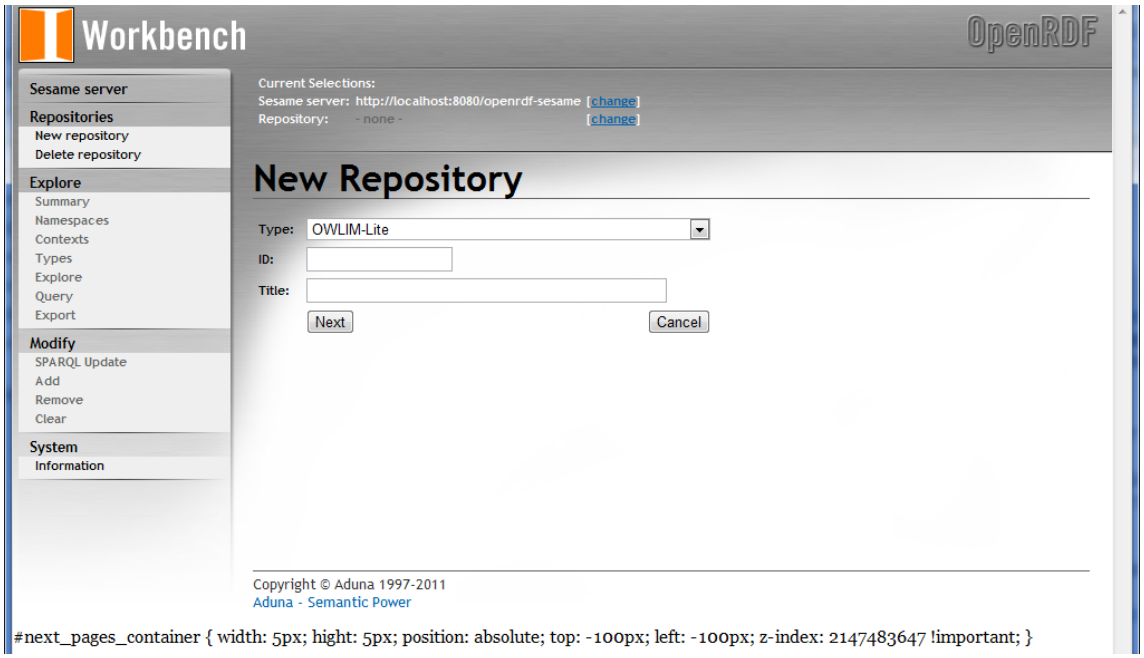
Pantalla de Tomcat Manager con Sesame instalado

Ahora podemos ir a la URL http://localhost:8080/openrdf-workbench y arrancar el Workbench de Sesame. En la información del servidor aparece la otra URL de la otra aplicación desplegada que es el servidor Sesame y por ahora no tenemos ningún repositorio.



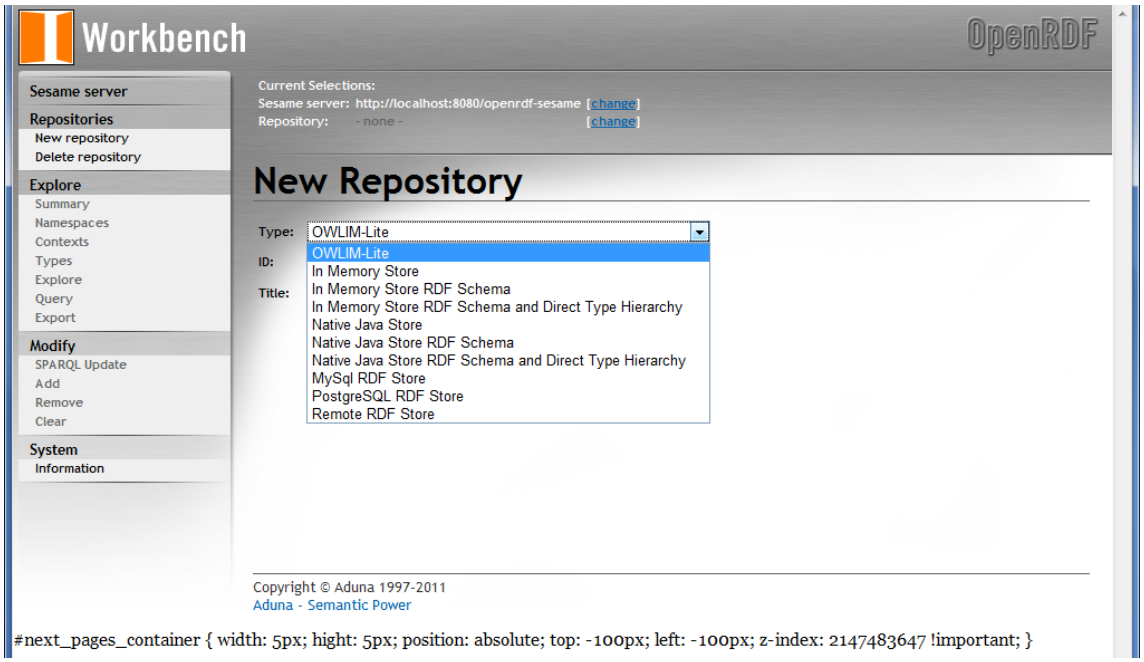
Centro de control de Sesame

Vamos a crear nuestro primer repositorio, para ellos pulsamos la opción “New repository” del menú de la izquierda y nos aparece la siguiente pantalla:



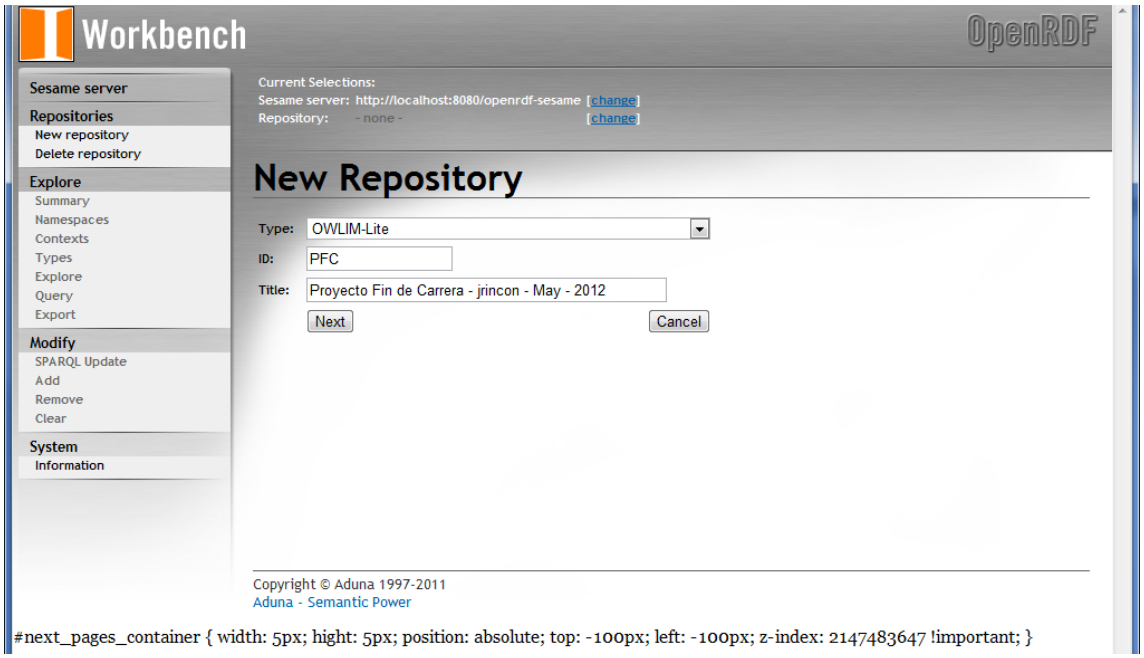
Pantalla para la creación de nuevos repositorios

Aunque la opción de OWLIM está seleccionada, si desplegamos las opciones de tipos de repositorios veremos las propias de Sesame a las que se ha añadido OWLIM. Es interesante ver que ofrece varias bases de datos relacionales como opción.



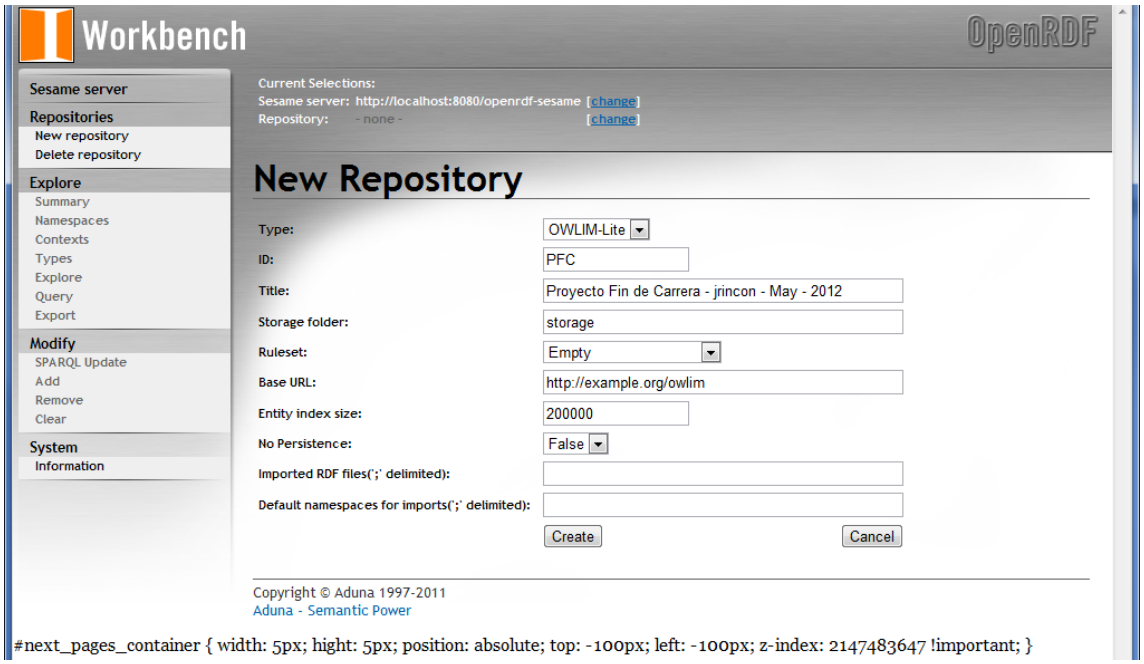
Tipos de repositorio con los que puede trabajar Sesame.

Introducimos los datos del nuevo repositorio y pulsamos “Next”.



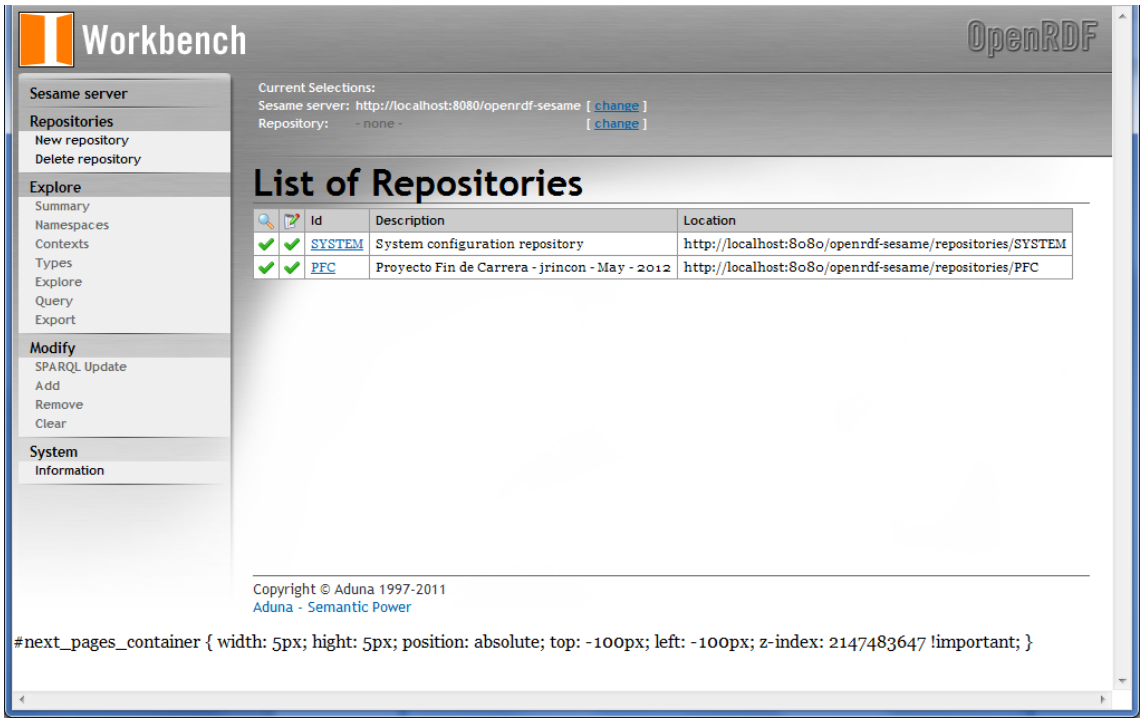
Datos del nuevo repositorio

Después de los datos iniciales, ahora pasamos a una pantalla en donde nos piden algunos detalles más del repositorio:



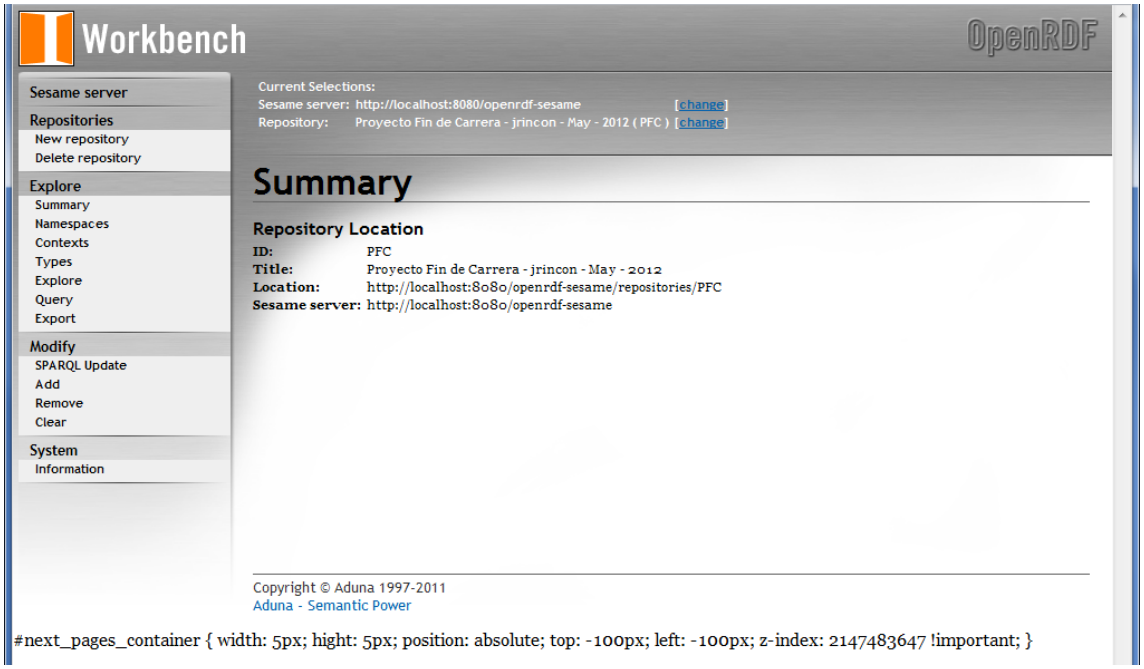
Segunda pantalla de creación de repositorios

En principio dejamos todos los valores ofrecidos por defecto. Pulsamos “Create” y ya tenemos nuestro primer repositorio creado.



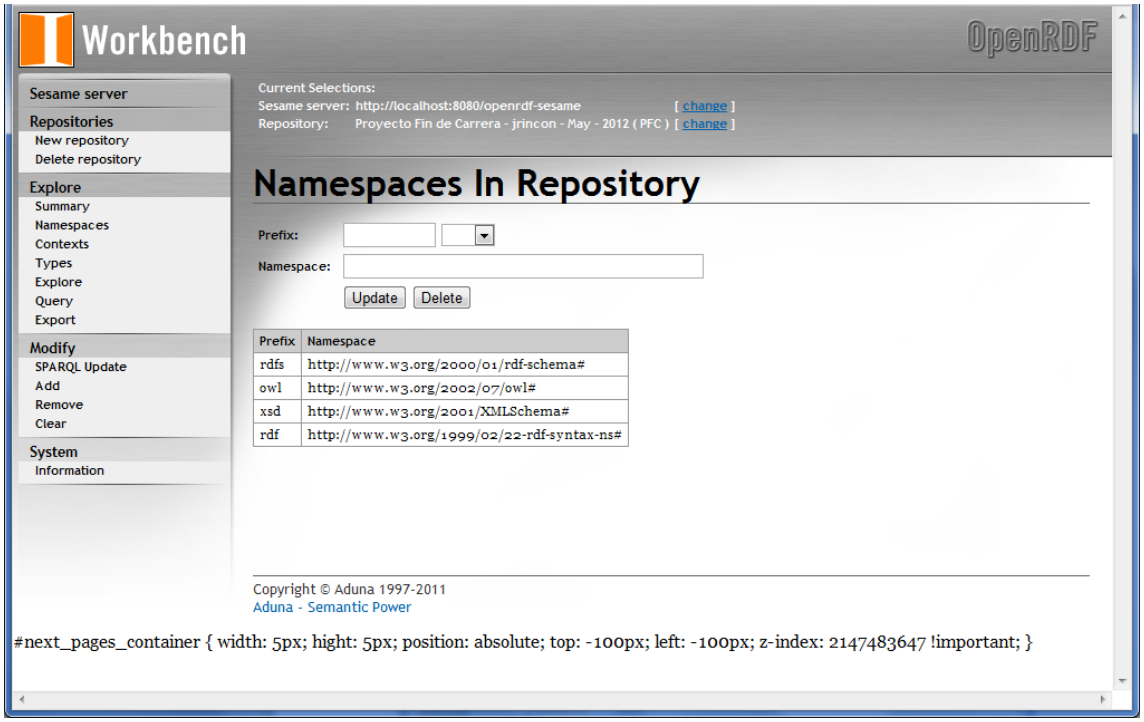
Listado de repositorios disponible

Si pulsamos el nuevo repositorio creado nos aparece una pantalla con un resumen de la información de ese repositorio:



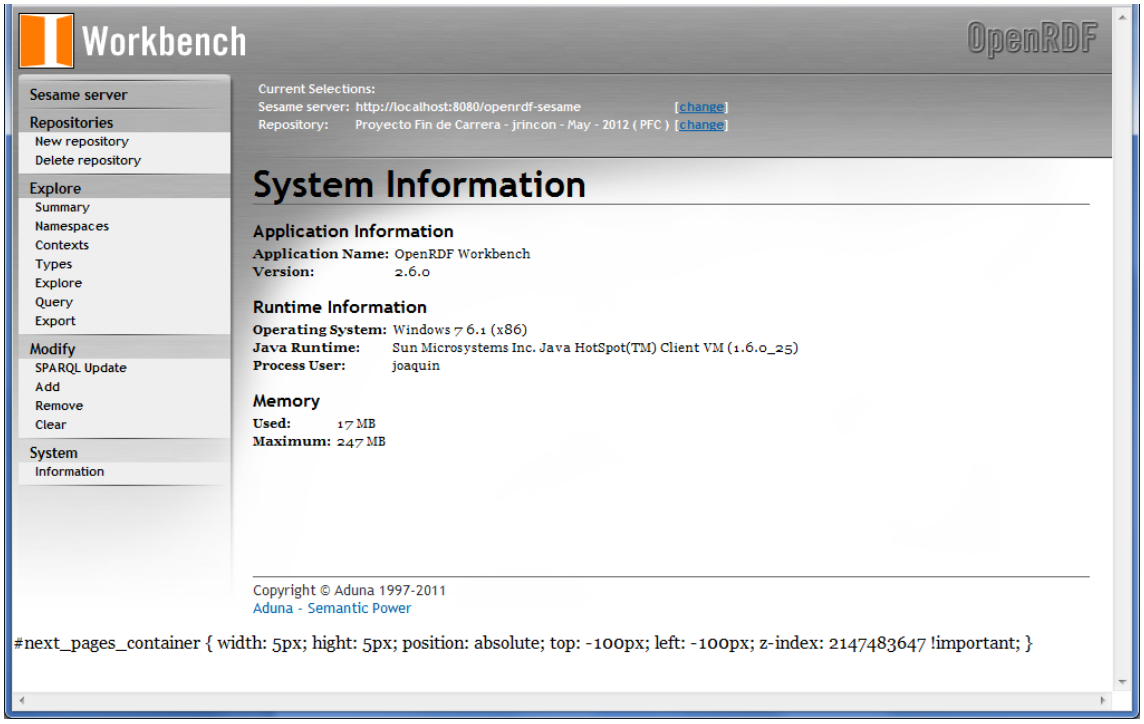
Resumen de la información de un repositorio

Si ahora pulsamos en la opción “Namespaces” (Vocabularios), vemos que por defecto se han añadido cuatro muy usados y ya se les han asignado prefijos.



Vocabularios disponibles y pantalla para añadir nuevos

Pulsando en System -> Information, podemos ver la versión del Workbench y alguna otra información del sistema, como la memoria disponible y la utilizada.



Información del sistema dada por el Workbench de Sesame

Con este proceso hemos instalado Tomcat, hemos instalado OWLIM y hemos creado nuestro primer repositorio.

La instalación de Java la he obviado, pero sólo es necesario instalar un JRE (Java Runtime Environment) que es muy sencillo y que tiene un paquete de instalación automatizado.

c. Como se usa

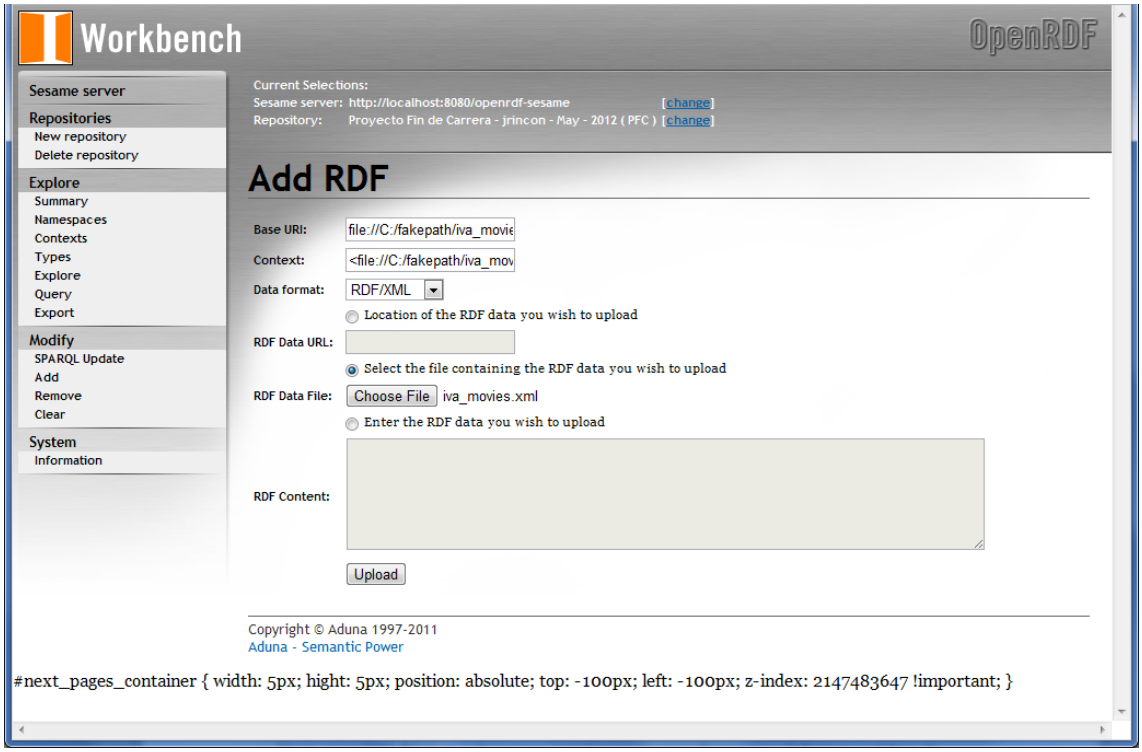
Sin duda la forma más fácil de usar OWLIM es a través del Workbench de Sesame. Sesame es uno de los pocos productos que además de ofrecer una librería y un sistema de almacenamiento ofrece además este GUI interactivo con el que poder administrar las bases de datos y poder ejecutar acciones, ya sean consultas o actualizaciones.

También veremos muy por encima como usar OWLIM desde Java

Sesame Workbench

Ya hemos visto al final de la instalación algunas pantallas del Sesame Workbench. Vamos a ver ahora más en detalle algunas de las funcionalidades que proporciona, extendiéndonos un poco más en la ejecución de consultas SPARQL.

Lo primero que tenemos que hacer es añadir datos a nuestro repositorio. Esto se hace pulsando el menú “Add” que aparece a la izquierda y tendremos la siguiente pantalla:

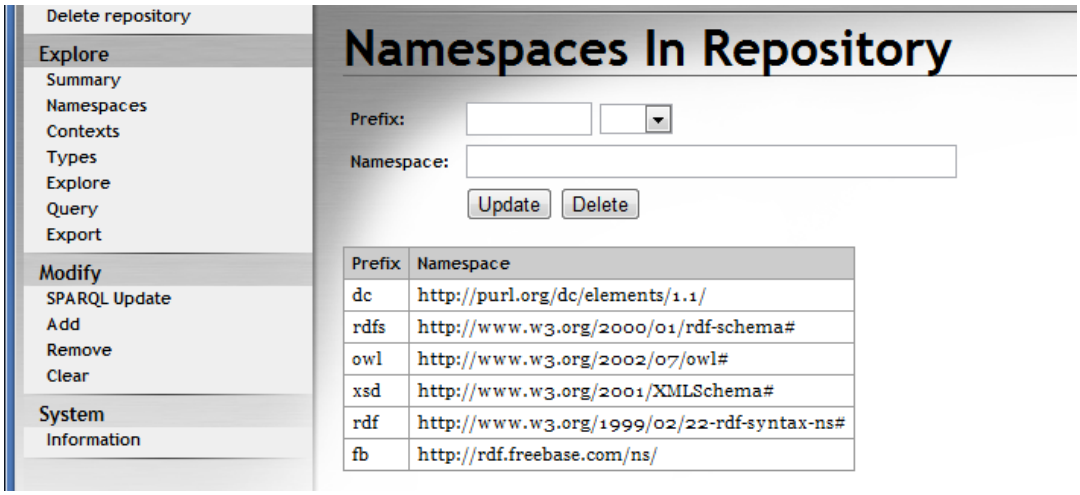


Vemos que desde esta pantalla podemos elegir distintas formas de añadir datos a nuestro repositorio: Desde una URL, desde un fichero o copiando un conjunto de tripletes directamente en la caja de texto que nos proporcionan. También tenemos que definir en que formato está la información que vamos a leer; Sesame es capaz de leer TriG, TriX, N-Triples, N3, RDF/XML, BinaryRDF y Turtle.

En nuestro caso vamos a leer un fichero en formato RDF/XML llamado `iva_movies.xml`[21] y que tiene información de películas. Pulsamos el botón “Upload” y el fichero se carga de forma inmediata. Este fichero tiene unas 2400 líneas, aunque el número de tripletes será menor (algo más de 1000), ya que algunos ocupan varias líneas del fichero. En este caso el tiempo de carga de este fichero es no perceptible.

Ahora podemos ver que se han añadido nuevos espacios de nombres al repositorio:

[21] Ese fichero está disponible aquí: http://semprog.com/psw/chapter7/iva_movies.xml, como parte de los ejemplos del libro mencionado en la nota (x)



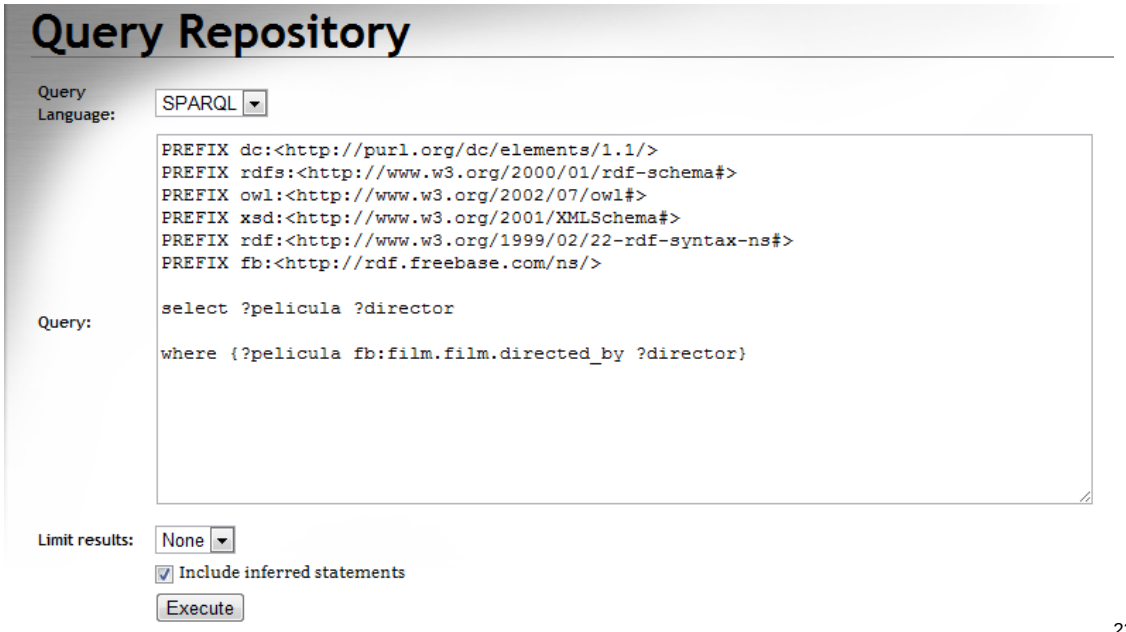
Y podemos empezar a ejecutar consultas usando SPARQL directamente.

Por ejemplo, vamos a sacar un listado de todas las películas y sus directores (solo saldrán aquellas de las que sabemos el director). La consulta SPARQL será algo así:

```
SELECT ?pelicula ?director

WHERE {?pelicula fb:film.film.directed_by ?director}
```

La relación `fb:film.film.directed_by` es la que establece el vocabulario de Freebase para definir la relación entre una película y su director.

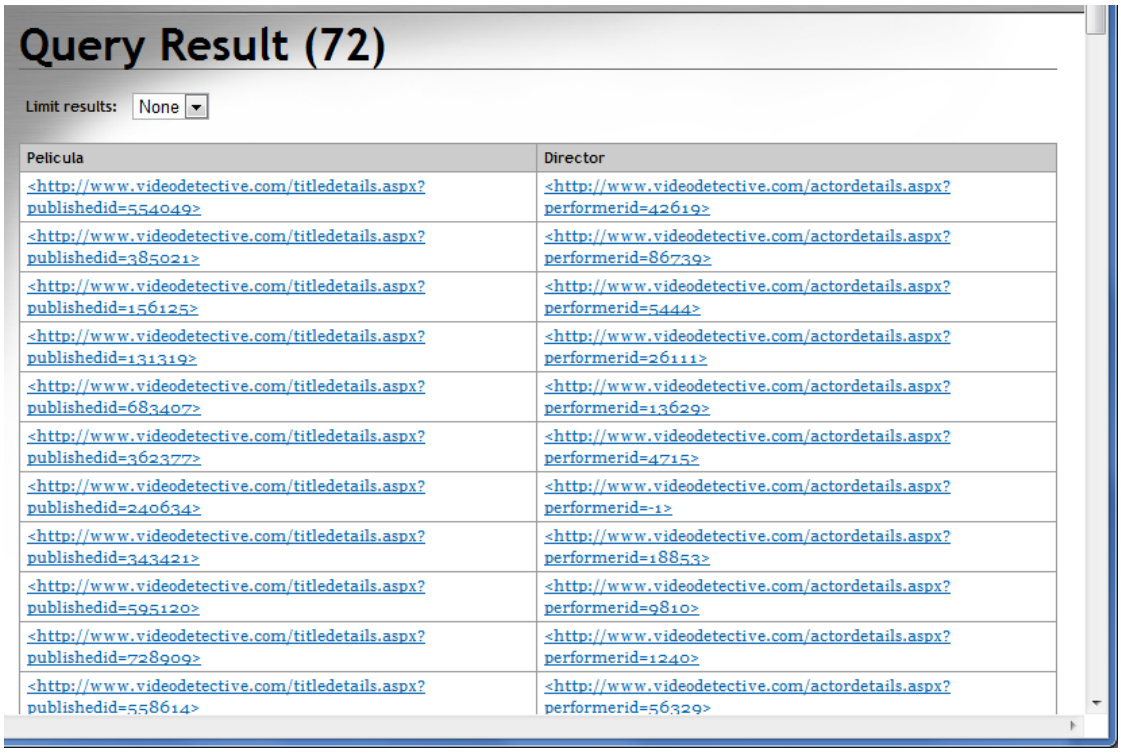
Se ejecuta en la siguiente pantalla, donde se puede ver que Sesame nos añade automáticamente todos los espacios de nombres a la consulta:



22

[22] Algunas de las capturas de pantalla están recortadas para centrarnos en un parte específica y mejorar la visualización.

El resultado es el siguiente:



Query Result (72)

Limit results: None

Pelicula	Director
<http://www.videodetective.com/titledetails.aspx?publishedid=554049>	<http://www.videodetective.com/actordetails.aspx?performerid=42619>
<http://www.videodetective.com/titledetails.aspx?publishedid=385021>	<http://www.videodetective.com/actordetails.aspx?performerid=86739>
<http://www.videodetective.com/titledetails.aspx?publishedid=156125>	<http://www.videodetective.com/actordetails.aspx?performerid=5444>
<http://www.videodetective.com/titledetails.aspx?publishedid=131319>	<http://www.videodetective.com/actordetails.aspx?performerid=26111>
<http://www.videodetective.com/titledetails.aspx?publishedid=683407>	<http://www.videodetective.com/actordetails.aspx?performerid=13629>
<http://www.videodetective.com/titledetails.aspx?publishedid=362377>	<http://www.videodetective.com/actordetails.aspx?performerid=4715>
<http://www.videodetective.com/titledetails.aspx?publishedid=240634>	<http://www.videodetective.com/actordetails.aspx?performerid=-1>
<http://www.videodetective.com/titledetails.aspx?publishedid=343421>	<http://www.videodetective.com/actordetails.aspx?performerid=18853>
<http://www.videodetective.com/titledetails.aspx?publishedid=595120>	<http://www.videodetective.com/actordetails.aspx?performerid=9810>
<http://www.videodetective.com/titledetails.aspx?publishedid=728909>	<http://www.videodetective.com/actordetails.aspx?performerid=1240>
<http://www.videodetective.com/titledetails.aspx?publishedid=558614>	<http://www.videodetective.com/actordetails.aspx?performerid=56329>

Que no es exactamente lo que estamos buscando. La consulta tal y como la hemos ejecutado nos muestra las URIs de las películas y de los directores, pero no sus nombres. Para conseguir un resultado legible tenemos que unir estas entidades con su propiedad Title (Title aplica también para asignar nombre a una persona en el diccionario “dc”), con lo que tendremos la siguiente consulta:

```
SELECT ?pelicula ?director

WHERE {?pelicula_aux fb:film.film.directed_by ?director_aux .

       ?pelicula_aux dc:title ?pelicula .

       ?director_aux dc:title ?director . }
```

En este caso el resultado es el esperado y como vemos el número de resultados (72) es el mismo que en el caso anterior:

Query Result (72)

Limit results: None

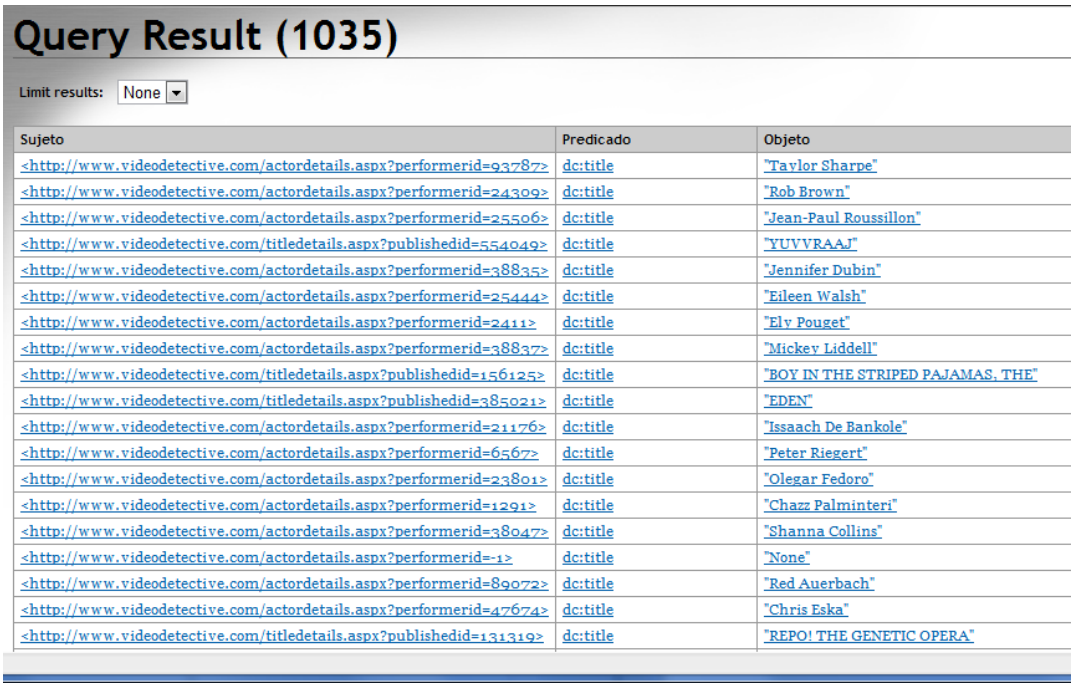
Pelicula	Director
"YUVVRAAJ"	"Subhash Ghai"
"EDEN"	"Declan Recks"
"BOY IN THE STRIPED PAJAMAS, THE"	"Mark Herman"
"REPO! THE GENETIC OPERA"	"Darren Lynn Bousman"
"SOUL MEN"	"Malcom D. Lee"
"HIGH SCHOOL MUSICAL 3: SENIOR YEAR"	"Kenny Ortega"
"UWE SCHOLZ: LE SACRE DU PRINTEMPS"	"None"
"MAX PAYNE"	"John Moore"
"EXPRESS, THE"	"Gary Fleder"
"SYNECDOCHE, NEW YORK"	"Charles Kaufman"
"DEAR ZACHARY"	"Kurt Kuenne"
"HOUSE"	"Robby Henson"
"WHO DOES SHE THINK SHE IS"	"Pamela Tanner Boll"
"TALENTO DE BARRIO"	"Jose Ivan Santiago"
"SEX DRIVE"	"Sean Anders"
"POLAR OPPOSITES"	"Fred Olen Ray"
"HEROES"	"Samir Karnik"
"QUARANTINE"	"John Erick Dowdle"
"CITY OF EMBER"	"Gil Kenan"

El tiempo de ejecución de esta consulta está por debajo de un segundo, siendo difícil precisar más con un cronómetro usado por un humano.

Con lo que hemos vistos, sabemos que la siguiente consulta nos devolverá el conjunto completo de tripletes que tenemos en el repositorio:

```
SELECT ?sujeto ?predicado ?objeto

WHERE { ?sujeto ?predicado ?objeto }
```

Vemos que son 1035 resultados que en esta ocasión han tardado entre tres y cuatro segundos en aparecer.



Query Result (1035)

Limit results: None

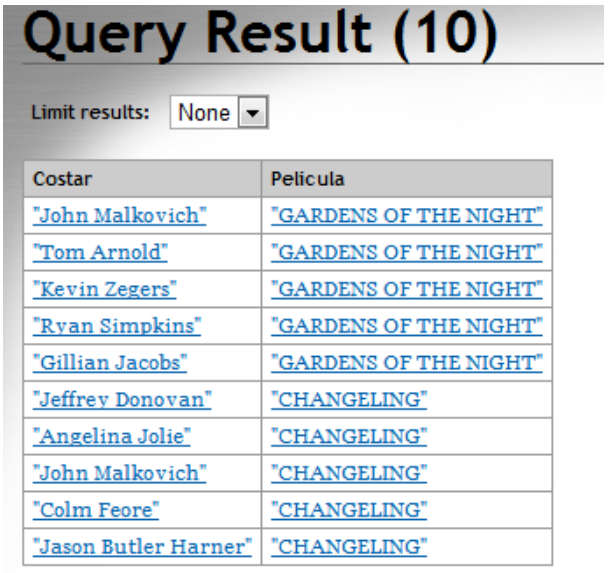
Sujeto	Predicado	Objeto
<http://www.videodetective.com/actordetails.aspx?performerid=93787>	dc:title	"Taylor Sharpe"
<http://www.videodetective.com/actordetails.aspx?performerid=24309>	dc:title	"Rob Brown"
<http://www.videodetective.com/actordetails.aspx?performerid=25506>	dc:title	"Jean-Paul Roussillon"
<http://www.videodetective.com/titledetails.aspx?publishedid=554049>	dc:title	"YUVVRAAJ"
<http://www.videodetective.com/actordetails.aspx?performerid=38835>	dc:title	"Jennifer Dubin"
<http://www.videodetective.com/actordetails.aspx?performerid=25444>	dc:title	"Eileen Walsh"
<http://www.videodetective.com/actordetails.aspx?performerid=2411>	dc:title	"Elv Pouget"
<http://www.videodetective.com/actordetails.aspx?performerid=38837>	dc:title	"Mickey Liddell"
<http://www.videodetective.com/titledetails.aspx?publishedid=156125>	dc:title	"BOY IN THE STRIPED PAJAMAS, THE"
<http://www.videodetective.com/titledetails.aspx?publishedid=385021>	dc:title	"EDEN"
<http://www.videodetective.com/actordetails.aspx?performerid=21176>	dc:title	"Issaach De Bankole"
<http://www.videodetective.com/actordetails.aspx?performerid=6567>	dc:title	"Peter Riegert"
<http://www.videodetective.com/actordetails.aspx?performerid=23801>	dc:title	"Olegar Fedoro"
<http://www.videodetective.com/actordetails.aspx?performerid=1291>	dc:title	"Chazz Palminteri"
<http://www.videodetective.com/actordetails.aspx?performerid=38047>	dc:title	"Shanna Collins"
<http://www.videodetective.com/actordetails.aspx?performerid=-1>	dc:title	"None"
<http://www.videodetective.com/actordetails.aspx?performerid=89072>	dc:title	"Red Auerbach"
<http://www.videodetective.com/actordetails.aspx?performerid=47674>	dc:title	"Chris Eska"
<http://www.videodetective.com/titledetails.aspx?publishedid=131319>	dc:title	"REPO! THE GENETIC OPERA"

Ahora vamos a ejecutar una consulta que tenga una lógica más compleja. La cojo prestada de este libro ya mencionado[xxvii].

Se trata de ver aquellos actores que han actuado en alguna película con John Malkovich. La consulta sería esta:

```
SELECT ?costar ?Pelicula

WHERE {?film fb:film.film.performances ?p1 .

      ?film dc:title ?Pelicula .

      ?p1 fb:film.performance.actor ?a1 .

      ?a1 dc:title "John Malkovich".

      ?film fb:film.film.performances ?p2 .

      ?p2 fb:film.performance.actor ?a2 .

      ?a2 dc:title ?costar .}
```

El resultado sería:

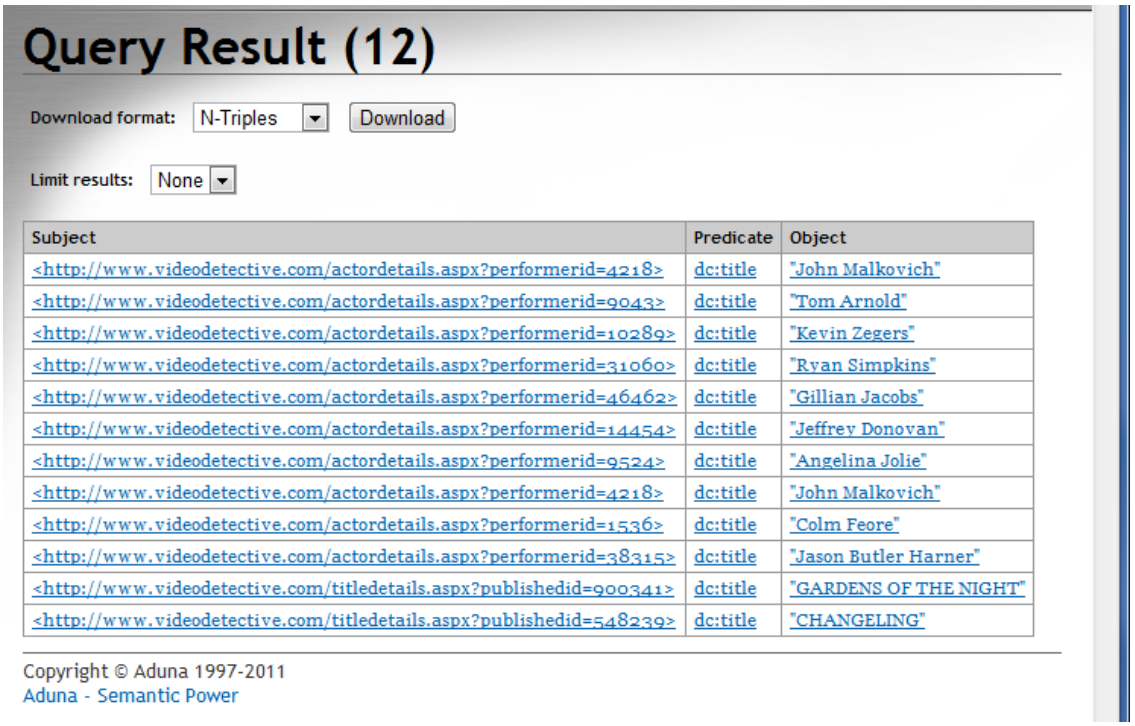


Query Result (10)

Limit results: None

Costar	Pelicula
"John Malkovich"	"GARDENS OF THE NIGHT"
"Tom Arnold"	"GARDENS OF THE NIGHT"
"Kevin Zegers"	"GARDENS OF THE NIGHT"
"Ryan Simpkins"	"GARDENS OF THE NIGHT"
"Gillian Jacobs"	"GARDENS OF THE NIGHT"
"Jeffrey Donovan"	"CHANGELING"
"Angelina Jolie"	"CHANGELING"
"John Malkovich"	"CHANGELING"
"Colm Feore"	"CHANGELING"
"Jason Butler Harner"	"CHANGELING"

Quiero utilizar esta consulta para mostrar el resultado que obtendríamos si ejecutamos la misma consulta usando la sentencia DESCRIBE en lugar de SELECT.



Query Result (12)

Download format: N-Triples Download

Limit results: None

Subject	Predicate	Object
<http://www.videodetective.com/actordetails.aspx?performerid=4218>	dc:title	"John Malkovich"
<http://www.videodetective.com/actordetails.aspx?performerid=9043>	dc:title	"Tom Arnold"
<http://www.videodetective.com/actordetails.aspx?performerid=10289>	dc:title	"Kevin Zegers"
<http://www.videodetective.com/actordetails.aspx?performerid=31060>	dc:title	"Ryan Simpkins"
<http://www.videodetective.com/actordetails.aspx?performerid=46462>	dc:title	"Gillian Jacobs"
<http://www.videodetective.com/actordetails.aspx?performerid=14454>	dc:title	"Jeffrey Donovan"
<http://www.videodetective.com/actordetails.aspx?performerid=9524>	dc:title	"Angelina Jolie"
<http://www.videodetective.com/actordetails.aspx?performerid=4218>	dc:title	"John Malkovich"
<http://www.videodetective.com/actordetails.aspx?performerid=1536>	dc:title	"Colm Feore"
<http://www.videodetective.com/actordetails.aspx?performerid=38315>	dc:title	"Jason Butler Harner"
<http://www.videodetective.com/titledetails.aspx?publishedid=900341>	dc:title	"GARDENS OF THE NIGHT"
<http://www.videodetective.com/titledetails.aspx?publishedid=548239>	dc:title	"CHANGELING"

Copyright © Aduna 1997-2011
Aduna - Semantic Power

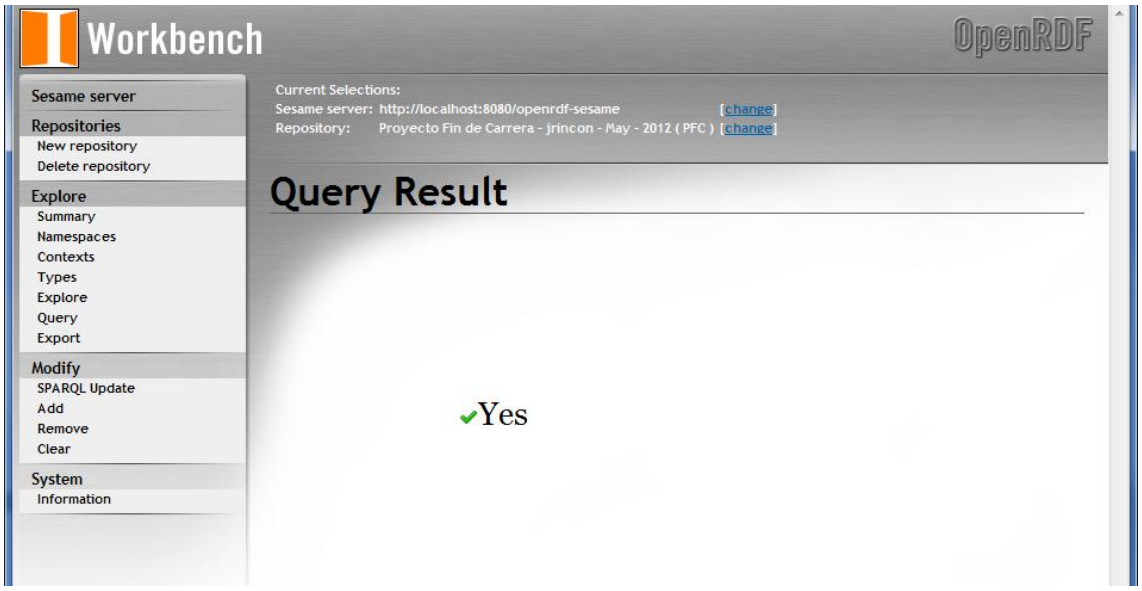
Lo que obtenemos es información relacionada con los 12 elementos que aparecían en la consulta inicial. Es interesante ver que la información aparece en forma de tripletes y que se puede exportar.

Si hacemos lo mismo con la consulta que nos daba las películas y sus directores, obtendremos 144 resultados, dando información de las 72 películas y de los 72 directores. Como ya se comentó antes, cada motor puede decidir como implementa la sentencia DESCRIBE.

Por último, vamos a ver algún ejemplo con la sentencia ASK. Por ejemplo, la siguiente consulta comprueba si “John Malkovich” actúa en la película “CHANGELING”

```
ASK

WHERE {?film fb:film.film.performances ?p1 .

      ?p1 fb:film.performance.actor ?a1 .

      ?a1 dc:title "John Malkovich".

      ?film dc:title "CHANGELING" .}
```

Y el resultado es:



Si cambiamos alguna letra del título de la película o del nombre del actor, la respuesta será “NO”, ya que no encontrará los literales exactos.

Si en la consulta que nos daba la lista de actores que han actuado con John Malkovich en alguna película cambiamos la sentencia SELECT por ASK, lo que obtenemos es una respuesta (afirmativa) a la pregunta de si alguien ha actuado con él.

Si ejecutamos la siguiente consulta:

```
ASK

WHERE {?film fb:film.film.performances ?p1 .

      ?p1 fb:film.performance.actor ?a1 .

      ?a1 dc:title "John Malkovich".}
```

Sólo estaremos preguntando si hay alguna película en la que actúe John Malkovich y obtendremos una respuesta afirmativa.

El poder ejecutar estas consultas es una ayuda muy buena para comprobar que la sintaxis es correcta antes de ponerlas en una aplicación y para comprobar que devuelven el tipo de resultados esperado.

En los resultados de las consultas, se pueden seguir los enlaces a los distintos recursos.

Aunque el Workbench es una herramienta útil y está muy por encima de la mayoría de las herramientas disponibles en los productos semánticos todavía tiene que mejorar la usabilidad, por ejemplo, cada vez que se ejecuta una consulta, el texto de la misma desaparece y hay que volver a escribirlo todo si se quieren hacer pequeñas modificaciones y ejecutarla de nuevo.

Java

Para usar OWLIM lo que tenemos que hacer es usar Sesame y definir que el tipo de repositorio es OWLIM en el momento en que se crea. Aparte de eso, todo el código puede ser igual a cualquier aplicación que use Sesame.

Aquí reproduzco un esqueleto de una aplicación Java que usa la librería Sesame y que define el tipo de repositorio como de tipo OWLIM.

```
//Estos son los import más comunes para usar Sesame
import org.openrdf.query.*;
import org.openrdf.model.vocabulary.*;
import org.openrdf.repository.*;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.inferencer.fc.ForwardChainingRDFSInferencer;
import org.openrdf.sail.memory.MemoryStore;
import org.openrdf.rio.*;
import org.openrdf.model.*;

import java.net.URL;
import java.net.URLConnection;
import java.util.*;
import java.io.*;

//... Definimos la clase Graph

       Graph graph = ...;
       Resource repositoryNode = ...;

       // Creamos un gestor repositorios locales
       RepositoryManager repositoryManager =
           new LocalRepositoryManager(new File("."));
       repositoryManager.initialize();


       // Creamos un objeto de configuración a partir de la configuración del
       // objeto "graph" y lo añadimos al gestor de repositorios
       RepositoryConfig repositoryConfig =
           RepositoryConfig.create(graph, repositoryNode);
       repositoryManager.addRepositoryConfig(repositoryConfig);

       // Instanciamos el repositorio que vamos a usar. Aquí definimos que
       // es de tipo OWLIM
```

```
        Repository repository = repositoryManager.getRepository("owlim");

        // Abrimos una conexión al repositorio
        RepositoryConnection repositoryConnection =
            repository.getConnection();

        // ... usamos el repositorio

        // Cerramos la conexión, el repositorio y el gestor
        repositoryConnection.close();
        repository.shutDown();
        repositoryManager.shutDown();

//Fin de la aplicación
```

d. Ventajas e inconvenientes

OWLIM es una herramienta razonablemente fácil de instalar, que está disponible para el entorno Windows y que con las versiones SE y Enterprise es capaz de ofrecer características de una herramienta profesional.

Su integración con Sesame y con otros frameworks hace que no sea necesario aprender nada específico de la solución para poder usarla si ya se conoce alguno de esos frameworks y Sesame es uno de los más extendidos.

El Workbench de Sesame es un entorno gráfico de trabajo con los repositorios que ofrece bastantes posibilidades y que ayuda a los desarrolladores, tanto en la creación de los repositorios, en su gestión y en las pruebas de consultas.

Aunque este entorno por si solo es un importante argumento en favor de OWLIM, aun le falta cuidar un poco más la experiencia de usuario para que sea incluso más cómoda de utilizar.

El Workbench es una herramienta de Sesame, no directamente una parte de OWLIM, pero Ontotext contribuye activamente a Sesame, que es un proyecto open source, y sería conveniente mejorar pequeños detalles.

Sesame es potente, fácil de instalar y es fácil trabajar con él[xxviii]. Se puede usar como una librería Java o se puede usar como un conjunto de servicios web. Además, si sustituimos su sistema de almacenamiento por defecto con OWLIM ganamos en escalabilidad y rendimiento.

La documentación, tanto de Sesame como de OWLIM es bastante completa, estando disponible online.

Los inconvenientes son los típicos de un área poco madura. Hay que tener un alto conocimiento técnico para poder instalar y usar estas herramientas. Hoy por hoy se mantienen en un nicho bastante cerrado.

e. Rendimiento

En mi instalación he intentado cargar colecciones de datos grandes, como todos los artículos (solo el título y algún dato más) de la Wikipedia que están disponibles en DBPedia [23], pero con ese fichero se producía un error de falta de memoria.

Ha sido difícil encontrar unas colecciones de datos[xxix] de tamaño adecuado para hacer pruebas. Por ejemplo, he cargado una serie de ficheros con datos de países, continentes, monedas y capitales, que en total son algo más de 15.000 tripletes.

La carga de datos se produce a una velocidad increíble, la carga de un fichero con 9330 tripletes en formato RDF/XML es prácticamente instantánea, por debajo de un segundo. Sin embargo la consulta que muestra todos los tripletes (`SELECT ?a ?b ?c WHERE{?a ?b ?c}`) para esos mismos 9.330 tarda 14 segundos. Con 15.257 nos vamos a 25 segundos. Una cosa interesante de estas consultas es que impactan a los cuatro cores (núcleos) de la máquina de pruebas, lo que demuestra buena capacidad de OWLIM para paralelizar el procesamiento.

Una cosa interesante de la carga de datos, es que añadir nuevos datos no requiere nada, tan solo ir a la pantalla de añadir y elegir que otro fichero queremos incluir. No hay mapeos de campos, ni ninguna otra configuración que gestionar. De esa forma al fichero inicial de países le puedo añadir muy rápidamente el de monedas, capitales, etc.

Aunque estas operaciones sirven para tener ciertas sensaciones sobre que va rápido y que no en OWLIM no sirven para hacer un conjunto de pruebas serio de la herramienta. Usaremos la información de terceros disponible en internet para estudiar la cuestión de rendimiento.

Ontotext afirma en su web que OWLIM es el producto con mejor rendimiento del mercado[xxx] y para ello hace referencia a un conjunto de pruebas realizadas por terceros y por ellos mismos.

Ya vimos en el trabajo de la referencia (xxv) que OWLIM no queda el primero, aunque muestre buenos resultados y en todos los casos ofrezca resultados comparables a los de las otras herramientas del estudio.

Por supuesto esta situación es muy volátil, ya que se crean nuevas versiones con mejores capacidades muy rápidamente.

Es también muy interesante un trabajo de la Freie Universität de Berlín[xxxi], universidad que tiene varios artículos sobre benchmarks de bases de datos semánticas. En este trabajo, que es bastante reciente (Febrero de 2011) se hace una comparativa y OWLIM está entre las herramientas elegidas.

OWLIM es claramente el mejor en tiempo de carga de información (lo que coincide con mi propia limitada experiencia), pero en determinados tipos de consultas no es así. De

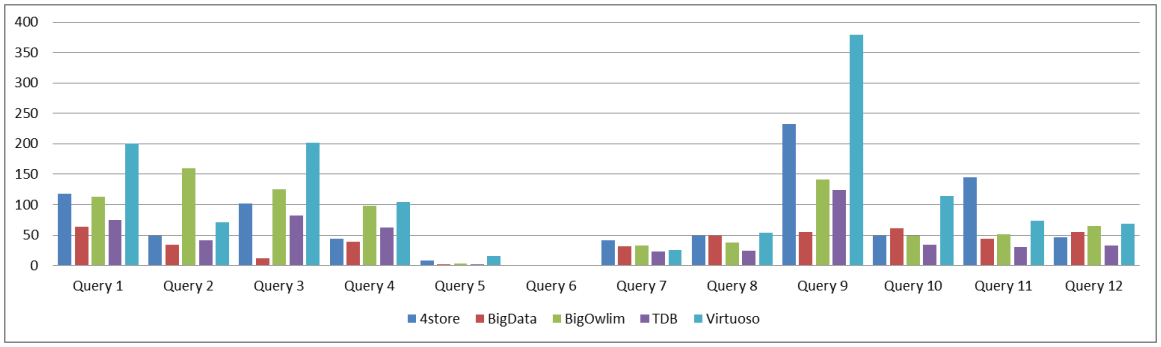
[23] http://dbpedia.org/About

hecho solo es el mejor en un tipo de consulta de doce tipos probados. Aunque es cierto que no es el peor en ninguna de las 12.

Estas tablas están extraídas de ese trabajo, los número mayores son mejores ya que indican número de consulta por unidad de tiempo. Tenemos una tabla para una base de datos con 100 millones de tripletes y otra con 200 millones. Los valores en negrita son los mejores de cada tipo de consulta.

100m

	4store	BigData	BigOwlim	TDB	Virtuoso
Query 1	117.6	64.2	112.5	75.1	**200.7**
Query 2	49.0	33.6	**159.3**	41.0	71.1
Query 3	102.4	12.4	125.0	82.2	201.4
Query 4	43.4	38.4	97.9	62.1	**103.9**
Query 5	7.8	2.3	3.0	2.0	**15.2**
Query 6	not executed	not executed	not executed	not executed	not executed
Query 7	**41.3**	31.3	32.6	22.6	24.9
Query 8	49.1	48.5	38.0	24.4	**54.0**
Query 9	233.0	54.8	141.8	124.6	**379.1**
Query 10	49.2	61.6	48.5	33.5	**113.7**
Query 11	**145.3**	43.8	51.3	30.0	73.6
Query 12	46.5	54.8	65.4	33.3	**68.0**



200m

	4store	BigData	BigOwlim	TDB	Virtuoso
Query 1	145.3	64.4	45.0	62.2	**163.0**
Query 2	55.7	35.3	**111.6**	44.1	73.8
Query 3	122.9	14.4	48.1	66.1	**195.4**
Query 4	62.9	40.0	37.8	47.3	**94.8**
Query 5	5.0	1.7	1.8	1.2	**9.3**
Query 6	not executed	not executed	not executed	not executed	not executed
Query 7	**49.3**	21.9	11.0	15.0	15.4

Query 8	**57.1**	14.1	12.6	15.9	22.5
Query 9	117.3	44.6	67.9	97.1	**160.1**
Query 10	52.8	28.8	22.4	26.4	**69.9**
Query 11	33.3	26.7	18.7	23.4	**39.5**
Query 12	**40.4**	31.1	29.1	28.3	39.8

Tal y como ocurría en el trabajo de [Patchigolla][xxxii], Virtuoso es el que tiene mejor comportamiento en general, por lo que la afirmación de la web de OWLIM parece un poco exagerada y responde más a una estrategia de marketing que a una realidad.

Estos datos demuestran que con OWLIM se puede trabajar con conjuntos de datos muy grandes. Estas pruebas están hechas con 200 millones de tripletes, pero afirman haber llegado a los 12.000 millones, en cualquier caso cantidades de datos muy considerables.

Independientemente de si es la base de datos semántica más rápida del mercado, lo que si podemos afirmar es que está a la altura de las mejores y que es una opción razonable para proyectos profesionales de grandes dimensiones.

Un comentario general sobre rendimiento, no específicamente relacionado con OWLIM, es que en varias de las comparativas que he visto se utilizan frameworks como Sesame o Jena, pero para la capa de persistencias se utilizan bases de datos relacionales tradicionales. En todos los casos estas soluciones ofrecen peor rendimiento que los productos nativos, como AllegroGraph, Virtuoso, OWLIM o el propio Sesame con su sistema de almacenamiento propio.

Está claro que estos productos están optimizados para trabajar con la información representada como tripletes y son superiores al uso generalista de las bases de datos relacionales disponibles actualmente.

f. Ejemplos de proyectos

En la web de Ontotext se pueden ver referencias a varios proyectos importantes en los que OWLIM es la herramienta elegida[xxxiii], entre ellos destaca que va a ser la base de datos que la BBC va a usar en su web[24] de cobertura de los juegos olímpicos de 2012. Es muy interesante el siguiente artículo del arquitecto de la BBC Jem Rayfield[xxxiv].

La web http://linkedlifedata.com consolida más de 25 bases de datos médicas y utiliza OWLIM para ello. Tiene al menos 10.052.200 tripletes (resultado de ejecutar una consulta global al repositorio).

[24] http://www.bbc.co.uk/sport/0/olympics/2012/

También se integra con alguna otra plataforma y herramientas como Gate[25] (data mining) y KIM[26], aunque en el caso de esta última, no es extraño ya que es otro producto de Ontotext.

[25] http://gate.ac.uk/
[26] http://www.ontotext.com/kim

4. Conclusiones

El área de las tecnologías semánticas y específicamente el de las bases de datos es todavía bastante nueva.

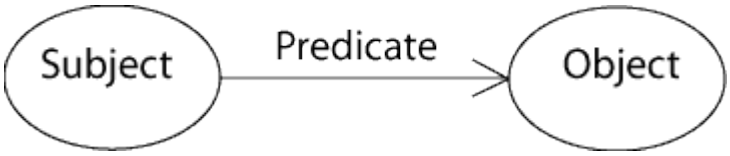
La inmadurez se refleja en la falta de oferta de productos comerciales serios y maduros, aunque hay algunos y podríamos considerar a OWLIM uno de ellos.

La mayoría de los productos existentes adolecen de una aproximación desde el I+D y no desde el área comercial. Son productos para expertos y no hacen fácil el acercamiento a usuarios esporádicos o poco técnicos.

Aunque la promesa de poder separar definitivamente la aplicación de los datos y permitir que evolucionen por separado tiene un valor enorme, se enfrenta al problema de la base instalada. Casi todas las aplicaciones que existen en la actualidad no se han diseñado para aprovechar las capacidades de un SGBD semántico y cambiarlas es un esfuerzo muy grande.

Este tipo de almacén de datos tiene sentido, sobre todo, para nuevas aplicaciones que desde el momento del diseño ya se piensan para que aprovechen las capacidades semánticas y esta separación entre aplicación y datos.

Es llamativo que desde un modelo tan simple como el de RDF, en el que tan solo tenemos tripletes, se pueda llegar a crear herramientas completas y almacenes de conocimiento sofisticados. Creo que en este caso aplica una de mis frases favoritas: “Simplicity is the ultimate sophistication” – (La simplicidad es la última sofisticación) – Atribuida a Leonardo Da Vinci. Desde algo tan sencillo se pueden crear construcciones que son superiores desde el punto de vista semántico a cualquier modelo de bases de datos actual.



Otra cosa muy llamativa es que un lenguaje como OWL, que sirve para definir el conocimiento de un área determinada y establecer las relaciones y las entidades básicas, se pueda expresar como un conjunto de tripletes, con lo que tenemos que **los modelos son datos**, que es otra de las características que le dan a este modelo su potencia.

Ya existen muchos ejemplos de aplicaciones que utilizan este tipo de tecnologías y, más específicamente, que usan bases de datos semánticas, pero siguen siendo testimoniales si se compara con las que usan bases de datos relacionales.

Habrá que esperar unos años para ver si las tecnologías semánticas explotan definitivamente y se usan de forma habitual en el desarrollo de aplicaciones de gestión y transaccionales y mientras tanto es muy interesante conocer estas herramientas para poder cubrir necesidades específicas.

Anexo 1.- Equipo de pruebas

Aunque las pruebas realizadas son informales y se dan solo para que los lectores tengan una aproximación, creo que es conveniente dar los detalles del equipo en que se ejecutaba OWLIM, Sesame y Tomcat

Equipo:	Dell XPS 420
Procesador:	Intel core Quad Q6600 a 2.40 GHz
Memoria:	3 GB
Sistema Operativo:	Windows 7 Home Premiun 64 bits
Software:	OWLIM-lite-4.3.4238
	Sesame 2.6.0
	Apache Tomcat Version 6.0.35
	Sun Microsystems Inc. Java HotSpot(TM) Client VM (1.6.0_25)

Anexo 2.- Bibliografía y referencias

Modelos de datos y contexto:

[i] Wikipedia – "*Data model*" - http://en.wikipedia.org/wiki/Data_model (a 1 de abril de 2012)

[ii] Wikipedia – "*Coceptual schema*" - http://en.wikipedia.org/wiki/Conceptual_schema (a 1 de abril de 2012)

[iii] Wikipedia – "*Entity-Relationship model*" - http://en.wikipedia.org/wiki/Entity-relationship_model (a 4 de abril de 2012)

[iv] Wikipedia – "*Modelo Entidad-Relación*" - http://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n (a 4 de abril de 2012)

[v] Wikipedia – "*Logical schema*" - http://en.wikipedia.org/wiki/Logical_schema (a 1 de abril de 2012)

[vi] Miguel, Adoración de; Piattini, Mario – "*Concepción y diseño de bases de datos. Del modelo E/R al modelo relacional*" – Madrid: RA-MA 1993. 989 p. ISBN: 84-7897-083-5. Páginas 425 y siguientes

Bases de datos semánticas:

[vii] W3C- Semantic Web Standards – "Main Page" - http://www.w3.org/2001/sw/wiki/Main_Page (a 12 de abril de 2012)

[viii] Wikipedia – "*Semantic Web*" - http://en.wikipedia.org/wiki/Semantic_Web (a 5 de abril de 2012)

[ix] W3C- W3C recommendation – "*Resource Description Framework (RDF): Concepts and Abstract Syntax*" - http://www.w3.org/TR/rdf-concepts/ (a 12 de abril de 2012)

W3C- W3C recommendation – "*RDF Primer*" - http://www.w3.org/TR/rdf-primer/ (a 12 de abril de 2012)

W3C- RDF Semantic Web Standards – "RDF" - http://www.w3.org/RDF/ (a 12 de abril de 2012)

Wikipedia – "*Resource Description Framework*" - http://en.wikipedia.org/wiki/Resource_Description_Framework (a 5 de abril de 2012)

[x] Segaran, Toby; Evans, Colin; Taylor, Jamie – "*Programming the semantic web*" – O'Reilly Media, Inc. 2009. 302 p. Ebook ISBN:978-0-596-80518-0. Posición 2016 y siguientes.

[xi] W3C – RDF Test cases – http://www.w3.org/TR/rdf-testcases/#ntriples (a 20 de mayo de 2012)

[xii] Berners-Lee, Tim – "Notation 3 logic" -2005 - http://www.w3.org/DesignIssues/Notation3 (a 20 de mayo de 2012)
W3C – "Primer: Getting into RDF & Semantic Web using N3"- http://www.w3.org/2000/10/swap/Primer (a 20 de mayo de 2012)

[xiii] W3C – "RDFa Primer" – http://www.w3.org/TR/xhtml-rdfa-primer/ (a 20 de mayo de 2012)

[xiv] FOAF –"The Friend of a Friend Project" - http://www.foaf-project.org/ (a 20 de mayo de 2012)
FOAF Vocabulary Specification 0.98 - http://xmlns.com/foaf/spec/ (a 20 de mayo de 2012)
Wikipedia – FOAF - http://en.wikipedia.org/wiki/FOAF_(software) (a 20 de mayo de 2012)

[xv]Wikipedia – Ontology - http://en.wikipedia.org/wiki/Ontology_(computer_science) (a 20 de mayo de 2012)

[xvi] Wikipedia – "Web Ontology Language" - http://en.wikipedia.org/wiki/Web_Ontology_Language (a 20 de mayo de 2012)
W3C – OWL 2 - http://www.w3.org/TR/owl2-overview/ (a 20 de mayo de 2012)

[xvii] W3C – "OWL Guide" - http://www.w3.org/TR/owl-guide/ (a 20 de mayo de 2012)

[xviii] Wikipedia – "Halting Problem" - http://en.wikipedia.org/wiki/Halting_problem (a 20 de mayo de 2012)
Wkipedia – "Problema de la parada" - http://es.wikipedia.org/wiki/Problema_de_la_parada (a 20 de mayo de 2012)

[xix] Wikipedia – "SPARQL" - http://en.wikipedia.org/wiki/SPARQL (a 20 mayo de 2012)
W3C – "SPARQL Query Language for RDF" - http://www.w3.org/TR/rdf-sparql-query/ (a 20 de mayo de 2012)
W3C – "SPARQL 1.1 Query Language" - http://www.w3.org/TR/sparql11-query/ (a 20 de mayo de 2012)

[xx] W3C – SPARQL 1.1 Update - http://www.w3.org/TR/sparql11-update/#insertData (a 20 de mayo de 2012)

[xxi] Segaran, Toby; Evans, Colin; Taylor, Jamie – "*Programming the semantic web*" – O'Reilly Media, Inc. 2009. 302 p. Ebook ISBN:978-0-596-80518-0. Posición 312 y siguientes.

[xxii] Clarke, Kendall – "The RDF database market", September,2010 - http://weblog.clarkparsia.com/2010/09/23/the-rdf-database-market/ (a 20 de mayo de 2012)

[xxiii] Wikipedia – ACID - http://en.wikipedia.org/wiki/ACID (a 20 de mayo de 2012)
[xxiv] W3C – "Large TripleStores" - http://www.w3.org/wiki/LargeTripleStores (a 20 de mayo de 2012)

[xxv] Patchigolla, Venkata - Comparison of clustered RDF data stores - Purdue University – http://docs.lib.purdue.edu/techmasters/43/ (a 20 de mayo de 2012)

[xxvi] OpenRDF.org – Sesame – "Chapter 8. HTTP Communications protocol for Sesame 2" - http://www.openrdf.org/doc/sesame2/system/ch08.html (a 20 de mayo de 2012)

[xxvii] Segaran, Toby; Evans, Colin; Taylor, Jamie – "*Programming the semantic web*" – O'Reilly Media, Inc. 2009. 302 p. Ebook ISBN:978-0-596-80518-0. Posición 5516 y siguientes

[xxviii] Segaran, Toby; Evans, Colin; Taylor, Jamie – "*Programming the semantic web*" – O'Reilly Media, Inc. 2009. 302 p. Ebook ISBN:978-0-596-80518-0. Posición 5595 y siguientes.

[xxix] W3C – "Dataset RDF Dumps" - http://www.w3.org/wiki/DataSetRDFDumps (a 20 de mayo de 2012)

[xxx] Ontotext – "Benchmark results" - http://www.ontotext.com/owlim/benchmark-results (a 20 de mayo de 2012)

[xxxi] Bizer, Chris. Schultz, Andreas – "BSBM V3 results (Feb – 2011)" - Freie Universität, Berlin - http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V6/#exploreBigowlim (a 20 de mayo de 2012)

Relacionado con el tema del rendimiento de bases de datos semánticas ver también la referencia (xxv) y las siguientes:

Bizer, Chris. Schultz, Andreas - "Berlin SPARQL Benchmark (BSBM) - Benchmark Rules" - http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/BenchmarkRules/index.html#datagenerator (a 20 de mayo de 2012)

W3C- "RDF Store Benchmarking" - http://www.w3.org/wiki/RdfStoreBenchmarking (a 20 de mayo de 2012)

[xxxii] Patchigolla, Venkata - Comparison of clustered RDF data stores - Purdue University – http://docs.lib.purdue.edu/techmasters/43/ (a 20 de mayo de 2012)

[xxxiii] Ontotext – "OWLIM in Action" - http://www.ontotext.com/owlim/usage (a 20 de mayo de 2012)

[xxxiv] Rayfield, Jem – "Sports Refresh: Dynamic Semantic Publishing" – abril, 2012 - http://www.bbc.co.uk/blogs/bbcinternet/2012/04/sports_dynamic_semantic.html (a 20 de mayo de 2012)