TFC

WEB SEMÁNTICA: RDF Y SGBD QUE LO SOPORTAN

TRABAJO REALIZADO POR:
Ana Beatriz Castelló Avilleira, estudiante de
I. T. Informática Gestión.

CONSULTOR: Oscar Celma Herrada

FECHA: Enero - 2006

RESUMEN

En este trabajo se realiza una breve introducción a la Web Semántica y su estado actual. Se analizan los lenguajes que se encuentran en la base de la pila de recomendaciones del World Wide Web Consortium (XML, XML Schema, RDF y RDF Schema) y se examinan diferentes sistemas de gestión de base de datos que dan soporte a los metadatos que son la base de la Web Semántica.

INDICE

1. INTRODUCCION

1.1. Justificación del TFC

El proyecto se basa en estudiar y evaluar diferentes sistemas gestores de bases de datos para guardar información dentro del contexto de la Web Semántica. La Web Semántica dota de significado al contenido textual de la Web, permitiendo que sea interpretable por una máquina.

Dentro del ámbito de la Web Semántica, la información se codifica siguiendo la notación RDF (*Resource Description Framework*). El lenguaje RDF provee interoperabilidad entre las aplicaciones que intercambian información a través de la Web. RDF puede ser representado utilizando diferentes sintaxis. Una de ellas es **RDF/XML** (sintaxis serializada básica y sintaxis abreviada) donde XML (Extensible Markup Lenguaje) es utilizado como medio de transporte o sintaxis.

En este trabajo se analizan tanto los lenguajes que utiliza la Web Semántica como los diferentes Sistemas de Gestión de Base de Datos que dan soporte a dichos lenguajes.

XML y Bases de Datos.

El concepto de "documento" en XML tiene una doble visión: Por un lado, tiene que ver con su contenido en un sentido tradicional (libros, revistas, grupos musicales, etc.) y por otro, una más centrada en los datos y en su gestión. Esta última visión enlaza con un campo más amplio de la Informática, como es el almacenamiento y la configuración de datos y en general, todo lo relacionado con las Bases de Datos. Por ello se han desarrollado herramientas para trabajar en XML con grandes bases documentales, tales como bibliotecas y almacenes de documentos. De hecho, la mayoría de proveedores de sistemas de gestión de bases de datos han trabajado muy duro para incorporar la compatibilidad de XML en sus productos (Oracle, IBM DB2, Microsoft, SQL Server etc.) En el caso de grandes documentos, como manuales técnicos, diccionarios, enciclopedias, etc., desde hace más de una década se ha venido trabajando con SGML (Standard Generalizad Markup Lenguaje) y ello ha permitido incorporar XML a estas técnicas sin mayores dificultades.

Sin embargo, la experiencia de trabajar con bases de datos documentales o tradicionales pone de manifiesto dos cuestiones claves: la necesidad de un diseño correcto de su estructura (el esquema), y que incluso disponiendo de muchos datos, pueden resultar inútiles sin un buen sistema de consulta. El W3C ha creado un lenguaje de consulta, XQuery, para proporcionar mecanismos que permitan localizar datos en documentos XML. El paso siguiente ha sido pasar a considerar esos documentos almacenados bien en Bases de Datos relacionales u orientadas a objetos, bien en sistemas de archivos más simples y proporcionar procedimientos que permitan su explotación, tratando de superar algunas de las deficiencias que surgen al comparar la gestión de documentos XML con las posibilidades del SQL en las bases de datos relacionales.

RDF y Web Semántica

Uno de los principales problemas con los que se encuentra la Web es su propio crecimiento, de forma que a medida que el número de sitios Web aumenta, la organización y procesado de la información accesible se incrementa exponencialmente necesitando cada vez más esfuerzo para acceder a lo que

realmente interesa en cada situación. Uno de los mecanismos que ha desarrollado W3C (World Wide Web Consortium) para gestionar el procesado automático de esta información es el Resource Description Framework (***RDF***). RDF consiste, básicamente, en un formato basado en XML para expresar metadatos referidos a la información existente en la Web. El objetivo de RDF es crear un modelo para describir información sobre la propia Web, de forma que se puedan describir distintos recursos de una forma consistente, con lo que, al menos teóricamente, se facilita la clasificación, localización y catalogación automática de los recursos existentes en la Web.

Por otra parte, en la búsqueda de una semántica que describa los recursos en la Web que vaya mas allá de una simple definición de una estructura de datos, se ha planteado la necesidad de definir relaciones entre conceptos, así como restricciones de dominio y rango entre ellos, que faciliten la definición de reglas y en consecuencia un procesamiento "inteligente" de estos recursos. En este contexto se ha propuesto el concepto de ***Web Semántica*** con la que se pretende automatizar al máximo el manejo de la información presente en la red, además de perseguir el objetivo de que los documentos incorporen un determinado significado semántico que pueda ser "comprendido" directamente por los ordenadores, sin necesidad de una intervención humana.

Dentro de la arquitectura de la Web Semántica se ha incluido a XML como capa básica para la definición sintáctica, a RDF, que proporciona información descriptiva simple sobre los recursos que se encuentran en la Web, como la siguiente capa, y para cumplir todos los objetivos de la Web Semántica ha surgido OWL (Ontology Web Lenguaje). OWL es un mecanismo para desarrollar temas o vocabularios específicos asociados a dichos recursos de la Web. Lo que hace OWL es proporcionar un lenguaje para definir ontologías estructuradas que pueden ser utilizadas a través de diferentes sistemas. Las ontologías permiten describir y formalizar conceptos que pertenecen a un dominio. Estas ontologías son utilizadas por los usuarios, las bases de datos y las aplicaciones que necesitan compartir información específica en un dominio determinado. Las ontologías incluyen definiciones de conceptos básicos en un campo determinado y la relación entre ellos.

El objetivo es superar la muy reducida capacidad actual de interpretación del contenido de la Web que tiene una máquina, para lo que se proporciona un vocabulario adicional, junto con una semántica formal. De hecho, RDF y OWL son los estándares de la Web Semántica al proporcionar un marco para que las diferentes máquinas puedan compartir la misma información de forma automática y "entenderla", incluso sin que utilicen el mismo software.

1.2. Objetivos del trabajo.

1. Estudiar los conceptos básicos de la Web semántica
2. Conocer la estructura y la representación de los lenguajes de representación de la información en la Web: XML y su extensión RDF.
3. Conocer la estructura y organización de los SGBD que trabajan con información basada en RDF/XML.
4. Evaluar la adecuación de uso de los SGBD para guardar y recuperar descripciones en RDF.

1.3. Enfoque y método a seguir:

Este trabajo se ha enfocado desde un punto de vista teórico, por lo que el método a seguir ha sido el estudio de los diferentes trabajos realizados hasta el momento y el análisis de las diferentes bases de datos que existen en la actualidad.

1.4. Planificación del proyecto:

Id	Tarea	Duración	Comienzo	Fin	Predecesoras
1	Plan de Trabajo	9 días	mi 21/09/05	lu 03/10/05	
2	**Pec 2**	30 días	ma 04/10/05	do 13/11/05	
3	XML	7 días	ma 04/10/05	mi 12/10/05	1
4	RDF	7 días	ju 13/10/05	vi 21/10/05	3
5	Lenguajes de Consulta	10 días	lu 24/10/05	vi 04/11/05	4
6	Confección Pec2	2 días	ma 08/11/05	mi 09/11/05	5
7	Entrega Pec2	1 día	do 13/11/05	do 13/11/05	6
8	**Pec3**	42 días	lu 24/10/05	lu 19/12/05	
9	Búsqueda SGBD	25 días	lu 24/10/05	ju 24/11/05	4
10	Análisis SGBD	10 días	vi 25/11/05	ju 08/12/05	9
11	Creación Consultas (No realizado)	5 días	vi 09/12/05	ju 15/12/05	10
12	Confección Pec3	1 día	vi 16/12/05	vi 16/12/05	11
13	Entrega Pec3	1 día	lu 19/12/05	lu 19/12/05	
14	**Documentación Final**	11 días	lu 26/12/05	lu 09/01/06	
15	Memoria	5 días	lu 26/12/05	vi 30/12/05	13
16	Entrega Documentación	1 día	lu 09/01/06	lu 09/01/06	15
17	Debate	5 días	lu 16/01/06	vi 20/01/06	16

1.5. Producto obtenido:

El producto obtenido es esta misma memoria.

1.6. Descripción del resto de capítulos:

En el resto de capítulos se realiza una breve introducción al estado actual de la Web Semántica, se analizan los lenguajes XML, XML Schema, RDF y RDF Schema, así como los diferentes Sistemas de Gestión de Bases de Datos que los soportan.

3. INTRODUCCION A LA WEB SEMANTICA

La Web semántica es la evolución de la Web actual en la que el contenido es procesable automáticamente a escala global. Su objetivo principal es permitir tanto a humanos como a máquinas encontrar, compartir y combinar información de manera sencilla y automatizada, es decir, tener una Web más útil.

La base de la Web Semántica son los Metadatos: Recursos que proveen información acerca de sí mismos. Deben estar en un formato común y procesable por las máquinas. Los vocabularios de estos metadatos deben estar definidos perfectamente. Los metadatos existen en las aplicaciones que utilizamos para generar los contenidos: Editores gráficos, procesadores de textos, etc. Por ejemplo, con respecto a una imagen fotográfica almacenada en .jpg los metadatos relativos a esa imagen serían: Nombre del archivo, fecha y hora de la captura de la imagen, fecha y hora de la descarga al ordenador, tamaño de la imagen, modelo de la cámara, etc.…

Para poder procesar estos metadatos, son necesarios los siguientes elementos:

- Recursos definidos unívocamente (URIs).
- Vocabularios controlados (Ontologías)
- Un formato común para expresar esos metadatos, como RDF.
- Infraestructuras para acceder a esa información

La tecnología que se ha creado para hacer posible la Web semántica incluye lenguajes para la representación de ontologías, parsers[1], lenguajes de consulta, entornos de desarrollo, módulos de gestión (almacenamiento, acceso, actualización) de ontologías, módulos de visualización, conversión de ontologías, y otras herramientas y librerías.

El primer lenguaje para la construcción de la Web Semántica fue SHOE[2], creado por Jim Hendler en la Universidad de Maryland en 1997. Desde entonces se han definido otros lenguajes y estándares con finalidad similar, como XML, RDF[3], DAML+OIL[4] y más recientemente OWL[5] por citar los más importantes.

XML representa una primera aproximación a la Web Semántica, y aunque no está expresamente pensado para definir ontologías, es el estándar más extendido hoy día. XML permite estructurar datos y documentos en forma de árboles de etiquetas con atributos. Con XML Schema[6] (XMLS) se pueden acordar de antemano las estructuras que se van a utilizar, así como manejar tipos de datos primitivos y derivados. Con el estándar XSLT[7] se pueden definir plantillas asociadas a las estructuras XML, que describen cómo generar código HTML para visualizar los contenidos en un navegador. Parsers como DOM[8] permiten moverse por las estructuras XML desde un programa Java o C++, y existen multitud de

[1] Analizador es Sintácticos.
[2] http://www.cs.umd.edu/projects/plus/SHOE/
[3] http://www.w3.org/RDF/
[4] http://www.w3.org/TR/daml+oil-reference/
[5] http://www.w3.org/2001/sw/WebOnt/
[6] http://www.w3.org/XML/Schema/
[7] http://www.w3.org/Style/XSL/
[8] http://www.w3.org/DOM/

herramientas para facilitar la compatibilidad de XML con bases de datos, Java Beans[9], etc.

En 1999 se publicó la primera versión de RDF (Resource Description Framework), un lenguaje para la definición de ontologías y metadatos en la Web. RDF es hoy el estándar más popular y extendido en la comunidad de la Web semántica. El elemento de construcción básica en RDF es la "tripleta" o sentencia, que consiste en dos nodos (sujeto y objeto) unidos por un arco (predicado), donde los nodos representan recursos, y los arcos propiedades. Encadenando estas tripletas se construyen grafos o redes semánticas para la Web.

Con RDF Schema (RDFS) se pueden definir jerarquías de clases de recursos, especificando las propiedades y relaciones que se admiten entre ellas. En RDF las clases, relaciones, y las propias sentencias son también recursos, y por lo tanto se pueden examinar y recorrer como parte del grafo, o incluso asertar sentencias sobre ellas. Se han definido diferentes formas sintácticas para la formulación escrita de RDF, pero quizás la más extendida es la basada en XML. Es por ello que RDF se presenta a menudo como una extensión de XML.

Una de las realizaciones pendientes desde hace años en relación con RDF es la creación de un lenguaje de consulta, similar al SQL de las bases de datos, que permita expresar búsquedas complejas sobre un grafo RDF mediante una sintaxis declarativa sencilla. A falta de alcanzar un acuerdo sobre un estándar comúnmente aceptado, se han consolidado de facto distintas iniciativas particulares como la del RDF Query Language (RDQL)[10], por Hewlett Packard; RDF Schema Query Language[11] [Karvounarakis 2002] (RQL), por el instituto ICS-FORTH de Grecia; Sesame RDF Query Language[12] (SeRQL), por la empresa holandesa Aidministrator, y el que posiblemente sea el estándar en el futuro, SPARQL[13], actualmente en discusión en el W3C.

A RDF le siguieron OIL[14] (Ontology Inference Language), desarrollado en Europa, y DAML[15] (DARPA Agent Markup Language), en EE.UU., dos lenguajes muy similares que de hecho se terminaron fundiendo en DAML+OIL. A partir de esta unión se definió el lenguaje OWL (Web Ontology Language), con el propósito de reunir todas las ventajas de DAML+OIL y resolver los problemas de este lenguaje. OWL se puede formular en RDF, por lo que se suele considerar una extensión de éste. OWL incluye toda la capacidad expresiva de RDF(S) y la extiende con la posibilidad de utilizar expresiones lógicas. OWL permite, por ejemplo, definir clases mediante condiciones sobre sus miembros mediante combinación booleana de clases, o por enumeración de las instancias que pertenecen a la clase (i.e. por extensión). Además OWL permite atribuir ciertas propiedades a las relaciones, como cardinalidad, simetría, transitividad, o relaciones inversas.

Para desarrollar aplicaciones basadas en RDF, OWL o lenguajes similares se precisan librerías para leer y procesar las ontologías definidas en estos lenguajes. Sin embargo con diferencia, el parser de RDF y OWL más popular es Jena2[16], desarrollado por Hewlett Packard, que permite leer, recorrer y modificar grafos tanto RDF como OWL desde un programa Java. Jena2 permite además guardar las

[9] http://java.sun.com/products/javabeans/
[10] http://www.hpl.hp.com/semweb/rdql.htm
[11] http://139.91.183.30:9090/RDF/RQL/
[12] http://sesame.aidministrator.nl/publications/users/ch05.html
[13] http://www.w3.org/TR/rdf-sparql-query/
[14] http://www.ontoknowledge.org/oil/
[15] http://www.daml.org/
[16] http://www.hpl.hp.com/semweb/jena2.htm

ontologías tanto en RDF textual como en formato de base de datos, lo que es importante para grafos muy grandes. Otra librería muy conocida de similares características para RDF y OWL es Sesame[17], desarrollado en el proyecto europeo Ontobroker[18]. Las últimas versiones de Jena y Sesame han incorporado también motores de razonamiento para las expresiones lógicas de OWL.

Escribir en lenguajes como RDF y OWL resulta sumamente difícil y propenso a errores. Afortunadamente se pueden utilizar entornos gráficos para visualizar y construir ontologías de forma mucho más razonable, como Protege 2000[19], OwlViz[20], SWOOP[21]...

Algunos ejemplos de aplicaciones actuales sobre la Web semántica son:

- La Iniciativa de Archivos Abiertos (OAI)[22]:Desarrolla y promociona estándares de interoperabilidad con el objetivo de facilitar la distribución eficiente de contenido.
- FOAF[23]: Proyecto basado en la creación de homepages entendibles por las máquinas que describe personas, los links entre ellos y las cosas que pueden crear y hacer.
- DOAP[24]:Es un proyecto para DOAP crear un vocabulario XML/RDF para describir proyectos "open source".
- Directorio Medioambiental SWED[25]
- SIMILE[26] Su propósito es aumentar la inter-operabilidad entre los recursos, vocabularios/ontologías/esquemas, metadatos y servicios
- Portales
  - Vodafone Live: Basado en RDF
  - Portal de Nokia
- XMP[27]: Metadatos basados en RDF desarrollado por Adobe para introducir metadatos en todas las imágenes.

[17] http://www.openrdf.org/
[18] http://ontobroker.semanticweb.org/
[19] http://protege.semanticweb.org//
[20] http://www.co-ode.org/downloads/owlviz/
[21] http://www.mindswap.org/2004/SWOOP/
[22] http://www.openarchives.org/
[23] http://www.foaf-project.org/
[24] http://usefulinc.com/doap
[25] DOAP is a project to create an XML/RDF vocabulary to describe open source projects.
[26] http://simile.mit.edu/
[27] http://www.adobe.com/products/xmp/main.html

4. INTRODUCCION A LOS LENGUAJES XML Y RDF

4.1. ORIGENES

Para comprender la evolución que han sufrido los lenguajes de Internet hasta llegar a XML, hay que hacer referencia al Consorcio World Wide Web y al papel que ha jugado en el desarrollo de Internet.

El Consorcio World Wide Web[28] (W3C) es una asociación internacional formada por organizaciones miembro del consorcio, personal y el público en general, que trabajan conjuntamente para desarrollar estándares Web. La misión del W3C es: **Guiar la Web hacia su máximo potencial a través del desarrollo de protocolos y pautas que aseguren el crecimiento futuro de la Web.**

En 1989 Tim Berners-Lee[29] creó la World Wide Web: Acuñó el término "World Wide Web", desarrolló el primer servidor para la World Wide Web, "httpd," y el primer programa de cliente (un navegador y un editor), "WorldWideWeb" en octubre de 1990. Creó la primera versión del "Lenguaje de Etiquetado de Hipertexto" (HTML), un lenguaje de formato que permite la utilización de enlaces de hipertexto y que se convirtió en el formato de publicación principal para la Web. Sus especificaciones iniciales para URI, HTTP y HTML, fueron revisadas y discutidas en grandes círculos según crecía la tecnología Web.

En octubre de 1994, Tim Berners-Lee fundó el Consorcio World Wide Web (W3C) en el Laboratorio de Ciencias Informática del Instituto de Tecnología de Massachusetts [MIT/LCS], en colaboración con el CERN[30] (European Organization for Nuclear Research) donde la Web tuvo su origen con la colaboración del DARPA (Defense Advanced Research Projects Agency)[31] y de la Comisión Europea[32]

Algunas de las acciones más importantes realizadas por el W3C en relación con el presente trabajo han sido las siguientes:

- **Diciembre 1997** - HTML 4.0 añade tablas, scripting, hojas de estilo, internacionalización y accesibilidad a la publicación en la Web.
- **Febrero 1998** - XML 1.0 promueve la interoperabilidad y etiquetado de dominio.
- **Mayo 2001** - Esquema XML (XML Schema) es la pieza esencial para que XML alcance su máximo potencial.
- **Febrero 2004** - RDF y OWL constituyen un importante avance para las aplicaciones de Web

Al ser XML un lenguaje de marcas, es importante ubicar el papel de sus antecesores en el tratamiento de los documentos de marcas.

GML Y Hojas de Estilo.
Los antecedentes del XML se sitúan a finales de los 60, cuando IBM desarrollo *Generalized Markup Language* (GML) para solucionar los problemas internos de publicación de sus manuales y comunicados de contratos legales y especificaciones

[28] Consorcio Word Wide Web: www.w3c.org
[29] Tim Berners-Lee: http://www.w3.org/People/Berners-Lee/
[30] CERN: http://public.web.cern.ch/Public/Welcome.html
[31] http://www.w3.org/Consortium/Prospectus/DARPA
[32] http://europa.eu.int/

de proyectos. GML fue diseñado de modo que los mismos ficheros fuente pudieran ser procesados para producir libros, informes y ediciones electrónicas. Una aportación de GML, que se mantiene en su idea básica en la actualidad, es la hoja de estilo, definida como un archivo separado del documento, que contiene información relacionada con formatos, de forma que a partir de un conjunto de esas hojas se puede formatear cada elemento y presentar completo un determinado documento. GML tenía una sintaxis de entrada simple para composiciones, incluyendo las etiquetas <> y </> que se conocen hoy en día en los lenguajes de marcado. Aunque los documentos fueran fáciles de leer y escribir, no se adaptaban a procesos de propósito general como por ejemplo aplicaciones de computador.

SGML y la definición de tipo de documento

Como surgieron muchos tipos de documentos, cada uno de los cuales requería de etiquetas apropiadas, se necesito la creación de una forma estándar para manipular y publicar cada definición del tipo de documento (DTD). A comienzos de los 80´s representantes de *GenCode* y *GML*, se unieron para formar el comité *American National Standar Institute* (ANSI) cuyo trabajo estaba orientado al tema de lenguajes de computador para el procesamiento de texto, y su objetivo fue estandarizar las formas de especificar, definir y usar marcas en los documentos.

SGML el *Standardized Generalized Markup language* fue publicado por la ISO 8879 en 1986. Desarrollado para definir y utilizar documentos con formatos portables, lo suficientemente formal para permitir pruebas de validación del documento, lo suficientemente estructurado para permitir el manejo de documentos complejos y lo suficientemente extensible para soportar el manejo de grandes almacenes de información.

HTML

Tim Berners escogió algunas etiquetas de marcado de una muestra del DTD de SGML (usado en la CERN) y en NeXUS (el visor y editor Web original) y utilizó etiquetas y hojas de estilo para componer una característica importante: "los enlaces".

En 1992 la Web no estaba preparada para un lenguaje de marcado genérico poderoso, lo que la Web necesitaba era un pequeño conjunto de etiquetas que fuera lo suficientemente simple y fácil de entender para la comunidad de autores. Ese pequeño conjunto de etiquetas es lo que conocemos como HTML.

Ya que HTML se basa en SGML, se puede decir que constituye el primer paso del lanzamiento de la comunidad SGML al World Wide Web. Esto trajo muchas ventajas para los implementadores, puesto que SGML tiene muchas características opcionales y la especificación es dura de leer.

HTML fue muy bien acogido, pero tiene limitaciones importantes, por ejemplo, que es poco flexible debido a que sus etiquetas son fijas, Dado que el desarrollo de las aplicaciones actuales está orientado a la Web, se inició una búsqueda por obtener un lenguaje más flexible que permitiera definir etiquetas para el desarrollo de aplicaciones en ámbitos concretos, pero orientados a la Web.

De HTML A XML

HTML fue muy bien acogido, pero tiene limitaciones importantes, por ejemplo, que es poco flexible debido a que sus etiquetas son fijas, mezcla el marcado estructural y el de presentación, es difícil de procesar, tiene inconvenientes en la visualización de documentos en determinados idiomas, etc.) Dado que el desarrollo de las aplicaciones actuales está orientado a la Web, se inició una búsqueda por obtener

un lenguaje más flexible que permitiera definir etiquetas para el desarrollo de aplicaciones en ámbitos concretos, pero orientados a la Web.

XML fue desarrollado por un Grupo de Trabajo de XML perteneciente al W3C, conocido originalmente como el Comité de Revisión Editorial de SGML, en 1996. Este grupo de trabajo estaba presidido por John Bosak de SUN. Los objetivos de diseño que este grupo se propuso fueron los siguientes:

- XML debía ser utilizable directamente sobre Internet
- XML debería soportar una amplia variedad de aplicaciones
- Tenía que ser compatible con SGML
- Tenía que ser sencillo escribir programas que procesasen documentos XML
- Debía existir el mínimo de características opcionales (mejor ninguna)
- Los documentos debían ser legibles para los seres humanos
- El diseño de los documentos tenía que ser formal y conciso
- Los documentos XML tenían que ser sencillos de crear.
- El formato de las etiquetas usadas tendría la mínima importancia.

La primera recomendación XML del W3C salió en febrero de 1998.

ORIGEN DE RDF

XML es un metalenguaje universal de definición de etiquetas. Proporciona un marco uniforme y un conjunto de herramientas como los parsers, para el intercambio de datos y metadatos entre aplicaciones. Sin embargo, XML no proporciona ninguna información sobre el significado (semántica) de los datos. Por ejemplo, no hay ninguna intención de significado en el anidamiento de las etiquetas; es tarea de cada aplicación el interpretar el anidamiento.

Aunque RDF (Resource Description Framework) es a menudo definido como un "lenguaje", es, en esencia, un modelo de datos. Un modelo abstracto de datos necesita una sintaxis concreta para ser representado y transmitido, y la sintaxis de RDF está basada en XML. Como resultado, hereda los beneficios asociados con XML. Sin embargo, es necesario entender que son posibles otras representaciones sintácticas de RDF, no basadas en XML.

PICS y MCF predecesores de RDF

Una revisión típica de la historia de RDF señala que no hay una persona o institución que sea responsable de su autoría, sino que es el resultado de múltiples colaboradores, tanto personas como instituciones. Sin embargo, es posible señalar a RAMANATHAN V. GUHA como el principal precursor de esta iniciativa. GUHA había trabajado en el proyecto CYC (Common Sense Applications) cuando desarrolló en APPLE el lenguaje MCF (Meta Content Framework), que propone una forma de representar estructuras de metadatos para fuentes de datos heterogéneas.

El desarrollo de PICS (Platform for Internet Content Selection)[33] fue motivado por las restricciones en algunas iniciativas legislativas en USA. PICS es un mecanismo para comunicación de calificaciones de sitios Web desde un servidor a clientes; estas calificaciones -o etiquetas de calificaciones- contienen información sobre el contenido de páginas Web. Por ejemplo, cuando una página particular contiene artículos de investigación; o está escrito por un investigador calificado; o contiene sexo, desnudos, violencia, etc. En lugar de ser un conjunto de criterios fijos, PICS introduce un mecanismo general para crear sistemas de calificación. Diferentes

[33] http://www.w3.org/PICS/

organizaciones pueden calificar contenido basados en su propios objetivos y valores, los usuarios (por ejemplo, los padres preocupados sobre el uso de la Web por parte de sus hijos) puede configurar sus browser para filtrar el contenido de páginas Web que no se ajustan a sus propios criterios. PICS fue una recomendación oficial del W3C en octubre de 1996.
MCF contribuyo al esfuerzo del W3C para definir el sucesor de PICS, una tecnología para etiquetado y filtrado. Esto, más el envío de XML Data desde Microsoft, se convirtió en RDF, que trata muchos de los mismos asuntos de MCF, pero tiene el beneficio de usar "*angle-brackets*" para el formato de datos y URLs para nombrar partes de vocabularios usados en los mapas de sitios. Otras contribuciones a RDF fueron las iniciativas de Microsoft: *Web Collections using XML y XML-Data.*

Modelo de metadatos RDF
Para mantener metadatos de forma distribuida e interoperables en la Web se ha propuesto RDF, el cual brinda un mejor modelo para el manejo de datos distribuidos a diferencia de XML. XML es un lenguaje para documentos semiestructurados, su modelo es un árbol y el orden de los elementos es importante; en cambio RDF es un lenguaje para metadatos, el modelo es de un grafo etiquetado y dirigido donde el orden no es relevante. XML aporta una semántica muy básica a los documentos, al incorporar etiquetas que pueden tratar sobre el significado de la información. Esto es un avance sobre HTML, donde las etiquetas expresan sólo la estructura del documento. Para un dominio particular, XML es una alternativa como lenguaje de marcado semántico, utilizando esquemas XML para definir vocabularios o bien una combinación de XML y RDF. Para permitir interoperabilidad entre aplicaciones es necesario soportar una diversidad de dominios. Esto significa que se debe permitir mantener distintos vocabularios y las relaciones a nivel lógico que existen entre ellos.

La W3C publicó una especificación para el modelo de datos RDF y la sintaxis XML como una recomendación en el año 1999. Se comenzó entonces un nuevo trabajo que fue publicado como un conjunto de especificaciones en el año 2004. A diferencia de otras recomendaciones de la W3C, estas especificaciones reemplazaron completamente las antiguas, y se les asignó un número de versión: RDF 2.0. Por lo tanto, muchas implementaciones basadas en las recomendaciones de 1999 no han sido todavía actualizadas.

4.2. INTRODUCCION AL LENGUAJE XML

a) Estructura de un documento XML.

Un documento XML es una información jerarquizada, en forma de texto, que constituye un objeto de datos que puede ser presentado mediante una estructura de árbol, que puede estar almacenado en un único archivo o estar divido en varios. Tanto su estructura física como lógica tienen la capacidad de anidar sus propiedades, lo que explica que XML organice sus documentos de forma no lineal y en múltiples piezas.

Un documento XML consiste en un prólogo, un número de documentos y un epílogo opcional.

Prólogo.
Un prólogo consiste en un encabezado XML y una referencia opcional a documentos externos de estructura. La forma mas simple de encabezado es: <?xml…?> incluyendo la definición de la versión de XML a la que se ajusta el documento. El encabezado puede tener dos atributos optativos: *encoding,* que determina el tipo de codificación del documento, y *standalone,* que indica si se necesita un documento externo para definir la estructura del documento. Por ejemplo:

<?xml versión:"1.0" encoding="ISO-8859-1" standalone=no ?>

En la declaración del tipo de documento se indica la forma en como una DTD se incorpora a dicho documento. Se declara a través de *DOCTYPE* con la sintaxis:

<!DOCTYPE nombre SYSTEM (o PUBLIC) uri>. Por ejemplo:
<!DOCTYPE película SYSTEM "película.dtd">

Cuerpo del documento.
Elementos y Atributos: En Xml un elemento es un componente lógico de la jerarquía de un documento, que a su vez se puede descomponer en otros elementos. Es una estructura compuesta de una etiqueta inicial, una etiqueta final y la información entre las etiquetas, que puede ser un texto u otros elementos anidados en él. Todo documento XML contiene uno o más elementos, delimitados e identificados por un nombre llamado *identificador genérico*,

<Titulo> XML y Web Semántica </Titulo>

El contenido de un elemento es cualquier cosa contenida entre sus etiquetas de inicio y final, y puede constar tanto de texto como de otros elementos. Se llama *atributo* a la forma en que los elementos incorporan información relacionada acerca de si mismos, describiendo sus propiedades, y acabando de dar significado a los nodos que constituyen el árbol del documento. Por tanto, un elemento, además de su identificador, puede tener un conjunto de atributos, cada uno con su nombre y valor respectivo.

<Elemento atributo1="valor" atributo2="valor2" />

Entidades: Una entidad general consiste en un nombre y un valor para su uso dentro del contenido de un documento. Las entidades en XML, al igual que los datos, pueden ser procesables o no procesables. El concepto de entidad no procesable se refiere a objetos no XML, es decir, elementos cuyos datos el analizador XML no puede leer (datos binarios .exe, graficos .gif, videos .mpeg, etc.). Una entidad es procesable cuando al empezar a analizar un documento, el procesador XML la reconoce como tal y por tanto asocia a su nombre un valor con

su texto de reemplazamiento, que puede ser desde un carácter a un archivo que se conoce también como expansión de la entidad.

Instrucciones de proceso: es un mecanismo que permite a los documentos XML contener instrucciones específicas para las aplicaciones que los van a usar, sin que estas formen parte de los datos del propio documento. El siguiente ejemplo la instrucción de proceso se utiliza para indicar a la aplicación que el documento se debe mostrar con una determinada hoja de estilo:
<?xml – stylesheet type=”text/xsl” ref.=”HojaEstilo.xsl” ?>

Secciones CDATA: Son complementarias al marcado, ya que permiten que determinados datos no sean procesados por los analizadores, con lo que pueden contener: texto, caracteres reservados y caracteres blancos. Empiezan con la cadena <![CDATA[y finalizan con la cadena]]>

Documentos bien formados y documentos válidos.

Un documento XML está bien formado si es sintácticamente correcto. Algunas reglas sintácticas son:

- Sólo contiene un elemento raíz
- Cada elemento tiene una etiqueta inicial otra final con nombres idénticos
- Las etiquetas no se entrecruzan, por lo que están correctamente anidados.
- Los atributos de un elemento tienen nombres únicos.
- No aparecen en el texto del documento los caracteres <, > y &
- Etc.

En un documento XML hay que distinguir entre documento válido y documento bien formado. En un documento válido, además de estar bien formado, se deben respetar las restricciones establecidas por la definición externa de un esquema (DTD o esquema XML).

b) DTD’s

Los componentes de una DTD se pueden definir en un archivo separado (DTD externa) o en el propio documento XML (DTD interna). Una DTD es una colección de declaraciones de elementos (ELEMENT), atributos (ATTLIST), entidades (ENTITY) y notaciones (NOTATION), a partir de las cuales se describe la “validez” de un documento.
Si la DTD está incluida en el archivo XML, debe ser definida con la siguiente sintaxis:

<!DOCTYPE elemento-raíz [declaraciones de elementos]>

Ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE nota [
  <!ELEMENT nota (para,de,cabecera,cuerpo)>
  <!ELEMENT para     (#PCDATA)>
  <!ELEMENT de    (#PCDATA)>
  <!ELEMENT asunto (#PCDATA)>
  <!ELEMENT cuerpo    (#PCDATA)>
]>
<nota>
  <para>Ana</para>
```

```
  <de>Alberto</de>
  <asunto>Recordatorio</asunto>
  <cuerpo>Nos vemos este finde</cuerpo>
</nota>
```

Esta DTD se interpreta de esta forma:

!DOCTYPE nota (en la línea 2) define que es un documento del tipo NOTA
!ELEMENT nota (en la línea 3) define que el elemento nota tiene cuatro elementos:"para ,de, asunto, cuerpo".
!ELEMENT para (en la línea 4) define el elemento **para** de tipo "#PCDATA".
!ELEMENT de (en la línea 5) define el elemento **de** de tipo "#PCDATA".
etc.....
Si la DTD no está incluida en el archivo XML, debe ser definida con la siguiente sintaxis:

```
<!DOCTYPE root-element SYSTEM "filename">
```

Este es el mismo documento que el anterior, pero con una DTD externa:

```
<?xml version="1.0"?>
<!DOCTYPE nota SYSTEM "nota.dtd">
<nota>
  <para>Ana</para>
  <de>Alberto</de>
  <asunto>Recordatorio</asunto>
  <cuerpo>Nos vemos este finde</cuerpo>
</nota>
```

y esta es una copia del fichero “nota.dtd” que la contiene:

```
  <!ELEMENT nota (para,de,cabecera,cuerpo)>
  <!ELEMENT para     (#PCDATA)>
  <!ELEMENT de    (#PCDATA)>
  <!ELEMENT asunto (#PCDATA)>
  <!ELEMENT cuerpo    (#PCDATA)>
```

c) XML SCHEMA

XML Schema proporciona un lenguaje muchísimo más rico para definir la estructura de los documentos XML. Una de sus características es que su sintaxis está basada en el propio XML. Este diseño proporciona una mejora importante en la legibilidad, pero lo más importante, significa la reutilización de la tecnología.

XML Schema, permite una serie de ventajas adicionales que se consideraron importantes:

- Una estructura de tipos mucho más rica. Los tipos base que se pueden emplear dentro de esquema de XML, son integer, bolean, string, date, etc.
- Permite tipos definidos por el usuario, llamados **Arquetipos**, dándoles un nombre y que se pueden emplear en distintas partes dentro del **Schema**.
- Es posible agrupar atributos, haciendo más comprensible el uso de un grupo de aspectos de varios elementos distintos, pero con denominador común, que deben ir juntos en cada uno de estos elementos.

- El trabajo con namespaces está especificado, permitiendo, dentro de la dificultad que conlleva trabajar con ellos, validar documentos con varios **namespaces**.
- Sin embargo, la característica que más resalta la utilidad de XML Schema es la posibilidad de extender **Arquetipos** de un modo específico, es decir permite lo que en términos de orientación a objetos se llama **herencia**. Considérese un esquema que extiende otro previamente hecho, se permite refinar la especificación de algún tipo de elemento para, por ejemplo, indicar que puede contener algún nuevo elemento del tipo que sea; pero dejando el resto del esquema antiguo completamente intacto.

Siguiendo con el ejemplo “nota”, a continuación se muestra un esquema XML Simple:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.program.com"
xmlns="http://www.program.com"
elementFormDefault="qualified">
<xs:element name="nota">
   <xs:complexType>
     <xs:sequence>
          <xs:element name="para" type="xs:string"/>
          <xs:element name="de" type="xs:string"/>
          <xs:element name="asunto" type="xs:string"/>
          <xs:element name="cuerpo" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

Y un documento xml que referencia el esquema:

```
<?xml version="1.0"?>
<nota
xmlns="http://www.program.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.program.com nota.xsd">

<to>Ana</to>
<from>Alberto</from>
<heading>Recordatorio</heading>
<body>Nos vemos este finde</body>
</note>
```

d) ESPACIO DE NOMBRES XML

Una de las ventajas principales de la utilización de XML es que se puede acceder a la información desde varias fuentes; es decir, un documento XML puede usar mas de una DTD o esquema. Aunque cada documento estructural se desarrolla independientemente, las colisiones de nombres son inevitables, es decir, que elementos de diferentes espacios de nombres puedan tener el mismo nombre. Para evitar estas colisiones, en XML se recurre al uso de prefijos adecuados, utilizando un prefijo diferente para cada DTD o esquema:

Prefijo:nombre.

En XML se llama *espacio de nombres* a una colección de nombres que proporciona un mecanismo por el que los nombres de elementos y atributos pueden asignarse para cada uso deseado, utilizando prefijos adecuados. Estos dominios nominales son objeto de una Recomendación del W3C elaborada para cumplir tres objetivos:

- Poder mezclar distintos vocabularios XML en un mismo documento
- Identificar unívocamente la etiqueta XML
- Disponer de nombres universales cuya panorámica se extienda más allá del documento que los contiene.
-

Los espacios de nombres se declaran en un elemento y pueden ser utilizados por ese elemento y cualquiera de sus hijos. Una declaración de un espacio de nombres tiene la forma:

xmlns:prefix="location"

Si no se especifica un prefijo (xmlns="location") entonces la localización se utiliza por defecto.

e) PROCESADO XML

Un procesador XML consiste en un módulo de software centrado en leer documentos, comprobar su sintaxis, informar de los posibles errores y de proporcionar acceso tanto a su contenido como a su estructura, y todo ello con capacidad de presentar los detalles tanto a un ser humano como a otra máquina.

Los principales componentes de un procesador XML son:

- Los gestores, responsables de localizar los datos que van a pasar al analizador, ya que acaba siendo necesario que algún módulo se encargue de funciones tales como declarar las entidades que se pueden pasar a la aplicación y cuales no y otras cuestiones parecidas
- El analizador, encargado de leer el flujo de símbolos de entrada (llamados *tokens*) y emitir los símbolos de salida basándose en las reglas gramaticales correspondientes.
- El *analizador léxico* que lee símbolos individuales y emite un símbolo por palabra (o grupo de caracteres) basándose en un conjunto de reglas léxicas.
- El *validador*, que comprueba las reglas del esquema y que incluso puede tener la capacidad de manejar ciertas entradas inaceptables.

f) XPATH

XPATH es un lenguaje declarativo basado en cadenas de expresiones (no basado en XML) que pueden usarse dentro de URIs y de atributos XML, y cuyo objetivo es localizar partes específicas de un documento utilizando para el procesado de sus valores un modelo de datos del documento basado en una estructura de árbol. Además de este objetivo, también proporciona facilidades básicas para manipular cadenas, números y booleanos.

El concepto central de los lenguajes de consulta XML son las expresiones de acceso (path expressions). Pueden ser:

- Absolutas (comenzando en la raíz del árbol); sintáctica mente, deben comenzar con el símbolo /, el cual se refiere a la raíz del documento, situado un nivel por encima del elemento raíz de un documento.
- Relativas a un nodo.

g) XQUERY

Se puede definir XQuery con el siguiente símil: XQuery es a XML lo mismo que SQL es a las bases de datos relacionales. XQuery es, pues, un lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML. Abarca desde archivos XML hasta bases de datos relacionales con funciones de conversión de registros (o tuplas) a XML. Su principal función es extraer información de un conjunto de datos organizados como un árbol n-ário de etiquetas XML. En este sentido XQuery es independiente del origen de los datos.
XQuery es un lenguaje funcional, lo que significa que, en vez de ejecutar una lista de comandos como un lenguaje procedimental clásico, cada consulta es una expresión que es evaluada y devuelve un resultado (un fragmento XML), al igual que en SQL. Diversas expresiones pueden combinarse de una manera muy flexible con otras expresiones para crear nuevas expresiones más complejas y de mayor potencia semántica. Una consulta XQuery tiene como entrada y salida sendos documentos XML.

XQuery es una recomendación del W3C. Los requisitos puestos por W3C a Xquery son:

- XQuery debe ser un lenguaje declarativo. Al igual que SQL hay que indicar que se quiere, no la manera de obtenerlo.
- XQuery debe ser independiente del protocolo de acceso a la colección de datos. Una consulta en XQuery debe funcionar igual al consultar un archivo local que al consultar un servidor de bases de datos que al consultar un archivo XML en un servidor Web.
- Las consultas y los resultados deben respetar el modelo de datos XML
- Las consultas y los resultados deben ofrecer soporte para los namespaces.
- Debe ser capaz de soportar XML-Schemas y DTDs y también debe ser capaz de trabajar sin ninguno de ellos.
- XQuery debe poder trabajar con independencia de la estructura del documento, esto es, sin necesidad de conocerla.
- XQuery debe soportar tipos simples, como enteros y cadenas, y tipos complejos, como un nodo compuesto por varios nodos hijos.
- Las consultan deben soportar cuantificadores universales (para todo) y existenciales (existe)
- Las consultas deben soportar operaciones sobre jerarquías de nodos y secuencias de nodos.
- Debe ser posible en una consulta combinar información de múltiples fuentes.
- Las consultas deben ser capaces de manipular los datos independientemente del origen de estos.
- Mediante XQuery debe ser posible definir consultas que transformen las estructuras de información originales y debe ser posible crear nuevas estructuras de datos.
- El lenguaje de consulta debe ser independiente de la sintaxis, esto es, debe ser posible que existan varias sintaxis distintas para expresar una misma consulta en XQuery.

4.3. XML y los SGBD

Al hablar de introducir el tema de la consulta en un documento XML, es obligado analizar las posibilidades que existen de emular en este terreno las posibilidades que ofrece el uso de SQL en el modelo relacional. Aunque XQuery y SQL puedan considerarse similares, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional. Por ejemplo, a diferencia de SQL, en XQuery el orden es que se encuentren los datos es importante y determinante, ya que no es lo mismo buscar una etiqueta <B> dentro de una etiqueta <A> que todas las etiquetas <B> del documento (que pueden estar anidadas dentro de una etiqueta <A> o fuera).

Las diferencias existentes entre datos XML y datos relacionales son las siguientes:

- *Metadatos*: Los datos relacionales presentan estructuras regulares y homogéneas (cada fila tiene las mismas columnas con los mismos nombres y tipos) lo que permite usar metadatos sin ningún problema, mientras que en XML los datos son heterogéneos e irregulares con estructuras diferentes que deben describirse caso a caso, de forma que estos metadatos se acaban describiendo en el propio documento.
- *Anidamiento*: Los documentos XML contienen distintos niveles de anidamiento, que son irregulares e impredecibles, mientras que los datos relacionales son “planos” al estar organizados a partir de tablas
- *Jerarquía*: En XML existe una jerarquía y un orden intrínseco que no se da en la estructura relacional, ello se refleja en la forma de trabajar de XPATH. Este orden y jerarquía carece de relevancia en el modelo relacional.
- *Densidad*: Los datos relacionales son densos (cada columna un valor) y los inexistentes se declaran como tales (null), en cambio los datos en XML son dispersos, y la información que no existe sencillamente carece de elemento. Como consecuencia, XML es más libre que el modelo relacional a la hora de enfrentarse con datos ausentes.
- *Mecanismo de consulta*: En XML el resultado de una consulta consiste inevitablemente en una secuencia heterogénea de elementos, atributos y valores primitivos, que a su vez pueden servir de intermediarios para procesar una expresión de mayor nivel, cosa que difiere de SQL, donde toda expresión dentro de una consulta devuelve una tabla. Ello significa que un lenguaje de consulta para XML debe proporcionar necesariamente funcionales constructoras que sean capaces de crear en el proceso estructuras anidadas que pueden ser complejas; se trata de requisitos que en cambio son irrelevantes en el caso relacional con el uso de SQL.

El mercado esta actualmente cargado de productos que soportan el formato XML como formato de entrada / salida. Mientras que esos productos ofrecen ventajas evidentes sobre otros que no soportan XML hay otra clase de productos los cuales se conocen como "XML nativo" que ofrecen significativas ventajas adicionales. Esos productos, que soportan XML en sus arquitecturas internas, son más escalables, confiables, y verdaderamente más ínteroperables que aquellos que solo utilizan XML como un formato para el intercambio de datos.

a) Bases de datos Relacionales habilitadas para XML

Muchos productos soportan actualmente XML como formato de entrada / salida, esto es, pueden traducir sus formatos de datos internos a XML y viceversa. Esos productos "habilitados para XML" tienen muchas ventajas sobre sus competidores

que no soportan XML: Pueden intercambiar datos mas fácilmente con otros productos que estén ejecutando en otras plataformas, y pueden programarse hasta cierto punto mediante código escrito de acuerdo con especificaciones XML. Esto ha producido la amplia utilización de XML como el "pegamento" para conectar los diferentes sistemas existentes en una empresa con otros sistemas dentro de la misma empresa, con los sistemas de sus clientes o proveedores y para presentar datos en vivo a los consumidores vía Internet. Un claro ejemplo de este tipo de utilización para XML es SOAP (Simple Object Access Protocol), un formato para la publicación serial basado en XML que puede ser utilizado para ejecutar mensajería asíncrona y llamadas a procedimientos remotos entre aplicaciones no XML utilizando la infraestructura de Internet.

Las principales ventajas e inconvenientes de almacenar datos XML en una base de datos relacional son:

- Los sistemas de gestión de bases de datos relacionales llevan mucho tiempo en el mercado, por lo que están “maduros” y son ampliamente usados.
- La posibilidad de utilización desde aplicaciones existentes.
- La conversión es sencilla si los datos se generan a partir de un esquema relacional y si se usa XML como formato de intercambio de datos. Si XML no se genera a partir de un esquema relacional, la transformación no es tan sencilla. Se producen problemas en la conversión, especialmente:
  - Elementos anidados
  - Elementos que se repiten (atributos multivaluados)

Los RDBMS habilitados para XML (Oracle 10.2, DB2 XML Extender y SQLServer 2005) son bases de datos tradicionales que definen un nuevo tipo de dato que permite el almacenamiento de información en formato XML. En todos ellos la información XML a almacenar sufre algún tipo de transformación, completamente transparente para el usuario, que en algunos sistemas implica la fragmentación del documento XML a fin de que éste pueda ser almacenado en tablas relacionales que posteriormente podrían ser indexadas y por tanto mejorar el rendimiento de la base de datos durante el proceso de consulta y extracción de este tipo de información. Esta transformación tiene dos planteamientos diferentes:

Uno es el aportado por Oracle y DB2. En ellos el tipo de dato XML (XML-Type en el caso de Oracle y XMLVARCHAR en el de DB2) posee una tabla adjunta en donde se almacena la información contenida en el documento XML (posteriormente se puede crear índices en estas tablas para que su acceso sea más eficiente). Esto exige una previa asociación entre el contenido de los elementos y atributos XML con los campos de dicha tabla.

El otro planteamiento es el utilizado por SQLServer 2005. En éste, el documento XML se almacena en un formato binario en el que los elementos se identifican a través de un número, que actúa a modo de índice, y la información embebida en el documento XML se convierte previamente al correspondiente tipo de dato en virtud de la naturaleza de dicha información.

En cuanto al lenguaje, embebido en las sentencias SQL, utilizado para consultar la información XML, sin duda SQLServer 2005 es el más potente. Esto se debe a que utiliza XQuery, en lugar de XPath como sucede en los otros sistemas.

En todos, además de permitir el almacenamiento XML, también se puede obtener este mismo formato a partir de su información puramente relacional. En este sentido, Oracle es quien mejor implementa esta característica ya que ofrece un

conjunto de funciones, que se incluirían en la sentencia SELECT, encargadas de definir la estructura del documento XML de salida. En los otros sistemas, la estructura del XML obtenido depende por completo de la estructura de la sentencia SQL (orden en que se seleccionen las tablas y los campos en la consulta).

b) Bases de datos nativas XML

"Base de datos XML Nativa" es aquella cuyas estructuras internas de datos se proyectan directamente al formato jerárquico de XML; los usuarios de una base de datos XML nativo no estarán obligados a distinguir entre algún formato externo de "intercambio" y un formato interno "eficiente", tampoco a diseñar aplicaciones que distingan entre "datos de negocios" y "contenido de documentos". En una base de datos XML Nativa, estas distinciones no tienen que hacerse.

Estas bases de datos permiten el almacenamiento, consulta y actualización de información XML. La diferencia fundamental entre ellas (Tamino, X-Hive/DB, dbXML, eXist, Xindice) es el lenguaje empleado tanto para la consulta como para la actualización de la información almacenada. Se observa que en todos se emplea XQuery, XPath o ambos lenguajes para consultar la información. Sin duda en los sistemas donde se utilice XQuery podrían plantearse consultas mucho más complejas que en aquellos que empleen XPath, al ser aquel un lenguaje más potente y versátil. En cuanto al lenguaje empleado para las actualizaciones, todavía no hay uno estandarizado aunque de momento XUpdate es el más extendido. Sin embargo, el hecho de emplear dos lenguajes diferentes, uno para la consulta (XQuery) y otro para la actualización de la información (XUpdate) no parece la solución más acertada, teniendo en cuenta además que el formato del segundo es XML y el del primero no. De ahí que sea preferible utilizar, como sucede en la base de datos Tamino, una extensión no estandarizada de XQuery para realizar las actualizaciones.

En la siguiente tabla se muestra una comparativa entre los lenguajes de consulta y actualización que utiliza cada base de datos.

Gestor de base de datos	Lenguaje de consulta	Lenguaje de actualización
Tamino	XQuery, X–Query (XPath extendido)	extensión de XQuery
X-Hive/DB	XQuery, XPath	XUpdate
dbXML	XPath	XUpdate
eXist	XQuery, XPath	XUpdate
Xindice	XPath	XUpdate

c) Archivos planos.

Otra alternativa de almacenamiento para datos XML sería los archivos planos. Dado que XML es un formato de archivo, es natural almacenar datos XML como un archivo plano. Sin embargo, presentan los inconvenientes asociados a la gestión de ficheros: falta de concurrencia, comprobaciones de integridad, atomicidad, recuperación, seguridad…

4.4. RDF

a) Introducción

RDF (Resource Description Framework) es un lenguaje para representar información sobre recursos en la World Wide Web. Se convirtió en Recomendación del W3C en febrero de 2004. Está especialmente dirigido a la representación de metadatos sobre recursos Web, como el título, autor y la fecha de modificación de una página Web, copyright e información de licencia sobre un documento Web, o la disponibilidad de algún recurso compartido. Sin embargo, mediante la generalización del concepto “recurso Web”, RDF también se puede utilizar para representar información sobre cosas que pueden ser identificadas en la Web, incluso aquellas que no pueden ser directamente recuperadas en la Web.

RDF está orientado a situaciones en las cuales esta información necesita ser procesada por aplicaciones, más que ser solo visualizada por seres humanos. RDF proporciona un marco común para expresar esta información para que pueda ser intercambiada entre aplicaciones sin pérdida de significado. Como es un marco común, los diseñadores de aplicaciones pueden valorar la disponibilidad de *parsers* RDF comunes y herramientas de proceso. La habilidad de intercambiar información entre diferentes aplicaciones quiere decir que la información puede estar disponible para otras aplicaciones diferentes a aquellas para las cuales fue previamente creada.

RDF está basado en la idea de identificar cosas utilizando identificadores Web (llamados *Uniform Resource Identifiers*, o *URIs*), y describir recursos en términos de propiedades simples y valores de dichas propiedades. Esto permite a RDF representar sentencias sencillas sobre recursos como un grafo de nodos y arcos representando los recursos, y sus propiedades y valores.

- Un **recurso** es cualquier cosa que puede tener un URI, esto incluye todas las páginas Web, todos los elementos individuales de cada documento XML y mucho más.
- Una **propiedad** es un recurso que tiene un nombre y que puede usarse como una propiedad, por ejemplo, autor o titulo. En muchos casos todo lo que nos importa en realidad es el nombre, pero una propiedad necesita ser un recurso de forma tal que pueda tener sus propias propiedades
- Una **sentencia** consiste en la combinación de un recurso, una propiedad y un valor. Estas partes son conocidas como el sujeto, predicado y el objeto de la sentencia. Una sentencia es por ejemplo *"El autor de http://www.textuality.com/RDF.why.html es Tim Bray"*. El valor puede ser un string por ejemplo "Tim Bray" o puede ser otro recurso por ejemplo *"El home page de http://www.textuality.com/RDF/Why.html es http://www.textuality.com."*

RDF esta diseñado para tener las siguientes características.

Independencia

Dado que una propiedad es un recurso, toda organización independiente o incluso cada persona puede inventarlas. Podemos inventar una propiedad llamada "Autor" y otros pueden inventar una propiedad llamada "Director" que podría aplicarse, por ejemplo, a recursos asociados con películas. Esta

libertad es necesaria porque no hay ninguna entidad/persona que se encargue de definir cuales son las propiedades a usar en todo el mundo

Intercambio

Dado que las sentencias RDF se escriben en XML pueden ser utilizadas fácilmente para intercambiar información.

Escalabilidad

Las sentencias RDF son simples, son registros con tres campos (Recurso, propiedad, valor) por lo que son fáciles de manejar y de usar para buscar objetos aun en volúmenes realmente grandes. La Web ya es lo suficientemente grande y continúa creciendo. Es probable que tengamos en algún momento miles de millones de RDFs flotando a nuestro alrededor algún día. Por eso la escalabilidad es importante

Las propiedades son recursos

Las propiedades pueden tener sus propias propiedades y pueden ser encontradas y manipuladas como cualquier otro recurso. Esto es importante porque tendremos muchísimos recursos que manejar. Demasiados como para buscarlos uno por uno. Por ejemplo, podríamos tener que saber si alguien tiene definido una propiedad que describa el género de una película, con valores como Comedia u Horror, son necesarios metadatos para resolver esta consulta.

Los valores pueden ser recursos

Por ejemplo, la mayoría de las páginas Web podrían tener una propiedad llamada "home" que apunte al home del sitio. Por lo tanto los valores de sus propiedades que podrían incluir el titulo y autor de la página también tienen que incluir recursos

Las sentencias pueden ser recursos

Las sentencias también tienen propiedades. Dado que no hay nadie que provea un standard para todos los recursos posibles y dado que la Web es demasiado grande como para que cada uno provea el suyo tendremos que realizar búsquedas basadas en los metadatos de otras personas, tal y como se hace hoy con Yahoo. Esto significa que querremos, dada una sentencia como "El tema de esta pagina es monos" poder preguntar "Quien lo dice", o "Cuando". Una forma útil de hace esto es mediante metadatos y por ello las sentencias deben poder tener sus propias propiedades

RDF no hace ninguna suposición sobre el dominio o campo de aplicación. Es tarea de los usuarios definir su propia terminología en un lenguaje de esquemas denominado RDF Schema (RDFS). Hay que hacer notar que RDFS no tiene la misma relación con RDF que XML Schema tiene con XML. XML Schema acota la estructura de documentos XML, mientras que RDF Schema define el vocabulario utilizado en modelos de datos RDF.

Por que no usar solo XML

XML nos permite inventar nuestros propios vocabularios, inventar nuestros propios *tags* que pueden contener texto, datos u otros *tags*. XML sin embargo falla en el objetivo de diseño de escalabilidad para metadatos. Debido a dos problemas:

El orden en el cual los elementos aparecen en un documento XML es significativo y muchas veces necesario. Esto es altamente antinatural en el mundo de los metadatos. Nadie se preocupa sobre si el director o el titulo de la película se lista primero siempre y cuando ambos datos estén disponibles para realizar búsquedas. Además mantener el orden correcto de millones de elementos de datos es caro y difícil.

Diferentes formas de ver una sentencia:

Consideremos la siguiente oración:

"Ana Castelló es la creadora del recurso http://www.programa.org/home/acastelloa

La oración tiene las siguientes partes:

Sujeto (recurso)	http://www.programa.org/home/acastelloa
Predicado (propiedad)	Creador
Objeto (literal)	Ana Castelló

A continuación se va a hacer un diagrama de esta sentencia RDF usando un grafo dirigido y etiquetado. En estos diagramas los nodos (dibujados como óvalos) representan recursos y los arcos representan propiedades con nombre (la etiqueta del arco es el nombre de la propiedad). Los nodos literales se dibujan como rectángulos. Para el ejemplo anterior el diagrama es:

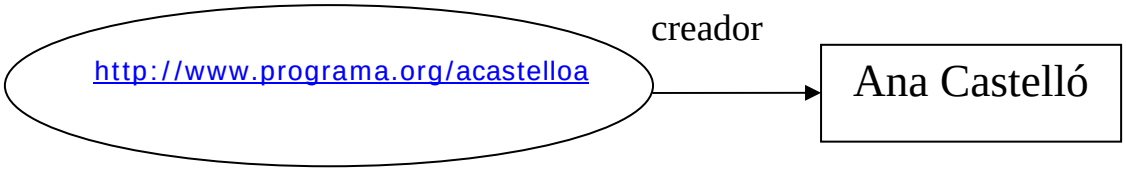

Figura 1. Representación mediante grafos de una tripleta

La dirección del arco es importante, siempre empieza en el sujeto y termina en el objeto de la sentencia. Este tipo de diagramas se lee de la forma TIENE: En este caso "http://www.programa.org/home/acastelloa" tiene como "Creador" a "Ana Castelló"

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.programa.org/acastelloa ">
    <s:Creador>Ana Castelló</s:Creador>
```

```
  </rdf:Description>
</rdf:RDF>
```

Una sentencia RDF raramente aparece en forma aislada, comúnmente varias propiedades de un recurso son indicadas simultáneamente. La sintaxis serializada RDF/XML ha sido diseñada para utilizar esta característica fácilmente agrupando varias sentencias sobre un mismo recurso en un elemento "Description". El elemento "Description" menciona en un atributo "about" el recurso para el cual las sentencias se aplican. Si el recurso no existe aún el elemento "Description" puede asignarle un identificador en el momento usando un atributo ID.

Aunque la sintaxis serializada muestra la estructura de un modelo RDF más claro, normalmente es mejor utilizar una forma XML más compacta. Esto se lleva a cabo a través de la *sintaxis abreviada* de RDF. Como valor añadido, la sintaxis abreviada permite a los documentos seguir DTDs de XML bien estructuradas que se interpretan como modelos RDF.

Ana Castelló es la creador del recurso http://www.programa.org/acastelloa

Se escribe en RDF/XML en la sintaxis abreviada básica de la forma:

```
<rdf:RDF>
      <rdf:Description      about=http://www.programa.org/acastelloa"
      s:Creador="Ana Castelló" />
 </rdf:RDF>
```

Pero existen también otras representaciones de los datos RDF, denominadas serializaciones. Ya se ha visto **RDF/XML** (sintaxis serializada básica y sintaxis abreviada) donde XML es utilizado como medio de transporte o sintaxis; además están: **NTRIPLES** que es sólo un listado de las tripletas; **Notation 3**, una notación que alivia el problema de lo difícil que resulta para los desarrolladores la serialización en XML, etc.

b) Sintaxis RDF basada en XML

Un documento RDF consiste en un elemento rdf:RDF, cuyo contenido es un número de descripciones. Como podemos observar en el ejemplo anterior, se utiliza el mecanismo de espacio de nombres de XML de una forma ampliada. En RDF se espera que los espacios de nombres externos sean documentos RDF que definen recursos, los cuales se utilizan en el documento RDF que los importa. Este mecanismo permite la reutilización de los recursos.

El elemento Description denomina, en un atributo about, el recurso para el cual se aplica cada una de las sentencias [o declaraciones]. Si el recurso no existe todavía (es decir, no tiene todavía un identificador de recursos) el elemento Description puede proporcionar el identificador para el recurso usando el atributoID. El atributo about del elemento: Description es el equivalente a un atributo identificador, pero se utiliza a menudo para sugerir que el objeto sobre el que trata la sentencia ya ha sido definido en otra parte. Formalmente hablando, un conjunto de sentencias RDF forman un gran grafo, relacionando cosas con otras cosas mediante propiedades.

La sintaxis serializada RDF básica toma la forma:

RDF ::= ['<*rdf*:RDF>'] description* ['</*rdf*:RDF>']

description ::= '<*rdf*:Description' idAboutAttr? '>'propertyElt*
'</*rdf*:Description>'
idAboutAttr ::= idAttr | aboutAttr
aboutAttr ::= 'about="' URI-reference '"'
idAttr ::= 'ID="' IDsymbol '"'
propertyElt ::= '<' propName '>' value '</' propName '>'
| '<' propName resourceAttr '/>'
propName ::= Qname
value ::= description | string
resourceAttr ::= 'resource="' URI-reference '"'
Qname ::= [NSprefix ':'] name
URI-reference ::= string, interpreted per [URI]
IDsymbol ::= (any legal XML name symbol)
name ::= (any legal XML name symbol)
NSprefix ::= (any legal XML namespace prefix)
string ::= (any XML text, with "<", ">", and "&" escaped)

El elemento **RDF** es un simple envoltorio que marca los límites en un documento XML entre los que el contenido está dispuesto a ser mapeado a una instancia de modelo de datos RDF. El elemento **RDF** es opcional si el contenido puede entenderse como RDF desde el contexto de la aplicación.

El elemento ***Description*** contiene los elementos sobrantes que posibilitan la creación de sentencias en la instancia [objeto específico de la categoría] del modelo. El elemento ***Description*** puede evocarse (con la finalidad de sintaxis RDF básica) simplemente como un lugar donde mantener la identificación de un recuso descrito. Normalmente habrá más de una sentencia [o declaración] sobre un recurso; el elemento ***Description*** proporciona una forma de dar el nombre justo de una vez a varias sentencias.

Cuando se especifica el atributo ***about*** con ***Description***, la sentencia [declaración] en el elemento ***Description*** se refiere al recurso cuyo identificador determina el elemento ***about***. El valor del atributo ***about*** se interpreta como una referencia-URI. El identificador de recursos correspondiente se obtiene resolviendo la referencia-URI a la forma absoluta. Un elemento ***Description*** sin un atributo ***about*** representa un nuevo recurso. Tal recurso podría ser un substituto, o delegado [proxy], para algunos recursos físicos que no tienen un URI reconocible. El valor del atributo ***ID*** del elemento ***Description***, en el caso de que se presente, es el link id de ese recurso"in-line".

Si otro elemento ***Description*** o valor de propiedad necesita referirse a el recurso in-line utilizará el valor del ***ID*** de dicho recurso en su propio atributo ***about***. El atributo ***ID*** indica la creación de un nuevo recurso y el atributo ***about*** se refiere a un recurso existente; por ello tanto el atributo ***ID*** como ***about*** pueden especificarse en el elemento ***Description*** pero no los dos juntos en el mismo elemento. Los valores para cada atributo ***ID*** no deben aparecer en más de un atributo ID dentro del mismo documento.

Un único elemento ***Description*** puede contener más de un elemento ***propertyElt*** con el mismo nombre de propiedad. Cada uno de **dichos *propertyElt*** añade un arco al gráfico. La interpretación de esta representación gráfica se definirá por el diseñador del esquema.

Dentro del elemento ***propertyElt***, el atributo ***resource*** especifica que otros recursos son el valor de esta propiedad; es decir, el objeto de la sentencia [o declaración] es otro recurso identificado por un URI preferentemente a la indicación

por un literal. El identificador del recurso del objeto se obtiene resolviendo el URI-de referencia del atributo ***resource*** de la misma forma que se mencionó anteriormente para el atributo ***about***. Los ***Strings*** deben ser XML bien formados; el XML convencional contiene mecanismos de citación y disgregación que pueden usarse si la serie de caracteres string contiene secuencias de caracteres (ej. "<" and "&") que transgreden las reglas de buena formación o que podría parecer "marcado".

Los nombres de propiedad pueden asociarse con un esquema. Esto puede hacerse cualificando los nombres de los elementos con un prefijo de *namespace* que asocie con claridad la definición de propiedad con el esquema RDF correspondiente o declarando un namespace por defecto.

El ejemplo anterior

Ana Castelló es la creador del recurso http://www.programa.org/acastelloa

se representa en RDF/XML:

```
<rdf:RDF>
  <rdf:Description about=" http://www.programa.org/acastelloa ">
    <s:Creator>Ana Castello</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Aquí el prefijo del namespace 's' se refiere a un prefijo específico elegido por el autor de esta expresión RDF y definido en una declaración XML del namespace como ésta:

```
xmlns:s="http://description.org/schema/"
```

Esta declaración del namespace podría incluirse normalmente como un atributo XML en el elemento ***rdf:RDF*** pero también puede incluirse con un elemento ***Description*** específico o incluso una expresión ***propertyElt*** concreta. El URI del nombre del namespace, en la declaración del namespace, es un identificador único universal para un esquema particular que la persona que define los metadatos [este autor de los metadatos] utiliza para definir el uso de la propiedad ***Creator***. Otros esquemas pueden definir igualmente la propiedad denominada ***Creator*** y las dos propiedades se diferenciarán gracias a sus identificadores de esquema. Nótese también que un esquema normalmente define también varias propiedades; una única declaración de namespace será suficiente para crear un amplio vocabulario de propiedades que podrán usarse.

El documento XML completo que contiene la descripción citada anteriormente, podría ser:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.programa.org/acastelloa ">
    <s:Creator>Ana Castello</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Utilizando la sintaxis de namespace definida por defecto en [NAMESPACES] para el propio namespace de RDF, este documento podría expresarse también así:

Además, la declaración de namespace puede asociarse con un elemento **Description** concreto e incluso un elemento ***propertyElt*** particular como en este ejemplo:

```
<?xml version="1.0"?>
<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Description about="http://www.programa.org/acastelloa ">
        <s:Creator                xmlns:s="http://description.org/schema/">Ana
        Castello</s:Creator>
  </Description>
</RDF>
```

Contenedores

Frecuentemente es necesario referirse a una colección de recursos, por ejemplo para decir que un trabajo fue realizado por más de una persona, o para listar los alumnos de un curso. RDF define contenedores para manejar listas de recursos o literales.

- **Bag**: Un *bag* es una lista de recursos o literales sin orden. Se permiten valores duplicados y no hay diferencias en cuanto al orden en que aparecen los recursos.
- **Sequence**: Una secuencia es una lista ordenada de recursos o literales. Una secuencia se usa para declarar un conjunto en el cual el orden de aparición de los miembros es importante por ejemplo para mantener el orden alfabético o de importancia. Se permiten valores duplicados.
- **Alternative**: Es una secuencia de recursos o literales para un "único" valor o propiedad, de la lista de recursos o literales debe elegirse uno. Por ejemplo para elegir el lenguaje de un texto o un *mirror site*.

La sintaxis de un contenedor RDF toma la forma:

```
container      ::= sequence | bag | alternative
sequence       ::= '<rdf:Seq' idAttr? '>' member* '</rdf:Seq>'
bag            ::= '<rdf:Bag' idAttr? '>' member* '</rdf:Bag>'
alternative    ::= '<rdf:Alt' idAttr? '>' member+ '</rdf:Alt>'
member         ::= referencedItem | inlineItem
referencedItem ::= '<rdf:li' resourceAttr '/>'
inlineItem     ::= '<rdf:li>' value '</rdf:li>'
```

Los contenedores pueden usarse en todos los casos en que se permita un elemento **Description**:

```
RDF            ::= '<rdf:RDF>' obj* '</rdf:RDF>'
value          ::= obj | string
obj            ::= description | container
```

Nótese como RDF/XML utiliza **li** como elemento de conveniencia para evitar que tenga cada miembro un número explícito. El elemento **li** asigna las propiedades **_1**, **_2**, y así si es necesario. El nombre del elemento **"li** se eligió para que fuera mnemotécnico con el término "list item" del HTLM.

Se necesita un contenedor **Alt** para tener al menos un miembro. Este miembro se identificará con la propiedad _1 y es el valor por defecto o preferido.

Por ejemplo sea la siguiente sentencia:

Los estudiantes del curso 4500 son Juan, José y Pedro

La sintaxis RDF/XML es:

```
<rdf:RDF>
  <rdf:Description about="http://www.una.edu/curso/4500">
    <s:estudiantes>
      <rdf:Bag>
       <rdf:li resource="http://www.una.edu/students/Juan" />
       <rdf:li resource="http://www.una.edu/students/Jose" />
       <rdf:li resource="http://www.una.edu/students/Pedro" />
      </rdf:Bag>
    </s:estudiantes>
</rdf:RDF>
```

Declaraciones sobre declaraciones [sentencias sobre sentencias]

Además de hacer sentencias sobre los recursos Web, RDF puede usarse para crear sentencias sobre otras declaraciones RDF; nos referiremos a éstas como sentencias de alto nivel [*higher-order statements*]. Para realizar declaraciones sobre otras declaraciones, tenemos que construir un modelo de la sentencia original; este modelo es un nuevo recurso al que podemos anexar propiedades adicionales.

Modelado de sentencias: Las sentencias se hacen sobre los recursos. Un modelo de una sentencia es el recurso que necesitamos para poder hacer una nueva declaración (una sentencia de alto nivel) sobre la declaración modelada.

Por ejemplo, consideremos la frase:

Ana Castelló es la creador del recurso http://www.programa.org/acastelloa

RDF estimaría esta sentencia como un hecho. Si, en lugar de eso, escribimos la frase:

Pepito Perez dice que Ana Castelló es la creador del recurso http://www.programa.org/acastelloa

no hemos dicho nada sobre el recurso *://www.programa.org/acastelloa*; en cambio, hemos expresado el hecho sobre la afirmación que hizo Pepito. Para expresar este hecho en RDF, tenemos que modelar la declaración original como un recurso con cuatro propiedades. Este proceso se denomina formalmente "*reification" [transformación de algo abstracto en concreto]* en el contexto de la Representación del Conocimiento. Un modelo de sentencia se denomina "*reified statement" [sentencia transformada de lo abstracto a lo concreto].*

RDF define las siguientes propiedades para modelar sentencias:

subject [sujeto] La propiedad *subject* identifica el recurso que describe la sentencia modelada; es decir, el valor de la propiedad *subject* es el recurso sobre el cual se hizo la sentencia original (en nuestro ejemplo, *http://www.programa.org/acastelloa*).

predicate [predicado] La propiedad *predicate* identifica la propiedad original en la declaración modelada. El valor de la propiedad *predica* es un recurso que representa la propiedad específica en la sentencia original (en nuestro ejemplo, el creador **creator**).

object [objeto] La propiedad *object* identifica el valor de la propiedad en la sentencia modelada. El valor de la propiedad *object* es el objeto en la sentencia original (en nuestro ejemplo "Ana Castelló").

type [tipo] El valor de la propiedad *type* describe el tipo del nuevo recurso. Todas las sentencias transformadas [de lo abstracto a lo concreto] son instancias de RDF: sentencias; es decir tienen una propiedad *type* cuyo objeto es RDF:sentencia. La propiedad *type* se usa más comúnmente también para declarar el tipo del recurso.

Un nuevo recurso con las cuatro propiedades anteriores representa la sentencia original y puede tanto usarse como objeto de otras declaraciones como tener una sentencia adicional sobre él. El recurso con estas cuatro propiedades no es un substituto de la sentencia original, es un modelo de la declaración. Una sentencia [declaración] y sus correspondientes sentencias transformadas [de lo abstracto a lo concreto, *reified*] existe independientemente sin las otras. La representación gráfica RDF se presenta para contener el hecho de la sentencia si y sólo si dicha sentencia se presenta en forma gráfica, sin considerar si la sentencia correspondiente transformada está presente.

Para modelar el ejemplo anterior, podemos adjuntar otra propiedad a la sentencia transformada (es decir, **"attributedTo"**) con un valor apropiado (en este caso "Pepito Perez"). Utilizando la sintaxis de nivel básico RDF/XML, podría escribirse así:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://description.org/schema/">
  <rdf:Description>
    <rdf:subject resource=" http://www.programa.org/acastelloa " />
    <rdf:predicate resource="http://description.org/schema/Creator" />
    <rdf:object>Ana Castello</rdf:object>
<rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"
/>
    <a:attributedTo>Pepito Perez</a:attributedTo>
  </rdf:Description>
</rdf:RDF>
```

La *reification* [transformación de algo abstracto en algo concreto] se necesita también para representar explícitamente en el modelo la sentencia de agrupación implícita en el elemento **Description**. El modelo gráfico RDF no necesita una construcción especial para los elementos **Description**; dado que los **Descriptions** son en realidad colecciones de sentencias, se usa un contenedor **Bag** para indicar que un conjunto de sentencias que provienen del mismo elemento (sintáctico) **Description**. Cada sentencia dentro de un **Description** se transforma [reified] y cada una de las sentencias transformadas es miembro del **Bag** que representa dicha **Description**

Nótese el nuevo atributo **bagID**. Este atributo especifica el recurso **id** del recurso contenedor:

```
description    ::= '<rdf:Description' idAboutAttr? bagIDAttr? propAttr* '/>'
                 | '<rdf:Description' idAboutAttr? bagIDAttr? propAttr* '>'
                     propertyElt* '</rdf:Description>'
bagIDAttr      ::= 'bagID="' IDsymbol '"'
```

BagID e **ID** no deben confundirse. **ID** especifica el identificador de un recurso in-line cuyas propiedades se detallan más ampliamente en la descripción [elemento Description]. **BagID** especifica la identificación del recurso contenedor cuyos miembros son sentencias transformadas [de lo abstracto a lo concreto] sobre otro recurso. Un elemento **Description** puede tener ambos atributos **ID** y **bagID**.

Abreviatura sintáctica para sentencias sobres sentencias

Dado que añadir un **bagID** a un elemento **Description** conlleva incluir en el modelo un Bag de la sentencia transformada [de lo abstracto a lo concreto] de **Description**, podemos usar esta como una abreviatura sintáctica cuando hagamos sentencias sobre sentencias. Por ejemplo, se queremos decir que Pepito Perez afirma que Ana Castelló es el creador de *http://www.programa.org/acastelloa* y que el título de dicho recurso es "La página de Ana", simplemente podemos añadir al ejemplo anterior:

```
<rdf:Description aboutEach="#D_001">
  <a:attributedTo>Pepito Perez</a:attributedTo>
</rdf:Description>
```

c) RDF SCHEMA

Los esquemas RDF establecen un sistema de tipos simples para definir vocabularios RDF que son un conjunto de recursos, propiedades y clases. RDFS define una notación de clases y permite la definición y descripción de nuevas clases. Clases y propiedades pueden tener subclases y subpropiedades, esto permite la definición de jerarquías. Las propiedades pueden ser restringidas a una clase tanto en el dominio como en el rango. RDFS es un vocabulario RDF. Clases y propiedades son iguales que otros recursos, así pueden ser descritos usando RDF.

El vocabulario RDFS dice como describir el uso con significado de propiedades y clases en datos RDF. Por ejemplo, un vocabulario RDF puede describir limitaciones en el tipo de valores que son apropiados para algunas propiedades, o las clases para las cuales tiene sentido atribuir tales propiedades. El lenguaje RDFS proporciona un mecanismo para describir esta información (usando rdfs:domain y rdfs:range), pero *no dice como* debe utilizarla una aplicación. Por ejemplo, mientras un esquema RDF puede afirmar que una propiedad del autor se utiliza para indicar los recursos que son instancias de clase persona, no dice cómo una aplicación debe actuar en el procesado de esa información. Distintas aplicaciones utilizarán esta información de maneras distintas. Por ejemplo, una herramienta de verificación de datos puede usar esto para ayudar a encontrar errores en algún conjunto de datos, un editor interactivo puede sugerir los valores apropiados, y una aplicación de razonamiento puede utilizarlo para inferir información adicional desde instancias de datos.

4.5 LENGUAJES DE CONSULTA PARA RDF

Los lenguajes de consulta para RDF deberían tener en cuenta las características de RDF que se muestran a continuación:

Modelo de datos abstracto. La estructura subyacente de cualquier documento RDF es una colección de tripletas. Esta colección se suele llamar el Grafo RDF. Cada tripleta establece una relación entre dos nodos en el grafo. Este modelo abstracto de datos es independiente de la serialización de la sintaxis. Además, los lenguajes de consulta normalmente no proporcionan habilidad para hacer consultas sobre características específicas de la serialización, por ejemplo, sobre el orden de la serialización.

Semántica formal e inferencia. RDF tiene una semántica formal que proporciona una base dependiente para razonar sobre el significado de un grafo RDF. Este razonamiento se suele llamar "entailment". Las reglas de "entailment" especifican que información implícita se puede inferir de información explícita.

Soporte para tipos de datos XML Schema. Los tipos de datos XML se pueden utilizar para representar valores de datos en RDF. XML Schema también proporciona un marco extensible para definir nuevos tipos de datos que se puedan usar en RDF. Estos tipos de datos deberían ser soportados en un lenguaje para consultas RDF.

Soporte para hacer sentencias libres sobre recursos. En general, no se puede asumir que toda la información sobre cualquier recurso esté disponible en una consulta RDF. Un lenguaje de consulta debería tener esto en cuenta, y debería tolerar información incompleta o contradictoria.

Además, las siguientes propiedades de cada lenguaje también deben tenerse en cuenta:

- Expresividad. Indica como se pueden formular consultas potentes en un lenguaje dado. Típicamente, un lenguaje debería como mínimo proporcionar los resultados ofrecidos por el álgebra relacional. Normalmente, la expresividad se restringe para mantener otras propiedades como seguridad y para permitir la ejecución eficiente y optimizable de consultas.
- Cierre. Esta propiedad requiere que los resultados de una operación sean de nuevo elementos del modelo de datos. Esto quiere decir que si un lenguaje de consulta opera en el modelo de datos de los grafos, los resultados de la consulta deberían ser de nuevo grafos.
- Adecuación. Un lenguaje de consulta se denomina adecuado i utiliza todos los conceptos del modelo de datos subyacente. Esta propiedad además complementa la propiedad del cierre: Para el cierra, el resultado de la consulta no puede salirse del modelo de datos, para la adecuación, se necesita explotar todo el modelo de datos.
- Ortogonalidad. La ortogonalidad de un lenguaje de consulta requiere que todas las operaciones puedan se usadas independientemente del contexto de uso.
- Seguridad. Un lenguaje de consulta se considera seguro si cada consulta que es sintácticamente correcta devuelve un conjunto de resultados finito (sobre un conjunto de datos finito

A continuación se muestra una comparación de varios lenguajes de consulta para RDF, subrayando sus características comunes y diferencias entre las dimensiones

generales para los lenguajes de consulta y los requerimientos en particular para RDF.

RQL

Para una explotación total del conocimiento almacenado en datos RDF/RDFS, se requiere un lenguaje de consulta a nivel semántico (debe ser sensible a la semántica de las primitivas RDF/S). RQL es un lenguaje de consulta declarativo para RDF/S que explícitamente captura esta semántica en su diseño. RQL fue desarrollado en el instituto ICS-FORTH, y su potencia semántica está basada en la evaluación de caminos de expresiones sobre grafos RDF. Permite el uso de variables tanto para denotar nombres de nodos (es decir, clases), como arcos (es decir propiedades). Permite consultar esquemas RDF y descripciones RDF (es decir, instancias) en una misma consulta. RQL está definido por medio de un conjunto de consultas básicas e iteradores que permiten construir otras consultas a través de una composición funcional.

RQL se apoya sobre un modelo de gráfico formal que permite la interpretación de Descripciones de recursos por medio de uno o más esquemas. La novedad de RQL reside en su habilidad para combinar con facilidad consultas de esquemas y datos mientras explota las taxonomías de etiquetas y la clasificación múltiple de los recursos. RQL sigue una sintaxis similar a OQL. RQL es ortogonal, pero no es cerrado, y las consultas devuelven datos variables en vez de grafos. Sin embargo, la semántica RQL no es completamente compatible con la semántica RDF: Hay restricciones adicionales que se aplican a los modelos RDF para permitir consultas RQL.

SeRQL

SeRQL (Sesame RDF Query Languaje) es un lenguaje de consulta y trasformación basado en diferentes lenguajes existentes, sobre todo RQL, RDQL y N3. Sus principales objetivos de diseño, son la unificación de las mejores prácticas para los lenguajes de consulta y proporcionar un lenguaje de consultas “ligero”. Su sintaxis es similar a la de RQL, aunque se han hecho modificaciones para que el lenguaje sea más fácil de procesar. Al igual que RQL, SeRQL está basado en una interpretación formal de los grafos RDF, pero la interpretación formal de SeRQL está basada directamente en el modelo teórico de RDF.

SeRQL soporta expresiones path generalizadas y constantes booleanas, así como dos filtros básicos: select-from-where y construct-from-where. El primero devuelve una tabla variable, el segundo devuelve un subgrafo. Por lo tanto, las consultas construct-from-where proporcionan el cierre y la ortogonalidad, y además permite la composición de consultas. SeRQL no es seguro, ya que proporciona varias funciones recursivas.

TRIPLE

El término Triple denota tanto el lenguaje de consulta y sus reglas como la aplicación en si. El lenguaje deriva de F-Logic. Las tripletas RDF (S, P, O) están representadas como expresiones F-Logic S [P->O], las cuales pueden ser anidadas.

Por ejemplo, la expresión S[P1->O1, P2->O2[P3->O3]] se corresponde con las tripletas:
(S,P1,O1), (S,P2,O2), y (O2,P3,O3).

Triple no distingue entre reglas y consultas, que son simplemente reglas sin encabezado, donde los resultados son los valores que se corresponden de las variables libres en la consulta. Por ejemplo: FORALL X <- (X[rdfs:label->"arbol"])@default:ln. Devuelve todos los recursos que contienen la etiqueta “arbol”.

Debido a que el output es una tabla de variables y sus posibles vínculos, TRIPLE no cumple la propiedad del cierre. Además, tampoco es seguro, ya que permite reglas no seguras ,como: FORALL X (X[rdfs:label->"arbol"] <- (a[rdfs:label->"arlbol"])@default:ln..

Mientras que TRIPLE es adecuado y cerrado para su propio modelo de datos, el mapeo de RDF a TRIPLE es no tiene pérdidas. Por ejemplo, los nodos anónimos RDF se hacen explícitos. TRIPLE es capaz de manejar varios modelos RDF simultáneamente, los cuales se identifican mediante el sufijo @model.

TRIPLE no codifica semánticas RDF fijas. Las semánticas tienen que ser especificadas mediante un conjunto de reglas a lo largo de la consulta. TRIPLE no soporta conjuntos de datos.

RDQL

La sintaxis de RDQL sigue un patrón como SQL, donde una cláusula FROM se omite. Por ejemplo select ?p where (?p, <rdfs:label>, "arbol") devuelve todos los recursos con la etiqueta arbol en la variable libre p. La cláusula SELECT al principio de la query permite proyectar las variables. Las abreviaciones de los namespace pueden ser definidas en una consulta mediante una cláusula using independiente. La información RDF Schema no es interpretada. Ya que la salida es una tabla de variables, y sus posibles correspondencias, RDQL no cumple por completo la propiedad del cierre y la ortogonalidad. RDQL es seguro y ofrece un soporte prelimitar para tipos de datos.

N3

Notation3 (N3) proporciona una sintáxis basada en texto para RDF. Sin embargo, el modelo de datos N3 se ajusta al modelo de datos RDF. Además, N3 permite definir reglas, las que se denotan utilizando una sintaxis especial, por ejemplo: ?y rdfs:label "arbol"
?y a :QueryResult.
Estas reglas, aunque no son un lenguaje de consulta per se, pueden ser utilizadas con el propósito de consulta. Para este propósito las consultas tienen que ser almacenadas como reglas en un archivo dedicado, el cual se utiliza conjuntamente con los datos. Los comandos CWM Filter permiten automáticamente seleccionar los datos que son generados por reglas. Incluso aunque N3 cumple las propiedades de cierre, seguridad y ortogonalidad, la utilización de N3 como lenguaje de consulta es incómodo.

Versa

Versa tiene una característica interesante, y es que el bloque principal del lenguaje es una lista de recursos RDF. Las tripletas RDF tienen un rol en las llamadas operaciones transversas, que tienen la forma ListExpr - ListExpr -> BoolExpr. Estas expresiones devuelven una lista de todos los objetos que cumplen las tripletas. Por ejemplo, la expresión trasversal : all() - rdfs:label -> *
Devolvería una lista conteniendo todas las etiquetas.

En una expresión transversal, podemos seleccionar alternativamente los sujetos poniendo una barra vertical al principio del símbolo flecha. Así: all() |- rdfs:label -> eq("arbol") devolvería como resultado todos los recursos que contengan la etiqueta “arbol”. El hecho de que una expresión transversa se utiliza como una lista de expresiones, permite anidar expresiones para crear consultas más complejas.

El árbol de estructura de datos y de expresiones hace que sea complicado proyectar varios valores de una vez. Versa utiliza el operador de la distribución para superar esta limitación. Crea una lista de listas, lo que permite seleccionar varias propiedades de una lista de recursos dada.

Versa ofrece soporte para reglas, ya que permite transversar predicados transitivamente. No tiene implementadas vistas, múltiples modelos y manipulación de datos. Sin embargo, versa cumple los criterios de ortogonalidad y seguridad.

SPARQL

El protocolo SPARQL y el lenguaje de consulta SPARQL están actualmente bajo discusión en el W3C Working Draft. El pasado 28 de noviembre de 2005 han publicado un nuevo borrador. En estos momentos SPARQL es el lenguaje de consulta más potente para RDF/OWL. Además la tendencia es clara y buena parte de los sistemas de almacenamiento de RDF soportan ya SPARQL.

Está basado en anteriores lenguajes de consulta para RDF, como dfDB, RDQL, y SeRQL, y aporta nuevas características muy importantes. Existen varias implementaciones de SPARQL en diferentes plataformas y lenguajes, como ARQ, que implementa el lenguaje para Jena, y SPARQL para Sesame.

Proporciona las siguientes facilidades:

- Extrae información en forma de URIs, nodos en blanco y literales planos y tipados.
- Extrae subgrafos RDF
- Construye nuevos grafos RDF basados en la información de los grafos consultados.

Está basado en la correspondencia entre patrones de grafos. El patrón más simple de un grafo es el patrón triple, que es como una tripleta RDF pero con la posibilidad de una variable en cualquier posición en el sujeto, predicado u objeto. Con esta combinación se proporciona un patrón de gráfico básico.

Por ejemplo, un par de características especialmente interesantes son:

- CONSTRUCT: que permite construir un grafo de respuesta en vez de devolver la típica tabla de variables y valores de SQL.
- DESCRIBE: que permite recuperar todos los metadatos asociados a un recurso (URI).

5. SGDB CON SOPORTE PARA RDF

5.1. INTRODUCCIÓN

RDF se convirtió en una recomendación del W3C en Febrero del 2004. Es un estándar para representar información que puede ser identificada utilizando un URI (Universal Resource Identifier). En particular, está orientado a la representación de metadatos en los recursos Web. Se utiliza para representar datos en numerosas áreas, como bibliotecas digitales, ciencias naturales, comercio electrónico, etc. RDF proporciona la base para la construcción de la Web Semántica. La simplicidad de su estructura promociona la interoperabilidad entre aplicaciones, y su formato fácilmente compresible por los computadores facilita el procesado automatizado de los recursos Web.

Para representar datos en RDF, cada sentencia está dividida en una tripleta de la forma <sujeto, predicado, objeto>. Esta tripleta se modela como un gráfico dirigido: El *sujeto* y el o*bjeto* de la tripleta son nodos, y el *predicado* (o propiedad) como una conexión dirigida que describe la relación entre los nodos. La dirección de la conexión siempre apunta hacia el objeto.

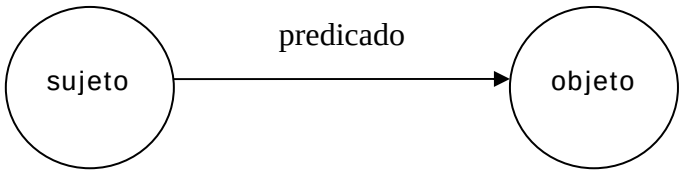

Fig. 1. Representación mediante grafos de una tripleta

En este momento, hay disponible una gran variedad de sistemas de almacenamiento para RDF: Jena2[34], Kowari[35], Sesame[36], Oracle[37], etc. Muchos de los sistemas implementan el almacenamiento persistente utilizando bases de datos relacionales, donde los datos se almacenan en tablas planas relacionales; las tripletas se almacenan en una tabla de sentencias con links a las tablas de soporte; esta tabla de sentencias tiene columnas para el sujeto, predicado y el objeto, y cada fila representa una tripleta.

El almacenamiento de los metadatos en una base de datos relacional típicamente requiere un diseño con pocas tablas (desde una hasta seis), muy estrechas y largas. Cada tabla tendría de dos a cinco columnas, la mayoría de ellas indexadas más de una vez para conseguir mayor rendimiento. Una consulta básica requeriría que todas las tablas fueran unidas con otras tablas para realizar joins anidados. Pueden ser necesarias una gran cantidad de consultas individuales para realizar consultas más complejas, lo que redundaría en bajos tiempos de respuesta.

[34] http://jena.sourceforge.net/
[35] http://www.kowari.org/
[36] http://www.openrdf.org/
[37] http://www.oracle.com/index.html

Actualizar metadatos en una base de datos relacional es costoso, ya que todas las columnas están indexadas. Para hacerlo se requiere que las páginas de datos y las columnas indexadas se bloqueen.

Las actualizaciones en una base de datos relacional se ralentizan cuando múltiples bloqueos de datos intentan obtener un mismo bloqueo de página ("hot spot"). Estos tipos de bloqueos también afectan el rendimiento de la lectura y el mantenimiento.

Los optimizadores de consultas de bases de datos relacionales no funcionan bien con tablas largas y estrechas, con gran variación en la distribución de datos. Esto sucede porque las columnas que almacenan una gran variedad de datos (como en los metadatos) están diluidas con respecto a las columnas con una unicidad alta (como una columna con números de teléfono). Además, algunos optimizadores de RDBMS están diseñados para examinar todos los posibles planes de ejecución basados en reglas y costes. Ya que las consultas de metadatos tienden a ser recursivas y tienen numerosos caminos, a los optimizadores no les resulta sencillo determinar el mejor plan de ejecución. En algunos casos determinar el plan de ejecución lleva mas tiempo que ejecutar la consulta.

El mantenimiento en caché de información para que sea útil a las consultas posteriores es también un problema. La mayoría de las bases de datos relacionales cachean las tablas recientemente accedidas. Esto no se ajusta a las bases de datos que almacenan metadatos, ya que la tabla relevante es estrecha y profunda.

El mayor problema con el almacenamiento de metadatos en bases de datos relacionales es la seguridad. Se necesitan tablas adicionales en el diseño para asegurar los metadatos a nivel de tabla y columnas. Esto incrementa el número de joins realizados y la complejidad de las consultas.

5.2. ORACLE 10G:

Oracle en su release 10.2 introduce un nuevo tipo de objeto (SDO_RDF_TRIPLE_S) para almacenar datos RDF. Cada tripleta RDF <sujeto, propiedad, objeto> se trata como un único objeto de la base de datos. Como consecuencia, un único documento RDF con varias tripletas dará como resultado múltiples objetos de la base de datos. Las tripletas RDF son mapeadas en un grafo mediante el almacenamiento de los sujetos y objetos como nodos, y las propiedades como links. El almacenamiento de los datos RDF lo gestiona Oracle: todos los datos RDF se gestionan desde un esquema central, y se proporcionan funciones de acceso a nivel de usuario y constructores para hacer consultas y actualizar los datos RDF.

Las tripletas RDF se almacenan en una base de datos Oracle como una red lógica (utilizando Oracle Spatial Network Data Model - NDM), soportando grafos dirigidos y no dirigidos. NDM es la solución propuesta por Oracle para almacenar, manipular y analizar redes o grafos en la base de datos.

Las tripletas son parseadas y almacenadas en el sistema como entradas en las tablas *rdf_node$* y *rdf_link$* Los nodos en la red RDF se almacenan una sola vez y se reutilizan cuando se encuentran en nuevas tripletas. En las tablas de las aplicaciones definidas por el usuario solo se almacenan referencias en el objeto SDO_RDF_TRIPLE_S para señalar a la tripleta almacenada en el esquema central.

Otra peculiaridad es que Oracle implementa la reificación (descripción de una tripleta RDF utilizando vocabulario RDF) utilizando *Oracle XML DB DBUri*, que referencia directamente las tripletas en la base de datos: sólo se almacena una nueva tripleta por cada reificación.

Las ventajas que se derivan de gestionar los datos RDF como objetos de la base de datos son las siguientes:

- Simplifica la modelización de aplicaciones RDF.
- Los datos RDF se pueden integrar fácilmente con otros datos de la empresa.
- La reusabilidad de los objetos RDF hace posible que se desarrollen nuevas aplicaciones con más facilidad.
- La abstracción de los objetos y la encapsulación de los comportamientos RDF facilita la comprensión y el mantenimiento de las aplicaciones.
- No se requieren mapeos entre los objetos RDF de la parte del cliente y las tablas y columnas de las bases de datos que contienen las tripletas.
- No se requieren configuraciones adicionales para almacenar datos RDF.

Oracle soporta los siguientes conceptos RDF: modelo de datos en grafos, vocabularios basados en URI, tipos de datos RDF y reificación. También soporta un subconjunto de consultas típicas en SQL.

Oracle Spatial RDF Data Model.

En Oracle Database 10g Release 2 se ha desarrollado un Nuevo modelo para almacenar los datos RDF y OWF. Esta funcionalidad reside en el Oracle Spatial Network Data Model (NDM). El modelo de datos RDF soporta tres tipos de objetos de base de datos: modelo, conjunto de reglas e índice de reglas.

Modelo

Hay solo un universo para todos los datos RDF almacenados en la base de datos. Todas las tripletas RDF son parseadas y almacenadas en el sistema como entradas

en tablas bajo el esquema MDSYS (Fig. 3). Una tripleta RDF (sujeto, predicado, objeto) se trata como un objeto de la base de datos.

RDF_MODEL$ es una tabla de sistema creada para almacenar información en todos los modelos RDF y OWL de la base de datos. Cuando se crea un nuevo modelo RDF, se genera automáticamente un MODEL_ID y se añade una entrada en la tabla del modelo.

La tabla RDF_NODE$ almacena el VALUE_ID para valores de texto que participan en sujetos u objetos de las sentencias. El NODE_ID es el mismo que VALUE_ID. Los valores NODE_ID son almacenados una sola vez, independientemente del número de sujetos u objetos en los que participe. La tabla de nodos permite a los datos RDF ser expuestos a todas las funciones analíticas y APIS disponibles en NDM.

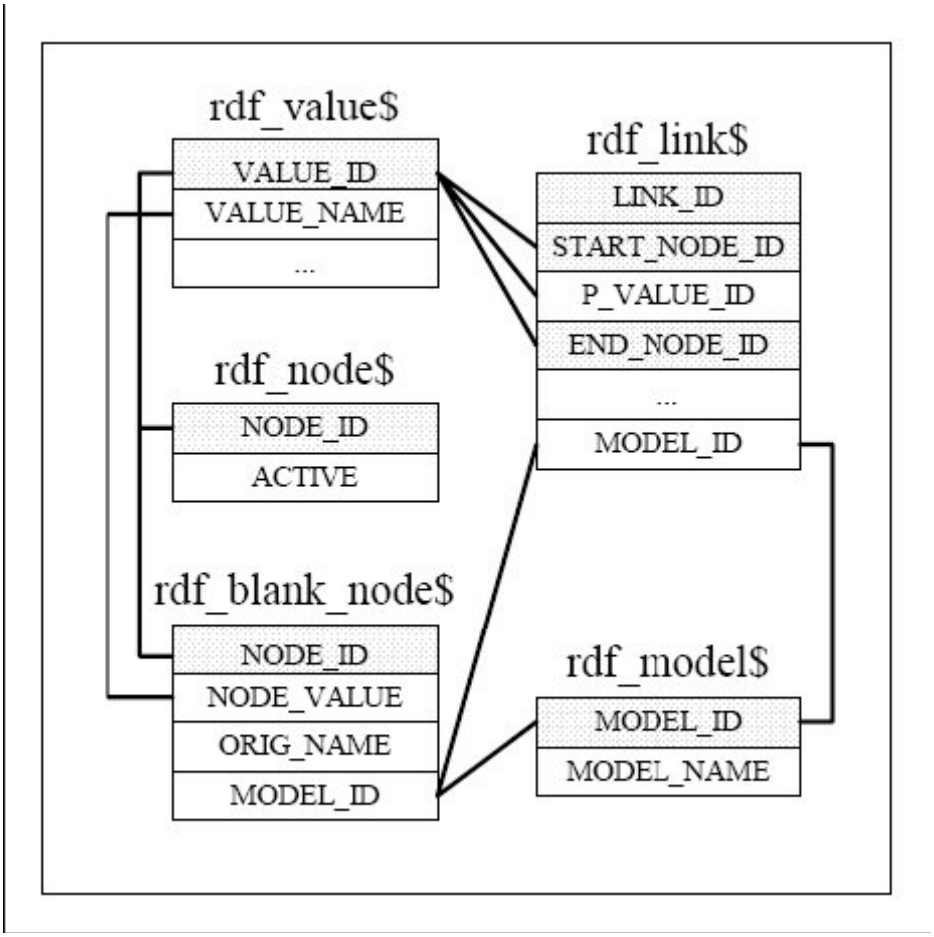

Fig 2. RDF_VALUE$, RDF_NODE$, RDF_LINK$ y RDF_MODEL$ son las tablas necesarias para almacenar RDF en el Oracle RDF Data Model.

La tabla LINKS$ almacena las tripletas para todos los modelos de la base de datos. Además, el MODEL_ID lógicamente particiona la tabla RDF_LINK$. Seleccionando todos los links de un MODEL_ID específico, devuelve la red RDF de ese modelo en particular.

La tabla RDF_VALUE$ almacena los valores de texto, es decir, el URI o los literales para cada parte de la tripleta. Cada valor de texto se almacena una sola vez, y un único VALUE_ID se genera para las entradas de texto. URIs, nodos en blanco, literales planos y literales tipados son diferentes posibles entradas de VALUE_TYPE.

Los nodos en blanco se utilizan para representar objetos desconocidos, y cuando la relación entre un nodo sujeto y otro nodo es n-aria. Los nuevos nodos en blanco se

generan automáticamente si se encuentran en las tripletas. Sin embargo, es posible que los usuarios reutilicen nodos en blanco, por ejemplo, cuando se insertan datos en contenedores o colecciones. La tabla RDF_BLANK_NODE$ almacena los nombres originales de los nodos en blanco que se reutilizan cuando se encuentran en las tripletas.

Para representar una sentencia reificada, se crea un recurso utilizando el LINK_ID de la tripleta. El recurso puede ser utilizado como el sujeto o el objeto de una sentencia. Para procesar una sentencia reificada, en primer lugar, en el sujeto de una nueva tripleta se inserta el valor del recurso de la sentencia, como propiedad rdf:type y como objeto: rdf:Statement. Entonces, una nueva tripleta se introduce por cada aserción sobre la sentencia reificada. Sin embargo, cada sentencia reificada tendrá solo un rdf:type asociado a un rdf:Statement, independientemente del numero de aserciones hecho utilizando este recurso.

El modelo de datos RDF de Oracle soporta contenedores y colecciones. Un contenedor o colección tendrá un rdf:container_name o rdf:collection_name asociado a un rdf:type, y un LINK_TYPE de un RDF_MEMBER.

Dos nuevos tipos de objetos se han definido para los datos RDF. SDO_RDF_TRIPLE sirve como la representación de la tripleta de un dato RDF, SDO_RDF_TRIPLE_S se define para almacenar datos persistentemente en la base de datos. La función GET_RDF_TRIPLE() se puede utilizar para devolver un tipo SDO_RDF_TRIPLE.

Conjunto de reglas (Rulebase)

Cada rulebase RDF consiste en un conjunto de reglas. Cada regla es identificada por un nombre, i consiste en un patrón 'IF' para los antecedentes, una condición de filtro opcional que además restringe los subgrafos, y un patrón THEN para los consecuentes.

Cuando se aplica una regla a un modelo RDF puede producir tripletas adicionales. Un modelo RDF aumentado con un conjunto de reglas es equivalente al conjunto de tripletas original además de las tripletas inferidas mediante la aplicación del rulebase al modelo. Las reglas de un rulebase pueden aplicarse al rulebase en si mismo para generar tripletas adicionales.

Oracle soporta tanto un rulebase RDF que implementa las reglas que conlleva RDF, y un rulebase RDF Schema (RDFS), que implementa las reglas de RDFS. Ambos rulebases se crean cuando se añade soporte RDF a la base de datos. También es posible crear un rulebase adicionales para habilidades de inferencia especializadas.

Para cada rulebase, se crea una tabla de sistema, además de una vista de sistema de la rulebase. Esta vista se utiliza para insertar, borrar y modificar reglas en la rulebase.

Índice de Reglas.

Un índice de reglas es un objeto que contiene tripletas pre-procesadas que pueden ser inferidas aplicando un conjunto de rulebases a un conjunto específico de modelos. Si una consulta se refiere a cualquier rulebase, debe existir un índice de reglas para cada combinación de rulebase-modelo en la consulta.

Cuando se crea un índice de reglas, una vista se crea también en las tripletas RDF asociadas con el índice bajo el esquema MDSYS. Esta vista es visible únicamente por el propietario de del índice de reglas, y para los usuarios que tengan este privilegio. La información sobre todos los índices de reglas e mantiene en la vista de índice de reglas. La información sobre todos los objetos de la base de datos, como modelos y rulebases, relacionada con los índices de reglas se mantienen en la vista Rule Index Datasets.

Consultas

La utilización de la función de tabla SDO_RDF_MATCH permite embeber una consulta de grafos en una consulta SQL. Esta función tiene la habilidad de buscar patrones contra los datos RDF, incluyendo inferencia, datos RDF, RDFS y reglas definidas por el usuario. La función SDO_RDF_MATCH se ha diseñado para cumplir la mayoría de los requerimientos identificados por W3C en SPARQL para consultas de grafos.

También se proporciona una API Java para representaciones de grafos y análisis de grafos, que incluyen la posibilidad de encontrar un path entre dos recursos, o de encontrar un path entre dos recursos cuando los links son de un tipo determinado.

5.3. JENA2

Jena es un framework Java para construir aplicaciones para la Web Semántica. Proporciona un entorno de programación para RDF, RDFS y OWL, incluyendo un motor de inferencia basado en reglas. El código de Jena es libre, y ha ido evolucionando a partir de trabajo con *HP Labs Semantic Web Programme*[38].

El framework de Jena incluye:

- Una API RDF
- Lectura y escritura de RDF en formato RDF/XML, N3 y N-Triples.
- Una API OWL.
- Almacenamiento en memoria y persistente.
- Consultas a través de RQL.

Jena es el toolkit más importante de los programadores de la Web Semántica. Actualmente, va por la segunda generación del producto. Jena2 está diseñado para solucionar algunos de los problemas de Jena1. Específicamente, el diseño del esquema se ha modificado para mejorar problemas de rendimiento debido a demasiadas uniones de tablas y el incremento del almacenamiento resultante de la implementación de la reificación.

Jena1 almacena las tripletas de una forma normalizada. Una tabla de sentencias almacena referencias al sujeto, predicado y objeto, y el valor real de las URIS y de los literales se almacena en dos tablas adicionales. Este diseño es eficiente en espacio, porque los valores de texto se almacenan una única vez, independientemente del número de veces que aparezcan en las tripletas. Sin embargo, una unión a tres bandas se requiere para operaciones de búsqueda. Además, la tabla de sentencias no es escalable para grandes bases de datos.

Jena 2 utiliza una visión desnormalizada, multi-modelo y de triple almacenamiento. Los modelos se almacenan en tablas separadas, y cada modelo almacena las sentencias afirmadas en una tabla y las sentencias reificadas en otra, aunque este comportamiento por defecto puede ser alterado mediante la configuración. La tabla de sentencias confirmadas almacena los valores textuales reales de las tripletas en columnas de sujeto, predicado y objeto. Los valores textuales se almacenan además redundantemente, si el mismo valor textual aparece en diferentes tripletas. Jena2 además consume mas espacio de almacenamiento que Jena 1; sin embargo, el número de joins de las tablas requerido se reduce en tiempo de consulta, mejorando el rendimiento.

Las sentencias reificadas se almacenan en una tabla de propiedades, que tiene columnas StmtURI (que representan la sentencia reificada), rdf:subject, rdf:predicate, rdf:objetct and rdf:type. Una fila simple con todos los atributos presentes representa una tripleta reificada.

Además de las tablas para sentencias afirmativas y reificadas, Jena2 puede configurarse para incluir tablas de propiedades en la creación de grafos. Estas tablas almacenan pares de sujeto-valor para predicados específicos. La tabla columna incluye el sujeto y cada predicado que va a ser representado en la tabla. Por ejemplo, una tabla de propiedades Dublín Core puede incluir las columnas sujeto, dc:title, dc:Publisher, y dc:description, donde una fila simple almacena los valores del predicado para un sujeto común. Las tablas de propiedades están diseñadas para proporcionar una ligera reducción del almacenamiento, ya que los

[38] http://www.hpl.hp.com/semweb/

predicados URIs no se almacenan. Se intenta agrupar propiedades que son accedidas conjuntamente y de esta forma incrementar el rendimiento.

Estas son las tablas básicas que componen el esquema de Jena2, aunque hay otras tablas de soporte. Jena2, así como otros sistemas que implementan almacenamiento persistente en base de datos relacionales, utiliza tablas relacionales planas.

El modulo de bases de datos de Jena proporciona una implementación de la interface del modelo Jena, pero con la capacidad de almacenar y recuperar datos RDF utilizando una base de datos. Actualmente, soporta MySql, Oracle y PostgreSQL para almacenamiento persistente, y se ejecuta bajo Windows y Windows XP.

El subsistema de persistencia soporta la capacidad Fastpath para las consultas RDQL que genera consultas SQL dinámicamente para mejorar el rendimiento de las consultas RDQL tanto como sea posible en el motor de la base de datos.

El algoritmo Fastpath trabaja dividiendo las variables y constantes de una consulta en uno o más grupos, llamados *stages*, donde cada stage consiste en variables que pueden ser procesadas juntas. Stages que consisten en más de una variable son procesadas mediante una consulta simple generada dinámicamente. Las stages que consisten justamente en una variable son procesadas mediante una operación Find. Esto se considera más eficiente que generar una consulta SQL. El algoritmo Fastpath también determina el orden de la ejecución para intentar minimizar el tiempo de ejecución.

También hay que mencionar ARQ[39], que es un motor de consulta para Jena que soporta el lenguaje SPARQL.

Algunas de las características de ARQ son:

- Soporta varios lenguajes de consulta: SPARQL, RDQL, ARQ (el lenguaje propio del motor de base de datos).
- Múltiples motores de consulta: Motor de propósito general, motores de acceso remoto y traductor a SQL.

[39] http://jena.sourceforge.net/ARQ/

5.4. SESAME

Sesame es una arquitectura modular basada en Java y de código libre para almacenar y hacer consultas sobre RDF y RDF Schema. Soporta tres lenguajes de consulta (RQL, SeRQL y RDQL) y puede usar la memoria principal de la máquina o una base de datos relacional para el almacenamiento de los datos RDF.

El la mayoría de los casos, Sesame depende de una base de datos relacional para almacenar los datos RDF. Actualmente, Sesame soporta PostgreSQL[40], MySQL[41], Oracle (9i o superior) y SQL Server[42]. Tanto PostgreSQL y MySql se distribuyen con licencia Open Source. En general, Sesame rinde más con MySql que con PostgreSQL. No hay estudios disponibles sobre el rendimiento sobre Oracle o SQLServer.

Si se desea utilizar Sesame utilizando el almacenamiento en memoria, no se necesita la instalación de un sistema de gestión de base de datos.

La Necesidad de Lenguaje de consulta para RDFS
Los documentos RDF y los esquemas RDF pueden considerarse sobre tres niveles diferentes de abstracción:

a. En el nivel sintáctico, son documentos XML.
b. En el nivel de estructura, que considera un conjunto de tripletas.
c. En el nivel semántico, que constituye uno o más grafos con semánticas parcialmente predefinidas.

a. Consultas a nivel Sintáctico.
El modelo RDF puede ser descrito mediante la notación XML, por lo que parece razonable pensar que se pueden realizar consultas RDF empleando para ello lenguajes de consulta para XML, como por ejemplo Xquery. Pero este enfoque no considera el hecho que de XML maneja una estructura en árbol mientras que RDF es un grafo etiquetado. Este no es el único inconveniente, también existe el problema que la serialización de RDF en XML se puede hacer de diferentes formas.

b. Consultas a Nivel de Estructura.
Un documento RDF se representa como un conjunto de tripletas; cada tripleta representa una sentencia de la forma sujeto-predicado-objeto. Diferentes lenguajes de consulta han sido propuestos e implementados teniendo en cuenta que los documentos RDF son un conjunto de tripletas, y que estas tripletas pueden ser consultadas de diferentes formas.

Aplicando un lenguaje de consulta como Squish, se podría formular la siguiente consulta:

```
SELECT ?x
FROM somesource
WHERE (rdf::type ?x Algo)
```

La gran ventaja de esta consulta es que directamente referencia el modelo RDF y es independiente de la sintaxis empleada para la representación de los datos.

[40] http://www.postgresql.org/
[41] http://www.mysql.com/
[42] http://www.microsoft.com/sql/default.mspx

Sin embargo, una desventaja de cualquier lenguaje de consulta a éste nivel es que interpretan cualquier modelo RDF como un conjunto de tripletas, inclusive aquellos elementos que tienen una semántica especial en RDFS.

c. Consultas a nivel Semántico.

Un requerimiento fundamental es que se necesitan consultas a nivel semántico, de modo que permitan obtener todo el conocimiento representado a través de un modelo RDFS. Se pueden considerar dos opciones para alcanzar este objetivo:
- Procesar y almacenar la clausura del grafo dado como base para la consulta.
- Permitir un procesador de consultas que infiera nuevas sentencias de acuerdo a la necesidad.

La arquitectura de SESAME

En la siguiente figura se muestra la arquitectura de Sesame. Uno de sus componentes principales es SAIL (Storage And Inference Layer) que es el nivel que se encarga de interactuar con los manejadores de bases de datos (DBMS) permitiendo de este modo mantener la arquitectura de Sesame independiente del DBMS. SAIL es una API que ofrece métodos RDF específicos a sus "clientes" y traduce estos métodos a llamadas de un DBMS específico. Los módulos funcionales de Sesame son "clientes" de SAIL. Actualmente existen tres módulos: EL motor de consultas RQL, el Módulo de Administración RDF y el módulo de exportación RDF.
Dependiendo del ambiente en el cual se implante Sesame ofrece diversas formas de comunicación, sea vía HTTP para contexto Web, o Remote Method Invocation (RMI) o Simple Object Acces Protocol (SOAP).

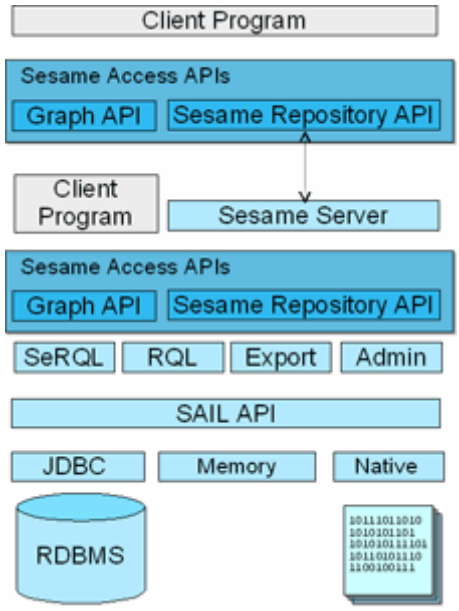

Fig. 3: Arquitectura de Sesame.

Módulos Funcionales de Sesame

a. Módulo de Consulta RQL

La versión de RQL de Sesame está caracterizada por cumplir con los estándares de la W3C, sin embargo no soporta la tipificación de datos. El proceso de consulta (Fig.4) es el siguiente: después del parsing y la construcción del árbol de consulta, pasa a un optimizador de consultas que trasforma la consulta en un modelo equivalente pero más eficiente. La optimización consta de un conjunto de

heurísticas, éstas pre-evaluaciones de optimización no dependen de ningún método de almacenamiento en particular.

EL modelo optimizado es subsecuentemente evaluado siguiendo la estructura de árbol generada, cada objeto representa una unidad básica en la consulta original y se evalúa a si mismo extrayendo información por medio de SAIL cuando lo necesita. La gran ventaja de este enfoque es que los resultados se retornan de la misma forma, en vez de construir primero la consulta completa en memoria.

El grueso de las consultas son resueltas por el motor RQL, por ejemplo cuando una consulta contiene una operación de semi-join, cada sub-consulta es evaluada y la operación de semi-join es ejecuta entonces por el motor de consultas sobre los resultados.

Otra alternativa sería aprovechar la optimización a partir del DBMS, pero esto haría que el proceso de consulta fuera dependiente de un DBMS específico.

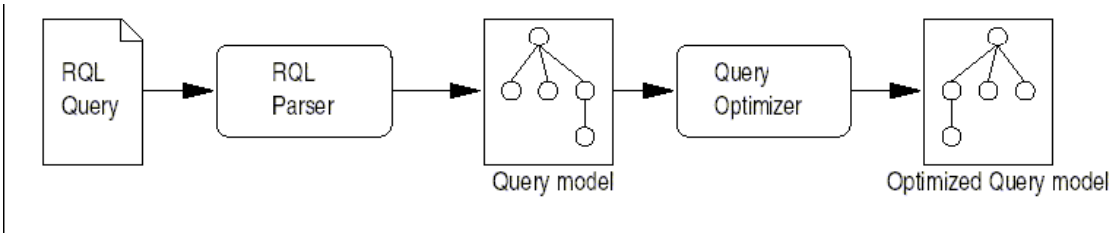

Fig. 4: Parsing de las consultas y optimización del modelo.

b. Módulo de Administración.
Para posibilitar la inserción de datos RDF e información de esquemas en un repositorio, Sesame utiliza el módulo de administración, ofreciendo dos operaciones fundamentales:
- Adición incremental de datos y esquemas RDF.
- Limpiar un repositorio.

Hay que señalar que una opción borrado (por sentencias) funcional aun no esta disponible.
El módulo extrae la información de una fuente RDF(S), generalmente en XML serializado, y se hace un parsing usando un parser RDF (por ejemplo, el ARP RDF parser que es parte del Toolkit de Jena), que entrega información al módulo de administración de la forma básica (objeto, atributo, valor), y luego al repositorio por medio de la capa SAIL reportando los errores que pudieron haber ocurrido.

c. Módulo de Exportación RDF.
Permite exportar los contenidos de un repositorio en formato XML serializado, el objetivo de este modulo es combinar Sesame con otras herramientas RDF, dado que la mayoría reconoce el formato. Además de permitir la exportación de datos, de esquemas, o de ambos, dependiendo si es necesario la semántica o no.

SAIL API
LA SAIL API consta de un conjunto de interfases Java que han sido específicamente diseñados para el almacenamiento y extracción de información basada en RDFS. A continuación, se presentan los principios más relevantes de la API de SAIL:

- Define una interfase básica para almacenar, borrar y recuperar RDF y RDFS de repositorios persistentes.
- Una abstracción del mecanismo actual de almacenamiento, debiendo ser aplicable a RDBMS, sistema de archivos, almacenamiento en memoria, etc.

- Aplicable a diferentes plataformas hardware y software, como PDAs, pero ofreciendo suficiente libertad para la optimización en el caso de gran cantidad de información, por ejemplo, cluster de bases de datos a nivel de empresas.
- Ser extensible a otros lenguajes basados en RDF como DAML+OIL.

Una característica importante que diferencia a SAIL API, por ejemplo, de Jena, es que ofrece métodos para consulta de clases y propiedades, así como restricciones de dominio y rango. Lo que no ocurre con otros productos que están enfocados al conjunto de tripletas, dejando el trabajo de interpretación al usuario. Esta ventaja se basa en la fuerte relación entre la eficiencia de la inferencia y el modelo de almacenamiento empleado.

Otra característica de SAIL, se relaciona a su tamaño “ligero”, dado que solamente implementa cuatro interfases predefinidas, ofreciendo almacenamiento y recuperación, así como funcionalidad y soporte transaccional.

5.5. KOWARI

Kowari es una base de datos construida con el propósito de almacenar, recuperar y analizar metadatos. Está escrita en Java, y su código es libre. Soporta tanto metadatos RDF como OWL.

A continuación, se señalan sus características principales:

Generales

- Soporte nativo de RDF
- Múltiples modelos de bases de datos por servidor.
- Lenguaje de consultas similar a SQL: Tucana Query Lenguaje, aunque se pretende dar soporte a SPARQL en el futuro.
- Funcionalidad búsqueda completa de texto.
- Soporta tipos de datos.
- Soporta las especificaciones de W3C y cumple sus recomendaciones de uso.

Rendimiento y escalabilidad.

- Gran capacidad de almacenamiento
- Está optimizado para el almacenamiento y recuperación de metadatos.
- Soporta multi-procesador.
- Está preparado para arquitecturas tanto de 32-bits como de 64-bits.
- Bajos requerimientos de memoria.

Confianza

- Soporte completo de transacciones
- Clustering y almacenamiento fail-over
- Integridad permanente

Conectividad

- JRDF: Java RDF, un conjunto de API e implementaciones base para RDF utilizando Java.
- SOAP
- Software Developers Kit (SDK)

Manejabilidad

- Mínima administración.
- Configuración basada en Web y herramientas de monitorización.

Soporte para diferentes sistemas operativos

- Microsoft® Windows NT®, Windows® 2000 and XP
- UNIX® and Linux®
- Solaris™
- Mac OS® X
- IRIX®

Escalabilidad

La máquina de almacenamiento de Kowari es un almacenamiento de las tripletas transaccional denominado :XA Triplestore. En gran parte, la escalabilidad de Kowari se debe a las siguientes características de XA.

Estructura de datos de 64-bit

Todos los campos relevantes el la memoria y en el disco son de 64 bits de amplitud, lo que asegura que Kowari puede almacenar grandes cantidades de datos hasta llegar a los límites impuestos por el sistema operativo de la máquina que lo aloja.

Múltiples Sesiones sin bloqueo de contenido.

Una simple sesión de escritura además de varias sesiones de lectura puede acceder al almacén de tripletas concurrentemente sin que las sesiones de lectura requieran adquirir un bloque global mientras procesan una consulta. Esto elimina la posibilidad de cualquier bloqueo. En general, cada sesión se ejecuta en su propio hilo. La carencia de bloqueo quiere decir que el número máximo de lecturas activa está limitado en exclusiva por el sistema operativo de la máquina y el subsistema de entrada-salida.

Cuando una sesión inicia una consulta, la cual puede involucrar múltiples solicitudes al almacén de datos, en primer lugar toma una instantánea de la base de datos completa. Esto asegura que todas las consultas al almacén durante el procesado de la consulta vean a la base de datos en un estado consistente.

El almacén de tripletas está designado para que la obtención de la instantánea sea una operación muy rápida y no cause operaciones de entrada/salida. Debería tomar menos de un milisegundo en cualquier hardware actual, independientemente del tamaño de la base de datos.

La sesión solo mantiene un bloqueo global durante este breve periodo de tiempo mientras obtiene la instantánea. Una vez que se obtiene, no se necesitan bloqueos adicionales independientemente del número de operaciones requeridas o de la cantidad de tiempo requerida para ejecutar la consulta.

La existencia de una instantánea no provoca la necesidad de un almacenamiento extra, pero provoca algunas modificaciones en la semántica. Las estructuras en el disco están diseñadas para minimizar la cantidad de copiado requerido para realizar una modificación, además de mejorar el rendimiento mediante la maximización de la cantidad de almacenamiento compartido entre instantáneas.

La instantánea se libera una vez que ha terminado el procesado de la consulta. Cualquier espacio utilizado por la instantánea y no compartido con otra instantánea está inmediatamente disponible para su utilización. Este proceso de liberalización es tan rápido como el de la obtención de la instantánea, pero la sesión no necesita mantener un bloqueo global durante esta operación.

Se utiliza un bloqueo global por separado (el bloqueo de escritura) para asegurar que hay solo una sesión de escritura en un momento dado. Este bloque se libera después de que la transacción se confirme o se anule.

Backup On-Line

El XA Triplestore permite que las modificaciones y las consultas se procesen concurrentemente con una operación de backup. La sesión que realiza el backup adquiere una instantánea de la base de datos completa como si estuviese realizando una consulta.

Integridad permanente.

Las interrupciones repentinas causadas por fallos de corriente eléctrica y algunos tipos de fallos de hardware no provocan corrupción de los datos. La estructura de los datos en el disco está diseñada para mantenerse en un estado consistente todo el tiempo, mediante se minimiza el coste requerido para conseguir la instantánea.

Utilización de Java NIO

El XA Triplestore utiliza el API Java NIO (new I/O) API que fue introducido en JAVA2 SDK V.1.4. La API NIO proporciona acceso a facilidades avanzadas de entrada-salida que sólo estaban accesibles con anterioridad a los programas escritos en C. La utilización de NIO permite a XA Triplestore proporcionar transacciones, integridad permanente y un buen rendimiento además de mantener una implementación pura en Java.

Extensibilidad

Kowari puede ser extendido programáticamente para tratar almacenes de datos externos como si fuesen grafos RDF vía el Resolver SPI. Bases de datos, sistemas de archivos, recursos Web, pueden ser consultados como metadatos nativos RDF.

5.6. TUCANA

Tucana Knowledge Server[43] (TKS) es una base de datos diseñada específicamente para el almacenamiento gestión y análisis de metadatos. Ha sido construido desde su base para este propósito, y no es una capa abstracta en la parte superior de una base de datos relacional. Es un producto comercial, basado parcialmente en Kowari. TKS mantiene los datos en el formato RDF. En otras palabras, un conjunto de datos TKS está compuesto de un número de *cosas* y las *relaciones* entre ellos. Estas *cosas* se almacenan mediante un grafo dirigido, donde los links entre los nodos del grafo representan las relaciones.

TKS permite que las definiciones de esquema sean muy flexibles, a diferencia de las bases de datos relacionales. Mientras una base de datos relacional necesita tanto la creación de un esquema antes de introducir algún dato como tener un conocimiento completo para poder consultarlo, TKS permite introducir e incluso consultar los datos sin la necesidad de un esquema fijo. Los esquemas son a menudo muy útiles, sin embargo, especialmente para que las consultas sean breves y fácilmente comprensibles. TKS soporta la adición de un esquema e incluso la información ontológica para ayudar a la comprensión de los datos una vez que son almacenados. La información del esquema es almacenada como metadatos en TKS, al contrario que en las bases de datos relacionales.

La velocidad de las consultas es muy rápida. Lectura y escritura de metadatos es cerca de diez veces más rápida que en las bases de datos relacionales que almacenan la misma información.

El Tucana Knowledge Server es seguro, permitiendo que se apliquen permisos basados en usuarios y grupos de usuarios a grupos de metadatos. Los metadatos pueden ser asegurados hasta las sentencias individuales. TKS utiliza una capa abstracta de seguridad para conectar con un amplio rango de servicios de autenticación, sobre todo directorios de servicios que Lightweight Directory Access Protocol (LDAP).

Con respecto a la escalabilidad, es masivamente escalable: Una instancia simple de TKS puede almacenar hasta un billón de metadatos sobre un sistema operativo de 64 bits (como Sun Solaris) o diez millones de sentencias sobre sistemas operativos de 32-bits (como Microsoft Windows NT/2K/XP).

Varios servidores TKS pueden ser tratados como una base de datos virtual. Es decir, una consulta puede ser lanzada contra más de una base de datos simultáneamente y se recibirá un resultado coordinado. Cualquier base de datos TKS puede servir como punto de entrada para esta consulta. Esta capacidad, combinada con el modelo de seguridad, permite un control completo sobre la administración y el acceso de los datos almacenados en TKS.

Lo más importante es que Tucana Knowledge Server puede actuar como un punto de unión para los datos de la empresa. Los datos en TKS pueden venir de:

- Fuentes estructuradas: por ejemplo, bases de datos Oracle.
- Fuentes no estructuradas: documentos office, correos electrónicos, etc.

Los datos estructurados pueden importarse a TKS mediante un conector de base de datos relacional, accesible mediante la Consola de Administración de TKS. Una vez almacenados los datos pueden ser combinados con datos de otras fuentes para su análisis.

[43] http://tucana.es.northropgrumman.com/

Los datos no estructurados no se almacenan directamente en TKS. En vez de esto, el contenido de los datos no estructurados es representado mediante entidades automáticamente extraídas. Estas entidades pueden consistir en metadatos de documentos tradicionales (como el tamaño de un archivo o la fecha de la última modificación) o representación de contenido mas rico (como los nombres mencionados en un documento, localizaciones referenciadas o sujetos diseccionados). Las entidades pueden ser extraídas automáticamente de documentos no estructurados utilizando el Tucana Metadata Extractor u otra herramienta de extracción de entidades capaz de producir una salida XML o RDF.

El TKS Query Engine puede ser utilizado para acceder a Fuentes de datos remotas (como máquinas de búsqueda de texto u otros Tucana Knowledge Servers). Las consultas federadas se habilitan permitiendo a cualquier instancia TKS consultar otras instancias TKS.

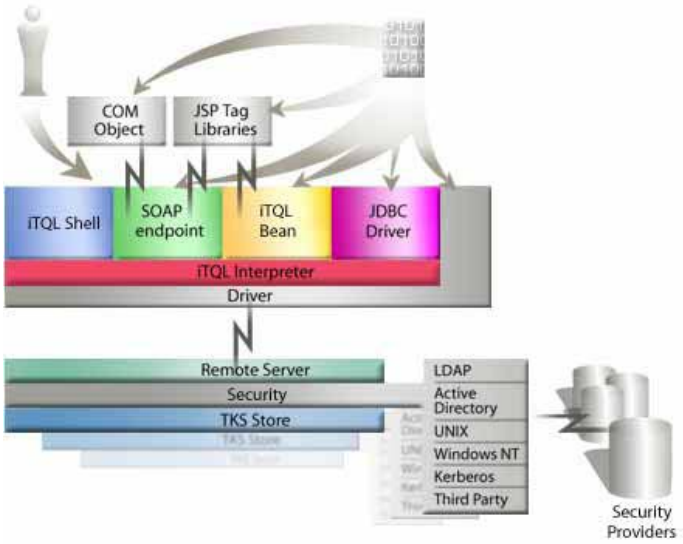

Fig.5 : Arquitectura de un servidor TKS.

La arquitectura de un Tucana Knowledge Server consiste en tres capas: La TKS Query Engine acepta consulta desde una serie de APIs y las distribuye ha uno o mas servidores TKS. TKS Stores, como sugiere su nombre, almacena metadatos de forma nativa.

Cada TKS Store gestiona su propia seguridad, almacenando usuarios y autorizando su acceso a la información requerida.

Muchos estándares de la industria y las APIs estándares son soportadas por TKS. Estas incluyen Web Services vía SOAP[44] y una interfaz WSDL[45] interface, un Java Bean y Java API, y soporte para otros lenguajes comúnmente utilizados, como Perl[46] y Java Server Pages[47].

El Tucana Knowledge Server soporta los estándares de metadatos standards del World Wide Web Consortium, incluyendo:

- Resource Description Framework (RDF)
- RDF Schema (RDFS)
- Web Ontology Language (OWL)

[44] http://www.w3.org/TR/soap/
[45] http://www.w3.org/TR/wsdl
[46] http://www.perl.com/
[47] http://java.sun.com/products/jsp/

Almacen de metadatos

Los datos originales (por ejemplo, un documento o un email) se almacenan separadamente del TKS, y son referenciados por una URL (Uniform Resource Locator). Los datos pueden ser almacenados en un sistema de archivos que sea accedido por la máquina. Esto evita el almacenamiento de grandes documentos en bases de datos relacionales o almacenes de objetos, e incrementa la velocidad de acceso a la información.

5.7. RDF GATEWAY

RDF Gateway es una plataforma diseñada para la Web Semántica. Es un servidor de aplicaciones, un servidor Web, y una servidor de bases de datos RDF.

El motor de base de datos de RDF Gateway no es como los de los Sistemas de Gestión de Bases de datos Relacionales (RDBMS). Soporta un lenguaje de comandos similar a SQL, múltiples bases de datos, tablas almacenadas en memoria o disco, cursores y transacciones.

La principal diferencia entre RDF Gateway y una base de datos relacional es como se representa la semántica de un modelo de datos. Una base de datos relacional codifica la semántica implícitamente utilizando elementos de almacenamiento como tablas, columnas, y claves ajenas. Las aplicaciones cliente tienen que ser desarrolladas para comprender la semántica de la información relacional. Esto hace que sea difícil compartir información entre sistemas que no comparte un esquema relacional idéntico. RDF proporciona un estándar abierto para especificar la semántica de un modelo de datos. Además, RDF Gateway no necesita codificar la semántica utilizando tablas, columnas y claves ajenas. Con RDF Gateway todas las tablas tienen el mismo formato y las claves ajenas no son necesarias.

Ya que cualquier modelo puede ser descrito utilizando tripletas RDF, todas las tablas tienen el mismo formato. Cada tabla tiene cuatro columnas: Contexto, predicado, sujeto y objeto.

Con RDF Gateway las tablas no se asocian con una clase simple de objeto en el modelo de datos, ya que no pueden alojar objetos de muchas clases diferentes. Una tabla de RDF Gateway puede incluso mantener tablas de varios esquemas diferentes. Las tablas de RDF no son más que fuentes de datos.

Alguna de sus características referentes a la gestión de los datos RDF son las siguientes:

Motor de Base de datos Nativo.

- Utiliza comandos similares a SQL para realizar consultas de datos RDF.
- Razonamiento deductivo vía reglas de inferencia (librerías para OWL y RDFS)
- Alto rendimiento en disco y tablas de memoria.
- Transacciones con niveles aislados.
- Tablas virtuales que integran fuentes de datos externas (páginas Web, documentos locales, bases de datos relacionales, etc.)
- Consultas federadas a través de múltiples fuentes de datos internas y externas.
- Búsquedas de texto.
- Encriptación de datos
- Soporte para la serialización RDF/XML, N3 y NTriples
- Seguridad a nivel de tabla y de sentencias.
- Acceso a datos desde otras aplicaciones vía http, Microsoft ADO[48] o Sun Microsystems JDBC[49].

Arquitectura:

[48] http://msdn.microsoft.com/data/Default.aspx
[49] http://java.sun.com/products/jdbc/

RDF Gateway es un servidor de alto rendimiento que combina el conjunto de características de un sistema de gestión de base de datos y un servidor Web. El objetivo de este diseño es proporcionar una aplicación completa para agrupar, consultar, trasformar y entregar información. RDF Gateway proporciona un lenguaje muy rico de script denominado RDFQL. Está basado en Java Script, con extensiones de consulta para realizar búsquedas federadas entre diferentes fuentes de recursos. Las páginas RDFQL pueden escribirse y utilizarse para crear aplicaciones muy completas llamadas packages.

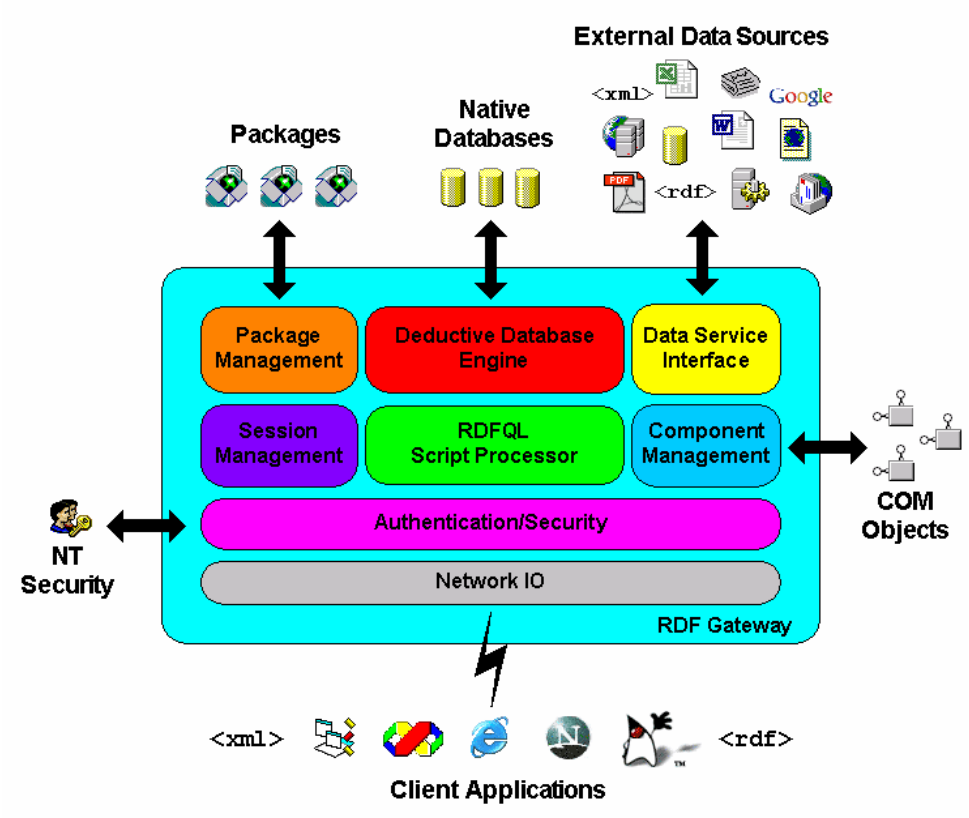

Fig. 6: Arquitectura de RDF Gateway

RDFQL Script Processor

El procesador de RDFQL script es una máquina virtual que compila, cachea y ejecuta RDFQL scripts. RDFQL integra extensiones de consulta similares a SQL para proporcionar acceso a la base de datos. También RDFQL permite que las páginas contengan una combinación de scripts y contenido estático similar a Microsoft Active Server Pages (ASP).

Base de datos deductiva

RDF Gateway tiene un motor de base de datos que ha sido diseñada desde el principio para soportar RDF. Su diseño semi-estructurado es ideal para consultar y almacenar tripletas RDF. El motor deductivo de la base de datos realiza una evaluación de las consultas que es federada a lo largo de todas las fuentes de datos especificadas. Las capacidades de inferencia lógica del motor proporcionan suporte para la sintaxis declarativa de RDFQL. Implementa si propia persistencia en archivos, eliminando la necesidad de un sistema de almacenamiento de datos externo.

Interfaz de servicio de datos.

La interfaz de servicio de datos permite a fuentes de datos externas integrarse con RDF Gateway. Un proveedor de servicio de datos es un módulo que implementa su interface y expone el contenido de un tipo concreto de datos como datos RDF. RDFQL permite que se realicen consultas federadas sobre múltiples servicios de datos Esta interfaz abierta hace posible utilizar cualquier proveedor de servicio de datos disponible actualmente o desarrollar un proveedor personalizado para una fuente de datos.

Autenticación / Seguridad

RDF Gateway tiene un modelo de seguridad basado en derechos y permisos para controlar el acceso al servidor y a los recursos de la base de datos. Soporta sus propios usuarios y roles, además de grupos y usuarios de NT. Un usuario de NT se autentica siempre utilizando las credenciales NT para la cuenta. Esta utilización de grupos y usuarios de NT hace posible que se gestione una administración de seguridad externa.

Network IO

RDF Gateway tiene un interfaz de red optimizado para el rendimiento y la escalabilidad. Soporta tanto http como un protocolo propietario basado en TCP/IP. La capa de red IO soporta esquemas de autenticación de red seguros como NT Challenge/Response (NTLM). RDF Gateway tiene drivers clientes para Microsoft ADO y Sun Microsystems JDBC. Esto permite a RDF Gateway soportar una gran cantidad de clientes, como navegadores Web, aplicaciones Windows, aplicaciones Java y clientes basados en XML o RDF.

Package Management

RDF Gateway permite que las aplicaciones se desarrollen y se distribuyan como paquetes. Un paquete consiste en páginas RDF Server, páginas HTML, imágenes o cualquier otro tipo de archivo.

Component Management

RDFQL soporta COM en sus scripts para el servidor. Esto permite que la funcionalidad de RDF se extienda para que las aplicaciones se integren con RDF Gateway.

6. COMPARACIÓN DE LOS SGBD

En el apartado anterior, se han descrito las diferencias entre los sistemas de gestión de bases de datos nativos para RDF y los Sistemas de Gestión Relacionales; también se ha examinado la arquitectura de cada uno de ellos, los lenguajes de consulta que soportan, las diferentes opciones para el almacenamiento de los datos así como las características que los diferencian. A continuación, en una tabla se expone un resumen de todas estas características.

PRODUCTO			ALMACENAMIENTO					LENGUAJES CONSULTA			
Nombre	Empresa	Versión	Nativo	Oracle	MySql	Postgres	Otros	SPARQL	RDQL	RQL	Otros
Oracle	Oracle	10g.2	No	Si	No	No	No	No	No	No	SQL Extendido
Jena	Open Source	2	No	Si	Si	Si	No	Si	Si	No	ARQL
Sesame	Open Source	1.2.3	No	Si	Si	Si	Sql Server	No	Si	SI	SeRQL
Kowari	Open Source	1.1.0	Si	No	No	No	No	No	No	No	iTQL
Tucana	Northrop Grumman Corp.		Si	No	No	No	No	No	No	No	iTQL
Rdf Gateway	Intellidimension Inc	2	Si	No	No	No	No	No	No	No	Sql propio

7. CONCLUSION

A la vez que la Web crece, convirtiéndose en un almacén de conocimiento humano cada vez más rico, se necesitan herramientas más potentes que realicen búsquedas y sean capaces de interpretar grandes cantidades de datos; esto se aplica tanto a intranets como a la Web en general. Han surgido dos modelos para ayudar a gestionar estos datos a escala global: La Web Semántica y los Servicios Web.

La Web Semántica proporciona una infraestructura común que permite compartir y reutilizar datos a través de aplicaciones, empresas y comunidades. Es una extensión de la Web actual en la cual la información está bien definida, las máquinas están mejor equipados y las personas trabajan en cooperación. La Web Semántica está centrada en los datos.

Los Servicios Web proporcionan un camino estándar que hace posible que diferentes aplicaciones de software interactúen entre sí, y funcionen en una variedad de plataformas y/o infraestructuras. Los Servicios Web están centrados en el mensaje.

Ambos modelos son importantes para sistemas distribuidos y de red, por lo que el W3C trabaja para asegurar una correcta integración, tanto entre ellos, como con infraestructuras Web existentes. Por ejemplo, los servicios Web se benefician de la capacidad de compartir vocabularios comunes, nombres bien definidos, y un modelo de datos común, todo ello expresado con tecnologías de Web Semántica.

A pesar de las grandes ventajas que la Web Semántica promete, su éxito o fracaso será determinado -como con la WWW- en gran medida por la facilidad de acceso y la disponibilidad de contenido variado y de gran calidad. Hay aún varios problemas que resolver antes de conseguir que esto suceda, incluyendo, pero no limitados a:

- La disponibilidad de contenido. Actualmente, hay poco contenido de la Web Semántica disponible. El contenido de las webs existentes debe ser actualizado a contenido para la Web Semántica, incluyendo páginas de HTML estático, contenido existente XML, y contenido dinámico, multimedia y servicios Web.
- Disponibilidad, desarrollo y evolución de ontologías. Las ontologías se convertirán en una pieza clave, ya que permiten explícitamente la semántica del contenido de la Web Semántica. Se debe realizar un gran esfuerzo en la creación de ontología comúnmente usadas para la Web Semántica, en proporcionar infraestructuras adecuadas para el desarrollo, gestión y mapeo de ontologías, y, en el entorno distribuido de la Web, en el adecuado control de la evolución de las ontologías y las anotaciones que las refieren.
- Escalabilidad. Será necesario realizar un esfuerzo significativo para organizar el contenido de la Web Semántica, almacenarlo y proporcional los mecanismos para encontrarlo. Todas esas tareas deben ser realizadas y coordinadas de forma escalable, ya que las soluciones deben estar preparadas para el enorme crecimiento de la Web Semántica.
- Soporte multilenguaje. Este problema ya existe en la Web actual, y también debe ser tenido en cuenta en la Web Semántica. Cualquier aproximación a la Web Semántica debe proporcionar mecanismos para acceder a la información en diferentes lenguajes, permitiendo la creación y el acceso al contenido independientemente del lenguaje original de los proveedores de contenido o los usuarios.

- Visualización. La visualización intuitiva del contenido de la Web Semántica será cada vez más importante para resolver la creciente sobrecarga de información, al requerir los usuarios un reconocimiento sencillo del contenido relevante para sus propósitos. Deben explorarse nuevas técnicas que difieren de la visualización usual de la estructura mediante hipertexto de la Web actual.
- Estabilidad de los lenguajes de la Web Semántica. Finalmente, deben realizarse urgentemente esfuerzos de estandarización en este campo emergente, para permitir la creación de la tecnología necesaria que soporte la Web Semántica.

Con respecto a los sistemas de gestión de bases de datos que existen en la actualidad, se observa una clara tendencia a soportar las nuevas tecnologías de la Web Semántica. En concreto, la mayoría de los que ya soportan RDF/XML se plantean (si no lo hacen ya) dar soporte a SPARQL (¿Recomendación de W3C en 2006?), por lo que es de esperar que continúen evolucionando en ese sentido.

En este momento, las empresas españolas están comenzando a interesarse por la Web Semántica. A finales del 2005, asistieron a las jornadas sobre "Gestión inteligente de Contenidos: la Web semántica" convocada por ISOCO[50] -Intelligent software for the networked economy- en Madrid, numerosos representantes de distintos sectores, que buscan en esta tecnología una aplicación práctica para su negocio.

Una de las compañías que ha apostado más fuerte por comenzar a aplicar esta tecnología en España es Bankinter. El banco lleva varios años con este proyecto, apoyado por ayudas financieras de la Comunidad Europea, institución que ya ha subvencionado bastantes proyectos de Web semántica. En la actualidad la mayoría de las empresas se muestran reticentes a invertir en una tecnología de la que no ven "negocio" a corto plazo. Los expertos opinan que será Google, una vez más, la que dará el primer paso. La empresa ya está desarrollando proyectos con esta tecnología, y no se tardará mucho en ver sus resultados. El hecho de que Google, el buscador que manejan más internautas, pueda indexar y devolver información con significado, será clave para que las empresas de los distintos sectores comiencen a aplicarla en sus webs.

Hay otras empresas españolas que también están desarrollando aplicaciones de Web Semántica, como la Residencia de Estudiantes, que dispondrá en un breve periodo de tiempo de una aplicación de Web semántica disponible para cualquier usuario. Concretamente aplicada a su "Archivo virtual de la Edad de Plata de la Cultura Española". El internauta podrá realizar búsquedas en lenguaje natural sobre esta inmensa colección de documentos relacionados con el mundo de la cultura.

También El Real Instituto Elcano, presentó en estas jornadas su buscador semántico para relaciones internacionales. Un localizador de información especializada que ya funciona en la actualidad y que está disponible en su Web: www.realinstitutoelcano.org.

[50] http://www.isoco.com/index.html

8. BIBLIOGRAFIA

Antoniou G., van Harmelen F. (2004). *A semantic web primer.* The MIT Press.

Powers, S. (2003). *Practical RDF*. O'Reilly

Hayes P., McBride B.: RDF Semantics. W3C Recommendation, World-Wide Web Consortium, Feb. 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

Davies J., Fensel D., van Harmelen F. (2003) , Towards the Semantic Web. John Wiley & Sons, Ltd.

Bray T., Paoli J., Sperberg-McQueen C.M., Maler E., Yergeau F.: Extensible Markup Language (XML) 1.0 (Third Edition), World-Wide Web Consortium, Feb. 2004. http://www.w3.org/TR/2004/REC-xml-20040204/

World-Wide Web Consortium, Semantic Web: http://www.w3.org/2001/sw/

World-Wide Web Consortium, RDF: http://www.w3.org/RDF

RDF Data Access group: http://www.w3.org/2001/sw/DataAccess/