Metodología de diseño para el desarrollo de mashups semánticos

Luis E. Nieto Peñalver

Directora: Dra. Alicia Díaz

Tesis presentada para obtener el grado de
Magíster en Ingeniería de Software

Facultad de Informática

Universidad Nacional de La Plata

2012

AGRADECIMIENTOS

A mi directora, Alicia Díaz, por su apoyo, experiencia, dedicación y paciencia para iniciarme en el mundo de la Web Semántica.

A los que, de una forma u otra, forman parte de mi camino: mis padres, mis hermanos (Carlos, Julieta, Carolina, Ricardo), Lucía y mis 2 soles y medio (Agus, Guille y el más chiquito, todavía sin nombre), les agradezco el apoyo constante, la paciencia y las horas quitadas.

RESUMEN

Durante los últimos años, y cada vez más rápido, vienen surgiendo nuevas tecnologías Web. La aparición de la Web 2.0 y la integración de estas tecnologías han abierto nuevas e innovadoras formas de colaboración que merecen ser consideradas.

Entre el conjunto de tecnologías Web 2.0 se encuentran los mashups, definidos como aplicaciones Web formadas por APIs, contenido y componentes de aplicación con el fin de extraer información de diversas fuentes, combinarla y presentarla en otras formas. Normalmente, un mashup mejora la presentación visual de la información, ofrece valor agregado a sus usuarios combinando la información de diferentes fuentes o ambas cosas.

Por ejemplo, un mashup podría buscar el precio de un determinado producto en Amazon.com y en BestBuy.com (a través de las APIs de ambos), comparar los 2 precios y devolverle al usuario esta información para que pueda decidir dónde comprar el producto. El mashup incluso podría buscar los precios de estos 2 proveedores periódicamente, para que el usuario pueda ir viendo el cambio de los mismos y decida el mejor momento para comprar. Estos tipos de mashups, también llamados *mashups Web 2.0*, tienen algunas limitaciones, como ser pobre escalabilidad a medida que aumenta la cantidad de fuentes de información con las que trabaja.

Con el surgimiento de la Web Semántica aparecieron los *mashups semánticos*, los cuales carecen de las limitaciones mencionadas anteriormente. El concepto de Web Semántica hace referencia a información publicada en la Web de tal forma que resulte "comprensible" a una máquina (es decir, que su significado esté explícitamente definido) y que pueda conectarse con otra información externa.

Así, un mashup semántico se define como una aplicación Web que extrae información de diversas fuentes, la combina (integra) y presenta en otras formas. La característica principal de esta información es que resulta "comprensible" a una máquina (su significado está explícitamente definido).

A la estructura donde cada fuente organiza su información se la denomina *ontología*, y en consecuencia la tarea de un mashup semántico consiste en integrar la información de las ontologías de las distintas fuentes con las que trabaja. Este es el problema más complejo a resolver por parte de un mashup semántico.

En el ámbito de la Ingeniería de Software se emplean distintas metodologías para el desarrollo de aplicaciones, por ejemplo el Proceso Unificado Rational. Estas metodologías no siempre resultan válidas para el caso de las aplicaciones Web debido no sólo a que no proporcionan todas las técnicas y notaciones requeridas, sino también a la burocracia y rigidez de las mismas.

Actualmente existen metodologías para aplicaciones Web, pero la gran mayoría no contempla el caso de las aplicaciones Web semánticas, y en consecuencia se necesita de una metodología que ayude al diseñador durante el desarrollo de las mismas.

El objetivo de este trabajo consiste en definir una metodología para el diseño de un mashup semántico, que sea lo suficientemente general para independizarse de la plataforma utilizada, y que defina cada proceso y actividad en forma precisa, indicando su propósito, entradas y salidas, actores

involucrados y el conjunto de métodos y técnicas que se utilicen para realizar cada actividad.

Índice

CAPÍTULO 1 – INTRODUCCIÓN

La Web está cada vez más presente en la vida de las personas. Desde el momento en que surgió, cambió la forma de trabajar, y al mismo tiempo, sufrió una maduración tal que le permitió convertirse en una plataforma muy atractiva y dominante para el desarrollo e implantación de aplicaciones de negocios y sociales [Murugesan, 2008].

Los sistemas y aplicaciones Web ofrecen actualmente una variedad de contenido y funcionalidad a una gran cantidad de usuarios, quienes esperan que funcionen cada vez mejor, que resulten más confiables, más seguras, que se puedan personalizar, que sean sensibles al contexto, etc.

Debido a que se ha incrementado la dependencia con estos tipos de aplicaciones durante el último tiempo, su rendimiento, confiabilidad, calidad, mantenimiento y escalabilidad se han vuelto muy importantes. Además, muchas aplicaciones Web están fuertemente integradas con otros sistemas "tradicionales" de información, como bases de datos y sistemas de procesamiento de transacciones, con lo cual, su diseño, desarrollo, implementación y mantenimiento resulta cada vez más complejo [Murugesan, 2008].

En los últimos años, y cada vez más rápido, vienen surgiendo nuevas tecnologías Web. La aparición de la Web 2.0 y la integración de estas tecnologías han abierto nuevas e innovadoras formas de colaboración que merecen ser consideradas [Raza *et al.*, 2008].

Entre el conjunto de tecnologías Web 2.0 se encuentran los mashups. El término "mashup" tiene su origen en la música: 2 o más fuentes musicales unidas en un único trabajo[1]. En el ámbito de la Web, un mashup se define como una aplicación Web formada por APIs[2], contenido y componentes de aplicación con el fin de extraer información de diversas fuentes, combinarla y presentarla en otras formas [Makki, 2008]. Normalmente, un mashup mejora la presentación visual de la información, ofrece valor agregado a sus usuarios combinando la información de diferentes fuentes, o ambas cosas [Yu, 2011].

Por ejemplo, un mashup podría buscar el precio de un determinado producto en Amazon.com y en BestBuy.com a través de las APIs de ambos, comparar los 2 precios y devolverle al usuario esta información para que pueda decidir dónde comprar el producto. El mashup incluso podría buscar los precios de estos 2 proveedores periódicamente, para que el usuario pueda ir viendo el cambio de los mismos y decida el mejor momento para comprar [Yu, 2011].

Según las APIs que empleen los mashups, los hay de distinto tipo, por ejemplo[3]:

- Mashups que emplean mapas

  Muchas personas acumulan una gran cantidad de información sobre distintas cosas y

1 http://www.wikipedia.org
2 Application Programming Interface: Interfaz de Programación de Aplicaciones
3 http://www.masternewmedia.org/news/2007/08/09/mashups_what_are_they_mashup.htm

actividades, y están acostumbrados a anotar toda esta información junto con sus ubicaciones, la cual puede presentarse gráficamente empleando mapas. Uno de los grandes impulsores para estos tipos de mashups fue Google con su API Google Maps, permitiendo a los desarrolladores Web combinar todo tipo de información en un mapa. También aparecieron las APIs de Microsoft (Virtual Earth), Yahoo (Yahoo Maps) y AOL (MapQuest).

Ejemplo de mashup que emplea una API de mapas es "*Live train map for the London Underground*[4]", el cual muestra en un mapa el movimiento de los trenes de la red de subtes de Londres casi en tiempo real.

- Mashups de videos y fotos

  La aparición de sitios que guardan fotos y redes sociales, como Flickr[5], ha dado lugar a una variedad de mashups interesantes. Debido a que estos proveedores de contenido guardan metadatos asociados a las imágenes (por ejemplo, quién sacó la foto, una descripción, dónde y cuándo fue sacada, etc) los diseñadores pueden combinar las fotos con otra información que se pueda asociar con los metadatos.

  Ejemplo de mashup que emplea una API de fotos es "*AlphaLearnr*[6]", el cual ayuda a aprender el alfabeto mediante fotos.

- Mashups para realizar búsquedas y compras

  Los mashups para realizar búsquedas y compras han existido desde mucho antes que se adoptara el término mashup. Antes del surgimiento de APIs para la Web, herramientas de comparación de compras, como BizRate[7], PriceGrabber[8] y Froogle de Google[9] combinaban tecnologías del tipo *business-to-business* (B2B) y *screen scraping* para mezclar datos.

  Para facilitar la creación de estos mashups, eBay y Amazon (entre otros) lanzaron APIs para acceder a sus contenidos.

  Ejemplo de mashup que emplea una API de Amazon es "*22books*[10]", el cual permite crear, compartir y ver listas de libros.

- Mashups de noticias

  Fuentes de noticias (como el New York Times, la BBC o Reuters) utilizan tecnologías de sindicación como RSS y Atom para la difusión de noticias relacionadas con diversos temas.

4 http://traintimes.org.uk/map/tube/
5 http://www.flickr.com/
6 http://www.rapidmonkey.com/alphalearnr/#/C
7 http://www.bizrate.com/
8 http://www.pricegrabber.com/
9 http://www.google.com/shopping
10 http://www.22books.com/

Ejemplo de mashup que emplea APIs de noticias es "*Diggdot.us*[11]", el cual combina canales de fuentes relacionadas con información tecnológica como Digg.com[12], Slashdot.org[13] y Del.icio.us[14].

Estos tipos de mashups, también llamados *mashups Web 2.0*, presentan las siguientes características y ventajas[15]:

- Reutilización

  Se construyen con partes y servicios de otras aplicaciones Web ya existentes, agregando código para integrar los mismos. Al igual que el mercado de componentes visuales construido alrededor de los componentes ActiveX en los '90, los mashups son una forma de reutilización, pero a una escala mucho más grande.

- Modelo simple y liviano

  Generalmente se construyen usando técnicas como "cortar y pegar", fragmentos Javascript, canales (*feeds*) y XML para conectar las diferentes partes. Además, no requieren instalación, actualizaciones, plug-ins, derechos de administrador, etc, salvo un navegador Web y la URL del mashup.

- Facilidad de desarrollo

  Permiten la creación de aplicaciones que nunca podrían haber sido justificadas según la relación comprar/construir.

Sin embargo, los mashups Web 2.0 también tienen sus limitaciones [Yu, 2011]:

- Escalabilidad pobre

  Como los diferentes proveedores publican su información empleando diferentes APIs, esto implica un proceso de aprendizaje constante, debiendo aprender una API distinta cada vez que surge un nuevo proveedor. La construcción de estos tipos de mashups no resulta escalable, a la vez que exige un mantenimiento costoso.

- Difícil integración

  A medida que aumenta el número de fuentes de información con las que trabaja el mashup, los usuarios se benefician ya que cuentan con más información, pero al mismo tiempo aumenta el esfuerzo necesario para realizar la integración de las mismas, ya que cada una se encuentra organizada de distinta forma, empleando diferentes estándares.

11 http://doggdot.us/
12 http://digg.com/
13 http://slashdot.org/
14 http://delicious.com/
15 http://www.zdnet.com/blog/hinchcliffe/mashups-the-next-major-new-software-development-model/106

Con el surgimiento de la Web Semántica aparecieron los *mashups semánticos*. El concepto de Web Semántica, propuesto por Tim Berners-Lee en el año 2006[16], hace referencia a información publicada en la Web de tal forma que resulte legible a una máquina, que su significado esté explícitamente definido y que pueda conectarse con otra información externa. Conceptualmente, la Web Semántica se refiere a un conjunto de mejores prácticas para publicar y conectar información estructurada en la Web.

En la práctica, la idea de la Web Semántica se puede resumir como [Yu, 2011]:

- Usar el modelo de datos RDF[17] para publicar información
- Usar enlaces RDF para interconectar información de diferentes fuentes

Para aclarar el concepto de Web Semántica, se presenta a continuación una comparación con la Web "tradicional" [Yu, 2011]:

- En la Web tradicional, cualquiera puede publicar cualquier cosa en cualquier momento

  Lo mismo es cierto para la Web Semántica: cualquiera, en cualquier momento, puede publicar cualquier información, con la excepción que la misma tiene que estructurarse según un modelo llamado RDF. Al organizarse la información en un modelo RDF, la misma puede ser usada por máquinas, no por personas.

- Para acceder a la Web tradicional se emplean navegadores Web

  Lo mismo es cierto para la Web Semántica, sin embargo, como está organizada en documentos RDF, se utilizan navegadores semánticos que puedan entenderlos, pudiendo seguir los enlaces RDF para navegar las diferentes fuentes de información. Los navegadores Web tradicionales, por otro lado, están diseñados para tratar con documentos HTML, con lo cual no resultan como mejor opción para acceder a la Web Semántica.

- En la Web tradicional todo está relacionado entre sí

  Lo mismo es cierto para la Web Semántica. Un hecho importante, sin embargo, es que los documentos HTML contenidos en la Web tradicional están conectados por enlaces sin tipo. Para la Web Semántica se utilizan enlaces con tipo que conectan cosas arbitrarias en el mundo, con lo cual se pueden construir aplicaciones mucho más inteligentes.

Teniendo en cuenta estas características de la Web Semántica, la diferencia entre un mashup Web 2.0 y uno semántico es que en este último la información resulta legible a una máquina (su significado está explícitamente definido). Es decir, un mashup semántico es una aplicación Web que extrae información de diversas fuentes, la combina (integra) y presenta en otras formas, pero esta

16 http://www.w3.org/DesignIssues/LinkedData.html
17 Resource Description Framework: Framework de Descripción de Recursos

información presenta la característica de poder ser "interpretada" por una máquina [Yu, 2011].

Ejemplo de un mashup semántico es "*Revyu*[18]", desarrollado usando tecnologías y estándares de la Web Semántica, en el que cualquiera puede ingresar para calificar y realizar una crítica sobre cualquier cosa en el mundo. Toda crítica que se ingresa se expresa, además de su forma "tradicional", como un grafo RDF. *Revyu* también consume datos semánticos de la Web para mejorar la experiencia por parte del usuario.

Las principales características de los mashups semánticos son [Yu, 2011]:

- Escalabilidad buena

  En la Web Semántica, la información se expresa usando grafos y estándares RDF (un único conjunto de estándares), con lo cual no existe una API específica para que cada proveedor exponga su estructura de datos. Por lo tanto, la construcción y el mantenimiento resultan muy escalables, no habiendo necesidad de un aprendizaje constante de nuevas APIs.

- Fácil integración

  A medida que aumenta el número de fuentes de información con las que trabaja el mashup semántico, los usuarios se benefician ya que cuentan con más información, y el esfuerzo necesario para realizar la integración de las mismas permanece prácticamente constante, ya que el mashup semántico trata a las fuentes de manera uniforme.

1.1 Motivación e hipótesis de esta investigación

La Web Semántica representa un desarrollo importante sobre la Web tradicional, y surgió en el año 2001 a partir de un trabajo de Tim Berners-Lee [Berners-Lee *et al.*, 2001] en el cual se describía la idea de agregar más significado a la información disponible en la Web, y hacerlo de tal forma que la misma no sólo pudiera ser *leída* por las aplicaciones de software, sino también *interpretada*.

En este trabajo se presenta el concepto de Web Semántica, el cual implica tener la información definida y organizada de tal forma que pueda ser usada por las aplicaciones de software no sólo para su visualización, sino también para automatización, integración y reutilización en varias aplicaciones.

De esta forma, Berners-Lee, Hendler y Lassila vieron a la Web Semántica como una evolución, natural y necesaria, de la Web, con menos limitaciones sintácticas en cuanto al manejo de la información.

Se dijo que un mashup semántico es una aplicación Web que extrae información (legible a una máquina y con un significado explícitamente definido) de diversas fuentes, la combina (integra) y presenta en otras formas.

18 http://revyu.com/

Con el fin de aclarar el funcionamiento de un mashup semántico, se presenta a continuación el siguiente ejemplo: suponiendo que existe un agente "inteligente" que recorre la Web buscando información, se quiere saber todo lo relacionado con una persona determinada [Yu, 2011].

Si el agente, al ir recorriendo la Web, cada página fuera un típico documento Web, no podría recopilar nada útil. Lo único que podría entender de cada página serían las etiquetas HTML, las cuales, además de informar al navegador sobre cómo presentar la información, no transmiten nada útil sobre los recursos subyacentes (para el agente, cada página Web es una cadena de caracteres).

Si en cambio las páginas Web no fueran las "tradicionales", además de las etiquetas HTML, contendrían algunas "declaraciones" que podrían ser recopiladas por el agente. Por ejemplo, suponiendo que la primera página que accede el agente sea la página personal de la persona (http://www.liyangyu.com), ésta podría proporcionar las siguientes declaraciones:

```
ns0:LiyangYu ns0:nombre "Liyang Yu".
ns0:LiyangYu ns0:sobrenombre "LaoYu".
ns0:LiyangYu ns0:autor <ns0:_x>.
ns0:_x ns0:ISBN "978-1584889335".
ns0:_x ns0:editorial <http://www.crcpress.com>.
```

Así, la primera declaración se puede leer:

El recurso `ns0:LiyangYu` tiene un `ns0:nombre` cuyo valor es `Liyang Yu`.

O así:

El recurso `ns0:LiyangYu` tiene una propiedad `ns0:nombre` cuyo valor es `Liyang Yu`.

La tercera declaración es distinta a las 2 primeras. Al especificar el valor de la propiedad `ns0:autor` para el recurso `ns0:LiyangYu`, en lugar de utilizar una cadena de caracteres como valor, utiliza otro recurso, el cual está identificado por `ns0:_x` (encerrado entre < y > para que resulte más evidente).

Suponer que el siguiente sitio Web que recorre el agente sea www.amazon.com. Igual que antes, si la información de este sitio fuera un típico documento Web, el agente no podría hacer mucho, pero si proporcionara las siguientes declaraciones:

```
ns1:libro-1584889330 ns1:ISBN "978-1584889335".
ns1:libro-1584889330 ns1:precio USD62.36.
ns1:libro-1584889330 ns1:revision "4.5 estrellas".
```

Al comparar estas declaraciones con las primeras, se aprecia que el recurso `ns0:_x` representa exactamente el mismo ítem indicado por el recurso `ns1:libro-1584889330`.

Una vez que se hace esta conexión surgen otras. Por ejemplo, una persona que tiene una página Web con la URL http://www.liyangyu.com, tiene un libro publicado y el precio actual del mismo (en

Amazon) es U$S 62.36. Lo más importante es que este hecho no se menciona explícitamente en ninguno de los sitios Web por separado. Es la persona quien ha integrado la información de ambos para llegar a esta conclusión.

Para el agente, esta integración tampoco resulta difícil: al ver el número `ISBN 978-1584889335` en ambas declaraciones, hace una conexión entre las 2, agregando automáticamente la siguiente declaración:

```
ns0:_x sameAs ns1:libro-1584889330.
```

En este ejemplo, el agente combina (integra) la información proveniente de más de una fuente. Para que el agente pueda hacer esta integración, la información se debe estructurar de tal forma que no sólo la pueda leer, sino también "comprender".

En este ejemplo, el funcionamiento del agente es el de un mashup semántico: extraer información (estructurada de tal forma que la pueda "comprender") de distintas fuentes, integrarla y presentarla.

La estructura donde cada fuente organiza su información con esta característica se denomina *ontología*, y en consecuencia la tarea del mashup semántico consiste en integrar las distintas ontologías, y es el problema más complejo de resolver por parte del mashup semántico. Este problema recibe el nombre de *mediación de ontologías*.

El objetivo de este trabajo consiste en definir una metodología para el diseño de mashups semánticos, que ayude a los desarrolladores a construirlos desde cero.

Esta metodología estará formada por una serie de procesos, los cuales se descompondrán en un conjunto de actividades. Para la descripción de cada proceso y actividad, se especificará una introducción; directrices detalladas propuestas para realizar el proceso o la actividad (definición, objetivo principal a lograr, entrada, salida, actores involucrados, el momento en que se tiene que realizar y los detalles de realización).

Finalmente, la metodología propuesta se evaluará mediante un caso de estudio.

1.2 Objetivos

El objetivo principal de este trabajo es proponer una metodología de diseño para mashups semánticos, para lo cual habrá que:

- Analizar el principal problema a resolver en el diseño de mashups semánticos, es decir, la integración de la información proveniente de las distintas fuentes.

- Analizar metodologías existentes para el desarrollo de aplicaciones Web e investigar cuáles contemplan el manejo de información semántica.

- Proponer una metodología de diseño específica para el caso de los mashups semánticos.
- Validar la metodología de diseño a través de un caso de estudio.

1.3 Aporte de esta tesis

Del cumplimiento de los objetivos de este trabajo se desprenden las siguientes contribuciones:

- Explicar las distintas razones por las cuales se dificulta la integración de información proveniente de distintas fuentes.
- Especificar una metodología para el diseño de mashups semánticos.
- Explicar las soluciones posibles para distintos casos donde se deba integrar información.

1.4 Organización

En el Capítulo 2, “Estado del Arte”, se presenta brevemente la evolución de la Web y sus aplicaciones, se mencionan características propias de los mashups semánticos y los desafíos que impone el desarrollo de estos tipos de aplicaciones. Finalmente se analizan brevemente algunos de los métodos principales para el diseño de aplicaciones Web y se discute su adecuación a los mashups semánticos.

En el Capítulo 3, “Ontologías y la Web Semántica”, se presenta el concepto de ontología y algunos de los lenguajes para su representación, como ser RDF, RDFS y OWL.

En el Capítulo 4, “Mediación de ontologías” se explica el principal problema a resolver por parte de un mashup semántico: la mediación de ontologías. Se muestran las técnicas existentes y adoptadas en el caso práctico desarrollado en el Capítulo 6.

En el Capítulo 5, “Metodología para el diseño de mashups semánticos” se presenta una propuesta metodológica para el diseño de mashups semánticos.

En el Capítulo 6, “Caso de estudio”, se presenta un caso práctico, desarrollado íntegramente siguiendo la propuesta metodológica presentada en el Capítulo 5.

En el Capítulo 7, “Conclusiones”, se incluyen las conclusiones y limitaciones al presente trabajo, y trabajos futuros.

En el Apéndice A, “RDF”, se desarrollan en detalle los conceptos de RDF.

En el Apéndice B, “RFDS”, se desarrollan en detalle los conceptos de RDFS.

En el Apéndice C, “Técnicas para el mapeo de ontologías”, se nombran algunas técnicas para el mapeo de ontologías. El mapeo de ontologías es una forma que puede tomar la mediación de ontologías.

CAPÍTULO 2 - ESTADO DEL ARTE

Desde hace unos años, la mayor parte de la infraestructura de datos se basa en la Web, formada tanto por usuarios como por proveedores de datos. La Web se ha convertido en parte fundamental de la vida diaria de la gente, al igual que las redes que proporcionan la energía eléctrica y las comunicaciones, por ejemplo. Hoy en día, las personas se informan, se educan, se entretienen por medio de aplicaciones y servicios Web [Rossi *et al.*, 2008].

Toda esta infraestructura está unida mediante una compleja interconexión de hardware, software, estándares internacionales y prácticas aceptadas. Para hacer frente al crecimiento de los sistemas Web y asegurar su eficacia, confiabilidad y mantenimiento, ha surgido la disciplina de "Ingeniería Web", la cual combina las prácticas tradicionales de gestión de proyectos y desarrollo de software con un proceso que va evolucionando a través de innovaciones, tales como la "Web Semántica" y la "Web 2.0".

Muchos desarrolladores siguen viendo al desarrollo Web sólo como la creación de páginas HTML, pasan por alto los requisitos a nivel sistema y consideraciones clave de diseño, no hacen uso de metodologías de desarrollo y diseño Web, etc. Además, erróneamente, desarrollan estos sistemas de la misma forma que los sistemas "tradicionales" [Murugesan, 2008].

El objetivo de este capítulo consiste en exponer los principales trabajos relacionados con metodologías para el desarrollo de aplicaciones Web, presentando, con el fin de facilitar su comprensión, una breve reseña sobre la evolución de la Web y sus aplicaciones, comentando sus características propias en cuanto a desarrollo y los principales aspectos a tener en cuenta.

2.1 Evolución de la Web y sus aplicaciones

La evolución de la Web se puede analizar desde distintas perspectivas [Murugesan, 2008]:

- el crecimiento (número) de sitios y páginas Web
- el número de usuarios
- la cantidad de visitas
- la funcionalidad y la interactividad que ofrecen las aplicaciones Web
- las tecnologías utilizadas para la creación de aplicaciones Web
- la repercusión social y en los negocios de la Web
- una combinación de todos estos

Asimismo, esta evolución ha reunido distintas disciplinas, tales como tecnologías de medios de comunicación, ciencias de la información y ciencias de la comunicación, facilitando la creación, mantenimiento, distribución y uso de diferentes tipos de información desde cualquier lugar y momento, empleando una variedad de dispositivos como computadoras de escritorio, portátiles y de bolsillo, PDAs, teléfonos móviles, etc.

En lo que refiere al diseño Web, resulta útil clasificar los sistemas y aplicaciones Web según sus características y tecnologías utilizadas para su creación de la siguiente manera:

- la Web estática
- la Web dinámica
- la Web 2.0
- la Web móvil
- la Web Semántica

2.1.1 La Web 1.0

La "Web estática" es, ante todo, una colección de páginas HTML (estáticas) que proporcionan información sobre productos, servicios ofrecidos, etc. Al comienzo, la mayoría de los sitios Web eran simplemente una colección de páginas estáticas. Después de un tiempo, la Web se volvió dinámica, generando páginas "sobre la marcha". La capacidad de crear páginas Web a partir del contenido almacenado en bases de datos permitió a los desarrolladores proporcionar información personalizada a los visitantes. Estos tipos de sitios son conocidos como "Web dinámicos" [Murugesan, 2008].

A pesar que un visitante de estos sitios Web obtiene información personalizada según sus requisitos, la interacción en los mismos es principalmente en un solo sentido, y la interactividad del usuario se encuentra limitada (los usuarios no cumplen ningún papel en la generación de contenido). A los sitios Web estáticos o dinámicos, que no tienen, o tienen muy poca, interacción con el usuario, se los conoce como Web 1.0.

Las principales características de la Web 1.0 son [Taylor *et al.*, 2008]:

- Aplicaciones de una única función
- Publicación de bases de datos privadas
- El intercambio e integración de información sólo es posible en la mente del usuario (la información se encuentra en "silos")

2.1.2 La Web 2.0

En los últimos años han surgido unos nuevos tipos de aplicaciones Web, conocidos como aplicaciones Web 2.0 (o Aplicaciones Orientadas a Servicios, SOA), las cuales permiten a las personas colaborar y compartir información. Ejemplos incluyen sitios de redes sociales, sitios de colaboración, etc [Murugesan, 2008].

Esta segunda generación de aplicaciones Web ofrece interfaces más inteligentes y facilidades para los usuarios para generar y editar contenido y, así, enriquecer los mismos.

Además de aprovechar el potencial de los usuarios para generar contenido, facilitan su mantenimiento (por ejemplo, mediante el uso de etiquetas) y acceso (por ejemplo mediante la suscripción a canales de noticias). Estos nuevos tipos de aplicaciones Web son también capaces de integrar múltiples servicios en una interfaz de usuario rica (ver Sección 2.1.5).

Con la incorporación de nuevas tecnologías Web como AJAX, Ruby, blogs, wikis, marcadores sociales y etiquetas, la Web se está volviendo cada vez más dinámica e interactiva, donde los usuarios no sólo pueden acceder al contenido de un sitio, sino también contribuir al mismo.

Todo este conjunto de tecnologías/estándares/formatos también constituye una desventaja al momento de integrar toda esta información, ya que la misma se organiza de distintas formas, y por sobre todo, está pensada para ser interpretada por las personas, no por máquinas. En la sección 1.1 se presenta un ejemplo sobre esto.

Otra característica de la nueva Web es la proliferación de APIs, las cuales facilitan a los desarrolladores la obtención de información y la creación de nuevas aplicaciones basadas en la misma.

Las principales características de la Web 2.0 son [Taylor *et al.*, 2008]:

- Se potencian los “silos” de contenido
- Las APIs facilitan la integración de información, aunque son los usuarios quienes realizan la mayor parte de esta tarea, ya que la información (generada por usuarios o por propietarios), no resulta fácilmente accesible ni transferible (aún permanece en silos)
- Surge la “Web social”

2.1.3 La Web móvil

Los avances en la computación móvil y las comunicaciones inalámbricas, junto con la adopción generalizada de dispositivos móviles (teléfonos inteligentes, PDAs y PCs de bolsillo), están permitiendo a un número creciente de usuarios acceder a la Web utilizando dispositivos de mano [Murugesan, 2008].

Los teléfonos móviles compiten con las computadoras como plataforma para el acceso a la Web. Esta tendencia va en aumento, a medida que los teléfonos inteligentes se vuelvan cada vez más asequibles, que un mayor número de personas los usan y que más aplicaciones Web migran a la Web móvil e inalámbrica.

Las aplicaciones Web móviles pueden también ofrecer algunas características adicionales en comparación a las aplicaciones Web tradicionales de escritorio, tales como servicios de localización, capacidades sensibles al contexto y personalización.

2.1.4 La Web Semántica

En las aplicaciones Web actuales, la información se presenta en lenguaje natural, lo cual resulta fácilmente procesable a los seres humanos, no así a las aplicaciones. La Web Semántica tiene por objetivo superar esta barrera [Murugesan, 2008].

Según Tim Berners-Lee [Berners-Lee *et al.*, 2001], la Web Semántica es una extensión de la Web actual, donde a la información se le da un significado bien definido, facilitando a las aplicaciones y a la gente trabajar en cooperación.

El concepto de Web Semántica implica tener información definida y organizada de tal forma que pueda ser usada por las aplicaciones de software no sólo para su visualización, sino también para automatización, integración y reutilización en varias aplicaciones. Asociar significado con contenido o establecer una capa de datos comprensibles a las aplicaciones permite aplicaciones más inteligentes, facilitando también los servicios interoperables.

El objetivo último de la Web Semántica es soportar el intercambio global de información de una manera escalable, adaptable y extensible, para que la misma pueda ser utilizada para un descubrimiento más efectivo, para automatización, integración y reutilización a través de varias aplicaciones.

2.1.5 Aplicaciones RIA (Aplicaciones Enriquecidas)

Las aplicaciones RIA[19] son aplicaciones Web que se ejecutan en un navegador, no requieren la instalación de software adicional y tienen las mismas características y funcionalidad de las aplicaciones tradicionales de escritorio. RIA representa la evolución del navegador, desde una interfaz estática del tipo pedido-respuesta, a una interfaz dinámica y asincrónica. Algunos ejemplos de aplicaciones RIA incluyen a Google Earth, Gmail, etc [Murugesan, 2008].

La construcción de aplicaciones Web utilizando este tipo de tecnología, sin embargo, no garantiza una mejor experiencia por parte del usuario. Para agregar valor, los desarrolladores deben abordar las necesidades reales de los usuarios e implementar técnicas estructuradas para la realización de pruebas a fin de comprender y validar el uso y diseño apropiados para las mismas.

2.2 Aspectos propios del desarrollo de aplicaciones Web

Para el diseño de mejores sistemas y aplicaciones Web resulta esencial comprender correctamente las características y demandas puestas sobre estos tipos de aplicaciones, los cuales tienen ciertas características únicas que diferencian su desarrollo del software tradicional.

El entorno operativo de las aplicaciones Web, junto con su enfoque de desarrollo e implantación

19 Rich Internet Application: Aplicaciones de Internet Enriquecidas

más rápido, las diferencian de las aplicaciones tradicionales. Además, se pone mayor énfasis en la seguridad, ya que resultan más susceptibles a violaciones que las aplicaciones tradicionales.

Algunas de las características principales de las aplicaciones Web son [Murugesan, 2008]:

- La mayoría son evolutivas por naturaleza, requiriendo frecuentes cambios en cuanto a contenido, funcionalidad, estructura, navegación, presentación o implementación. En particular, evolucionan en términos de sus requisitos y funcionalidad, especialmente después que se ponen en uso. En la mayoría de los casos, la frecuencia y la magnitud de los cambios son mucho mayores que en las aplicaciones tradicionales. Por lo tanto, gestionar con éxito su evolución, cambio y los nuevos requisitos resulta un desafío grande en lo técnico, en lo organizativo y en lo relativo a la gestión.

- Están pensadas para ser utilizadas por una comunidad de usuarios grande, diversa y remota, con diferentes necesidades, expectativas y habilidades. Por lo tanto, la interfaz de usuario y las características de usabilidad deben satisfacer las necesidades de una comunidad diversa y anónima. Por otra parte, el número de usuarios que acceden en cualquier momento es impredecible, pudiendo variar bastante, creando problemas de rendimiento.

- Exigen la presentación de la una variedad de contenido: texto, imágenes, audio, video, etc. Por lo tanto, su desarrollo incluye la creación y gestión de contenido y presentación de una manera atractiva, y una posterior gestión de cambios en forma continua después del desarrollo inicial e implantación.

- Por lo general demandan un mayor grado de atractivo estético, junto con una fácil navegación.

- Frecuentemente necesitan contemplar el uso de múltiples idiomas, diferentes sistemas de unidades, etc.

- Las necesidades de seguridad y privacidad son en general más exigentes que en las aplicaciones tradicionales.

- Necesitan soportar una variedad de dispositivos de visualización y formatos, hardware, software y redes con velocidades de acceso muy diferentes.

- Las ramificaciones en cuanto a las fallas o la insatisfacción de los usuarios pueden ser mucho peores que para el caso de los sistemas convencionales. Por otra parte, las aplicaciones Web pueden fallar por muchas razones diferentes.

- Los tiempos de desarrollo son más cortos, y esto influye significativamente en las metodologías de diseño y desarrollo y en el proceso que se adopte para su desarrollo.

- La proliferación de nuevas tecnologías Web, normas y la presión competitiva para usarlas trae sus propias ventajas y también desafíos adicionales para el desarrollo y mantenimiento.

- La naturaleza evolutiva requiere un proceso de desarrollo incremental.

2.3 Aspectos propios del desarrollo de mashups semánticos

A diferencia de muchos estándares y protocolos de Internet, los mashups no surgieron a partir de un proceso de diseño, sino a medida que la gente empezó a combinar los estándares y protocolos existentes en formas innovadoras [Feiler, 2008].

Desde la visión del usuario, un mashup generalmente se caracteriza por presentar información específica sin requerir del usuario mucha interacción. Es decir, es el mashup y no el usuario quien realiza la síntesis de la información para poder presentarla. Una forma de describir los mashups consiste en considerarlos como "administradores de complejidad", con lo cual se incluyen los 2 tipos principales: los multifuente y los de presentación [Feiler, 2008]:

- Mashups multifuente

  Muchos mashups combinan información de 2 o más fuentes. Por ejemplo, para un código postal dado, un mashup podría combinar la información del censo junto con la información económica; otro podría combinar los restaurantes y las salas de cine, etc. El corazón de un mashup multifuente consiste en la combinación de fuentes de información agregando valor.

- Mashups de presentación

  Un mashup no necesita combinar 2 fuentes de información. En su lugar, puede utilizar las tecnologías de la Web 2.0 para presentar la misma información de maneras diferentes, por ejemplo, en forma de texto y en un mapa.

Con el fin de caracterizar el funcionamiento de un mashup, se muestra a continuación un mashup existente llamado *Beardscratchers Compendium*[20], el cual busca y filtra información relacionada a la música en distintas fuentes de información en la Web (YouTube, Yahoo Search, Yahoo Music, Wikipedia, Twitter y Flickr entre otras). Este mashup se encuentra disponible en un repositorio llamado *ProgrammableWeb*[21], y en la figura 2.3.1 puede verse la pantalla inicial del mismo:

20 http://beardscratchers.com/

21 http://www.programmableweb.com/

Figura 2.3.1 – Pantalla inicial del mashup "Beardscratchers Compendium"

Como se puede ver de la figura 2.3.1, el mashup solicita que se ingrese el artista musical sobre el que se quiere obtener información, luego consulta las distintas fuentes de información y muestra en una segunda pantalla (figura 2.3.2) información relacionada con el perfil del artista, discografía, fotos, videos, fechas de sus próximas presentaciones, etc.

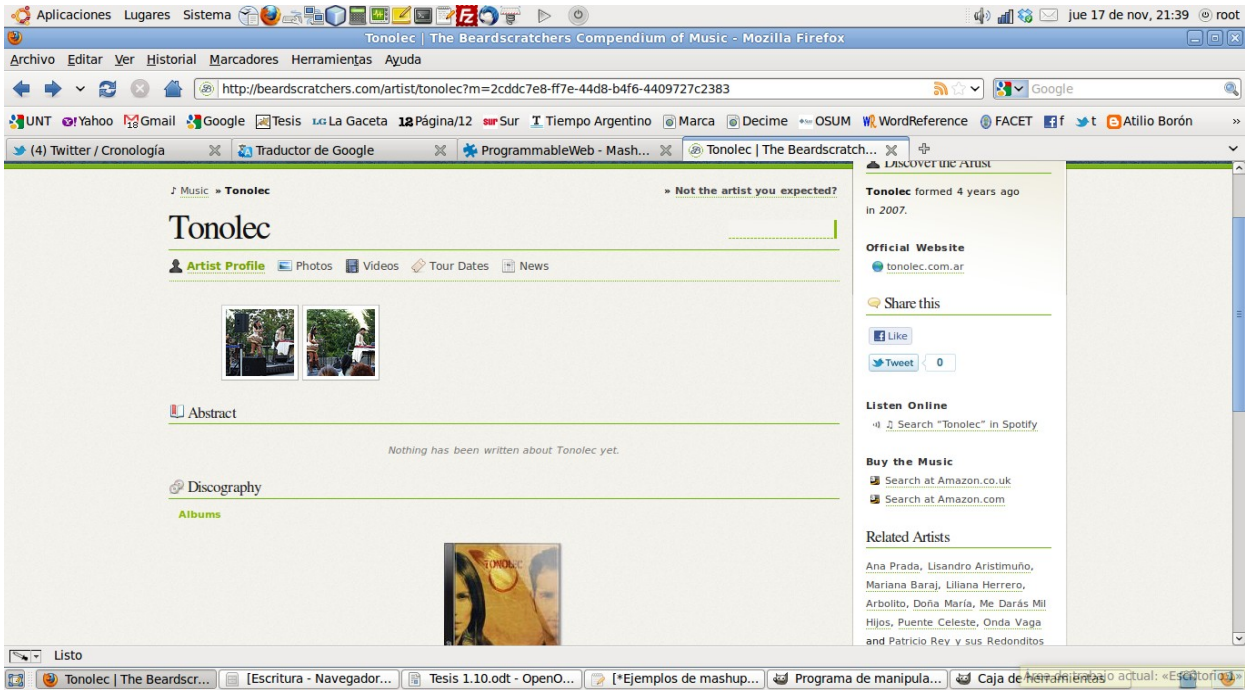

Figura 2.3.2 – Segunda pantalla del mashup "Beardscratchers Compendium"

Para que este (o cualquier otro) mashup pueda tomar información de distintas fuentes y combinarla, la misma debe representarse de tal forma que pueda ser leída y procesada por el mismo.

Uno de los problemas que surge es que cada fuente brinda su información organizándola en distintas estructuras y empleando diferentes lenguajes. Así, hay fuentes que emplean estructuras relacionales usando bases de datos (SQL Server, MySQL, Oracle, PostgreSQL, etc), otras que emplean hojas de cálculo como Excel, otras que emplean canales (*feeds*) de noticias como RSS/Atom, otras que emplean servicios Web, etc.

Por esta razón, si bien la utilidad de los mashups Web 2.0 aumenta a medida que crece la cantidad de fuentes de información (los usuarios se benefician ya que cuentan con más información para sus aplicaciones), también aumenta la complejidad de integración de las mismas (con un número pequeño de fuentes, la integración de la información no presenta gran complejidad, pero a medida que aumenta la cantidad de fuentes, aumenta la complejidad de integración [Taylor *et al.*, 2008]).

Debido al problema de integración que presentan los mashups Web 2.0, surgen los mashups semánticos. Como se dijo antes, la Web Semántica hace referencia a información publicada en la Web de tal forma que resulte legible a una máquina, que su significado esté explícitamente definido y que pueda conectarse con información externa. En la práctica, este concepto se puede resumir como [Yu, 2011]:

- El uso de un modelo de datos RDF para publicar datos estructurados en la Web

- El uso de enlaces RDF para interconectar datos de diferentes fuentes de datos

Es decir, todo lo que se publica en la Web Semántica son documentos RDF (documentos para ser usados por máquinas, no por personas). Al emplear un único conjunto de estándares (RDF), cada fuente expone su información en una misma estructura, llamada ontología, la cual puede representarse en distintos lenguajes (RDF, RDFS, OWL) según su complejidad.

Una vez que las fuentes tienen organizada su información en ontologías, un mashup semántico las deberá integrar. Este problema, llamado *mediación de ontologías*, es el más complejo a resolver, ya que cada ontología por lo general emplea sus propios términos y relaciones.

Según lo explicado anteriormente, se puede hacer la siguiente caracterización de un mashup semántico:

- Trabaja con distintas fuentes, cada una de las cuales organiza su información en su propia ontología, empleando sus propios términos y relaciones, lo cual da lugar al problema conocido como "*mediación de ontologías*" al momento de integrar toda esta información.

- La información con la que trabaja se encuentra estructurada en un único conjunto de estándares (RDF), con lo cual puede tratar a todas las fuentes de información de forma uniforme [Taylor *et al.*, 2008].

- Al igual que un mashup Web 2.0, en cuanto a la interfaz de usuario suele ser más simple que

una aplicación Web tradicional, requiriendo una navegación fácil y rápida sin demasiada interacción por parte de los usuarios.

- Exige la presentación de una variedad de contenido: texto, imágenes, audio, video, etc.

2.4 Desafíos del desarrollo de mashups semánticos

Como se dijo antes, un mashup semántico es una aplicación Web que extrae información de diversas fuentes, la combina (integra) y presenta en otras formas, pero esta información presenta la característica de poder ser "interpretada" por una máquina.

A las estructuras donde cada una de estas fuentes organizan toda su información se las denomina ontologías. Las ontologías modelan la estructura de la información (por ejemplo, la representación de un conjunto de clases y sus propiedades o atributos), la semántica de la información (en forma de axiomas que expresan restricciones, como relaciones de herencia o restricciones sobre las propiedades) y las instancias de la información (llamadas generalmente individuos). Para integrar las ontologías se debe entender la relación entre las estructuras (clases, propiedades, relaciones) en las diferentes ontologías, y usar la semántica de las mismas para modelar estas relaciones y así crear una ontología integrada y consistente [Udrea *et al.*, 2007].

Como ejemplo, considerar 2 fuentes de información que clasifican diferentes tipos de cámaras de foto [Walton, 2007]. Suponer además que ambas fuentes guardan su información empleando diferentes formatos, por ejemplo, la fuente 1 emplea una base de datos relacional SQL, mientras que la fuente 2 emplea XML [Euzenat & Shvaiko, 2007].

Como un mashup semántico es una aplicación Web que integra información (información semántica) de distintas ontologías, el primer paso consiste en "transformar" la representación de cada fuente en una ontología (ver figura 2.4.1).

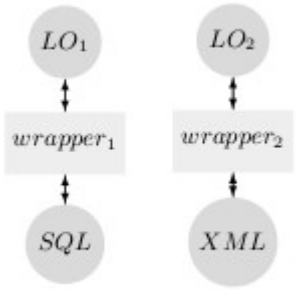

Figura 2.4.1 – Transformación de la estructura de cada fuente en una ontología

Así, la fuente 1, que organiza su información en una base de datos relacional, mediante el "*wrapper*$_1$" transforma esta representación en la ontología LO_1, y el "*wrapper*$_2$" hace lo mismo con la fuente 2, obteniendo la ontología LO_2.

Luego, suponer que las ontologías LO_1 y LO_2 tienen la estructura mostrada en las figuras 2.4.2 y

2.4.3 respectivamente:

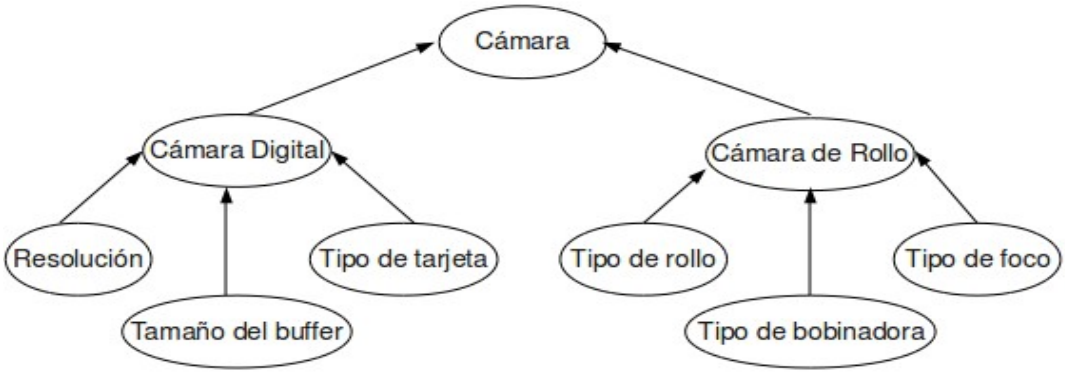

Figura 2.4.2 – Ontología LO_1 para clasificar los diferentes tipos de cámaras de foto

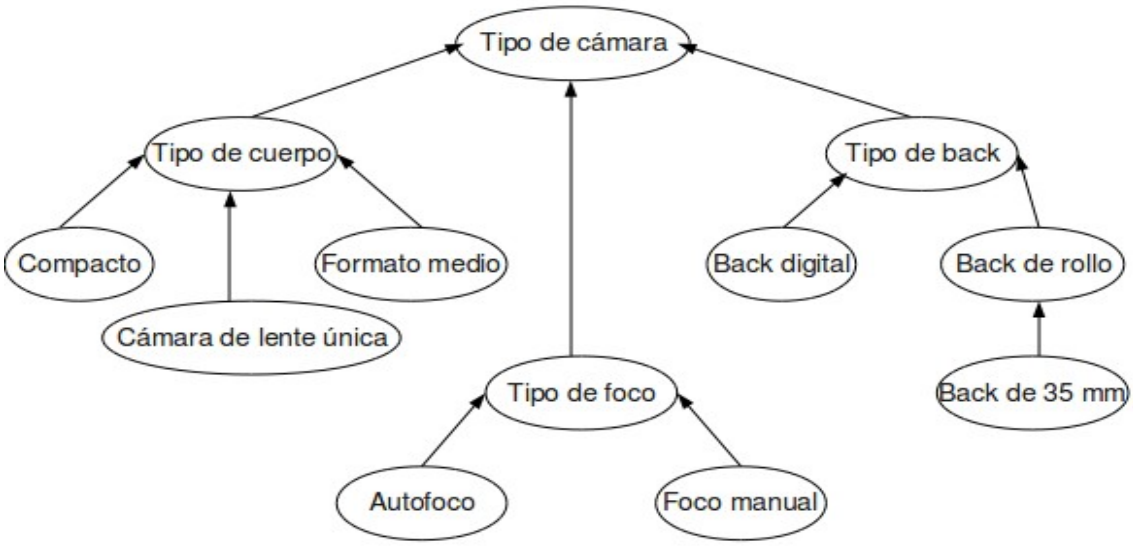

Figura 2.4.3 – Ontología LO_2 para clasificar los diferentes tipos de cámaras de foto

La integración de estas ontologías consiste en la iteración de los siguientes pasos [Klein, 2001]:

1. Encontrar los lugares donde ambas se solapan.
2. Relacionar los conceptos semánticamente "cercanos" mediante relaciones de equivalencia y subsunción.
3. Verificar la consistencia, coherencia y no redundancia del resultado.

Para la implementación de estos pasos se han propuesto distintas soluciones, cada una desde puntos de vistas distintos, abarcando bases de datos, sistemas de información, inteligencia artificial, etc. Estas soluciones aprovechan las diversas propiedades de las ontologías (estructura, instancias de datos, semántica, etiquetas, etc), usan y comparten técnicas de diferentes campos (estadística y

análisis de datos, aprendizaje automático, razonamiento automatizado, lingüística, etc) y atacan problemas similares difiriendo en la forma en que combinan y explotan sus resultados. Como consecuencia, son difíciles de comparar y describir [Euzenat & Shvaiko, 2007].

El objetivo de este trabajo consiste en definir una metodología para diseñar un mashup semántico, la cual, por todo lo explicado antes, deberá contemplar el problema de la integración de las distintas ontologías.

2.5 Acceso a las fuentes de información

Se dijo que cada fuente puede organizar su información de distinta forma, empleando estructuras relacionales (bases de datos SQL Server, MySQL, Oracle, PostgreSQL, etc), hojas de cálculo como Excel, canales (*feeds*) de noticias como RSS/Atom, servicios Web, etc.

Para acceder a la información que proporciona una fuente, un mashup puede [Feiler, 2008]:

- *Utilizar un servicio Web para obtener información desde un recurso del cual no tiene control*. En estos casos, el mashup accede a la información que brinda el servicio Web mediante una API proporcionada por el mismo. Ejemplos de estos tipos de servicios son los proporcionados por eBay, Amazon, Flickr, Google, etc.

- *Obtener información desde un recurso sobre el cual sí tiene control*. Puede ser una base de datos, un archivo o datos que estén en el mismo código del mashup. Como el mashup sí tiene control sobre estos recursos, puede usar una API o no.

- *Obtener información desde un recurso sobre el cual no tiene control sin usar una API*. Ejemplos de esto son los canales (feeds) de noticias RSS. Si estos canales son públicos, cualquiera puede leerlos sin utilizar una API propietaria.

2.5.1 Servicios Web

Originalmente la Web consistía de documentos (páginas Web) y enlaces entre los mismos. La idea inicial fue desarrollar una base de datos con información universal que pudiera ser accedida de forma simple y confiable por los consumidores [Cardoso, 2007].

Distintas organizaciones aprovecharon estas tecnologías para gestionar, organizar y distribuir información entre sus clientes y socios, y a medida que comenzaron a implementar soluciones para comercio electrónico se dieron cuenta que las mismas no resultaban suficientes para vender sus productos por Internet.

Por otro lado, debido a la globalización, las organizaciones fueron fusionándose progresivamente, creando entornos compuestos por sistemas, aplicaciones, procesos y fuentes de datos heredados, autónomos, distribuidos y heterogéneos, surgiendo la necesidad de interconectarlos a fin de

satisfacer las crecientes expectativas de sus clientes y del negocio y mejorar su productividad y eficiencia, lo cual dio lugar al desarrollo y despliegue de soluciones para integración de aplicaciones empresariales (EAI).

La mayoría de las soluciones para EAI requerían de protocolos y formatos costosos y propietarios, que presentaban muchas dificultades al momento de integrar sistemas internos con sistemas externos ejecutándose en otras computadoras. Para lograr este nivel de integración, se desarrollaron soluciones del tipo *business-to-business* (B2B), basadas en el uso de XML para representar datos, que al ser independientes de la plataforma y del proveedor, permitían procesar cualquier documento que se ajustara al mismo.

Las soluciones de B2B muchas veces presentaban un fuerte acoplamiento entre las aplicaciones, limitando la flexibilidad de los sistemas. Como resultado, y para superar estas limitaciones, surgió el concepto de *arquitectura orientada a servicios* (SOA[22]), el cual define un método de diseño, desarrollo, despliegue y gestión de componentes computacionales discretos para lograr la estructuración, bajo acoplamiento y estandarización de funcionalidad entre las aplicaciones de software.

Si bien hay muchas posibilidades diferentes para implementar una arquitectura SOA (servicios Web, Java RMI, DCOM y CORBA), los servicios Web son actualmente la solución más deseable, ya que eliminan muchos de los problemas de interoperabilidad entre las aplicaciones.

Los servicios Web están basados en tecnología de computación distribuida y proporcionan un medio estándar de interoperabilidad entre diferentes aplicaciones de software, utilizando protocolos y formatos XML, y cumplen con varios estándares de la Web, como WSDL[23] y SOAP[24], haciendo a esta plataforma de servicios independiente del lenguaje y del proveedor.

Se puede definir a los servicios Web como entidades computacionales accesibles a través de Internet mediante interfaces independientes de la plataforma y del lenguaje de programación [Preist, 2004].

2.5.2 Servicios Web semánticos

Los servicios Web prometen hacer frente a las necesidades de integración de aplicaciones proporcionando un marco basado en estándares (WSDL, UDDI[25], SOAP, etc) para intercambiar información entre las mismas de forma dinámica. Actualmente, estos estándares están diseñados para representar la información sobre las interfaces de los servicios, la forma en que se despliegan y cómo llamarlos, pero están limitados para expresar cuáles son las capacidades y requerimientos de los servicios, impidiendo la integración automática de aplicaciones [Cardoso, 2007].

Si todos los proveedores de servicios acordaran un formato estándar para la representación de sus

22 Service Oriented Architecture: Arquitectura Orientada a Servicios
23 Web Services Definition Language: Lenguaje de Definición para Servicios Web
24 Simple Object Access Procotol: Protocolo de Acceso Simple a Objetos
25 Universal Description, Discovery and Integration: Descripción Universal, Descubrimiento e Integración

servicios, no habría necesidad de esta semántica. Como no todas las aplicaciones, y sus servicios correspondientes, se pueden estandarizar, esto lleva a disparidades en las especificaciones de los servicios tanto en los proveedores como en los solicitantes.

Un primer paso hacia la solución de este problema es "elevarse por encima de estas diferencias en la representación de las interfaces de los servicios e identificar las similitudes semánticas entre los mismos" [Paolucci *et al.*, 2002]. El agregado de semántica para representar las necesidades y capacidades de los servicios Web es esencial para la automatización del descubrimiento y ejecución de los mismos.

Esta necesidad de semántica en los servicios Web ha llevado a la convergencia de los conceptos "servicios Web" y "Web semántica", dando lugar a los "servicios Web semánticos". Los servicios Web semánticos son servicios Web cuyas "propiedades, capacidades, interfaces y efectos están codificados de forma no ambigua y son interpretables por una máquina" [McIlraith *et al*., 2001].

Si a los servicios Web y a la información disponible al mashup se les agrega esta semántica, se podrá lograr que la aplicación tome decisiones inteligentes. Por ejemplo, el mashup semántico comprenderá qué servicios representan información de mapas, cuáles representan tiendas en línea, etc. Y a partir de estos servicios, el mashup semántico sabrá qué información representa un título, una descripción, un precio, etc [Chase, 2007A].

Actualmente con 2 tendencias, OWL-S[26] y WSMO[27], los servicios Web semánticos prometen un sistema que comprenda qué representan los datos subyacentes, pudiendo utilizarlos de maneras específicas.

2.6 Estado del desarrollo de mashups

En general, el desarrollo de mashups sigue siendo en gran medida una actividad *ad hoc*: se generan manualmente utilizando tecnologías como HTML, CSS y JavaScript, resultando una tarea que consume mucho tiempo y difícil al usuario típico de Internet.

En los últimos años, se han propuesto una serie de herramientas para simplificar el proceso de creación de mashups [Fischer *et al.*, 2009]:

- Herramientas que emplean programación

  Son herramientas que utilizan un IDE para crear los mashups. IBM WebSphere sMash[28] es un entorno de desarrollo y ejecución para aplicaciones Web dinámicas que permite una fácil reutilización y rápida integración de diferentes servicios Web. BungeeConnect[29] es otra plataforma que permite a los usuarios crear aplicaciones Web, automatizando la importación de servicios Web públicos, así como también bases de datos y almacenes de datos.

26 OWL orientado a Servicios
27 Web Service Modelling Ontology: Ontología de Modelado de Servicios Web
28 http://www-01.ibm.com/software/webservers/smash
29 http://www.bungeelabs.com/

- Herramientas que emplean lenguajes de *scripting*

  Existen herramientas que emplean lenguajes de scripting como WSO2 Mashup Server[30], la cual permite componer servicios, desplegar servicios desarrollados en JavaScript, acceder a servicios REST, canales (feeds), etc.

- Herramientas que conectan componentes

  Herramientas que conectan componentes, como Yahoo Pipes[31], Openkapow[32], Proto Financial[33], Anthracite[34] y Lotus Mashups[35] mezclan y combinan datos, funcionalidad o presentación conectando componentes gráficamente, los cuales proporcionan distintas funcionalidades (recuperación, transformación y presentación de datos) y se deben conectar para lograr la coordinación deseada. Generalmente soportan diferentes tipos de fuentes de datos, tales como servicios Web REST y/o SOAP.

- Herramientas para hojas de cálculo

  Herramientas para hojas de cálculo, como Extensio Excel Extender[36], se centran en la mezcla de datos. A diferencia de las que conectan componentes, los datos provenientes de una fuente se insertan en las celdas de una hoja de cálculo. Extensio Excel Extender utiliza servicios Web del tipo SOAP para crear mashups, y puede proporcionar acceso a SAP, varias bases de datos y archivos planos.

- Herramientas que emplean programación por demostración

  La programación por demostración permite a los usuarios aprender un sistema mediante ejemplos. Dapper[37] es un servicio online que puede crear una API para cualquier sitio Web, seleccionando gráficamente los campos que se deben extraer.

- Herramientas basadas en la generación automática de mashups

  PiggyBank[38], el cual es una extensión para Firefox que convierte al navegador en un mashup semántico, puede procesar datos RDF y mezclarlos entre sí automáticamente. Los datos RDF que procesa PiggyBank pueden provenir de archivos RDF o scrapers que extraen datos no semánticos y los transforman en RDF.

30 http://ws02.org/projects/mashup
31 http://pipes.yahoo.com/pipes/
32 http://openkapow.com/
33 http://www.protosw.com/
34 http://www.metafy.com/products/anthracite
35 http://www-01.ibm.com/software/lotus/products/mashups/
36 www.extensio.com/
37 http://www.dapper.net/
38 http://simile.mit.edu/piggy-bank/

2.6.1 Conclusiones

El desarrollo de mashups que emplean programación o lenguajes de scripting requiere experiencia en tecnologías Web y programación si se quieren crear mashups complejos, resultándole difícil a un usuario sin experiencia su creación.

Las herramientas semiautomáticas (las que emplean programación por demostración, las que conectan componentes y las pensadas para hojas de cálculo) están pensadas para usuarios finales sin muchos conocimientos técnicos, pero requieren de otros conocimientos previos: el usuario final debe seleccionar manualmente las fuentes de información, lo cual no sólo requiere la ubicación (URL, por ejemplo) de cada fuente, sino también conocimiento sobre la estructura y semántica de la información que contienen. Además, es probable que el usuario final no conozca muchas fuentes disponibles, encontrándose limitado a un conjunto de fuentes conocidas, las cuales pueden no satisfacer sus requisitos para una situación específica.

En las herramientas que conectan componentes, se especifican los mashups mediante un flujo de diferentes componentes para recuperar, transformar, combinar y presentar información a partir de diferentes fuentes. A medida que aumenta la complejidad del mashup, aumenta la cantidad de componentes a conectar, resultándole cada vez más difícil a un usuario seleccionar los mismos.

Otra limitación importante de los mashups que emplean programación, lenguajes de scripting, programación por demostración, que conectan componentes y los pensados para hojas de cálculo es la falta de adaptabilidad, lo cual significa que si las fuentes de información y funcionalidades del proveedor cambian su estructura o comportamiento, el mashup tiene que ser rediseñado, requiriendo conocimientos sobre tecnologías Web y programación.

En la generación automática de mashups las fuentes se seleccionan automáticamente y la información recuperada se combina sin interacción del usuario. Esta generación automática requiere la utilización extensiva de semántica, pero promete superar las limitaciones generales indicadas anteriormente.

2.7 Métodos de diseño de aplicaciones Web

Para el desarrollo de aplicaciones Web, la Ingeniería Web ha adaptado los métodos de la Ingeniería de Software, que fueron concebidos inicialmente para dar soporte al desarrollo de software “tradicional”, es decir, no pensado para la Web.

Los métodos de la Ingeniería Web se basan en un principio similar [Fons *et al.*, 2008]: las aplicaciones Web deben ser desarrolladas comenzando por una descripción sólida, precisa y sin ambigüedades a partir de un esquema conceptual. Luego, este esquema conceptual se debe transformar adecuadamente en su producto software correspondiente haciendo corresponder las primitivas conceptuales con sus representaciones software. Para lograr esto, los esquemas conceptuales tradicionales, que se centraban principalmente en la captura de la estructura estática y el comportamiento del sistema, se extendieron con nuevos modelos y mecanismos de abstracción

para capturar los nuevos aspectos introducidos por las aplicaciones Web.

Entre las metodologías más importantes para el diseño de aplicaciones Web, se pueden nombrar las siguientes, discriminándolas según tengan en cuenta, o no, el aspecto semántico de la información:

- No tienen en cuenta el aspecto semántico de la información:
  - OOWS
  - OOHDM
  - UWE
  - IDM
  - WebML

- Sí tienen en cuenta el aspecto semántico de la información:
  - HERA
  - WSDM

A continuación se presenta una descripción de cada una de las metodologías nombradas anteriormente.

2.7.1 OOWS (Object Oriented Approach for Web Solutions Modeling)

OOWS [Fons *et al.*, 2008] es un método de la Ingeniería Web, guiado por modelos (*MDA*), que proporciona un soporte metodológico para el desarrollo de aplicaciones Web que utiliza técnicas del modelado conceptual.

Algunas características:

- Proporciona una guía metodológica y precisa para ir del problema a la solución (producto software final).

- Se centra en la integración del modelado conceptual y del diseño de la navegación: al modelado de la navegación se lo considera como una única fase genérica.

- Extiende el método de modelado conceptual orientado a objetos llamado *OO-Method*, introduciendo un modelo de navegación para especificar características específicas a las aplicaciones Web.

OO-Method es un método, orientado a objetos, de producción de software que proporciona capacidades de generación de código basadas en modelos e integra las técnicas de especificación formales con las notaciones convencionales del modelado orientado a objetos.

OO-Method proporciona un modelo independiente de la plataforma (*PIM*) donde se capturan los aspectos estáticos y dinámicos del sistema por medio de 3 vistas complementarias, las cuales están

definidas por los siguientes modelos:

- Un **Modelo Estructural** que define la estructura del sistema (sus clases, operaciones y atributos) y las relaciones entre clases (especialización, asociación y agregación) por medio de un diagrama de clases.

- Un **Modelo Dinámico** que describe las secuencias de vida válidas para un objeto para todas las clases del sistema usando diagramas de transición de estados. Las interacciones entre los objetos (comunicaciones entre objetos) también se representan mediante diagramas de secuencias en este modelo.

- Un **Modelo Funcional** que captura la semántica de los cambios de estado para definir los efectos de servicio usando una especificación textual formal.

Las aplicaciones Web introducen propiedades adicionales que no son tratadas por estos tipos de métodos durante el proceso, con lo cual los mismos requieren algunas extensiones para hacerles frente. Estas nuevas propiedades hacen referencia a aspectos como la navegación, presentación, y otras características avanzadas como la personalización. Para lograr esto, OOWS introduce 3 nuevos modelos al PIM:

- **Modelo de usuarios**: permite especificar una categorización sobre los tipos de usuarios que pueden interactuar con el sistema, así como las relaciones entre los mismos.

  Un diagrama de usuarios permite especificar los tipos de usuarios que pueden interactuar con el sistema, los cuales se organizan jerárquicamente a través de relaciones de herencia, permitiendo especificar la especialización de la navegación. Los subtipos de usuarios pueden heredar la semántica de navegación asociada a sus supertipos, permitiendo reutilizar las descripciones de navegación.

  Este modelo clasifica los tipos de usuarios en 3 grupos:

  - *Usuarios anónimos* (representados con un '?'). Representan a los usuarios que no están validados en el sistema.

  - *Usuarios registrados* (representados con un candado). Representan a los usuarios que se validaron (registraron) en el sistema.

  - *Usuarios genéricos* (representados con una 'X'). Representan usuarios abstractos (usuarios que no se pueden instanciar).

- **Modelo de navegación**: permite especificar la visibilidad del sistema (en términos de datos y funcionalidad) y los caminos válidos para atravesar la estructura del mismo (la semántica de navegación) para cada tipo de usuario.

  Se entiende por navegación a un salto de una página Web a otra, provocado por la selección

de un enlace, en el que se cambia de contenido (información y/o funcionalidad).

El modelo de navegación representa la semántica navegacional de una aplicación Web en base a un modelo de objetos y a los requisitos de navegación. El modelo de navegación se introdujo en el enfoque OOWS para especificar la vista sobre el sistema en términos de clases, atributos y las operaciones y relaciones entre las clases para cada tipo de usuario definido en el modelo de usuarios.

Este modelo se construye en 2 fases. La primer fase, *fase general*, define una visión global sobre la navegación. La segunda fase, *fase detallada*, hace una descripción detallada de los elementos definidos en la fase anterior.

- **Modelo de presentación**: se presenta para especificar los requisitos de presentación de los elementos definidos en el Modelo de Navegación.

  Una vez que está construido el modelo de navegación, hay que especificar los requisitos de presentación utilizando este modelo, el cual depende en gran medida del anterior, ya que utiliza los contextos de navegación para definir las propiedades de presentación.

  Los requisitos de presentación se especifican por medio de patrones. Los patrones de presentación básicos son los siguientes:

  - *Paginación de la información*: este patrón permite especificar la información para hacer el paginado. Todas las instancias recuperadas se dividen en bloques lógicos de modo que sólo un bloque sea visible a la vez. Se proporcionan mecanismos para avanzar o retroceder.

  - *Criterios de ordenamiento*: este patrón define un ordenamiento de población de clases ascendente o descendente, utilizando el valor de uno o más atributos. Se puede aplicar ya sea a las clases de navegación o a las estructuras de acceso, a los índices y a los filtros, especificando cómo se ordenarán las instancias recuperadas.

  - *Disposición de la información*: hay 3 patrones básicos de disposición y un operador de disposición. Los 3 patrones son: *registro*, *tabular* (*vertical* y *horizontal*) y *árbol*. El operador (*maestro-detalle*) se aplica a las relaciones "muchos a muchos" usando uno de estos patrones básicos de disposición para mostrar la parte del detalle.

2.7.2 OOHDM (Object Oriented Hypermedia Design Method)

El Método de Diseño Hipermedia Orientado a Objetos [Rossi & Schwabe, 2008] es un enfoque basado en modelos para desarrollar aplicaciones Web. Permite al diseñador especificar una aplicación Web, vista como una instancia de un modelo hipermedia, mediante el uso de varios metamodelos especializados que se centran en diferentes aspectos de la aplicación. Una vez que los mismos se han especificado para una aplicación dada, es posible generar el código que implementa

la aplicación.

OOHDM utiliza diferentes mecanismos de abstracción y composición en un framework orientado a objetos para permitir, por un lado, una descripción concisa de elementos de información complejos y, por otro, especificar los complejos patrones de navegación y las transformaciones de las interfaces.

En OOHDM una aplicación Web se construye en 5 pasos que dan soporte a un modelo de proceso incremental o de prototipos. Cada paso se centra en un aspecto particular del diseño, construyendo un modelo apropiado. Durante todo el proceso se utiliza la clasificación y la generalización/especialización para mejorar la capacidad de abstracción y las oportunidades de reutilización.

Los 5 pasos que conforman este método son:

1. *Obtención de los requisitos*: consiste en reunir las necesidades de todos los interesados (*stakeholders*). Para lograr esto, es necesario identificar primero a los actores y las tareas que deben realizar.

   Luego, se preparan los escenarios (o un borrador de los mismos) para cada tarea y tipo de actor.

   Luego, estos escenarios forman los casos de uso, que se representan mediante Diagramas de Interacción de Usuario (UIDs). Estos diagramas proporcionan una representación gráfica y concisa del flujo de información entre el usuario y la aplicación durante la ejecución de una tarea. Los UIDs son validados con los actores, y rediseñados en caso de ser necesario. En secuencia, se aplica un conjunto de pautas a los UIDs para extraer un modelo conceptual básico.

2. *Diseño conceptual*: se construye un modelo conceptual del dominio de la aplicación utilizando principios bien conocidos del modelado orientado a objetos. No se tienen en cuenta los tipos de usuarios y las tareas, sólo la semántica del dominio de la aplicación. Se construye un esquema conceptual a partir de los subsistemas, clases y relaciones. OOHDM utiliza UML, con algunas extensiones, para expresar el diseño conceptual.

3. *Diseño de la navegación*: en OOHDM, a una aplicación se la ve como una vista de navegación sobre el modelo conceptual. Esto refleja una gran innovación, que reconoce que los objetos (elementos) por los cuales navega el usuario no son objetos conceptuales, sino otros tipos de objetos que se “construyen” a partir de uno o más objetos conceptuales, para adaptarse a los usuarios y a las tareas a las cuales se debe dar soporte.

   En otras palabras, para cada perfil de usuario se puede definir una estructura de navegación diferente que refleje los objetos y las relaciones en el esquema conceptual de acuerdo a las tareas que debe realizar este tipo de usuario. La estructura de clases de navegación de una aplicación Web se define mediante un esquema que contiene las clases de navegación.

En OOHDM existe un conjunto predefinido de tipos básicos de clases de navegación: nodos, enlaces, anclas y estructuras de acceso. La semántica de nodos, enlaces y anclas es la habitual en las aplicaciones hipermedia. Los nodos en OOHDM representan "ventanas" lógicas (o vistas) sobre las clases conceptuales definidas durante el diseño conceptual. Los enlaces son realizaciones hipermedia de relaciones conceptuales, así como también asociaciones relacionadas con las tareas. Las estructuras de acceso, como los índices, representan las posibles formas de iniciar la navegación.

La estructura de navegación de una aplicación Web se describe en términos de contextos de navegación, que son conjuntos de nodos relacionados que poseen alternativas de navegación (opciones) similares, y resultan significativos para un determinado paso en alguna tarea que está realizando el usuario.

4. *Diseño abstracto de la interfaz*: el modelo abstracto de interfaz se construye definiendo los objetos perceptibles, también llamados *widgets*, que contienen información (por ejemplo, una imagen, un mapa de la ciudad, etc) en términos de interfaces. Las interfaces se definen como agregaciones recursivas de clases primitivas o de otras interfaces. Los objetos de interfaz se hacen corresponder con los objetos de navegación, brindándoles una apariencia perceptible, o a valores de entrada. El comportamiento de la interfaz se define especificando la forma de controlar los eventos externos y generados por los usuarios y cómo se realiza la comunicación entre la interfaz y los objetos de navegación.

5. *Implementación*: la implementación hace corresponder la interfaz y los objetos de navegación con objetos en tiempo de ejecución y puede involucrar arquitecturas elaboradas, por ejemplo, cliente-servidor, en la que las aplicaciones son clientes a un servidor de base de datos compartido que contiene los objetos conceptuales.

2.7.3 UWE (UML-based Web Engineering)

La Ingeniería Web Basada en UML [Koch *et al.*, 2008] surgió a fines de la década del 90 con la idea de encontrar una forma estándar para la creación de modelos de análisis y diseño de sistemas Web basada en los métodos vigentes en ese momento (OOHDM, RMM y WSDM). El objetivo fue usar un lenguaje común, o al menos definir las correspondencias basadas en metamodelos, entre los enfoques existentes.

En ese momento el Lenguaje Unificado de Modelado (UML), que evolucionó a partir de la integración de 3 técnicas diferentes de modelado (Booch, OOSE y OMT) parecía ser un enfoque prometedor para el modelado de sistemas. A partir de esos primeros esfuerzos de integración, UML se convirtió en la "lingua franca" de la Ingeniería de Software (orientada a objetos).

Una característica destacada de UML es que proporciona un conjunto de ayudas (los llamados mecanismos de extensión) para la definición de lenguajes de modelado de dominio específico (DSL). Además, los nuevos DSLs siguen siendo compatibles con UML, lo cual permite el uso de todas sus características (extensiones Web específicas, por ejemplo).

Tanto la aceptación de UML como estándar para el desarrollo de software, como la flexibilidad que ofrecen los mecanismos de extensión, son las razones de la elección de UML en lugar de técnicas de modelado propietarias. La idea seguida por UWE de adherirse a normas no se limita a UML, también utiliza XMI como formato de intercambio de modelos (con la esperanza de futuras herramientas de interoperabilidad, posibles gracias a un XMI verdaderamente portable), MOF para el metamodelado, los principios guiados por modelos dados por el enfoque de la Arquitectura Guiada por Modelos (MDA) de OMG, el lenguaje de transformación QVT y XML.

El desarrollo de sistemas Web está sujeto a cambios continuos en los requisitos del usuario y en la tecnología, con lo que los modelos, en cualquier etapa del proceso de desarrollo, tienen que ser fácilmente adaptables a estos cambios. Para hacer frente eficientemente a esta flexibilidad requerida, UWE aboga por una estricta separación de incumbencias en las primeras fases del desarrollo e implementa un proceso de desarrollo guiado por modelos.

La metodología propuesta por UWE comprende los siguientes pasos:

1. *Especificación de requisitos*: el primer paso hacia el desarrollo de un sistema Web es la identificación de los requisitos, los cuales se especifican en UWE mediante un *modelo de requisitos*. Los requisitos se pueden documentar en diferentes niveles de detalle. UWE propone 2 niveles: en primer lugar, se genera una descripción general de las funcionalidades, las que se modelan con casos de uso UML. En una segunda fase, se desarrolla una descripción más detallada de los casos de uso, por ejemplo, diagramas de actividad UML que describen las responsabilidades y acciones de las partes interesadas.

2. *Definición del contenido*: los modelos de análisis proporcionan la base para los *modelos de diseño*, en particular, el modelo de contenido de un sistema Web. El objetivo del modelo de contenido es proporcionar una especificación visual de la información del dominio relevante para el sistema Web que incluye principalmente el contenido de la aplicación Web. Sin embargo, muy a menudo también incluye a las entidades del dominio necesarias para aplicaciones Web personalizadas. Estas entidades constituyen el llamado perfil de usuario o modelo de usuario.

3. *Establecimiento de la estructura de navegación*: basado en el análisis de requisitos y el modelado del contenido, se modela la estructura de navegación de una aplicación Web. Las clases de navegación representan nodos navegables de la estructura de hipertexto; los enlaces de navegación indican enlaces directos entre las clases de navegación. Cada clase de procesos está asociada con un caso de uso que modela un proceso de negocio.

Estos modelos se utilizan para representar diferentes vistas de la misma aplicación Web correspondientes a las diferentes incumbencias (contenido, estructura de navegación y presentación):

- El modelo de contenido se utiliza para especificar los conceptos que son relevantes al dominio de la aplicación y las relaciones entre los mismos.

- La estructura de navegación se modela por separado del contenido, aunque se deriva del mismo.

- El modelo de navegación representa los caminos de navegación del sistema Web que se está modelando.

- El modelo de presentación tiene en cuenta las tareas de representación y la comunicación hombre-máquina.

UWE propone por lo menos un tipo de diagrama UML para la visualización de cada modelo para representar los aspectos estructurales de las diferentes vistas. Sin embargo, muy a menudo se usan diagramas de interacción UML o máquinas de estado para representar los aspectos de comportamiento del sistema Web.

2.7.4 IDM (Interactive Dialogue Model)

IDM [Bolchini & Garzotto, 2008] es el resultado de una larga experiencia en la construcción y utilización de modelos para el diseño de aplicaciones hipermedia. A principios de los 90, se comenzó con HDM (*Hypertext Design Model*), que fue el primer modelo para el diseño conceptual de esta clase de aplicaciones. HDM era relativamente simple y, en algunos aspectos, un tanto "ingenuo". Aún así, proponía algunos conceptos centrales que inspiraron a muchos modelos posteriores.

HDM evolucionó progresivamente en otros modelos (HDM+, HDM2 y W2000) que, para abordar la creciente complejidad de las aplicaciones hipermedia, eran mucho más ricos y sofisticados que su antecesor, pero este aumento de capacidad expresiva tuvo algunos inconvenientes: la dificultad de aprender la teoría y la práctica de estos modelos no se compensaba en términos de una mayor calidad de diseño; la producción de documentación sobre el diseño, en proyectos industriales, se hizo cada vez más consumidora de tiempo (ya que las especificaciones eran cada más detalladas), al mismo tiempo que su poder como medio de comunicación entre las partes interesadas disminuía drásticamente (especialmente entre las personas que no tenían ninguna capacitación formal en modelización).

Así, después de pasar de la simplicidad (HDM) a la complejidad (W2000), se volvió progresivamente a la simplicidad, siendo IDM el "final" de esta "parábola". IDM se centra en los conceptos de diseño verdaderamente fundamentales para hacer rentable un proceso. No se limita a ofrecer una herramienta de especificación para que los diseñadores generen sus soluciones: los ayuda a crear representaciones abstractas, mínimas y expresivas y, sobre todo, a entender cómo pensar al momento de hacer un diseño.

IDM concibe las interacciones del usuario con una aplicación hipermedia como un diálogo: una secuencia de "actos" del tipo pregunta-respuesta: la selección de un enlace es la contra parte operacional a una pregunta que el usuario se "hace" a sí mismo y vuelve al "sistema". El efecto de la selección de un enlace, es decir, la visualización de la página destino del enlace, es la respuesta

que materialmente ofrece el sistema, según los diseñadores, sobre cómo responder a la pregunta del usuario.

El proceso de diseño previsto por IDM se compone de 3 actividades principales:

- *Diseño conceptual (C-IDM)*: asumiendo un enfoque orientado a diálogos, las primeras cuestiones que debe abordar el diseñador se pueden resumir en las siguientes preguntas:

    - ¿Cuál es el tema del diálogo?, es decir, ¿qué puede (o debe) decirle al usuario la aplicación?
    - ¿Cuáles son los cambios de temas relevantes a los cuales se debe dar soporte durante el diálogo entre el usuario y la aplicación?
    - ¿Cuáles son las posibles formas, diferentes, de organizar el diálogo?, es decir, agrupar los diferentes temas a través de los cuales el usuario pueda iniciar el flujo real de la conversación

    Se pueden proporcionar respuestas precisas y detalladas a las preguntas anteriores sólo cuando se haya elegido un canal de entrega específico (factores determinantes como el tamaño de la pantalla, los mecanismos de señalización, los medios disponibles, el rendimiento, etc.) Sin embargo, se pueden hacer determinadas decisiones con antelación, lo que se llama "diseño conceptual".

    En esta fase inicial, se debe definir un esquema conceptual de la aplicación interactiva a fin de transmitir todas las "estrategias de diálogos" necesarias, sin entrar en detalles que puedan depender de cuestiones técnicas del dispositivo real de entrega.

- *Diseño lógico (L-IDM)*: a diferencia del diseño conceptual, el diseño lógico empieza tomando decisiones que generalmente dependen de un canal específico a través del cual se puede transmitir la aplicación (ya sea la Web tradicional, un canal oral, TV interactiva, un canal móvil, etc).

    Mientras que un esquema de diseño conceptual C-IDM define toda la estrategia de interacción a la cual se debe dar soporte durante el diálogo de la aplicación con el usuario, los diseñadores pueden desarrollar uno o más diseños "lógicos", uno para cada canal específico para el cual desean diseñar la aplicación. El diseño "lógico" IDM se puede ver como una versión detallada del diseño conceptual, donde se deciden los detalles en base a una variedad de factores dependientes de los canales, tales como las limitaciones impuestas por el tipo de dispositivo disponibles sobre un determinado canal (por ejemplo, tamaño de la pantalla), los dispositivos de señalización (por ejemplo, teclado, lápiz inteligente, mouse, scroller, entrada de audio, punteros táctiles, punteros de seguimiento ocular), los medios que se pueden utilizar (por ejemplo, audio, texto visual, imágenes, gráficos o video), el rendimiento esperado, y de suma importancia, los escenarios típicos de uso (por ejemplo, doméstico o de oficina, para ir caminando, uso móvil en el auto, etc.)

- *Diseño de páginas (P-IDM)*: el diseño de páginas IDM (P-IDM) significa definir los

elementos que deben comunicarse al usuario en un solo acto de diálogo. Ahora los diseñadores tienen que crear las páginas reales que contienen los elementos necesarios para mantener el diálogo.

2.7.5 WebML (Web Modeling Language)

El Lenguaje de Modelado Web (WebML) [Brambilla *et al.*, 2008] es una metodología de diseño Web de tercera generación, concebida en 1998, como consecuencia de los primeros modelos y los trabajos pioneros en hipermedia y diseño Web, como HDM y RMM. La meta original de WebML era dar soporte al diseño y a la implementación de las llamadas aplicaciones Web intensivas en datos, definidas como sitios Web para acceder y mantener grandes cantidades de datos estructurados, generalmente almacenados como registros en bases de datos, como las aplicaciones de comercio electrónico y en línea, sitios Web institucionales de organizaciones privadas y públicas, bibliotecas digitales, portales corporativos y sitios de comunidades.

Para lograr este objetivo, WebML reutilizó los modelos de datos conceptuales existentes y propuso una notación original para expresar las características de navegación y composición de las interfaces hipertexto. El modelo hipertexto de WebML adoptó un enfoque muy diferente de los anteriores: en lugar de ofrecer un elevado número de primitivas para representar todas las posibles formas de organizar una interfaz hipertexto que se pueden presentar en las aplicaciones Web con gran volumen de datos, el foco se centró en la invención de un número mínimo de conceptos, que podían estar compuestos de formas bien definidas para obtener un número arbitrario de configuraciones.

Esta opción inicial de diseño influyó profundamente en la definición del lenguaje y en su evolución hacia clases más complejas de aplicaciones. Cuatro grandes versiones de WebML caracterizan la progresión del mismo:

- *WebML 1*: la versión inicial contaba sólo de un conjunto fijo de primitivas para la representación de sitios Web de gran volumen de datos y de sólo lectura; el foco se centraba en la organización modular de la interfaz, la definición de la navegación y la extracción y publicación de contenido en la interfaz.

- *WebML 2*: se agregó soporte para la representación de las acciones de negocio (llamadas operaciones) provocadas por la navegación del usuario; de esta manera, el poder expresivo se amplió para admitir características como la gestión de contenido, la autenticación y la autorización.

- *WebML 3*: la introducción del concepto de *plugin* transformó a WebML en un lenguaje abierto, extensible por los diseñadores con sus propias primitivas de nivel conceptual, a fin de ampliar el poder expresivo para cubrir las necesidades de las nuevas aplicaciones. Esta transición destacó el papel del modelado basado en componentes y fue la base de todas las extensiones subsiguientes.

- *WebML 4*: la noción de *plugin* fue explotada para agregar extensiones ortogonales al núcleo

de WebML, cubriendo los sectores y las aplicaciones no asociadas previamente con el desarrollo orientado a modelos. Por ejemplo, se incorporaron la interacción de servicios Web y las primitivas de modelado de flujo de trabajo como plugins para permitir la modelización y la implementación de aplicaciones distribuidas; otras extensiones apuntaron en la dirección de aplicaciones Web multicanal y sensibles al contexto.

WebML es un lenguaje visual para especificar la estructura del contenido de una aplicación Web y la organización y presentación de dicho contenido en forma de hipertexto. Un rasgo distintivo de WebML es la presencia de una línea industrial de desarrollo en paralelo a la investigación académica.

El enfoque WebML para el desarrollo de aplicaciones Web se compone de diferentes fases. Inspirado en el modelo espiral de Boehm y de acuerdo con los métodos modernos para el desarrollo Web y aplicaciones software, el proceso WebML se aplica de manera iterativa e incremental de manera que las distintas fases se repiten y refinan hasta que los resultados cumplan con los requisitos de las aplicaciones.

El ciclo de vida del producto, por lo tanto, se somete a varios ciclos, cada uno produciendo un prototipo o una versión parcial de la aplicación. En cada iteración, la versión actual de la aplicación se prueba y evalúa y luego se extiende o se modifica para hacer frente a los requisitos recogidos anteriormente, así como también los requisitos recientemente aparecidos. Este tipo de ciclo de vida iterativo e incremental resulta particularmente apropiado para el contexto Web, donde las aplicaciones deben desplegarse con rapidez (en “tiempo Internet”) y los requisitos pueden cambiar durante el desarrollo.

Las fases de WebML son:

- *Análisis de requisitos*: se centra en recoger información sobre el dominio de la aplicación y las funciones que se esperan y en especificarlas a través de descripciones fáciles de comprender. La entrada a esta actividad es el conjunto de los requisitos de negocio que motivan el desarrollo de la aplicación. Los resultados principales de esta fase son:

    - La identificación de los grupos de usuarios, cada uno de los cuales representa a usuarios con las mismas características o que juegan el mismo papel dentro de un proceso de negocio.

    - La especificación de requisitos funcionales que responden a las funciones que deben facilitarse a los usuarios. Para cada grupo de usuarios, se identifican y especifican las actividades relevantes a llevar a cabo.

    - La identificación de objetos de información básicos, es decir, los principales activos de información a ser accedidos, intercambiados y/o manipulados por los usuarios.

    - La descomposición de la aplicación Web en vistas, es decir, diferentes hipertextos diseñados para cumplir con un conjunto bien definido de requisitos funcionales y de usuario. Cada grupo de usuarios contará con al menos una vista del sitio que soporte

las funciones descritas para el grupo.

- *Modelado conceptual*: consiste en definir esquemas conceptuales, que expresan la organización de la aplicación a un nivel de abstracción alto, independiente de los detalles de implementación. Para WebML, el modelado conceptual consiste en:

    - *un diseño de datos*: se corresponde con la organización de los objetos de información básicos identificados previamente en el análisis de requisitos en un esquema de datos coherente y exhaustivo, posiblemente enriquecido a través de objetos derivados.

    - *un diseño de hipertexto*: el diseño hipertexto a continuación produce esquemas de vistas del sitio sobre el esquema de datos previamente definido. Las vistas del sitio expresan la composición del contenido y los servicios dentro de las páginas hipertexto, así como también la navegación y la interconexión de componentes. Para aplicaciones donde diferentes grupos de usuarios realizan múltiples actividades, o para aplicaciones multicanal, en la que los usuarios pueden adoptar diferentes dispositivos de acceso, el diseño hipertexto requiere la definición de múltiples vistas del sitio, tratando los grupos de usuarios involucrados y sus requisitos de acceso.

- *Implementación*: el desarrollo de aplicaciones con WebML es asistido por WebRatio, una herramienta comercial para diseñar e implementar aplicaciones Web. La arquitectura de WebRatio consta de 2 capas: una capa de diseño, que proporciona las funciones para la edición visual de las especificaciones, y una capa de tiempo de ejecución, que implementa los servicios básicos para ejecutar las unidades WebML sobre un framework de aplicaciones Web estándar.

2.7.6 Hera

El propósito de Hera [Houben *et al.*, 2008] es dar soporte al diseño de aplicaciones que proporcionen estructuras Web basadas en la navegación (presentaciones hipermedia) sobre datos estructurados semánticamente en una forma personalizada y adaptable. El enfoque de diseño se centra en los modelos que representan los aspectos centrales del diseño de la aplicación.

Antes de poder crear un modelo para especificar el diseño de la aplicación, es necesario como punto de inicio en Hera un modelo del dominio (DM), el cual describe la estructura de los datos. El único propósito del DM es definir la forma en que el diseñador percibe la estructura semántica de los datos: dice lo que se necesita saber acerca del contenido sobre el que se quiere que funcione la aplicación. Basado en este DM, el diseñador crea un modelo de la aplicación (AM) que describe una estructura de navegación basada en hipermedia sobre el contenido. Esta estructura de navegación está diseñada en aras de entregar y presentar el contenido al usuario de una forma que permita un acceso efectivo (semánticamente) al mismo.

A su vez, este acceso puede implicar la personalización o adaptación que se considere pertinente. Hera permite la personalización y la adaptación dinámica del contenido. A tal efecto, se mantienen

datos de contexto (bajo control de la aplicación) en un modelo de contexto (CM). Estos datos de contexto son normalmente actualizados en base a las (inter)acciones del usuario, así como a la información externa.

Por lo tanto, sobre la base del DM y el CM, el AM sirve como una receta que prescribe cómo se transforma el contenido en una estructura de navegación. Para ser más precisos, el instanciar el AM con contenido concreto resulta en páginas AM (AMP). Estas AMPs se pueden considerar como páginas que contienen contenido (que se muestra) y primitivas de navegación (basadas en las relaciones semánticas subyacentes del DM) que el usuario puede utilizar para navegar a otras AMPs y así "moverse" semánticamente a una parte diferente del contenido.

Una AMP en sí misma no está todavía lista para un navegador, pero se puede transformar en una presentación adecuada mediante un generador de presentación, es decir, un motor que ejecuta una especificación, por ejemplo, un modelo de presentación (PM) del diseño de la presentación concreta en términos de diseño y otros detalles de presentación (específicos del navegador). Para Hera, esta fase de generación de la presentación en sí misma no es específica y puede ser hecha de la manera que se prefiera. Así, el AM especifica la construcción (más conceptual o semántica) de la estructura de navegación sobre el contenido, mientras que la fase de presentación posterior, posiblemente especificada por un PM, es responsable de la transformación de esta estructura en los elementos que se ajustan a la situación concreta de navegación.

- **Modelado de Datos**

  Antes que el diseño de la aplicación pueda considerar la navegación personalizada sobre los contenidos del dominio, se deben especificar los datos relevantes. Como un primer paso necesario en el enfoque, el paso del modelado de datos lleva a la construcción de los modelos de datos para los contenidos del dominio y el contexto del usuario.

  El modelado de los contenidos del dominio utiliza RDFS y sobre todo está dirigido a la captura de la estructura semántica de los contenidos del dominio. Incluso permite al modelo que sea una ontología OWL. Por ejemplo, un modelo representado en UML se puede representar fácilmente como una definición RDFS u OWL mediante un proceso de conversión de UML a OWL [Hart *et al.*, 2004].

  El CM se modela e implementa de forma similar al DM. La principal diferencia entre ambos es que los datos de contenido están pensados para ser presentados al usuario, mientras que los datos de contexto tienen el propósito de dar soporte a la adaptación dependiente del contexto de la aplicación. Así, los datos de contenido normalmente contienen la información que finalmente se muestra al usuario, mientras que los datos de contexto contienen la información utilizada (internamente) para la personalización y adaptación de la entrega de contenido.

  Tanto los datos del DM como los del CM se implementan usando un repositorio Sesame[39]. Para el CM, que está bajo el control directo de la aplicación, esto permite que la aplicación gestione y actualice el contexto tal como ella lo percibe. Junto a esto, también proporciona

39 http://www.openrdf.org/about.jsp

los medios para que otros procesos utilicen y actualicen (partes de) esta información.

Otra gran ventaja del uso de Sesame es la posibilidad de combinar varias fuentes de datos (tanto datos de contenido como de contexto) al mismo tiempo. En esta forma, los diseñadores pueden acoplar fuentes de datos adicionales a las ya existentes y por lo tanto fácilmente extender el contenido del dominio. Esto también ofrece posibilidades para explotar los conocimientos adicionales cuando se realiza una búsqueda.

Al dar soporte irrestricto a DMs RDFS u OWL, Hera resulta especialmente adecuado para (re)utilizar ontologías de dominios existentes. Por otra parte, muchas fuentes de datos existentes que aún no están disponibles en RDFS u OWL pueden ser utilizadas a través de las técnicas de la Web Semántica [Thiran *et al.*, 2005].

- **Modelado de la Aplicación**

  Sobre la base de la definición del dominio, el modelado de la aplicación resulta en el modelo de aplicación (AM) que especifica el comportamiento de navegación de la aplicación Web. El AM permite a los diseñadores especificar cómo se estructura el acceso (de navegación) a los datos (recuperados dinámicamente desde el dominio) describiendo qué datos se muestran al usuario y a qué páginas Web puede navegar el mismo. Al mismo tiempo, el AM permite que esta especificación sea dinámica, de tal manera que el acceso navigacional a los datos se pueda personalizar para un usuario y adaptar a un contexto determinado.

2.7.7 WSDM (Web Semantics Design Method)

WSDM [De Troyer *et al.*, 2008] fue presentado por De Troyer y Leune en 1998 [De Troyer & Leune, 1998]. En ese momento las siglas eran por *Web Site Method* y sólo apuntaba a sitios Web que proporcionaran información. Desde entonces, el método evolucionó mucho y actualmente permite, además del diseño de aplicaciones Web tradicionales, el diseño de aplicaciones semánticas, de ahí el cambio del nombre a *Web Semantics Design Method*.

Más que un método de diseño Web, WSDM es una metodología, es decir, no sólo proporciona primitivas de modelado que permiten a un desarrollador Web construir modelos que describen el sitio/aplicación Web desde diferentes perspectivas y a diferentes niveles de abstracción, sino que también proporciona una forma sistemática de desarrollar estos tipos de aplicaciones. El desarrollo de un sitio/aplicación Web con WSDM comienza con la formulación de la declaración de la misión y sigue una filosofía de diseño bien definida que brinda al diseñador el soporte necesario para estructurar el sitio Web. El método consiste en una secuencia de fases, donde cada una tiene una salida bien definida. Para cada fase, se proporciona un (sub)método que describe cómo derivar la salida a partir de su entrada (la salida de una fase es la entrada de la fase siguiente). También es importante notar que WSDM permite desarrollar sistemas Web anotados semánticamente, permitiendo la Web Semántica.

WSDM sigue un enfoque de diseño “guiado por el público”. Esto significa que para el diseño se

toman como punto de partida a los distintos tipos de usuarios (visitantes) y sus necesidades, y que la estructura principal del sitio Web se deriva a partir de éstos. Concretamente, esto se traduce en diferentes rutas de navegación ofrecidas desde la página inicial, una para cada tipo diferente de visitante.

Las 5 fases que componen WSDM son:

1. *Declaración de la misión*: el objetivo de esta fase es identificar el propósito del sistema Web, así como también la temática y los usuarios finales. Los usuarios finales son los usuarios que se quieren abordar o los que estarán interesados en el sistema Web. La temática del sistema Web está relacionada con la finalidad y los usuarios finales del mismo: debe permitir el cumplimiento de la finalidad del sistema Web, y debe adaptarse a los usuarios finales. La salida de esta fase es la declaración de la misión. La misma estará formulada en lenguaje natural y deberá describir el propósito, la temática y destinatarios del sistema Web. De hecho, la declaración de la misión establece los límites del proceso de diseño. Permite (en las siguientes fases) decidir qué información o funcionalidad incluir o excluir, cómo estructurarla y cómo presentarla.

2. *Modelado de usuarios*: los usuarios finales identificados en la declaración de la misión son refinados en "clases de usuarios". Esto significa que los diferentes tipos de usuarios son identificados y clasificados en clases según sus requisitos: los usuarios con los mismos requisitos de información y funcionalidad pasan a ser miembros de la misma clase, los usuarios con más requisitos forman subclases. De este modo se construye una jerarquía de clases. Para cada clase, se dan las características relevantes (por ejemplo, edad, nivel de experiencia, etc).

3. *Diseño conceptual*: se utiliza para especificar la información, funcionalidad y estructura del sistema Web, a nivel conceptual. Un diseño conceptual hace una abstracción de cualquier tecnología de implementación o plataforma destino. La información y la funcionalidad se especifican durante la subfase "*Modelado de tareas e información*". Toda la estructura conceptual, incluidas las posibilidades de navegación para cada clase de público, se especifican durante la subfase "*Diseño de la navegación*".

    - **Modelado de Tareas e Información**

      En lugar de comenzar con un modelo conceptual de datos global, como hace la mayoría de los métodos de diseño Web, WSDM comienza analizando los requisitos de las diferentes clases de usuarios, lo cual da lugar a una serie de pequeñas descripciones conceptuales, llamadas fragmentos de objetos, que modelan la información y la funcionalidad necesarias para satisfacer estos requisitos.

        - **Modelado de Tareas**

          El propósito del modelado de tareas es modelar en detalle las diferentes tareas que deben ser capaces de realizar los miembros de cada clase de usuario y describir formalmente los datos y la funcionalidad que se necesita para esas

tareas.

- **Modelado de la Información**

  Cuando se completa el modelo de tareas, se crea un fragmento de objeto para cada tarea elemental en este modelo. El objetivo principal de un fragmento de objeto es describir formalmente la información y la funcionalidad necesarias por el usuario cuando tiene que realizar la tarea asociada. Si el requisito asociado a la tarea es un requisito de información puro (es decir, el usuario sólo busca información; no necesita realizar acciones), el fragmento de objeto puede considerarse como la descripción conceptual de la información que se muestra en (una parte de) la pantalla. A tal efecto, es suficiente un lenguaje de modelado conceptual estándar. Sin embargo, para poder tratar la funcionalidad (por ejemplo, llenar un formulario y procesarlo), también se necesita un lenguaje de manipulación de datos.

  WSDM utiliza OWL para modelar las necesidades de información. Su uso como lenguaje de especificación para los fragmentos de objetos permite una fácil integración y el uso de ontologías de dominios existentes.

4. *Diseño de la implementación*: los modelos del diseño conceptual se complementan con la información necesaria para una implementación real. Consta de 3 subfases: "*Diseño de la estructura del sitio*", "*Diseño de la presentación*" y "*Diseño lógico de datos*". Durante el diseño de la estructura del sitio, se hace una correspondencia entre la estructura conceptual del sistema Web y las páginas, es decir, el diseñador decide qué componentes (que representen información y funcionalidad) y enlaces se van a agrupar en páginas Web. Para el diseño conceptual, se pueden definir diferentes estructuras del sitio, dirigidas a distintos dispositivos, contextos, o plataformas. El diseño de la presentación se utiliza para definir la apariencia del sistema Web así como también el diseño de las páginas. El diseño lógico de datos sólo es necesario para sistemas Web intensivos en datos. En el caso que los datos se mantengan en una base de datos, se construye un esquema de la misma (o se puede utilizar uno ya existente), y se establece una correspondencia entre el modelo conceptual de datos y la fuente de datos real.

5. *Implementación*: la implementación real puede generarse automáticamente a partir de la información recogida durante las fases anteriores.

Las 5 fases anteriores, mostradas de forma secuencial, en la práctica se ejecutan en forma bastante iterativa.

2.8 Análisis de las distintas metodologías

Según las descripciones de las distintas metodologías, se puede realizar el siguiente análisis:

- OOWS

  Metodología basada en modelos, basada en otra llamada OO-Method. A los 3 modelos de OO-Method, le agrega otros 3 para hacerle frente a las características propias de las aplicaciones Web: **modelo de usuarios** (para categorizar los distintos tipos de usuarios: anónimos, registrados y genéricos), **modelo de navegación** (para especificar la visibilidad del sistema y los caminos válidos para atravesar la estructura del mismo) y **modelo de presentación** (para especificar los requisitos de presentación de los elementos definidos en el modelo de navegación).

  Esta metodología no tiene en cuenta el aspecto semántico de la información, propio de los mashups semánticos. Un mashup semántico, a diferencia de otro tipo de aplicación Web más "tradicional", generalmente no requiere hacer una distinción entre distintos tipos de usuarios, ya que suele trabajar con un único tipo (anónimo), y por lo tanto no se necesita definir un modelo de usuarios.

  Por lo general, los mashups semánticos, en cuanto a la interfaz de usuario, suelen estar organizados en 2 o 3 pantallas, donde en la primera y/o segunda se ingresan las distintas opciones de búsqueda/clasificación/ordenamiento/etc, y en la pantalla siguiente se muestra toda la información ya condensada. En consecuencia, tampoco resulta necesario un modelo de navegación, o al menos uno tan detallado como el que propone esta metodología.

- **OOHDM**

  Metodología basada en modelos, permite construir una aplicación Web en 5 pasos: 1) **obtención de los requisitos** (reúne las necesidades de los interesados, identificando los actores y las tareas que deben realizar para luego elaborar un borrador con los escenarios para los mismos. Estos escenarios forman los casos de uso), 2) **diseño conceptual** (arma un modelo conceptual del dominio sin tener en cuenta los usuarios y las tareas), 3) **diseño de la navegación** (para cada perfil de usuario define una estructura de navegación diferente que refleja los objetos y las relaciones en el esquema conceptual según las tareas que debe realizar este tipo de usuario), 4) **diseño abstracto de la interfaz** (define los *widgets* que contienen información en términos de interfaces), 5) **implementación** (hace corresponder la interfaz y objetos de navegación con objetos en tiempo de ejecución).

  Al igual que la metodología anterior, no tiene en cuenta el aspecto semántico de la información. Por las mismas razones explicadas anteriormente, el paso 3 (diseño de la navegación) sólo se aplica para un tipo de usuario, lo mismo para el paso 1 (obtención de los requisitos) en cuanto a la identificación de los distintos actores y tareas.

- UWE

  Metodología basada en modelos, permite construir una aplicación Web en los siguientes pasos: 1) **especificación de requisitos** (se especifican mediante un modelo de requisitos, los cuales se pueden documentar con diferentes niveles de detalle), 2) **definición del contenido**

(proporciona una especificación visual de la información del dominio relevante para el sistema Web), 3) **establecimiento de la estructura de navegación** (se modela la estructura de navegación de la aplicación Web según el análisis de requisitos y el modelado del contenido). Para todos estos pasos utiliza UML.

Al igual que las metodologías anteriores, no tiene en cuenta el aspecto semántico de la información. Esta metodología complementa a una para aplicaciones "tradicionales" en el hecho de contemplar el diseño de la estructura de navegación.

- IDM

Metodología donde el proceso de diseño se compone de 3 actividades principales: 1) **diseño conceptual** (asumiendo un enfoque orientado a diálogos, se responde a preguntas como ¿qué puede (o debe) decirle al usuario la aplicación?, ¿cuáles son los cambios de temas relevantes a los cuales se debe dar soporte durante el diálogo entre el usuario y la aplicación?, ¿cuáles son las posibles formas, diferentes, de organizar el diálogo?, etc), 2) **diseño lógico** (toma decisiones que generalmente dependen de un canal específico a través del cual se puede transmitir la aplicación, por ejemplo la Web tradicional, un canal oral, TV interactiva, un canal móvil, etc), 3) **diseño de páginas** (se crean las páginas reales que contienen los elementos necesarios para mantener el diálogo).

Al igual que las metodologías anteriores, no tiene en cuenta el aspecto semántico de la información. Similar a la metodología UWE en cuanto al diseño de la estructura de navegación.

- WebML

Metodología de diseño Web de tercera generación, compuesta de diferentes fases, inspirada en el modelo espiral de Boehm. Las fases son: 1) **análisis de requisitos** (recoge información sobre el dominio de la aplicación y las funciones que se esperan y las especifica a través de descripciones fáciles de comprender), 2) **modelado conceptual** (define esquemas conceptuales que expresan la organización de la aplicación a un nivel de abstracción alto, independiente de los detalles de implementación), 3) **implementación** (el desarrollo es asistido por WebRatio, una herramienta comercial para diseñar e implementar aplicaciones Web).

Tampoco tiene en cuenta el aspecto semántico de la información. Similar a las anteriores en cuanto al diseño de la estructura de navegación.

- Hera

Metodología que sí tiene en cuenta el aspecto semántico de la información. El primer paso consiste en definir un **modelo del dominio** (describe la estructura semántica de los datos). Para el modelado del dominio se puede utilizar RDFS u OWL. Sobre el modelo del dominio se define un **modelo de la aplicación** (describe una estructura de navegación sobre el

contenido, la cual está diseñada con el fin de entregar y presentar el contenido al usuario de una forma que permita un acceso efectivo -semánticamente- al mismo). Este acceso puede implicar la personalización o adaptación que se considere pertinente, para lo cual se pueden mantener datos de contexto en un **modelo de contexto**, el cual se implementa de forma similar el modelo del dominio.

El aspecto a señalar de esta metodología es que no tiene en cuenta el problema de la integración de la información de las distintas fuentes (ontologías) que utiliza un mashup semántico. Es decir, trabaja sobre una ontología y no un grupo de éstas.

- WSDM

  Metodología que sí tiene en cuenta el aspecto semántico de la información. Permite, además del diseño de aplicaciones Web tradicionales, el diseño de aplicaciones Web semánticas. El método consiste en una secuencia de fases siguiendo un enfoque "guiado por el público": 1) **declaración de la misión** (identifica el propósito del sistema Web, la temática y los usuarios finales), 2) **modelado de usuarios** (los usuarios finales identificados en la declaración de la misión son refinados en "clases de usuarios"), 3) **diseño conceptual** (especifica la información, funcionalidad y estructura del sistema Web a nivel conceptual), 4) **diseño de la implementación** (los modelos del diseño conceptual se complementan con la información necesaria para una implementación real. Consta de 3 subfases: "diseño de la estructura del sitio", "diseño de la presentación" y "diseño lógico de datos"), 5) **implementación** (la implementación real puede generarse automáticamente a partir de la información recogida durante las fases anteriores). Para realizar todo el modelado de la información (fase 3) emplea OWL.

  Al igual que la anterior, esta metodología tampoco tiene en cuenta el problema de la integración de las distintas ontologías. Además, algunas tareas no son necesarias para un mashup semántico, como la identificación de los distintos tipos de usuarios y la categorización de los mismos en clases.

En base a este análisis, se concluye que resulta necesario contar con una metodología propia para el caso de los mashups semánticos, la cual brinde soporte al principal problema a resolver durante su desarrollo: la integración de la información de las distintas ontologías.

2.9 Conclusiones

En este capítulo se presentó una reseña sobre la evolución de la Web y sus aplicaciones, señalando las características y tecnologías utilizadas para su creación.

También se vieron aspectos específicos al desarrollo de aplicaciones Web y mashups semánticos, como ser el acceso a las fuentes de información, la integración de la información de las distintas fuentes y características de la interfaz de usuario.

Luego se vio el estado del desarrollo de mashups semánticos, donde se mencionaron algunos proyectos que están siendo desarrollados por distintas organizaciones, comentando sus ventajas y desventajas.

Finalmente se dio una breve descripción sobre un conjunto de metodologías para el diseño de aplicaciones Web, realizando un análisis de cada una tras lo cual se concluyó que ninguna de las metodologías analizadas se adapta al caso de los mashups semánticos, porque entre otras cosas, no contemplan el manejo de múltiples fuentes de información, no tienen en cuenta el aspecto semántico de la información o no contemplan el problema de la integración de las distintas ontologías, en virtud de lo cual resulta necesario contar con una metodología propia para el desarrollo de mashups semánticos.

CAPÍTULO 3 – ONTOLOGÍAS Y LA WEB SEMÁNTICA

Se dijo que un mashup semántico es una aplicación Web que extrae información de diversas fuentes, la combina (integra) y presenta en otras formas. La característica principal de esta información es que resulta legible a una máquina y su significado está explícitamente definido [Yu, 2011].

Cada fuente organiza su información en ontologías. Para que el mashup semántico pueda integrar las distintas ontologías, necesita "comprender" el significado de la información que representa cada una, para lo cual debe "comprender" su semántica.

Las personas, de alguna manera, representan simbólicamente el mundo, sus objetos y las relaciones entre los mismos. Por ejemplo, una persona al consultar determinada información, puede distinguir qué parte de la misma hace referencia a una dirección, gracias a un "conocimiento extra" o "contexto" que puede utilizar.

Si se quiere que una aplicación de software, en este caso un mashup semántico, "comprenda" el significado de la información, se necesita, al menos parcialmente, automatizar el proceso de interpretación semántica.

El objetivo de este capítulo consiste en definir el concepto de ontología y explicar brevemente algunos de los principales lenguajes para su representación (RDF, RDFS y OWL). Para una explicación más detallada sobre RDF y RDFS se pueden consultar los Apéndices A y B respectivamente.

3.1 Ontologías

Una ontología intenta capturar el significado (la semántica) de un área de conocimiento determinada, la cual se corresponde con lo que una persona sabe acerca de ese dominio [Daconta *et al.*, 2003]. Esta semántica es caracterizada por la ontología definiendo los términos (las "cosas" sobre las que se quiere hablar) y las relaciones que se pueden establecer sobre los mismos. Generalmente, estos conceptos y sus relaciones se implementan como clases, relaciones, propiedades, atributos y valores (de las propiedades/atributos).

Existen varias definiciones para el concepto de ontología. Por ejemplo, la más conocida: "*una ontología es una formalización de una conceptualización*" [Gruber, 1993].

En el entorno de la Web Semántica, la definición presentada en el documento "Casos de Uso y Requisitos en OWL"[40] por la W3C resulta muy adecuada:

"*Una ontología define formalmente un conjunto común de términos que se utilizan para describir y representar un dominio. Define los términos utilizados para describir y representar un área de*

40 http://www.w3.org/TR/webont-req/

conocimiento".

Hay varias cosas que se deben aclarar a partir de esta definición [Yu, 2011]:

- La ontología es específica del dominio, y se utiliza para describir y representar un área de conocimiento. Un dominio es un área específica de conocimiento, por ejemplo, la fotografía, la medicina, la educación, etc.

- La ontología contiene los términos, llamados clases o conceptos, y las relaciones entre los mismos. Las relaciones entre estas clases se pueden expresar usando una estructura jerárquica: las superclases representan conceptos de alto nivel (conceptos generales) y las subclases representan conceptos más específicos, los cuales tienen todos los atributos y características de los conceptos generales.

- Además de las relaciones entre las clases, hay otro nivel de relación expresado usando un grupo especial de términos, las propiedades, las cuales describen diversas características y atributos de las clases, y también se pueden utilizar para asociar diferentes clases juntas. Por lo tanto, las relaciones entre las clases no sólo son del tipo superclase o subclase, sino también relaciones expresadas en términos de propiedades.

Al tener los términos y las relaciones entre éstos claramente definidos, la ontología codifica el conocimiento del dominio de tal manera que el mismo pueda ser "comprendido" por una computadora. Esta es la idea básica de una ontología.

Se pueden resumir los beneficios de una ontología de la siguiente manera [Yu, 2011]:

- Proporciona una comprensión/definición común y compartida sobre algunos conceptos clave del dominio

- Proporciona los términos que se pueden utilizar al crear los documentos RDF en el dominio

- Proporciona una forma de reutilizar el conocimiento del dominio

- Explicita los supuestos que se hacen sobre el dominio

- Junto con los lenguajes de descripción de ontologías, como RDFS y OWL, proporciona una forma de codificar el conocimiento y la semántica de tal manera que una máquina los pueda comprender

- Posibilita un procesamiento a gran escala

La figura 3.1.1 muestra un fragmento de la ontología del caso práctico desarrollado en el Capítulo 6, en donde las elipses representan las clases (conceptos), los rectángulos las propiedades (características de los conceptos) y las flechas las relaciones:

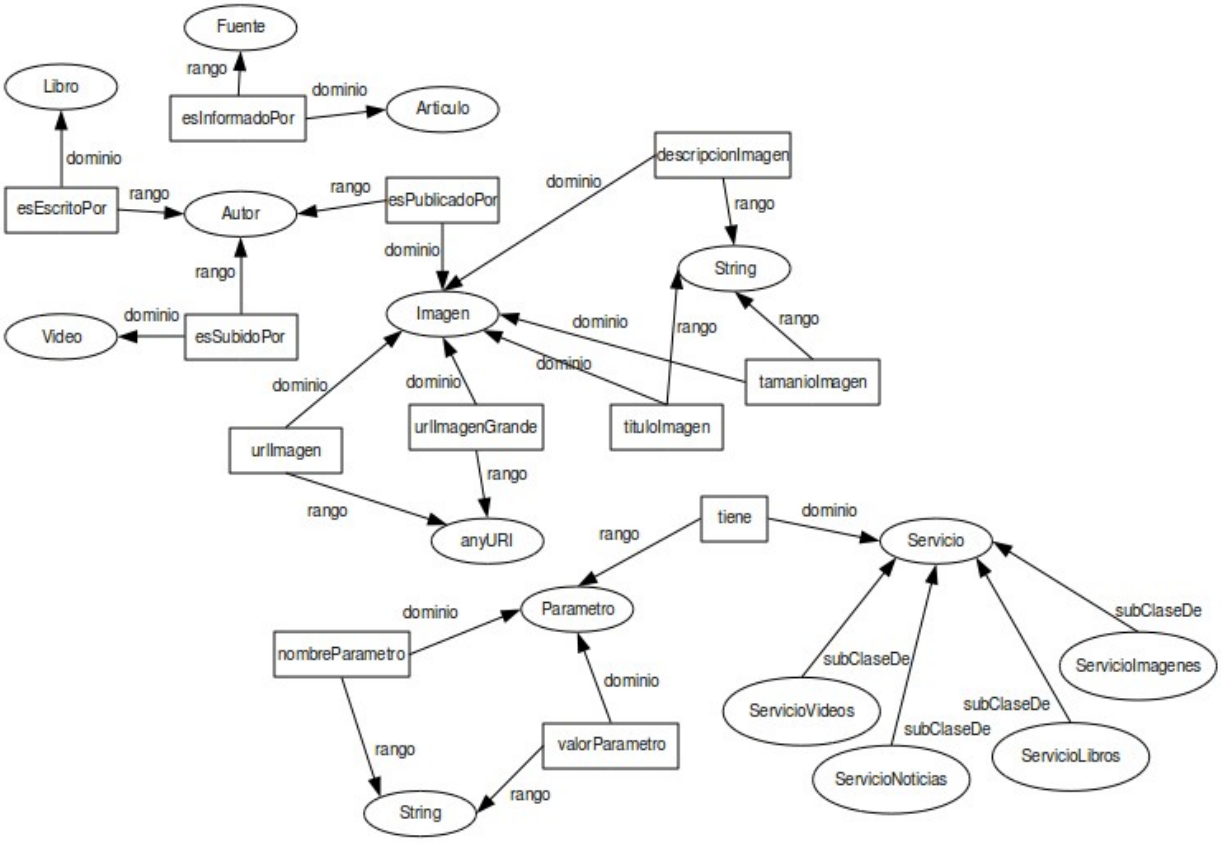

Figura 3.1.1 – Fragmento de ontología desarrollada en el caso práctico

En la figura 3.1.1 se pueden ver, por ejemplo, las clases `ServicioVideos`, `ServicioNoticias`, `ServicioLibros` y `ServicioImagenes`, los cuales representan las fuentes de información correspondientes. A su vez, cada una de estas clases es un tipo especial de `Servicio`. Se puede ver que un `Servicio` cualquiera tiene un `Parametro` (o varios), el cual tiene un nombre (`nombreParametro`) y un valor (`valorParametro`), y que el tipo de datos de cada uno de estos es una cadena (`String`).

En la figura 3.1.1 también aparecen las clases `Libro`, `Video`, `Imagen` y `Articulo`, y se puede ver que un `Libro` es escrito por un `Autor`, una `Imagen` es publicada por un `Autor`, un `Video` es subido por un `Autor`, y que un `Articulo` es informado por una `Fuente` de noticias.

También, según la ontología, una `Imagen` se caracteriza por tener una descripción (`descripcionImagen`), un tamaño (`tamanioImagen`), un título (`tituloImagen`), y un par de direcciones: una para verla en tamaño normal (`urlImagen`) y otra para verla en tamaño grande (`urlImagenGrande`).

3.2 Ontologías bien conocidas de la Web Semántica

Se dijo que una ontología define un vocabulario común para compartir información en un dominio, el cual incluye definiciones (que pueden ser interpretadas por una máquina) de conceptos básicos en el dominio y las relaciones entre los mismos.

Algunas de las razones por las cuales se desarrolla una ontología son [Noy & McGuinness, 2001]:

- Para compartir un entendimiento común de la estructura de la información [Musen, 1992], [Gruber, 1993].

  Por ejemplo, suponer que distintos sitios Web contienen información médica o prestan servicios de asistencia médica. Si estos sitios comparten y publican la misma ontología con los términos que usan todos, agentes de software pueden extraer y agrupar toda esta información para responder a consultas de usuarios o utilizarla como dato de entrada a otras aplicaciones.

- Para permitir la reutilización del conocimiento del dominio

  Por ejemplo, diferentes dominios necesitan modelos para representar el tiempo, lo cual incluye los conceptos de intervalos de tiempo, puntos en el tiempo, medidas relativas de tiempo, etc. Si un grupo de investigadores desarrolla esta ontología en detalle, otros pueden reutilizarla para sus dominios. Además, si se tiene que construir una ontología general, se pueden integrar varias ontologías existentes que describan las partes del gran dominio.

- Para explicitar supuestos del dominio

  Explicitar supuestos del dominio subyacente a una aplicación posibilita cambiar fácilmente estos supuestos si cambia el conocimiento que se tiene sobre el dominio. Si los supuestos se insertan directamente en el lenguaje de programación, los mismos no sólo resultan difíciles de encontrar y comprender sino también de cambiar, en particular para alguien sin experiencia en programación. Además, las especificaciones explícitas sobre el conocimiento del dominio resultan útiles para los nuevos usuarios que tienen que aprender el significado de los términos del dominio.

- Para analizar el conocimiento del dominio

  Analizar el conocimiento del dominio es posible una vez que esté disponible una especificación declarativa de los términos. El análisis formal de los términos es extremadamente valioso tanto cuando se trata de reutilizar ontologías existentes como de extenderlas [McGuinness *et al.*, 2000].

La realización de la Web Semántica depende de un número de factores, uno de los cuales es la habilidad de reutilizar ontologías existentes [Staab *et al.*, 2004]. La construcción de ontologías es una tarea que consume tiempo y trabajo, por lo cual depende en gran medida de la posibilidad de

reutilizar ontologías, existiendo actualmente motores de búsqueda, como Swoogle[41] y Ontosearch[42] [Doran, 2006].

Al buscar ontologías existentes para evaluar su reutilización, las mismas vienen en una amplia variedad [Antoniou & van Harmelen, 2008]:

- Cuerpos de conocimiento especializados

  Algunas ontologías son desarrolladas por un amplio equipo de expertos durante muchos años. Por ejemplo, la *ontología del cáncer*[43] del Instituto Nacional del Cáncer (NCI) en Estados Unidos, el vocabulario *Iconclass*[44] para describir imágenes culturales, etc.

  La ontología del cáncer del NCI es una terminología pública, profunda y compleja en comparación con vocabularios clínicos más amplios, implementando ricas relaciones semánticas entre los nodos de sus taxonomías. Las relaciones semánticas están destinadas a facilitar la investigación y dar soporte a la infraestructura bioinformática del Instituto. Los tópicos descritos en la ontología incluyen enfermedades, medicamentos, productos químicos, diagnósticos, tratamientos, genes, anatomía, organismos y proteínas.

  Iconclass es un sistema de clasificación diseñado para el arte y la iconografía. Es la herramienta científica más ampliamente aceptada para la descripción y recuperación de temas representados en las imágenes (obras de arte, ilustraciones de libros, reproducciones, fotografías, etc) y es utilizado por museos e instituciones de arte de todo el mundo.

- Vocabularios integrados

  El *Sistema Unificado de Lenguaje Médico* (UMLS)[45] integra 100 vocabularios y clasificaciones biomédicas desarrolladas independientemente en una única ontología grande, con más de 750000 conceptos y más de 10 millones de enlaces entre los mismos. No es sorprendente que la semántica de un recurso que integra muchos vocabularios desarrollados independientemente sea bastante baja, pero no obstante ha resultado ser muy útil en muchas aplicaciones, al menos como punto de partida.

- Ontologías de nivel superior

  Mientras que las ontologías anteriores son muy específicas del dominio, se han hecho algunos intentos para definir ontologías de aplicación general (a veces conocidas como ontologías de nivel superior). Una ontología de nivel superior se limita a metaconceptos, genéricos, abstractos y filosóficos, y por lo tanto son lo suficientemente generales como para hacer frente (a un nivel alto) a un amplio rango de áreas de dominio. En estas ontologías no se incluyen aquellos conceptos específicos a un determinado dominio, pero brindan una

41 http://swoogle.umbc.edu/
42 http://www.ontosearch.com/
43 http://www.mindswap.org/2003/CancerOntology/
44 http://www.iconclass.nl/
45 http://umlsinfo.nlm.nih.gov/

estructura y un conjunto de conceptos generales sobre los cuales se pueden construir ontologías de dominio (ontologías médicas, financieras, de ingeniería, etc). Los 2 principales ejemplos son *Cyc*[46] y *SUO*[47].

OpenCyc es la versión código abierto de la tecnología CyC, la base de conocimiento general más grande y más completa, que se puede utilizar como base de una amplia variedad de aplicaciones inteligentes, como sistemas expertos, juegos, etc.

SUO (Standard Upperlevel Ontology) está desarrollando un estándar que especifique una ontología de nivel superior que de soporte a las aplicaciones informáticas tales como la interoperabilidad de datos, la búsqueda y recuperación de información, inferencia automatizada y el procesamiento del lenguaje natural.

- Recursos lingüísticos

  Algunos recursos fueron construidos originalmente no como abstracciones de un dominio en particular, sino más bien como recursos lingüísticos, los cuales han demostrado ser útiles como puntos de partida para el desarrollo de ontologías. El ejemplo principal de esta categoría es *WordNet*[48], con más de 90000 términos.

  WordNet es una gran base de datos léxica de inglés: sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos cognitivos, llamados *synsets*, cada uno expresando un concepto distinto. Estos *synsets* están interconectados entre sí por medio de relaciones conceptuales semánticas y léxicas. La red resultante de palabras y conceptos relacionados de manera significativa se puede navegar con el navegador y también se encuentra disponible para descargarse en forma libre y pública.

- Otras [Yu, 2011]

  En la Web hay muchísimas páginas personales, donde en cada una el autor proporciona generalmente cierta información personal, como correo electrónico, fotos, intereses, etc, creándose de esta forma una red social con la cual se puede responder a preguntas como "¿quién tiene los mismos intereses que yo?".

  Como todos los sitios Web personales se construyen para "los ojos humanos", hay que hacer todo lo anterior de forma manual, resultando muy difícil crear una aplicación que lo haga automáticamente.

  Para que estos documentos resulten comprensibles a una aplicación, se debe crear una ontología sobre personas y cada página personal tiene que estar conectada a algún documento RDF escrito mediante el uso de esta ontología (tiene que estar etiquetada).

46 http://www.opencyc.org/
47 http://suo.ieee.org/
48 http://wordnet.princeton.edu/

Esta fue la motivación detrás del proyecto FOAF (Friend of a Friend)[49]: un vocabulario (una ontología) que incluye los términos básicos que describen información personal, como quién es uno, a qué se dedica y quiénes son sus amigos. Sirve como estándar para todo aquel que quiera marcar sus páginas personales y volverlas documentos que puedan ser procesados por máquinas.

Otro ejemplo es *DBPedia*[50]. Probablemente, la wiki más exitosa y popular sea Wikipedia, la enciclopedia en línea más grande, creada y mantenida por una comunidad de autores distribuida a nivel mundial.

Sin embargo, similar a cualquier sitio wiki tradicional, su uso implica asimilar toda la información y buscar a través de las páginas para encontrar lo que se busca. Para solucionar esto, se desarrolló un agente para extraer la información estructurada existente en cada página (en lugar de agregar marcas semánticas a cada página de la wiki), con la característica que esta información obtenida toma la forma de un grafo RDF.

Lo anterior es la idea sobre cómo transformar Wikipedia para que resulte más útil.

3.3 Lenguajes para representar ontologías en la Web Semántica

Una ontología se representa mediante un *lenguaje de representación del conocimiento* y no mediante lenguaje natural, porque se quiere que esta representación sea lo más clara y precisa posible. Este lenguaje puede tomar distintas formas, por ejemplo, un gráfico, un texto expresado en un cierto código (RDF/S, OWL, OKBC, CycL, Prolog, etc), etc. Lo importante no es la forma que toma este lenguaje, sino cuán poderoso resulta para representar la ontología.

Una cuestión corolaria es que los lenguajes más sofisticados están respaldados por una lógica rigurosa y formal, lo que hace que la ontología pueda ser interpretada por una aplicación de software.

Esta sección presenta una introducción a los siguientes lenguajes para la representación de ontologías: RDF[51] y RDFS[52] y OWL[53].

Estos, y otros lenguajes, están organizados en una jerarquía según un *diagrama de capas de la Web Semántica*, bajo el siguiente principio: un lenguaje de una determinada capa aprovecha las ventajas de los lenguajes de las capas inferiores. La figura 3.3.1[54] muestra este diagrama de capas (de acuerdo a los estándares de la W3C), donde se puede ver que XML está en la parte inferior de la pila, con lo cual, el resto de los lenguajes utilizan la sintaxis XML (XML proporciona la base para la interoperabilidad en la Web).

49 http://www.foaf-project.org/
50 http://dbpedia.org/
51 Resource Description Framework
52 RDF Schema
53 Web Ontology Language
54 http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/layerCake-4.png

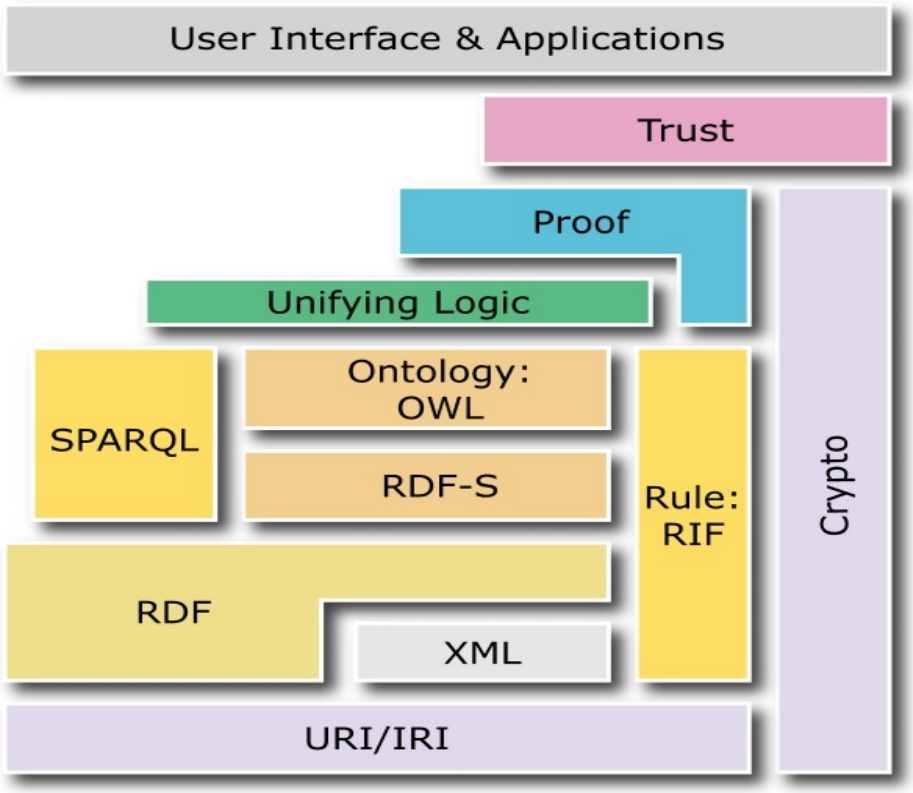

Figura 3.3.1 – Diagrama de capas de la Web Semántica

La capa que sigue a XML es la de RDF/S, la cual proporciona un lenguaje simple para expresar los conceptos y las relaciones de una ontología, junto con sus instancias. Sobre esta capa se encuentra OWL, el cual permite la definición de una ontología mucho más expresiva.

Hay que mencionar además que, a pesar que todas estas capas se expresan en XML, igualmente se debe recurrir a los intérpretes específicos para comprender el lenguaje particular con el fin de aprovechar realmente lo que ofrece cada uno. Por ejemplo, aunque RDF/S y OWL se pueden validar como XML legítimos, sólo los intérpretes de RDF/S y OWL pueden interpretar las respectivas capas. En general, los intérpretes más altos pueden interpretar correctamente todas las capas inferiores. Por lo tanto, un intérprete de OWL puede utilizar cualquier construcción RDF/S o XML, además del código OWL específico.

Finalmente, en la parte superior se encuentran los métodos de razonamiento y pruebas, la capa llamada “Web de confianza”, que utiliza pruebas automatizadas, así como también características de seguridad e identidad que aún siguen siendo relativamente menos conocidas y por lo tanto, menos maduras, que las tecnologías. En la parte superior de la pila están las aplicaciones, las cuales pueden utilizar todas las capas de la Web Semántica y, por lo tanto, mostrar un comportamiento más “inteligente” u ofrecer servicios más “inteligentes”.

3.4 RDF (Resource Description Framework)

La Web a la que se está acostumbrado está hecha de documentos enlazados entre sí. Cualquier conexión entre un documento y la(s) cosa(s) que describe del mundo la hace únicamente la persona que lee el documento. Por ejemplo, puede haber un enlace desde un documento sobre Shakespeare a un documento sobre Stratford, pero no existe la noción de "la entidad Shakespeare" o su vinculación con "la entidad Stratford" [Allemang & Hendler, 2008].

En la Web Semántica se emplea el concepto de *recurso* para referirse a las cosas del mundo. Un *recurso* puede ser cualquier cosa, tangible o intangible, sobre la cual se quiera hablar. Por ejemplo: autores, libros, editores, lugares, gente, hoteles, habitaciones, un puesto de trabajo, etc.

En una red de información, cualquiera puede contribuir al conocimiento sobre un recurso. Fue este aspecto de la Web actual lo que le permitió crecer tan rápido. Para implementar la Web Semántica se necesita un modelo de datos que permita que la información se distribuya a través de la Web.

Aunque se lo suele llamar lenguaje, RDF es esencialmente un modelo de datos muy simple, con varias representaciones. RDF es una forma, no ambigua, de establecer información sobre algo (un recurso), es decir, una forma de especificar información sobre un recurso, la cual se especifica mediante las *propiedades* del mismo. Las *propiedades* describen las relaciones entre los recursos, por ejemplo: "escrito por", "título", etc. [Chase, 2007B]

En conclusión, para describir un recurso en RDF, se lo hace mediante sus propiedades.

El bloque de construcción básico de RDF es una terna, compuesta por:

- Un *sujeto* (en gramática, el sujeto es la cosa sobre la cual trata la oración).
- Un *predicado* (en gramática, el predicado representa una característica sobre la cual se quiere hablar del sujeto).
- Un *objeto* (el valor del predicado de la terna).

A estas ternas se las puede organizar como un grafo dirigido, donde los arcos, dirigidos desde el nodo origen (el sujeto) hacia el nodo destino (el objeto), representan las propiedades (predicados).

En la figura 3.4.1 se puede ver un ejemplo de este grafo, donde tanto los nodos como los arcos se etiquetan. Para la figura se utilizó la ontología del caso práctico que se desarrolla en el Capítulo 6, mostrada en parte en la figura 3.1.1.

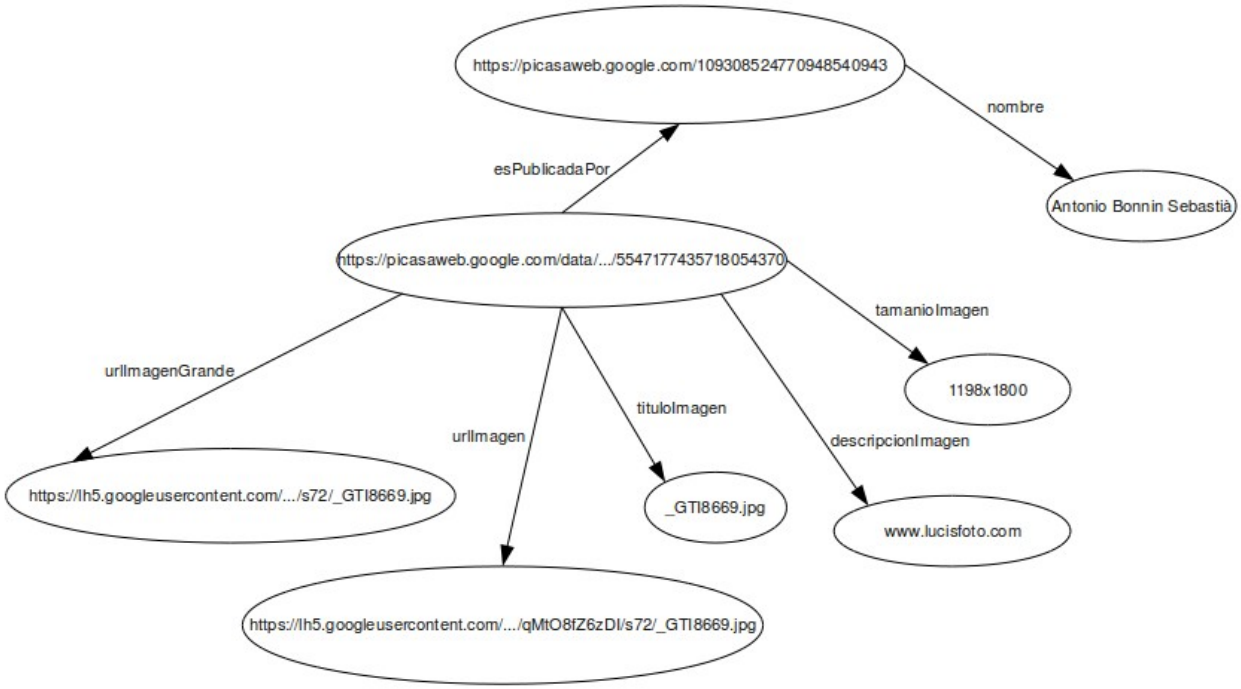

Figura 3.4.1 – Grafo RDF

En la figura 3.4.1 se han etiquetado los nodos y los arcos con nombres simples como "`www.lucisfoto.com`", "`1198x1800`", "`tituloImagen`", etc. En la Web Semántica, esto no resulta suficiente para identificar algo. Por ejemplo, al hablar sobre Shakespeare, ¿todos saben de lo que se está hablando? En la Web Semántica se tiene que ser más específico: ¿en qué sentido se hace referencia a la palabra "Shakespeare"?

RDF soluciona este problema mediante una tecnología de la Web actual: las URIs[55]. Un tipo especial de URI es la URL[56] u otro tipo de identificador único. Las URIs no sólo han sido definidas para los recursos en la Web sino también para objetos tan diversos como números de teléfono, números de ISBN y ubicaciones geográficas. Es decir, una URI es la identificación global de un recurso Web. Ejemplos de URIs:

- http://www.example.org/file.xml#element(home)
- http://www.example.org/file.html#home
- http://www.example.org/file2.xml#xpath1(//q[@a=b])

Lo importante de todo esto, es que RDF utiliza URIs para identificar los recursos. La forma que tomen las URIs no es importante.

Para publicar datos RDF en la Web, resulta adecuado la representación en forma de texto más que la representación en forma de grafo. Existen distintas maneras de expresar RDF en forma textual. Algunas de las representaciones textuales más comunes para RDF son:

55 Uniform Resource Identifier
56 Uniform Resource Locator

- **Ternas N**

  Esta notación referencia los recursos mediante sus URIs completas. Por ejemplo:

```
<http://www.WorkOnt.org/Examples/C3Manufacture.rdf#Producto1>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.WorkOnt.org/Examples/C3Manufacture.rdf#Producto> .
```

- **3 RDF (N3)**

  Combina la notación anterior con el uso de *qnames*, con lo cual debe haber una asociación entre éstos (locales) y las URIs (globales). Las ternas se expresan en el orden sujeto-predicado-objecto, seguido de un punto (.). Por ejemplo:

```
@prefix mfg:
<http://www.WorkOnt.com/Examples/C3/Manufacturing.rdf#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

mfg:Product1 rdf:type mfg:Product .
```

- **RDF-XML**

  Es la forma más popular y conveniente para la Web Semántica [Chase, 2007B]. Por ejemplo:

```
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex="http://example.com#"
   xmlns:dc="http://purl.org/dc/elements/1.1/#">
   <rdf:Description rdf:about="http://www.chaosmagnet.com">
      <ex:nombre>Chaos Magnet</ex:name>
      <dc:creador rdf:resource="http://www.nicholaschase.com" />
   </rdf:Description>
</rdf:RDF>
```

RDF describe cómo especificar información sobre un recurso, pero la información en sí misma no significa nada. Para definir los tipos de objetos y sus relaciones, como clases y subclases, es necesario hacer algunos agregados al vocabulario. Estos agregados se encapsulan en RDF Schema, también conocido como RDFS, el cual tiene su propio espacio de nombres y una serie de elementos y atributos predefinidos.

3.5 RDFS (RDF Schema)

El nombre oficial es RDF Vocabulary Description Language, pero por razones históricas se emplea el término “Schema”.

RDF crea simplemente una estructura en forma de grafo para representar los datos, mientras que RDFS proporciona el vocabulario que se utilizará en este grafo [Allemang & Hendler, 2008].

Analizar el caso de una universidad, en la cual hay profesores que imparten determinados cursos. Si sólo se quiere hablar sobre el profesor `P` o el curso `C`, se puede utilizar RDF. Pero RDF no tiene la capacidad suficiente para hablar sobre *los* cursos, *los* cursos de primer año, *los* profesores, etc., es decir, sobre los conjuntos (clases) que definen los tipos de recursos. Tampoco tiene RDF la capacidad de hablar sobre las relaciones entre estos conjuntos, es decir, las relaciones entre los profesores y los cursos. Para definir un vocabulario mediante el cual se puedan definir los distintos conceptos (clases) y sus relaciones, es necesario hacer algunos agregados, los cuales se encapsulan en RDFS.

La construcción básica para especificar un conjunto en RDFS es una clase, la cual se define en el espacio de nombres *rfds* (`rdfs:Class`).

En RDFS, al igual que en la mayoría de los lenguajes de programación, uno de los usos más importantes de las clases es imponer restricciones sobre qué se puede afirmar sobre un determinado concepto.

Una vez definidas las clases, se deberán establecer las relaciones entre las mismas mediante una jerarquía de superclase y subclase. RDFS proporciona un nuevo recurso, `rdfs:subClassOf`, para expresar la relación "*es un*" entre clases:

Sujeto	**Predicado**	**Objeto**
dep:Futbolista	rdfs:subClassOf	dep:Deportista

Tabla 3.5.1 – Ejemplo de uso del identificador `rdfs:subClassOf`

En RDFS no se requiere que las clases formen una jerarquía estricta, con lo cual una clase puede tener varias superclases.

RDFS proporciona un medio por el cual las clases se pueden relacionar entre sí por medio de la relación de subclase y las inferencias que definen el significado de esta construcción. Esto permite a quien modela en RDFS describir (utilizando la noción de conjuntos) las relaciones entre los elementos que se describen en una colección de ternas RDF. Pero si se quiere dar un significado a los datos, se necesita algo más que hablar sobre los elementos: se tiene que hablar sobre las propiedades que los vinculan, es decir, los predicados de las ternas.

RDFS proporciona un mecanismo sencillo para esto, basado en el recurso `rdfs:subPropertyOf`. La intuición dice que esta construcción es análoga a `rdfs:subClassOf`, pero para las propiedades. Por ejemplo la relación "*hermano*" es más específica que la relación "*pariente*": si alguien es mi *hermano*, entonces también es mi *pariente*.

Se puede decir que cuando se utiliza una propiedad, el sujeto de la terna viene de una cierta clase y el objeto proviene de otra. Esto se expresa en RDFS con los recursos `rdfs:domain` y

`rdfs:range`, respectivamente.

El dominio se refiere al sujeto de cualquier terna que utiliza P como su predicado, y el rango al objeto de cualquier terna que utiliza a P como predicado. Cuando se afirma que la propiedad P tiene dominio D (o rango R), se está diciendo que cada vez que se use la propiedad P, se puede inferir que el sujeto (o el objeto) de esa terna es un miembro de la clase D (o R).

3.6 OWL (Web Ontology Language)

La expresividad que brindan RDF y RDFS es muy limitada. RDF se limita a predicados binarios mientras que RDFS a las jerarquías de subclases y propiedades, con definiciones de dominio y rango de estas propiedades.

El Web Ontology Working Group del W3C[57] ha identificado una serie de casos de uso característicos que requieren mayor expresividad que la que brindan RDF y RDFS.

Un número de grupos de investigación tanto en Estados Unidos como en Europa ya había identificado la necesidad de un lenguaje de modelado de ontologías más poderoso. Esto condujo a una iniciativa conjunta para definir un lenguaje más rico, llamado DAML+OIL[58].

DAML+OIL, a su vez, se tomó como punto de partida para que el Web Ontology Working Group del W3C definiera OWL, el lenguaje destinado a ser el estándar y ampliamente aceptado para la Web Semántica.

Como se ve, las iniciales para el nombre no coinciden con el mismo. La elección de OWL surge como un homenaje al proyecto “One World Language” de William H. Martin del año 1970 [Mitri & Chase, 2006].

RDFS es muy útil, pero no soluciona todos los posibles requisitos. Las aplicaciones complejas suelen requerir más posibilidades:

- Caracterización de propiedades
- Identificación de objetos con diferentes URIs
- Disyunción o equivalencia de clases
- Poder construir clases, no sólo nombrarlas
- Que un programa pueda razonar sobre algunos términos. Por ejemplo: “si los recursos A y B tienen la misma propiedad `foaf:email`, entonces A y B son idénticos”

RDFS proporciona capacidades de inferencias bastante rudimentarias: tipos (en función de la pertenencia de clases) y propiedades. RDFS se puede considerar como un lenguaje ontológico simple. OWL provee características más avanzadas de modelización para describir cómo se pueden relacionar los datos.

57 http://www.w3.org/2001/sw/WebOnt/

58 http://www.daml.org/2001/03/daml+oil-index.html

Como lenguaje OWL tiene 2 objetivos complementarios y conflictivos: poder llegar a describir una amplia variedad de conceptos y relaciones, pero al mismo tiempo, hacer uso de estos conceptos requiere un software que pueda darle sentido a los mismos. Desafortunadamente, cuantas más opciones ofrezca un lenguaje, más difícil resulta escribir software que las contemple a todas.

Para resolver este problema, en la especificación inicial de OWL el W3C identificó 3 versiones (o "especies") de OWL: OWL Lite, OWL DL y OWL Full.

- *OWL Full*: es la versión más expresiva de las tres. Básicamente permite hacer cualquier cosa que permita RDF: se pueden definir clases, usar clases como propiedades e individuos, y construir ontologías que no sean necesariamente "decidibles", lo cual significa que un programa podría no contar con la suficiente información para responder todas las preguntas sugeridas por los datos. Incluso las especificaciones de OWL señalan que es poco probable que una sola herramienta alguna vez de soporte a todas las características de OWL Full.

- *OWL DL*: las siglas DL significan Description Logic (lógica de descripción). Esta versión tiene mucha de la expresividad de OWL Full, pero requiere que las ontologías sean decidibles. También exige que todas las clases se definan explícitamente, y tiene ciertas restricciones sobre algunas de las características más avanzadas de OWL.

- *OWL Lite*: es la versión más chica de OWL. Es para aquellos que necesiten crear una ontología más sencilla, y que no necesiten toda la expresividad del lenguaje. Por ejemplo, OWL Lite permite especificar que debe existir una propiedad para un objeto, y que la misma debe tener un valor, pero no se puede especificar qué valor debe ser. Esta es la versión más fácil para construir herramientas.

3.6.1 Algunas características de OWL

- OWL es una capa extra, casi como RDFS comparado con RDF
- Tiene su propio espacio de nombres, sus propios términos
- Se basa en RDFS
- Propiedades:
    - En OWL se puede caracterizar el comportamiento de las propiedades. Por ejemplo, una propiedad puede ser la inversa de otra, equivalente a otra, simétrica con otra, disjunta de otra, transitiva, etc.
    - OWL hace una distinción entre 2 tipos de propiedades: `DatatypeProperty` (su rango es un literal con tipo) y `ObjectProperty` (su rango es un individuo de una clase en particular).
- Clases
    - En RDFS se pueden crear subclases de clases existentes, y nada más.
    - En OWL, se pueden construir clases a partir de otras mediante:
        - La enumeración de su contenido (la clase consiste exactamente de

determinados individuos)
    - La intersección, unión, complemento (por ejemplo, se puede definir una clase
    - llamada `Literatura` como la unión de las clases `Novela`, `Poesía` e `Historias_Cortas`)
- En OWL, toda clase es automáticamente una subclase de `owl:Thing`. Esto significa que no es necesario definir explícitamente esta relación, se la asume (se puede pensar que `owl:Thing` es la clase base desde la cual se derivan todas las demás).
- OWL también define una segunda clase, `owl:Nothing`, la cual representa una clase vacía, sin miembros.
- Se puede declarar que una clase A es equivalente a una clase B, que 2 clases son disyuntas, etc.

Como se ve, OWL es muy complejo:

- La combinación de las distintas construcciones de clases con las distintas restricciones es muy potente.
- La lógica que sigue OWL es la misma a la seguida por RDF y RDFS:
    - Extender las posibilidades que dan RDF y RDFS con nuevas características
    - Definir la semántica, es decir, el "significado" en término de relaciones
    - Poder inferir nuevas relaciones en base a las anteriores

3.7 Contenido de una ontología

Como se dijo, una ontología está formada por clases, relaciones entre las mismas, propiedades de las clases, restricciones sobre las propiedades e instancias (individuos) de las distintas clases. A continuación se explican cada uno de estos conceptos, siguiendo una guía desarrollada por Horridge [Horridge, 2009].

3.7.1 Definición de las clases y sus relaciones

Una clase representa un término (importante para la ontología) que describe un grupo de objetos similares. Las clases se organizan en una jerarquía del tipo superclase/subclase (taxonomía), la cual representa una relación del tipo "*es un*": una clase `A` es una subclase de `B` si cada instancia de `A` es también una instancia de `B`. Otra forma de ver una relación taxonómica es mediante una relación "*tipo de*": `A` es un tipo de `B`.

Con la jerarquía de clases establecida, se pueden especificar ciertas características, por ejemplo:

- *Clases disyuntas*

  Las clases son disyuntas si no pueden tener instancias en común. Por ejemplo, dadas las

clases `A` y `B`, disyuntas, no puede existir una instancia que pertenezca a ambas. Muchos sistemas permiten especificar, explícitamente, que varias clases son disyuntas. Especificar que las clases son disyuntas permite al sistema validar mejor la ontología.

- *Clases complementarias*

  Una clase complementaria (de otra) contiene todos los individuos que no están contenidos en la clase a la que se está complementando. Por ejemplo se puede tener la superclase `A`, y las subclases `B` y `C`. Si la clase `B` es complementaria de `C`, ésta deberá contener todos los individuos que son `B` y que no sean `C`.

3.7.2 Definición de las propiedades de las clases

Una vez que se han definido algunas clases, se debe describir la estructura interna de los conceptos que representan las mismas. Hay 2 grandes tipos de propiedades:

- *Propiedades de tipo objeto*: los valores de estas propiedades son individuos (instancias) de otras clases.
- *Propiedades de tipo "datatype"*: los valores de estas propiedades son un tipo de datos (enteros, cadenas, etc).

Si el lenguaje con el que se modela la ontología es OWL, éste también tiene un tercer tipo de propiedad: la de tipo *anotación*. Las propiedades de tipo *anotación* se pueden utilizar para agregar información (metadatos) a las clases, a los individuos y a las propiedades. Ejemplos más comunes de estos metadatos son: comentarios, fecha de creación, autor, referencias a recursos como páginas Web, etc.

En una jerarquía de clases del tipo superclase/subclase, todas las subclases heredan las propiedades de sus superclases. Las propiedades, al igual que las clases, pueden tener subpropiedades, con lo cual también se pueden formar jerarquías (taxonomías) de propiedades. Al igual que las clases, las subpropiedades especializan a sus superpropiedades.

Las subpropiedades se pueden crear para los 2 tipos de propiedades, pero lo que no se puede hacer es mezclarlas, es decir, no se puede crear una subpropiedad de tipo *datatype* de una propiedad de tipo *objeto* por ejemplo.

Hay 2 conceptos que se aplican tanto a las propiedades de tipo objeto como a las de tipo *datatype*:

- *Dominio*: son las clases sobre las cuales se aplica una propiedad, o las clases a las que describe una propiedad.
- *Rango*: son las clases permitidas para los valores de una propiedad.

Sobre las propiedades de tipo *objeto* se pueden especificar ciertas características:

- *Propiedades inversas*

  Toda propiedad de tipo *objeto* tiene su correspondiente propiedad inversa. Si una propiedad vincula un individuo `a` con un individuo `b`, entonces su propiedad inversa vincula el individuo `b` con el `a`. Guardar la información "en los 2 sentidos" es redundante, sin embargo, desde la perspectiva de la adquisición del conocimiento es conveniente contar explícitamente con las 2 partes de la información.

  Cuando se establece el dominio y rango de una propiedad, la propiedad inversa tendrá por dominio el rango de la primera, y por rango el dominio de la primera.

- *Propiedades funcionales*

  Si una propiedad es funcional, para un individuo dado, no puede haber más de un individuo que esté relacionado con el primero a través de la propiedad. Por ejemplo, la propiedad `tienePorMadreBiológicaA`: algo sólo puede tener una madre biológica. Si se dice que el individuo `a` `tienePorMadreBiológicaA` `b`, y también se dice que el individuo `a` `tienePorMadreBiológicaA` a `c`, entonces, si `tienePorMadreBiológicaA` es funcional, se puede inferir que `b` y `c` deben ser los mismos individuos. También hay que señalar que si se especificó a `b` y `c` como 2 individuos diferentes, entonces la declaración anterior llevaría a una incoherencia.

  A las propiedades funcionales también se las conoce como propiedades de un solo valor, y es la única característica de las propiedades del tipo *objeto* que también se puede aplicar sobre las del tipo *datatype*.

- *Propiedades funcionales inversas*

  Si una propiedad es funcional inversa significa que la propiedad inversa es funcional.

- *Propiedades transitivas*

  Si una propiedad `P` es transitiva, y la misma relaciona un individuo `a` con un individuo `b`, y también al individuo `b` con el individuo `c`, entonces se puede inferir que el individuo `a` se relaciona con el individuo `c` a través de la propiedad `P`.

  Si una propiedad es transitiva, su inversa también lo es. Si una propiedad es transitiva, no puede ser funcional.

- *Propiedades simétricas*

  Si una propiedad `P` es simétrica, y la misma relaciona un individuo `a` con un individuo `b`, entonces el individuo `b` también se relaciona con el individuo `a` a través de la propiedad `P`. Dicho de otra manera, la propiedad es su propia propiedad inversa.

- *Propiedades antisimétricas*

  Si una propiedad `P` es antisimétrica, y la misma relaciona un individuo `a` con otro `b`, entonces el individuo `b` no se puede relacionar con el individuo `a` través de la propiedad `P`.

- *Propiedades reflexivas*

  Una propiedad `P` se dice que es reflexiva cuando la misma debe relacionar un individuo `a` consigo mismo.

- *Propiedades irreflexivas*

  Una propiedad `P` irreflexiva relaciona un individuo `a` con un individuo `b`, donde `a` y `b` no son los mismos.

Finalmente, para ambos tipos de propiedades se pueden especificar v*alores por defecto*:

> Si un valor en particular es el mismo para la mayoría de las instancias de una clase, se puede definir al mismo como valor por defecto de la propiedad. Luego, cuando se creen instancias de una clase que contenga esta propiedad, se asignará el valor por defecto de forma automática, el cual luego puede cambiar a cualquier otro que permitan las restricciones. Es decir, los valores por defecto están por conveniencia, no exigen nuevas restricciones sobre el modelo ni cambian el modelo de ninguna manera.

3.7.3 Definición de las restricciones sobre las propiedades

Las propiedades describen relaciones binarias: las de tipo *datatype* describen las relaciones entre individuos y valores de datos, mientras que las de tipo *objeto* describen relaciones entre 2 individuos. Por ejemplo, un individuo `a` está relacionado con un individuo `b` a través de la propiedad `P`. Ahora considerar todos los individuos que tienen una relación `P` con otros individuos: se puede ver a estos individuos como pertenecientes a la clase de individuos que tienen alguna relación `P`. La idea es que una clase de individuos se describe o se define por las relaciones en las cuales participan los mismos. En OWL se pueden definir estas clases utilizando restricciones.

Las restricciones pueden ser:

- *Restricciones existenciales*

  Las restricciones existenciales describen clases de individuos que participan en al menos una relación a lo largo de una propiedad especificada con individuos que son miembros de una determinada clase. En OWL a este tipo de restricción se lo conoce con el nombre de "`someValuesFrom`" (`algunosValoresDe`). En la herramienta para edición de ontologías Protégé[59] se emplea la palabra "`some`" para denotar las restricciones

59 http://protege.stanford.edu/

existenciales.

- *Restricciones universales*

  Las restricciones existenciales especifican la existencia de al menos una relación a lo largo de una determinada propiedad a un individuo que sea miembro de una clase específica. Sin embargo, las restricciones existenciales no exigen que las únicas relaciones que puedan existir para la propiedad dada deban ser con individuos que sean miembros de la clase especificada.

  Las restricciones universales describen clases de individuos que, para una determinada propiedad, sólo tienen relaciones a lo largo de esta propiedad con individuos que son miembros de una clase especificada.

  En OWL a este tipo de restricción se la conoce con el nombre de "`allValuesFrom`" (`todosLosValoresDe`). En Protégé se emplea la palabra "`only`" para denotar las restricciones universales.

  Para una determinada propiedad, las restricciones universales no especifican la existencia de una relación. Meramente afirman que si existe una relación para la propiedad, entonces debe ser con individuos que sean miembros de una clase específica.

- *Restricciones de cardinalidad*

  En OWL se puede describir la clase de individuos que tienen por lo menos (`min`), a lo sumo (`max`) o exactamente (`exactly`) un número específico de relaciones con otros individuos o valores de tipo de datos.

  Para una determinada propiedad `P`, una restricción de cardinalidad mínima indica el número mínimo de relaciones `P` en las que un individuo debe participar. Una restricción de cardinalidad máxima especifica el número máximo de relaciones `P` en las que un individuo puede participar. Una restricción de cardinalidad exacta especifica el número exacto de relaciones `P` en las que un individuo debe participar.

  Si además de especificar el número exacto de relaciones `P` en las que debe participar un individuo se especifican los tipos de objetos (clases) dentro de la restricción, se trata de una *restricción de calidad cualificada*.

- *Tipo de datos*

  Una restricción sobre tipo de datos describe qué tipo de valores puede tener la propiedad. A continuación se listan los tipos más comunes:

  Cadena: es el tipo más simple que se utiliza para las propiedades.
  Número: describe propiedades con valores numéricos.
  Booleano: son banderas del tipo sí/no.

Enumeraciones: especifican una lista de los valores específicos permitidos para la propiedad.
Instancia: permiten la definición de relaciones entre individuos. Las propiedades cuyos valores pueden ser una instancia deben definir una lista de las clases permitidas a partir de las cuales provienen las instancias.

3.7.4 Creación de instancias (individuos) de las distintas clases

La última actividad consiste en crear las instancias individuales que hicieran falta para las clases de la jerarquía. Definir una instancia individual de una clase requiere:

- la elección de una clase
- la creación de una instancia individual de esa clase
- completar los valores de las propiedades

3.8 Las ontologías y los mashups semánticos

Con el fin de explicar con mayor detalle la relación entre las ontologías y los mashups semánticos, se toma como ejemplo a continuación un trabajo que muestra cómo se puede relacionar información genética con información sobre secuencias biológicas para encontrar dependencias con la nicotina [Sahoo *et al.*, 2008] (se sabe que aproximadamente entre un 40% y 60% de la dependencia a la nicotina se debe a contribuciones genéticas, y el resto en gran parte al medio ambiente).

En este trabajo se explica cómo se puede dar soporte a la integración de información empleando tecnologías de la Web Semántica, mostrando cómo consultas biológicas complejas pueden ser respondidas por un mashup semántico. Por ejemplo: ¿Qué genes participan en un gran número de secuencias? ¿Qué genes (o productos génicos) interactúan entre sí? ¿Qué genes se expresan en el cerebro?

Estas consultas biológicas requieren la integración de gran cantidad de información proveniente de varias fuentes (por ejemplo, Entrez Gene [Maglott *et al.*, 2005], KEGG [Kanehisa *et al.*, 2004], HomoloGene[60], etc), las cuales no soportan el procesamiento automático de información, necesario para responder a estas preguntas complejas. Por ejemplo, un gran obstáculo para la integración es el formato utilizado por las fuentes de información. Así, los recursos disponibles para Entrez Gene y HomoloGene están en múltiples formatos, incluyendo XML. Aunque XML estandariza la representación de la información desde un punto de vista sintáctico, no explicita las relaciones entre los distintos tipos de entidades. Es decir, aunque el archivo XML de Entrez Gene sea procesable por una máquina, no se puede integrar fácilmente, o automáticamente, con otras fuentes de información sin intervención humana.

Una de las muchas ventajas al utilizar una ontología como modelo de representación del

60 http://www.ncbi.nlm.nih.gov/sites/entrez/query.fcgi?db=homologene

conocimiento para integrar información de fuentes heterogéneas es que la semántica formal de la misma permite a un mashup semántico interpretar consistentemente la información y razonar sobre la misma.

3.9 Conclusiones

En este capítulo se presentó el concepto de ontología, estructura sobre la cual están organizadas las fuentes de información que utiliza un mashup semántico, y que resultan fundamentales para la Web Semántica.

Se vio también que existen distintos lenguajes para representar ontologías, cada uno de los cuales proporciona distintas capacidades de expresividad. Así por ejemplo, RDF describe cómo especificar información sobre un recurso a través de las propiedades del mismo, mientras que RDFS permite establecer jerarquías de subclases y propiedades, con definiciones de dominio y rango para estas últimas.

A diferencia de estos 2 lenguajes, OWL brinda una capacidad de expresividad mucho más grande, a tal punto que el lenguaje está organizado en 3 grupos: OWL Lite, OWL DL y OWL Full.

Finalmente se vieron las entidades que forman una ontología (clases, relaciones, propiedades, instancias) y se mostró un ejemplo en el cual se señala la importancia de la relación entre las ontologías y los mashups semánticos.

Una vez comprendido el concepto de ontología, se debe resolver el problema de la integración de las mismas (mediación de ontologías), debido a que cada fuente organiza su información en su propia ontología.

CAPÍTULO 4 – MEDIACIÓN DE ONTOLOGÍAS

Un supuesto clave de la Web Semántica es que habrá muchas ontologías disponibles, las cuales serán construidas para diferentes colecciones de información y diferentes tipos de aplicaciones. Existen muchas razones para esta proliferación de ontologías: por ejemplo, por lo general resulta más fácil construir una ontología nueva que encontrar una existente que resulte apropiada para una tarea determinada. Del mismo modo, muchas veces se quiere tener control directo sobre la ontología de un dominio particular, en lugar de depender de terceros [Walton, 2007].

La tarea de un mashup semántico consiste en combinar (integrar) información de las distintas ontologías con las que trabaja, para lo cual deberá conciliar las diferencias entre las distintas ontologías, proceso que se conoce como *mediación de ontologías* [Davies *et al.*, 2006], y es el principal problema a resolver en el diseño de un mashup semántico.

El objetivo de este capítulo consiste en explicar las razones por las cuales se presentan discrepancias entre las ontologías y comentar algunos métodos para solucionarlas, mostrando algunos casos prácticos que se emplearon en el Capítulo 6.

En el Apéndice C se encuentra una clasificación de las distintas técnicas para mapear ontologías.

4.1 Razones por las que se producen discrepancias entre las ontologías

Para ilustrar por qué se debe realizar la mediación de ontologías, se pueden considerar los fragmentos de 2 ontologías que se muestran en las figuras 4.1.1 y 4.1.2, las cuales están pensadas para clasificar diferentes tipos de cámaras de foto [Walton, 2007].

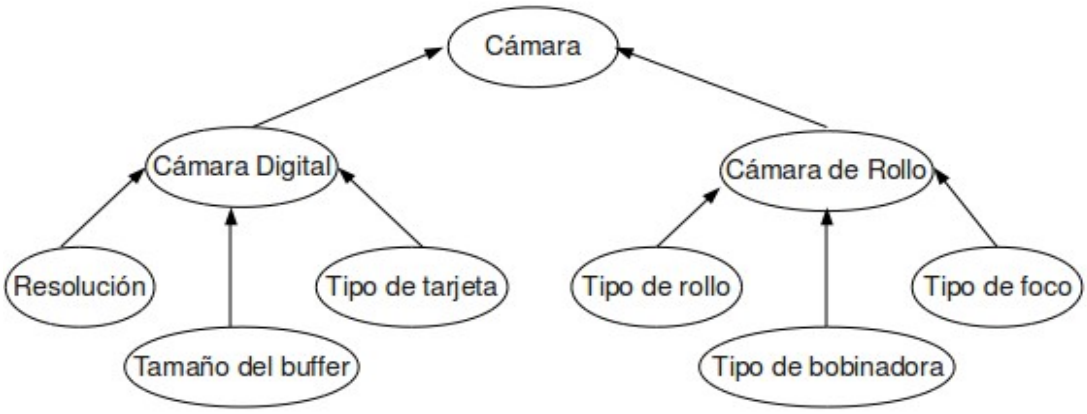

Figura 4.1.1 – Ontología 1 para clasificar diferentes tipos de cámaras de foto

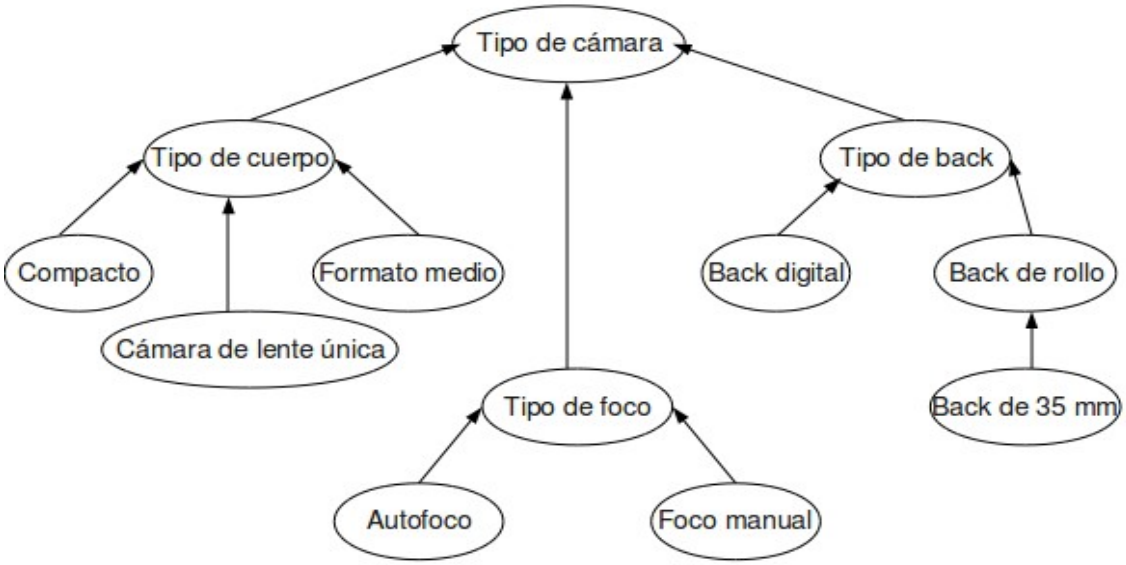

Figura 4.1.2 – Ontología 2 para clasificar diferentes tipos de cámaras de foto

Al examinar estas 2 ontologías, se pueden apreciar (al menos) los siguientes problemas:

- Algunos conceptos pueden tener nombres diferentes, por ejemplo, "`Cámara`" es equivalente a "`Tipo de cámara`". Esto se conoce como *heterogeneidad estructural* [Muthaiyah, 2008], ya que no hay una correspondencia entre las estructuras de las ontologías.
- Algunos conceptos pueden estar presentes sólo en una de las ontologías, por ejemplo, "`Tipo de tarjeta`" y "`Formato medio`".
- Algunos conceptos pueden ser similares, sin ser idénticos, por ejemplo "`Cámara de rollo`" y "`Back de rollo`".
- Algunos conceptos pueden estar relacionados, sin ser equivalentes, por ejemplo "`Tipo de rollo`", y "`Back de 35 mm`".
- Puede haber supuestos no declarados en una ontología, por ejemplo, todas las "`Cámaras digitales`" son "`Autofoco`".

Los problemas de heterogeneidad pueden producirse a diferentes niveles, sin embargo se los puede clasificar en las siguientes categorías [Bouquet *et al.*, 2004]:

- *Heterogeneidad sintáctica*: se produce cuando 2 ontologías se expresan en distintos lenguajes, por ejemplo OWL y F-Logic. Este tipo de heterogeneidad generalmente se trata en el plano teórico, cuando se establecen las equivalencias entre las construcciones de los diferentes lenguajes. Por lo tanto, a veces es posible traducir ontologías expresadas en diferentes lenguajes conservando el significado [Euzenat & Stuckenschmidt, 2003]. Este tipo de heterogeneidad se presenta muy poco en la Web Semántica debido a que el lenguaje más ampliamente utilizado es OWL.

- *Heterogeneidad terminológica*: se encuentran todas las formas de discrepancias relacionadas con el proceso de nombrar las entidades presentes en una ontología. Ejemplos típicos de discrepancias en este nivel son los siguientes:

    - Diferentes palabras que se utilizan para nombrar a la misma entidad (sinonimia), por ejemplo *Paper* vs *Artículo*
    - La misma palabra se utiliza para nombrar diferentes entidades (polisemia), por ejemplo *Sierra* (herramienta) vs *Sierra* (cordillera)
    - Palabras de idiomas diferentes se utilizan para nombrar a las entidades (*Paper* vs *Artículo*)
    - Variaciones sintácticas de la misma palabra (diferente ortografía, abreviaturas, uso de prefijos o sufijos opcionales, etc), por ejemplo *kinesiólogo* vs *quinesiólogo*

- *Heterogeneidad conceptual* (también llamada *heterogeneidad semántica* [Euzenat, 2001]): se encuentran discrepancias que tienen que ver con el contenido de una ontología, las cuales explican, por ejemplo, por qué diferentes ontologías sobre el mismo dominio pueden comenzar a partir de diferentes clases primitivas, o por qué diferentes ontologías pueden contener afirmaciones diferentes (posiblemente contradictorias) acerca de las mismas entidades. Estas discrepancias se pueden dividir en 2 tipos principales:

    - *Diferencias metafísicas*: tienen que ver con la "descomposición" del mundo (es decir, qué entidades, propiedades y relaciones están representadas en una ontología).

    - *Diferencias epistémicas*: tienen que ver con las afirmaciones que se hacen sobre las entidades seleccionadas.

  Las formas prácticas en las que pueden surgir diferencias metafísicas son innumerables, pero se las puede agrupar en 3 tipos abstractos:

    - *Cobertura*: 2 ontologías pueden diferir entre sí porque cubren diferentes partes, que posiblemente se superponen, del mundo (o incluso de un único dominio).

      2 ontologías que se diferencian sólo por la porción del mundo que describen pueden ser disyuntas o pueden solaparse. Un ejemplo sencillo de 2 ontologías disyuntas puede ser el caso de una ontología `O` sobre fútbol y una `O'` sobre cricket. Si se asume que no hay intersección entre las 2, una mediación dirá que las 2 ontologías no tienen ninguna relación.

      La situación es ligeramente diferente cuando las 2 ontologías se superponen (parcial o completamente). Un ejemplo sencillo es una ontología `O` que describa los deportes de equipo y una `O'` que describa los deportes de interior (algunos deportes como el voley pueden pertenecer a ambas ontologías). En este caso, se debe reconocer la parte común y resolver los posibles problemas sintácticos y terminológicos.

- *Granularidad*: 2 ontologías pueden diferir entre sí porque la primera ofrece una descripción más (o menos) detallada de las mismas entidades.

  Considerar el caso de 2 ontologías `O` y `O'` que describan la misma parte del mundo, pero a diferentes niveles de granularidad. Ejemplos sencillos son: cuando `O` caracteriza la posición de objetos físicos sólo mediante 2 coordenadas (latitud y longitud), mientras que `O'` tiene en cuenta también una tercera coordenada (altura sobre el nivel del mar); cuando `O` expresa medidas en centímetros y `O'` en milímetros; etc.

  Para la granularidad, la mediación debe proporcionar una forma de pasar del nivel de representación de una ontología al nivel de representación de la otra. Esta operación es más compleja que la operación requerida en el caso de la cobertura, ya que requiere poner en relación modelos que son intrínsecamente heterogéneos (por ejemplo, hechos de la forma `pos(x, y)` en `O` con hechos de la forma `pos(x, y, t)` en `O'`, donde `x` e `y` pueden expresarse en diferentes unidades de medida).

- *Perspectiva*: una ontología puede proporcionar un punto de vista sobre un dominio que sea diferente del adoptado en otra ontología.

  Suponer 2 ontologías `O` y `O'` que describen la misma región del mundo, al mismo nivel de granularidad, pero desde una perspectiva diferente. Un ejemplo puede ser una representación que utilice expresiones contextuales (como “aquí”, “ahora”, “ayer”), ya que el contenido de este tipo de expresiones depende esencialmente de dónde, cuándo, quién lo dijo, y su interpretación correcta generalmente requiere la capacidad de cambiar la perspectiva de uno. También hay ejemplos menos directos: partidarios de 2 partidos políticos diferentes harán descripciones opuestas de los mismos políticos; “frío” se aplicará a diferentes condiciones climáticas en Finlandia que en Grecia, etc.

  En este caso, la mediación debe proporcionar una forma de “cambiar” la perspectiva de una ontología, cambiar su punto de vista. Para algunas formas de heterogeneidad, esto puede hacerse de manera sistemática y en una forma relativamente sencilla (por ejemplo para descripciones contextuales), sin embargo, en general, el cambio de perspectiva es una tarea muy difícil para cualquier método de mediación.

- *Heterogeneidad semiótica* (también llamada *heterogeneidad pragmática*): se refiere a cómo interpretan las personas las entidades. De hecho, las entidades que tienen exactamente la misma interpretación semántica a veces se interpretan diferente por las personas en relación con el contexto. Por ejemplo, si el concepto `Europa` aparece en el esquema de clasificación de un repositorio multimedia a lo largo de una ruta como `Imágenes/ByN/Europa`, no se puede concluir que sea equivalente al concepto `Europa` en una ontología geográfica, ya que el significado pragmático del primero es el de un contenedor de imágenes en blanco y negro de Europa, mientras que el segundo se trata de un continente (esto no quiere decir que los 2 conceptos no estén conectados, sino que se debe tener en cuenta la estructura de ambos). Este tipo de heterogeneidad es difícil de detectar para una máquina, y aún más difícil de

resolver.

Los problemas de heterogeneidad terminológicos y semánticos son propios de este enfoque, y en mucho menor medida los debidos a sintaxis por el uso de OWL como lenguaje estándar para la representación de ontologías.

4.2 Formas que puede tomar la mediación de ontologías

La mediación de ontologías puede tomar distintas formas [Davies *et al.*, 2006]:

- *Correspondencia (mapeo) de ontologías*: el mapeo de ontologías consiste en encontrar las correspondencias entre las entidades (relacionadas semánticamente) de diferentes ontologías. Al descubrimiento (semi)automático de tales correspondencias se lo conoce como *alineación de ontologías*.

- *Unión de ontologías*: cuando se realiza la unión de ontologías, se crea una nueva ontología como unión de las ontologías originales, la cual captura todo el conocimiento de las primeras. El desafío en la unión de ontologías es asegurar que se reflejen todas las correspondencias y diferencias entre las ontologías en la ontología unión.

Para el caso de un mashup semántico, interesa el mapeo de ontologías, ya que el mismo tiene que trabajar con varias ontologías combinándolas. Como se dijo, el mapeo consiste en definir relaciones entre las distintas ontologías (representaciones heterogéneas), las cuales se pueden utilizar para transformar la expresión de una ontología en una forma compatible con la de otra. Esto puede ocurrir a cualquier nivel [Bouquet *et al.*, 2004]:

- sintáctico: a través de transductores que preserven la semántica
- terminológico: a través de funciones que mapeen información léxica
- conceptual: a través de la transformación general de las representaciones
- pragmático: a través de una transformación que cuide el contexto

Esto significa que cualquier proceso de mapeo de ontologías tiene 4 componentes:

- *Componente sintáctico*: las transformaciones sintácticas necesarias para transformar un formato de representación a otro.

- *Componente terminológico*: la parte del mapeo que expresa las relaciones terminológicas entre las expresiones utilizadas para nombrar las entidades que se van a mapear. Ejemplos sencillos son: el mismo nombre para 2 entidades; 2 entidades que se denominan con expresiones que son una traducción de la otra en un lenguaje diferente; el nombre de una entidad es una abreviatura del nombre de la otra; etc.

- *Componente conceptual*: la parte del mapeo que expresa la relación entre las entidades en diferentes ontologías. Ejemplos sencillos son: el concepto `c1` en la ontología `O1` es

`equivalente` al concepto `c2` en la ontología `O2`; el concepto `c1` en la ontología `O1` es `similar` al concepto `c2` en la ontología `O2`; el individuo `i1` en la ontología `O1` es el `mismo` individuo `i2` en la ontología `O2` ; etc.

- *Componente semiótico/pragmático*: la parte del mapeo que "tiende un puente" sobre el uso de las entidades en diferentes ontologías. Por ejemplo, que el concepto `c1` utilizado en una ontología `O1` para clasificar una colección de documentos se utiliza en un sentido definido por `c2` en la ontología `O2`; que el nombre usado para un concepto en una ontología se toma de un léxico `L`; etc.

Debe quedar claro que el problema del mapeo de ontologías no es trivial, especialmente si se intenta automatizar este proceso. Sin embargo, hay una gran variedad de técnicas que se pueden utilizar, y muchas veces se pueden hacer simplificaciones. Por ejemplo, se pueden requerir sólo mapeos parciales entre las ontologías, es decir, sólo las partes relevantes para la tarea a resolver.

4.3 Técnicas para el mapeo de ontologías

Se han propuesto muchas soluciones para el mapeo de ontologías, que abarcan diferentes puntos de vista: bases de datos, sistemas de información, inteligencia artificial, etc. Estas soluciones aprovechan las ventajas de las diversas propiedades de las ontologías: estructuras, instancias de datos, semántica, etiquetas, etc. y usan técnicas de diferentes campos: estadística y análisis de datos, aprendizaje automático, razonamiento automatizado, lingüística, etc. Estas soluciones comparten algunas técnicas y hacen frente a problemas similares, pero difieren en la forma en que combinan y explotan sus resultados. Como consecuencia, son muy difíciles de comparar y describir, careciendo de un marco uniforme [Euzenat & Shvaiko, 2007].

Una posible división de las técnicas para mapear ontologías puede ser [Euzenat & Shvaiko, 2007]:

- *Las que trabajan a nivel elemento*: consideran las entidades de las ontologías en forma aislada de sus relaciones con otras entidades.

- *Las que trabajan a nivel estructura*: consideran a las entidades de las ontologías para comparar sus relaciones con otras entidades, en lugar de, o además de comparar sus nombres o identificadores.

- *Las que trabajan a nivel instancia*: consideran las instancias para encontrar similitudes.

- *Las que se basan en la semántica*: la característica principal de estas técnicas es que usan la semántica de la teoría de modelos para justificar sus resultados.

4.4 Resolución del mapeo de ontologías

Para demostrar algunos pasos prácticos de mapeo de ontologías, se explica lo hecho en el diseño del mashup semántico del caso práctico mostrado en el Capítulo 6. En este ejemplo, el objetivo principal del mashup semántico consiste en permitirle a un usuario buscar un determinado concepto en distintas fuentes de información (Flickr, Picasa, Amazon, Google, Yahoo y YouTube) y que a su vez pueda intercambiar las mismas a voluntad, creando algo totalmente nuevo sin realizar ningún tipo de programación (de esta forma, el mashup semántico trata uniformemente a estas fuentes).

Los tipos de heterogeneidad que se presentan, no sólo en este caso ya que son propias de este enfoque, son:

- *Terminológica* (hay variaciones en los nombres para las mismas entidades en diferentes ontologías: hay sinonimia, polisemia, etc)
- *Semántica* (hay diferencias en el modelado del mismo dominio de interés: diferencias en la cobertura, granularidad y/o perspectiva)

A estos tipos se le debe agregar el *Sintáctico*, no propio de este enfoque (ver Sección 4.4.2)

4.4.1 Heterogeneidad terminológica

En este ejemplo, al buscar imágenes (Flickr y Picasa), la información que se necesita de las mismas es el título, el tamaño, una descripción, la imagen en sí misma y el autor (nombre real y/o nombre de usuario si lo tuviera). En la tabla 4.4.1.1 se puede ver para cada uno de estos atributos la información que devuelve cada una de las fuentes:

Atributo	**Flickr**	**Picasa**
título	Devuelve información sobre el `title` (título)	Devuelve información sobre el `title` (título)
tamaño	Devuelve información sobre el `height_t` (alto) y el `width_t` (ancho)	Devuelve información sobre el `height` (alto) y el `width` (ancho)
descripción	Devuelve información sobre `description` (descripción)	Devuelve información sobre `summary` (resumen)
imagen	Devuelve información sobre `url_l` (URL con la dirección de la foto)	Devuelve información sobre `url` (URL con la dirección de la foto)
nombre autor	No devuelve esta información	No devuelve esta información
usuario	Devuelve información sobre el `ownername` (dueño)	Devuelve información sobre el `name` (nombre) con respecto a otra sobre el `author` (autor)

Tabla 4.4.1.1 – Información devuelta por las fuentes Flickr y Picasa

Como se puede ver a partir de la tabla 4.4.1.1, hay algunas variaciones en los nombres para una misma entidad (`description` y `summary` por ejemplo).

También, se puede ver que Flickr para el tamaño de la foto devuelve `height_t` y `width_t`, y a la foto en sí misma la devuelve como `url_l`, ya que Flickr devuelve para una foto cualquiera distintos tamaños (desde muy chico a muy grande). Al momento de realizar la consulta a esta fuente, se optó por un único tamaño (dado por la terminación `_t`), y se eligió visualizar la foto en tamaño grande (`url_l`).

En este ejemplo, para resolver el problema de heterogeneidad terminológica, el mapeo entre los conceptos de las distintas ontologías se hizo manualmente, es decir, dado un concepto en una ontología, se buscó/determinó su mejor equivalente en la otra.

4.4.2 Heterogeneidad semántica

Para mostrar el problema de la heterogeneidad semántica, siguiendo con el ejemplo del caso práctico, se muestra a continuación el caso de la fuente YouTube. En este ejemplo, para describir un video cualquiera, interesa la siguiente información:

- título del video
- descripción del video
- una imagen del video
- fecha de publicación del video
- duración del video
- el video en sí mismo

Al hacer una consulta a esta fuente, la información devuelta tiene el siguiente formato XML (en negrita se encuentra resaltada la información relevante para el mashup semántico):

```
<?xml version='1.0' encoding='UTF-8'?>
  <feed xmlns='http://www.w3.org/2005/Atom'
        xmlns:app='http://www.w3.org/2007/app'
        xmlns:media='http://search.yahoo.com/mrss/'
        xmlns:openSearch='http://a9.com/-/spec/opensearch/1.1/'
        xmlns:gd='http://schemas.google.com/g/2005'
        xmlns:yt='http://gdata.youtube.com/schemas/2007'
        gd:etag='W/&quot;CkECR386fSp7ImA9Wx9UE0g.&quot;'>
    ...

    <entry gd:etag='W/&quot;C04CRX47eCp7ImA9Wx9UE0w.&quot;'>
      <id>tag:youtube.com,2008:video:iLmt4G2tXHY</id>
      <published>2011-02-10T04:26:04.000Z</published>
      <updated>2011-02-10T04:26:04.000Z</updated>
      <category scheme='http://schemas.google.com/g/2005#kind'
                term='http://gdata.youtube.com/schemas/2007#video'/>
      <category scheme='http://gdata.youtube.com/schemas/
                        2007/categories.cat'
                term='Entertainment' label='Entertainment'/>
```

```
<category scheme='http://gdata.youtube.com/schemas/
                  2007/keywords.cat'
          term='vos-ke'/>
<category scheme='http://gdata.youtube.com/schemas/
                  2007/keywords.cat'
          term='vlogs'/>
<category scheme='http://gdata.youtube.com/schemas/
                  2007/keywords.cat'
          term='divertido'/>
<category scheme='http://gdata.youtube.com/schemas/
                  2007/keywords.cat'
          term='random'/>
<category scheme='http://gdata.youtube.com/schemas/
                  2007/keywords.cat'
          term='mamadas'/>
<category scheme='http://gdata.youtube.com/schemas/
                  2007/keywords.cat'
          term='funny'/>
<title>calentamiento vacuno</title>
<content type='application/x-shockwave-flash'
         src='http://www.youtube.com/v/iLmt4G2tXHY?f=videos&
              amp;app=youtube_gdata'/>
<link rel='alternate' type='text/html'
      href='http://www.youtube.com/watch?v=iLmt4G2tXHY&
            amp;feature=youtube_gdata'/>
<link rel='http://gdata.youtube.com/schemas/2007#video.responses'
      type='application/atom+xml'
      href='http://gdata.youtube.com/feeds/api/videos/
            iLmt4G2tXHY/responses?v=2'/>
<link rel='http://gdata.youtube.com/schemas/2007#video.related'
      type='application/atom+xml'
      href='http://gdata.youtube.com/feeds/api/videos/
           iLmt4G2tXHY/related?v=2'/>
<link rel='http://gdata.youtube.com/schemas/2007#mobile'
      type='text/html'
      href='http://m.youtube.com/details?v=iLmt4G2tXHY'/>
<link rel='self' type='application/atom+xml'
      href='http://gdata.youtube.com/feeds/api/videos/
            iLmt4G2tXHY?v=2'/>
<author>
  <name>voskevlogs1</name>
  <uri>http://gdata.youtube.com/feeds/api/users/voskevlogs1</uri>
</author>
<yt:accessControl action='comment' permission='allowed'/>
<yt:accessControl action='commentVote' permission='allowed'/>
<yt:accessControl action='videoRespond' permission='moderated'/>
<yt:accessControl action='rate' permission='allowed'/>
<yt:accessControl action='embed' permission='allowed'/>
<yt:accessControl action='list' permission='allowed'/>
<yt:accessControl action='syndicate' permission='allowed'/>
<gd:comments>
  <gd:feedLink href='http://gdata.youtube.com/feeds/api/videos/
                     iLmt4G2tXHY/comments?v=2' countHint='0'/>
</gd:comments>
<media:group>
  <media:category label='Entertainment'
                  scheme='http://gdata.youtube.com/schemas/2007/
                          categories.cat'>
    Entertainment
  </media:category>
  <media:content url='http://www.youtube.com/v/iLmt4G2tXHY?f=videos&
```

```
                    amp;app=youtube_gdata'
                type='application/x-shockwave-flash' medium='video'
                isDefault='true' expression='full' duration='216'
                yt:format='5'/>
    <media:content url='rtsp://v4.cache6.c.youtube.com/
                  CiILENy73wIaGQl2XK1t4K25iBMYDSANFEgGUgZ2aWRlb3MM/
                    0/0/0/video.3gp'
                type='video/3gpp' medium='video' expression='full'
                duration='216' yt:format='1'/>
    <media:content url='rtsp://v8.cache8.c.youtube.com/
                  CiILENy73wIaGQl2XK1t4K25iBMYESARFEgGUgZ2aWRlb3MM/
                    0/0/0/video.3gp'
                type='video/3gpp' medium='video' expression='full'
                duration='216' yt:format='6'/>
    <media:credit role='uploader' scheme='urn:youtube'>
      voskevlogs1
    </media:credit>
    <media:description type='plain'>
      vos-ke vlogs video 1 alentamiento vacuno una vreve descipcion
      del calentamiento global y sus consecuencias.... no mames al
      chile que verdad tan desastrosa....
    </media:description>
    <media:keywords>
      vos-ke, vlogs, divertido, random, mamadas, funny
    </media:keywords>
    <media:player url='http://www.youtube.com/watch?
                  v=iLmt4G2tXHY&amp;feature=youtube_gdata_player'/>
    <media:thumbnail
                url='http://i.ytimg.com/vi/iLmt4G2tXHY/default.jpg'
                height='90' width='120' time='00:01:48'
                yt:name='default'/>
    <media:thumbnail
              url='http://i.ytimg.com/vi/iLmt4G2tXHY/hqdefault.jpg'
              height='360' width='480' yt:name='hqdefault'/>
    <media:thumbnail url='http://i.ytimg.com/vi/iLmt4G2tXHY/1.jpg'
                  height='90' width='120' time='00:00:54'
                  yt:name='start'/>
    <media:thumbnail url='http://i.ytimg.com/vi/iLmt4G2tXHY/2.jpg'
                  height='90' width='120' time='00:01:48'
                  yt:name='middle'/>
    <media:thumbnail url='http://i.ytimg.com/vi/iLmt4G2tXHY/3.jpg'
                  height='90' width='120' time='00:02:42'
                  yt:name='end'/>
    <media:title type='plain'>calentamiento vacuno</media:title>
    <yt:duration seconds='216'/>
    <yt:uploaded>2011-02-10T04:26:04.000Z</yt:uploaded>
    <yt:videoid>iLmt4G2tXHY</yt:videoid>
  </media:group>
  <yt:statistics favoriteCount='0' viewCount='1'/>
 </entry>
 <entry gd:etag='W/&quot;C08FRn47eCp7ImA9Wx9UEkQ.&quot;'>
  <id>tag:youtube.com,2008:video:D5xmLylim4U</id>
  ...
 </entry>
 ...
</feed>
```

Como se ve, por cada video dentro del elemento raíz `<feed>` hay un elemento `<entry>`, dentro del cual se encuentra detallada, en forma de atributos u otros elementos, la información sobre el mismo.

Si se compara la información proporcionada por esta fuente con la necesaria por la ontología, las discrepancias semánticas se deben a aspectos que tienen que ver con la cobertura (la información proveniente de YouTube cubre partes que se superponen con las de la ontología) y la granularidad (la información proveniente de YouTube contiene una descripción más detallada de las mismas entidades de la ontología).

Al igual que en el caso de la heterogeneidad terminológica, para resolver el problema de heterogeneidad semántica, el mapeo entre los conceptos de las distintas ontologías se hizo manualmente.

En todos los casos, la información devuelta por las distintas fuentes está estructurada en XML similar al ejemplo mostrado para YouTube. Es decir, la información está pensada para ser procesada por una persona, pero no por una máquina (la información no tiene definido explícitamente un significado). Siguiendo con el ejemplo de YouTube, si se tomara la información del primer video que se muestra en el fragmento anterior para crear una instancia de la clase `Video` en la ontología del ejemplo, la misma tendría el siguiente formato:

```
<owl:NamedIndividual
           rdf:about="http://www.semanticweb.org/ontologies/tesis#Video1">
  <rdf:type
        rdf:resource="http://www.semanticweb.org/ontologies/tesis#Video"/>
  <fechaVideo rdf:datatype="&xsd;string">
    2011-02-10T04:26:04.000Z
  </fechaVideo>
  <duracionVideo rdf:datatype="&xsd;string">216</duracionVideo>
  <tituloVideo rdf:datatype="&xsd;string">
    calentamiento vacuno
  </tituloVideo>
  <urlImagenVideo rdf:datatype="&xsd;anyURI">
    http://i.ytimg.com/vi/iLmt4G2tXHY/1.jpg
  </urlImagenVideo>
  <urlVideo rdf:datatype="&xsd;anyURI">
    http://www.youtube.com/v/iLmt4G2tXHY?
          f=videos&amp;amp;app=youtube_gdata
  </urlVideo>
  <descripcionVideo rdf:datatype="&xsd;string">
    vos-ke vlogs video 1 alentamiento vacuno una vreve descipcion del
    calentamiento global y sus consecuencias.... no mames al chile que
    verdad tan desastrosa....
  </descripcionVideo>
  <esSubidoPor
       rdf:resource="http://www.semanticweb.org/ontologies/tesis#Autor1"/>

</owl:NamedIndividual>


<owl:NamedIndividual
          rdf:about="http://www.semanticweb.org/ontologies/tesis#Autor1">
  <rdf:type
       rdf:resource="http://www.semanticweb.org/ontologies/tesis#Autor"/>
  <usuario rdf:datatype="&xsd;string">
    http://gdata.youtube.com/feeds/api/users/voskevlogs1
  </usuario>
  <nombreAutor rdf:datatype="&xsd;string">voskevlogs1</nombreAutor>
```

```
</owl:NamedIndividual>
```

Comparando los 2 últimos fragmentos de código (XML y OWL), se puede ver que para YouTube la información de cada video está en un elemento `<entry>`, mientras que para la ontología en un elemento `<NamedIndividual>`, y mediante un atributo `about` se distingue el tipo. Para el caso de YouTube, la información correspondiente a la fecha en la cual se subió el video se encuentra en el elemento `<published>`, y para la ontología dentro del elemento `<fechaVideo>` (`fechaVideo` es una de las propiedades que se asignó a la clase `Video`).

Un mashup semántico trabaja con fuentes que organizan su información de manera tal que la misma pueda ser "interpretada" por el mashup, es decir, información organizada en ontologías. Como se mostró en el fragmento anterior, estas 6 fuentes organizan su información en XML (también podrían haberlo hecho empleando estructuras relacionales, hojas de cálculo, canales de noticias, etc), apareciendo de esta forma también problemas relacionados a la heterogeneidad sintáctica debido al empleo de distintos lenguajes/formatos/estándares.

En el ejemplo del caso práctico, para resolver el problema de la heterogeneidad sintáctica, se convirtió el formato devuelto por cada fuente (XML) al formato de la ontología (OWL) mediante transformaciones XSL[61]. Así como para el caso de los documentos HTML se emplean hojas de estilo CSS, para los documentos XML se emplean hojas de estilo XSL.

A continuación se muestra la transformación XSL para el caso de la fuente de YouTube:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:owl="http://www.w3.org/2002/07/owl#"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
                xmlns="http://www.semanticweb.org/ontologies/tesis"
                xmlns:n1="http://www.w3.org/2005/Atom"
                xmlns:media='http://search.yahoo.com/mrss/'
                xmlns:yt='http://gdata.youtube.com/schemas/2007'>
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
      <rdf:RDF >
        <xsl:apply-templates select="/n1:feed/n1:entry"/>
      </rdf:RDF >
    </xsl:template>

    <xsl:template match="n1:entry">
      <xsl:element name="owl:NamedIndividual">
        <xsl:attribute name="rdf:about">
          http://www.semanticweb.org/ontologies/tesis#
          <xsl:value-of select="n1:id"/>
        </xsl:attribute>

        <xsl:element name="rdf:type">
          <xsl:attribute name="rdf:resource">
            http://www.semanticweb.org/ontologies/tesis#Video
          </xsl:attribute>
        </xsl:element>
```

61 http://www.w3.org/Style/XSL/

```
    <xsl:element name="duracionVideo">
      <xsl:attribute name="rdf:datatype">
        http://www.w3.org/2001/XMLSchema#nonNegativeInteger
      </xsl:attribute>
      <xsl:value-of select="media:group/yt:duration/@seconds"/>
    </xsl:element>

    <xsl:element name="fechaVideo">
      <xsl:attribute name="rdf:datatype">
        http://www.w3.org/2001/XMLSchema#date
      </xsl:attribute>
      <xsl:value-of select="n1:published"/>
    </xsl:element>

    <xsl:element name="descripcionVideo">
      <xsl:attribute name="rdf:datatype">
        http://www.w3.org/2001/XMLSchema#string
      </xsl:attribute>
      <xsl:value-of select="media:group/media:description"/>
    </xsl:element>

    <xsl:element name="tituloVideo">
      <xsl:attribute name="rdf:datatype">
        http://www.w3.org/2001/XMLSchema#string
      </xsl:attribute>
      <xsl:value-of select="media:group/media:title"/>
    </xsl:element>

    <xsl:element name="urlVideo">
      <xsl:attribute name="rdf:datatype">
        http://www.w3.org/2001/XMLSchema#anyURI
      </xsl:attribute>
      <xsl:value-of select="n1:content/@src"/>
    </xsl:element>

    <xsl:element name="urlImagenVideo">
      <xsl:attribute name="rdf:datatype">
        http://www.w3.org/2001/XMLSchema#anyURI
      </xsl:attribute>
      <xsl:value-of select="media:group/media:thumbnail[1]/@url"/>
    </xsl:element>

    <xsl:element name="esSubidoPor">
      <xsl:attribute name="rdf:resource">
        http://www.semanticweb.org/ontologies/tesis#
        <xsl:value-of select="n1:author/n1:uri"/>
      </xsl:attribute>
    </xsl:element>
  </xsl:element>

  <xsl:element name="owl:NamedIndividual">
    <xsl:attribute name="rdf:about">
      http://www.semanticweb.org/ontologies/tesis#
      <xsl:value-of select="n1:author/n1:uri"/>
    </xsl:attribute>

    <xsl:element name="rdf:type">
      <xsl:attribute name="rdf:resource">
        http://www.semanticweb.org/ontologies/tesis#Autor
      </xsl:attribute>
```

```
      </xsl:element>

      <xsl:element name="nombreAutor">
        <xsl:attribute name="rdf:datatype">
          http://www.w3.org/2001/XMLSchema#string
        </xsl:attribute>
        <xsl:value-of select="n1:author/n1:name"/>
      </xsl:element>

      <xsl:element name="usuario">
        <xsl:attribute name="rdf:datatype">
          http://www.w3.org/2001/XMLSchema#string
        </xsl:attribute>
        <xsl:value-of select="n1:author/n1:uri"/>
      </xsl:element>
    </xsl:element>

  </xsl:template>
</xsl:stylesheet>
```

De esta forma la transformación traduce la representación de la información de una fuente (XML en este caso) en la representación ontológica correspondiente. En esta transformación a su vez se encuentran los mapeos necesarios para resolver los problemas por heterogeneidad terminológica y semántica.

4.5 Conclusiones

Un mashup semántico es una aplicación que trabaja con información proveniente de diversas fuentes, cada una de las cuales la organiza en su propia ontología. Al tratar con varias ontologías, al momento de integrar la información de las mismas el mashup debe resolver los problemas de heterogeneidad debido a las distintas discrepancias que se presentan, proceso que se conoce como mediación de ontologías.

Estas discrepancias pueden deberse a cuestiones relacionadas con el aspecto sintáctico (debido al uso de distintos lenguajes para representar las ontologías), el aspecto terminológico (debido al nombre con el cual se designan las distintas entidades presentes en una ontología), el aspecto semántico (debido al contenido de las ontologías, pudiéndose presentar discrepancias por cobertura, granularidad y/o perspectiva) y/o el aspecto pragmático (debido al contexto en el cual las personas interpretan la información).

En el caso de un mashup semántico, los tipos de heterogeneidad que se presentan con mayor frecuencia son los debidos a cuestiones terminológicas y semántica, y en mucho menor medida los debidos a cuestiones sintácticas.

Para la mediación de ontologías, se pueden buscar las correspondencias semánticas entre las distintas entidades (mapeo de ontologías) o se puede crear una nueva ontología como unión de las ontologías que se busca mediar, la cual capture todo el conocimiento (unión de ontologías). Para el caso de un mashup semántico, interesa el primer caso (el mapeo), ya que el mismo tiene que trabajar con varias ontologías combinándolas.

Para el mapeo de ontologías, se pueden emplear técnicas que trabajan a nivel elemento, a nivel estructura, a nivel instancia o las que se basan en la semántica. Debido al alcance de estos temas, se explicaron algunos de los casos que se presentaron en el ejemplo práctico, mostrando como se los resolvió.

También, se debe mencionar que se han desarrollado una serie de herramientas basadas en la idea que las personas están mejor equipadas para mapear ontologías. Estas herramientas están diseñadas de tal manera que resulta fácil definir los mapeos, y también se pueden combinar con métodos automatizados que generan mapeos candidatos. Ejemplos de estas herramientas incluyen: ConcepTool[62], PROMPT[63], OntoMap[64] y MAFRA[65].

62 http://www.aktors.org/technologies/conceptool/
63 http://protege.stanford.edu/plugins/prompt/prompt.html
64 http://ontomap.sourceforge.net/
65 http://mafra-toolkit.sourceforge.net/

CAPÍTULO 5 - METODOLOGÍA PARA EL DISEÑO DE MASHUPS SEMÁNTICOS

El objetivo de este trabajo consiste en definir una metodología para el diseño de mashups semánticos, que ayude a los desarrolladores a diseñarlos desde cero. Como se dijo antes, un mashup semántico es una aplicación Web que extrae información de diversas fuentes, la combina (integra) y presenta en otras formas. La característica principal de esta información es que resulta legible a una máquina y su significado está explícitamente definido.

Los siguientes principios guiarán la construcción de esta metodología:

- Deberá ser lo suficientemente general para permitir a un desarrollador diseñar un mashup semántico independientemente de la plataforma de desarrollo utilizada.

- Deberá definir cada proceso y actividad en forma precisa, indicando claramente su propósito, entradas, salidas y los actores involucrados, adoptándose las siguientes definiciones:

    - *Proceso*: conjunto de actividades cuyo objetivo es el desarrollo o evolución del software [Sommerville, 2007].

    - *Actividad*: cuerpo definido del trabajo a realizar, incluyendo su información de entrada y salida requeridas [IEEE, 1997]. Las actividades se pueden dividir en cero o más tareas.

La metodología propuesta, que se evaluará en un caso de estudio tratado en el Capítulo 6, se estructura en los siguientes procesos:

1. Definición de los objetivos del mashup semántico
2. Modelado del dominio
3. Mediación de ontologías
4. Acceso a las fuentes de información
5. Diseño de la presentación
6. Implementación
7. Pruebas

Este trabajo no tratará los últimos 3 procesos de la metodología propuesta ("*Diseño de la presentación*", "*Implementación*" y "*Pruebas*") debido a que estos temas se encuentran tratados en otros trabajos.

En la figura 5.1 puede verse la estructura general de la metodología, basada en la propuesta por NeOn [García-Castro *et al.*, 2009] para el desarrollo de aplicaciones semánticas:

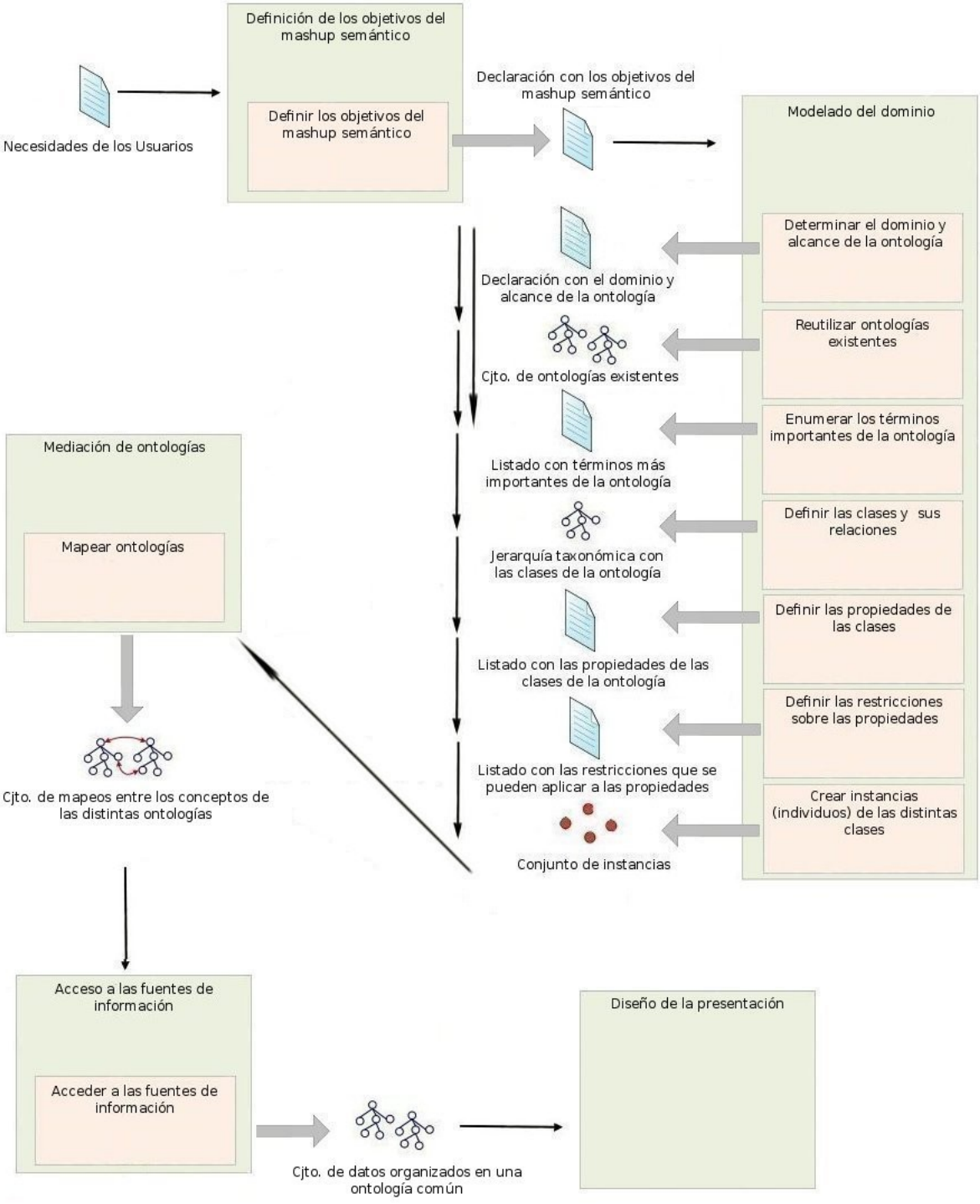

Figura 5.1 – Estructura general de la metodología propuesta

El objetivo de este capítulo consiste en explicar cada uno de los procesos y actividades que forman la metodología propuesta para el diseño de un mashup semántico, describiendo cada una de las actividades que conformarán los procesos utilizando la siguiente plantilla [García-Castro *et al.*, 2009]:

Nombre de la actividad	
Objetivo	
Explica el objetivo principal de la actividad	
Entrada	*Salida*
Recursos necesarios para realizar la actividad	Resultados obtenidos después de realizar la actividad
Participantes	
Identifica la gente involucrada en la realización de la actividad	
Cuándo	
Explica en qué momento se realiza la actividad	

Tabla 5.1 – Plantilla para especificar un proceso o actividad

5.1 Definición de los objetivos del mashup semántico

Durante este proceso, se debe formular la declaración de los objetivos del mashup semántico, ya que de lo contrario, no se contará con una base adecuada para tomar decisiones referidas al proceso siguiente ("*Modelado de la ontología de referencia*"), o para evaluar la efectividad del mismo.

También, esta declaración servirá como base para decidir qué información incluir (o no), cómo estructurarla y cómo presentarla.

La finalidad de cualquier mashup (semántico o 2.0) es presentarle información al usuario de la forma más útil posible, debiendo ser la interacción con el mismo lo más simple y sólo con lo que resulte necesario. Por esta razón, para establecer los objetivos se empleará el lenguaje natural utilizando la terminología del usuario [Feiler, 2008].

Como los mashups (semánticos o 2.0) son aplicaciones Web simples, con metas bien específicas, tienen pocos objetivos. Por ejemplo, en el caso práctico desarrollado en el Capítulo 6, se especifica que el objetivo principal del mashup semántico consistirá en permitirle a un usuario buscar en distintas fuentes de información (libros, artículos de noticias, imágenes y videos), pudiendo intercambiar las mismas a voluntad, (usar una, todas, o un grupo de estas fuentes) sin realizar ningún tipo de programación (característica de cualquier mashup). Por esta razón, este proceso estará compuesto por una sola actividad: *"Definir los objetivos del mashup semántico"*.

5.1.1 Definir los objetivos del mashup semántico

Definir los objetivos del mashup semántico	
Objetivo	
Esta actividad establece los objetivos que debe cumplir el mashup semántico	
Entrada	*Salida*
Las necesidades del o los usuarios	Una declaración con los objetivos del mashup semántico
Participantes	
Desarrolladores del mashup semántico y opcionalmente el cliente o los usuarios finales. Es necesario que algunos de los desarrolladores tengan experiencia en tecnología semántica	
Cuándo	
Esta actividad se debe realizar al comienzo, en el proceso de "*Definición de los objetivos del mashup semántico*"	

Tabla 5.1.1.1 – Actividad "*Definir los objetivos del mashup semántico*"

5.2 Modelado del dominio

Una vez establecidos los objetivos del mashup semántico, la meta de este proceso es convertirlos en descripciones formales, que se puedan utilizar más tarde para implementar el mashup semántico. Este proceso trata con el modelado del dominio del mashup semántico.

Ya sea que se desarrollen sistemas "tradicionales" o aplicaciones para la Web, aparece el concepto de *modelo*, el cual se utiliza en el contexto de "*modelo del dominio*". Para un sistema software, un modelo del dominio representa los conceptos y las estructuras de datos relacionadas a partir de un dominio de aplicación. También codifica el conocimiento que guía el comportamiento de la aplicación, y se expresa mediante un conjunto de clases y una descripción de las interacciones entre las mismas, por ejemplo, mediante una colección de diagramas de diseño UML. El modelo del dominio es la parte central de un sistema software.

En el caso de la Web Semántica, las aplicaciones utilizan ontologías y modelos RDF, es decir, en cualquier aplicación para la Web Semántica, la ontología es el modelo del dominio [Yu, 2011].

Para un mashup semántico, esta ontología representa su modelo del dominio (contiene los conceptos y las relaciones entre los mismos) y es la que permite establecer las correspondencias (mapeos) entre los distintos conceptos y relaciones entre las ontologías de las fuentes de información con las cuales trabaja.

Así, un mashup semántico que trabaje con 3 ontologías, A, B y C, deberá integrar la información de las mismas estableciendo los mapeos correspondientes tomando su ontología como referencia.

La finalidad de este proceso es modelar el dominio del mashup semántico en una ontología, la cual se utilizará como referencia para luego poder realizar la mediación de las distintas ontologías de cada una de las fuentes de información.

El desarrollo de una ontología incluye:

- la definición de las clases;
- la organización de las clases en una jerarquía taxonómica (subclase – superclase);
- la definición de las propiedades y la descripción de los valores permitidos para las mismas;
- la asignación de valores a las propiedades para las distintas instancias.

Teniendo en cuenta esto, y siguiendo la guía propuesta por Horridge [Horridge, 2009], este proceso estará compuesto de las actividades *"Determinar el dominio y alcance de la ontología", "Reutilizar ontologías existentes", "Enumerar los términos importantes de la ontología", "Definir las clases y sus relaciones", "Definir las propiedades de las clases", "Definir las restricciones sobre las propiedades"* y *"Crear instancias (individuos) de las distintas clases"*.

5.2.1 Determinar el dominio y alcance de la ontología

Para determinar el dominio y alcance de la ontología de referencia se deberán responder algunas preguntas:

- ¿Cuál es el dominio que abordará la ontología?
- ¿Para qué se va a utilizar la ontología?
- ¿A qué tipo de preguntas debe dar respuesta la información en la ontología?
- ¿Quién utilizará y mantendrá la ontología?

Las respuestas a estas preguntas pueden cambiar durante el diseño de la ontología, pero en cualquier instante de tiempo ayudan a limitar el alcance del modelo.

En el caso práctico, para determinar el dominio y alcance de la ontología se tuvo en cuenta que se debía tratar con información relacionada con libros, videos, imágenes y artículos de noticias. La ontología debía permitir responder a preguntas como:

- ¿Qué libros, videos, artículos de noticias y/o imágenes hay relacionados a un concepto?

- ¿Cuál es la información detallada de los libros, videos, artículos de noticias y/o imágenes que tienen información sobre un concepto?

Las respuestas a estas preguntas ayudan a determinar el dominio y alcance de la ontología.

Determinar el dominio y alcance de la ontología	
Objetivo	
Esta actividad determina el dominio y alcance de la ontología	
Entrada	*Salida*
La salida del proceso *"Definición de los objetivos del mashup semántico"* (una declaración con los objetivos del mashup semántico)	Una declaración con el dominio y alcance de la ontología
Participantes	
Desarrolladores del mashup semántico y opcionalmente el cliente o los usuarios finales	
Cuándo	
Esta actividad se debe realizar una vez definidos los objetivos del mashup semántico (al comienzo)	

Tabla 5.2.1.1 – Actividad "*Determinar el dominio y alcance de la ontología*"

5.2.2 Reutilizar ontologías existentes

Casi siempre conviene considerar lo que han hecho otros y comprobar si se puede refinar y extender para un dominio y tarea en particular. Reutilizar ontologías existentes puede ser un requisito si el sistema necesita interactuar con otras aplicaciones comprometidas con ontologías particulares o vocabularios controlados.

Hay muchas ontologías disponibles en formato electrónico que se pueden importar en el entorno que se esté utilizando. El formalismo en el que se exprese una ontología no importa, ya que muchos sistemas de representación del conocimiento pueden importar y exportar ontologías. Incluso si un sistema de representación del conocimiento no puede trabajar directamente con un formalismo particular, la tarea de traducir una ontología de un formalismo a otro no suele ser difícil.

Como se dijo antes, a veces también resulta más fácil construir una ontología nueva que encontrar una existente que resulte apropiada para una tarea determinada. También se puede querer tener el control directo sobre la ontología de un dominio particular, en lugar de depender de terceros.

Con respecto a las fuentes de información con las que trabaja el mashup semántico, se pueden presentar los siguientes casos:

1. Que ninguna cuente con un modelo semántico (es decir, que ninguna fuente cuente con una ontología)
2. Que todas cuenten con un modelo semántico (es decir, que todas las fuentes cuenten con una ontología)

3. Una combinación de los 2 casos anteriores.

Si alguna fuente de información contara con una ontología, la misma podría reutilizarse (total o parcialmente) para modelar el dominio del mashup semántico, y por lo tanto servir como ontología de referencia contra la cual integrar el resto. Aún cuando todas las fuentes contaran con una ontología, también se podría reutilizar otra ontología existente para modelar el dominio.

En el caso que ninguna fuente contara con una ontología, se podría tomar el modelo de datos subyacente de una de las fuentes (u otro modelo de datos existente) para modelar la ontología de referencia.

Si se toma la decisión de reutilizar una ontología existente (o un modelo de datos que sirva para modelar la ontología de referencia), en caso de existir más de una primero habrá que seleccionar la que mejor se ajuste a los objetivos del mashup semántico, y luego determinar si la misma se reutilizará total o parcialmente según el resultado de la actividad anterior ("*Determinar el dominio y alcance de la ontología*"). Con esta ontología de referencia, habrá que realizar la mediación con el resto de las ontologías de las otras fuentes de información (ver la actividad "Mediación de ontologías").

En el caso práctico, ninguna fuente de información contaba con una ontología, y se tomó la decisión de crear una ontología nueva, debiendo realizar las actividades "*Enumerar los términos importantes de la ontología"*, "*Definir las clases y sus relaciones"*, "*Definir las propiedades de las clases"*, "*Definir las restricciones sobre las propiedades"* y "*Crear instancias (individuos) de las distintas clases"* a fin de obtener la ontología de referencia. Se espera que en un futuro cada vez más fuentes cuenten con una ontología.

Reutilizar ontologías existentes	
Objetivo	
Esta actividad busca ontologías previas que se puedan adaptar al dominio y alcance de la ontología de referencia	
Entrada	*Salida*
Salida de la actividad "*Determinar el dominio y alcance de la ontología*" correspondiente al proceso "*Modelado del dominio*"	Un conjunto de ontologías o modelos de datos subyacentes (en caso de existir)
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez definidos el dominio y alcance de la ontología de referencia que se busca modelar	

Tabla 5.2.2.1 – Actividad "*Reutilizar ontologías existentes*"

5.2.3 Enumerar los términos importantes de la ontología

Resulta útil hacer una lista de aquellos términos sobre los que se desean hacer declaraciones:

- ¿Cuáles son los términos sobre los que se quiere hablar?
- ¿Qué propiedades tienen esos términos?
- ¿Qué se querría decir sobre esos términos?

Inicialmente, es importante obtener una lista completa de estos términos sin preocuparse del solapamiento entre los conceptos que representan, las relaciones entre los mismos o si un determinado concepto se debe considerar como una clase o como una propiedad.

Las siguientes 2 actividades, "*Definir las clases y relaciones entre las mismas*" y "*Definir las propiedades de las clases*", están estrechamente relacionadas. Es difícil hacer una de ellas primero y luego la otra. Por lo general, se crean algunas definiciones de los conceptos en la jerarquía y luego se continúan describiendo las propiedades de estos conceptos y así sucesivamente. Estas 2 actividades son también las más importantes en este proceso.

Por ejemplo, algunos de los términos importantes para el caso práctico eran: libro, número de ISBN, título, idioma, editorial, precio, autor, usuario, video, fecha de publicación, duración, imagen, tamaño, artículo de noticia, resumen, etc.

Enumerar los términos importantes de la ontología	
Objetivo	
Esta actividad busca obtener una lista con los términos más importantes para la ontología, concentrándose en lo importante dejando de lado los detalles innecesarios	
Entrada	*Salida*
Salida de la actividad "*Determinar el dominio y alcance de la ontología*" correspondiente al proceso "*Modelado del dominio*"	Un listado con los términos más importantes para la ontología
Participantes	
Desarrolladores del mashup semántico y opcionalmente el cliente o los usuarios finales.	
Cuándo	
Esta actividad se debe realizar una vez definidos el dominio y alcance de la ontología que se busca modelar	

Tabla 5.2.3.1 – Actividad "*Enumerar los términos importantes de la ontología*"

5.2.4 Definir las clases y sus relaciones

Hay varios enfoques posibles en el desarrollo de una jerarquía de clases:

- *De arriba hacia abajo (top-down)*: comienza con la definición de los conceptos más generales en el dominio y la posterior especialización de los mismos.
- *De abajo hacia arriba (bottom-up)*: comienza con la definición de las clases más específicas (las hojas de la jerarquía) y la posterior agrupación de las mismas en conceptos más generales.
- *Una combinación de los 2 enfoques anteriores*: primero se definen los conceptos más relevantes y luego se los generaliza y especializa de manera apropiada.

Ninguno de los enfoques anteriores es, intrínsecamente, mejor que cualquiera de los otros. El enfoque más adecuado depende en gran medida de la visión que se tenga sobre el dominio. Sea cual sea el método que se elija, por lo general se comienza por la definición de las clases. A partir de la lista creada en la actividad 5.2.3 ("*Enumerar los términos importantes de la ontología*"), se seleccionan los términos que describen los objetos, los cuales serán las clases en la ontología de referencia, que se organizarán en una jerarquía del tipo superclase/subclase (taxonomía).

No hay una única jerarquía correcta de clases para un dominio dado. La jerarquía depende de los posibles usos de la ontología, del nivel de detalle necesario para la aplicación, de las preferencias personales, y muchas veces de requisitos de compatibilidad con otros modelos.

Con la jerarquía de clases establecida, se pueden especificar ciertas características, por ejemplo qué clases son disyuntas o cuáles complementarias. A continuación, se enumeran algunas características/recomendaciones a tener en cuenta al desarrollar una jerarquía de clases:

- *Transitividad de las relaciones jerárquicas*

  Si `B` es subclase de `A` y `C` es subclase de `B`, entonces `C` es subclase de `A`.

- *Evitar los ciclos de clases*

  Se dice que hay un ciclo en una jerarquía cuando una clase `A` tiene una subclase `B` y al mismo tiempo, `B` es una superclase de `A`. La creación de este ciclo es igual a declarar que las clases `A` y `B` son equivalentes: todas las instancias de `A` son instancias de `B`, y todas las instancias de `B` son también instancias de `A`.

- *Cuándo introducir una nueva clase (o no)*

  Una de las decisiones más difíciles de tomar durante el modelado es cuándo introducir una nueva clase o cuándo representar una distinción a través de los diferentes valores de una propiedad. Resulta difícil navegar tanto una jerarquía muy anidada con muchas clases como

una jerarquía muy plana con muy pocas clases con demasiada información codificada en propiedades. Encontrar el equilibrio adecuado no resulta fácil.

Hay varias reglas que ayudan a decidir cuándo introducir nuevas clases en una jerarquía:

- Las subclases de una clase generalmente (1) tienen propiedades adicionales que la superclase no tiene, o (2) tienen restricciones diferentes a las de la superclase, o (3) participan en relaciones diferentes que las superclases.

  En otras palabras, se introduce una nueva clase en la jerarquía generalmente sólo cuando hay algo que se puede decir sobre esta clase que no se puede decir sobre la superclase. En la práctica, cada subclase debería tener nuevas propiedades, o nuevos valores para esas propiedades, o sobrescribir algunas restricciones para las propiedades heredadas.

- Sin embargo, a veces puede resultar útil crear nuevas clases, incluso si no introducen ninguna propiedad nueva. Por ejemplo, una ontología que trata sobre un sistema electrónico de registros médicos puede incluir una clasificación de las diversas enfermedades. Esta clasificación puede ser sólo eso, una jerarquía de términos, sin propiedades (o con el mismo conjunto de propiedades). En ese caso, sigue siendo útil organizar los términos en una jerarquía en lugar de tener una lista plana porque (1) permitirá una exploración y navegación más fácil y (2) permitirá al médico elegir fácilmente un nivel de generalidad del término que sea apropiado para la situación.

- Otra razón para introducir nuevas clases sin nuevas propiedades es para modelar conceptos entre los cuales los expertos del dominio comúnmente hacen distinciones, a pesar que se puede haber decidido no modelar tal distinción.

- Por último, no se deben crear subclases de una clase para cada restricción adicional.

En el caso práctico, algunas de las clases que se definieron fueron: `Libro`, `Imagen`, `Video`, `Articulo`, `Servicio`, `ServicioLibros`, `ServicioImagenes`, `ServicioVideos`, `ServicioNoticias`, `Parametro`, `Autor`, `Fuente` (las clases `ServicioLibros`, `ServicioImagenes`, `ServicioVideos` y `ServicioNoticias` representan las fuentes de donde se obtendrá la información correspondiente). Se establecieron las clases `ServicioLibros`, `ServicioImagenes`, `ServicioVideos` y `ServicioNoticias` como subclases de `Servicio`. Por ejemplo, entre las clases `Servicio` y `Parametro` se definió una relación llamada `tiene` (un `Servicio tiene Parametros`).

Definir las clases y sus relaciones	
Objetivo	
Esta actividad busca obtener una lista con las clases de la ontología organizadas en una estructura jerárquica del tipo superclase/subclase, y sus relaciones	
Entrada	*Salida*
Salida de la actividad "*Enumerar los términos importantes de la ontología*" correspondiente al proceso "*Modelado del dominio*" (el conjunto de términos relevantes para el dominio y alcance de la ontología que se busca modelar)	Una jerarquía taxonómica con las clases de la ontología y sus relaciones
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez definidos los términos más importantes de la ontología	

Tabla 5.2.4.1 – Actividad "*Definir las clases y sus relaciones*"

5.2.5 Definir las propiedades de las clases

Las clases por sí solas no proporcionan información suficiente para responder las preguntas de la actividad 5.2.1 ("*Determinar el dominio y alcance de la ontología*"). Una vez que se han definido algunas clases, se debe describir la estructura interna de los conceptos que las mismas representan.

Una vez seleccionadas las clases a partir de la lista de términos creada en la actividad 5.2.3 ("*Enumerar los términos importantes de la ontología*"), es probable que la mayor parte de los términos restantes sean propiedades de estas clases.

En el caso práctico, algunas de las propiedades que se definieron para la clase `Libro` fueron: `titulo`, `isbn`, `idioma`, `precio`, `editorial`, `url`.

Definir las propiedades de las clases	
Objetivo	
Esta actividad busca obtener una lista con las propiedades de las clases de la ontología, las cuales al igual que las clases, pueden organizarse en una estructura jerárquica del tipo superpropiedad/subpropiedad	
Entrada	*Salida*
Salida de la actividad "*Enumerar los términos importantes de la ontología*" correspondiente al proceso "*Modelado del dominio*" (conjunto de términos relevantes para el dominio y alcance de la ontología que se busca modelar)	Un conjunto con las propiedades de las clases de la ontología. Estas propiedades también pueden organizarse en una estructura jerárquica
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez definidos los términos más importantes de la ontología y obtenidos a partir de éstos las clases que los representan	

Tabla 5.2.5.1 – Actividad "*Definir las propiedades de las clases*"

5.2.6 Definir las restricciones sobre las propiedades

Una restricción describe una clase anónima (una clase sin nombre). La clase anónima contiene todos los individuos que satisfacen la restricción, es decir, todos los individuos que tienen las relaciones necesarias para ser un miembro de la clase. De esta forma, las restricciones se emplean en las descripciones de clases OWL para especificar superclases anónimas de la clase que se está describiendo.

Ejemplos de restricciones:

- La clase de individuos que tiene al menos una relación `P`.
- La clase de individuos que tiene al menos una relación `P` con los miembros de la clase `A`.
- La clase de individuos que sólo tiene relaciones `P` con individuos que son `A`.
- La clase de individuos que tiene más de 3 relaciones `P`.

Definir las restricciones sobre las propiedades	
Objetivo	
Esta actividad busca refinar las propiedades obtenidas en la actividad "*Definir las propiedades de las clases*", estableciendo restricciones que se pueden aplicar a las mismas	
Entrada	*Salida*
Salida de la actividad "*Definir las propiedades de las clases*", correspondiente al proceso "*Modelado del dominio*" (el conjunto de propiedades que describen las clases de la ontología que se busca modelar)	Un conjunto actualizado con las restricciones que se pueden aplicar a las propiedades de las clases de la ontología
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez definidos los términos más importantes de la ontología de referencia y obtenidos a partir de éstos las clases que los representan junto con sus propiedades	

Tabla 5.2.6.1 – Actividad "*Definir las restricciones sobre las propiedades*"

En el caso práctico, debido a los objetivos del mashup semántico, no hubo necesidad de definir restricciones adicionales.

5.2.7 Crear instancias (individuos) de las distintas clases

La última actividad, al igual que la actividad 5.2.2 ("*Reutilizar ontologías existentes*"), es opcional, y consiste en crear las instancias individuales que hicieran falta para las clases de la jerarquía. Esta actividad es opcional porque no siempre es necesario que una ontología tenga individuos pertenecientes a las clases que tiene definidas.

En el caso práctico se definieron algunos individuos para las clases `ServicioImagenes`, `ServicioNoticias`, `ServicioLibros`, `ServicioVideos` y `Parametro`, los cuales actúan como valores iniciales para poder realizar luego la búsqueda en las distintas fuentes de información (representan los parámetros para realizar las búsquedas).

Definir una instancia individual de una clase requiere:

- la elección de una clase
- la creación de una instancia individual de esa clase
- completar los valores de las propiedades

Por ejemplo, para la clase `ServicioImagenes` se crearon 2 instancias: una llamada `Flickr` y la otra `Picasa`, las cuales tienen los datos necesarios para poder acceder a esas fuentes de información.

Crear instancias (individuos) de las distintas clases	
Objetivo	
Esta actividad busca crear las instancias necesarias para las clases obtenidas en la actividad "*Definir las clases y sus relaciones*"	
Entrada	*Salida*
Salida de las actividades "*Definir las clases y sus relaciones*", "*Definir las propiedades de las clases*" y "*Definir las restricciones sobre las propiedades*", correspondiente al proceso "*Modelado del dominio*"	Un conjunto de instancias correspondientes a las clases de la ontología que se está diseñando
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez definidas las clases, propiedades y restricciones que se puedan establecer sobre las mismas	

Tabla 5.2.7.1 – Actividad "*Crear instancias (individuos) de las distintas clases*"

5.3 Mediación de ontologías

En la Web Semántica toda la información está organizada en ontologías. Por lo tanto, el mashup semántico al realizar la mediación de las distintas ontologías con las que trabaja, los tipos de heterogeneidad que se presentarán en mucho mayor medida serán los debidos a las discrepancias terminológicas y conceptuales (semánticas), y en mucho menor medida los debidos a discrepancias sintácticas (en el caso que las distintas ontologías se expresen en un lenguaje distinto al OWL). Este proceso se muestra en la figura 5.3.1.

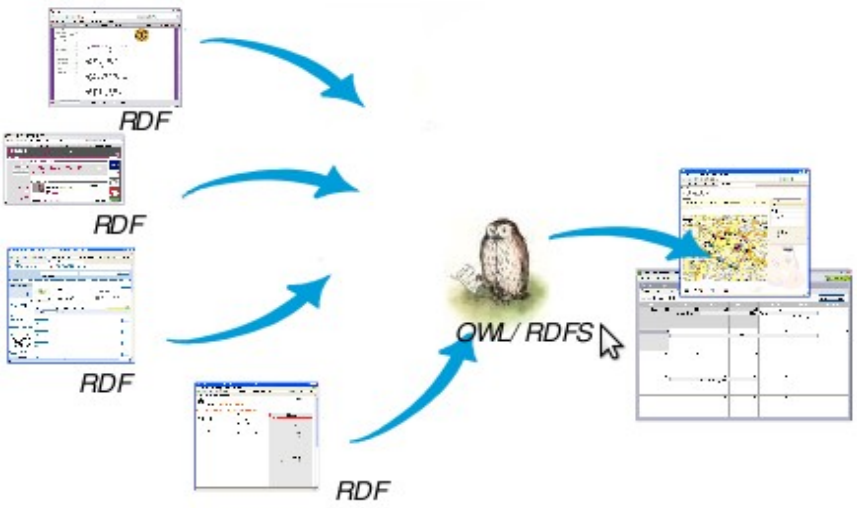

Figura 5.3.1 – Mediación en la Web Semántica

Sin embargo, en la Web 1.0 y 2.0 la información está organizada de manera tal que sólo resulta comprensible a las personas (por ejemplo, información organizada en documentos HTML, XML, etc). Como se dijo antes, la información organizada de esta forma no transmite nada útil sobre los recursos subyacentes.

Un mashup semántico trabaja con información estructurada en ontologías, por lo que se deberán traducir cada una de las representaciones que emplean las distintas fuentes en su correspondiente representación ontológica, para luego poder realizar la mediación entre las mismas. Este proceso es lo que se muestra en la figura 5.3.2.

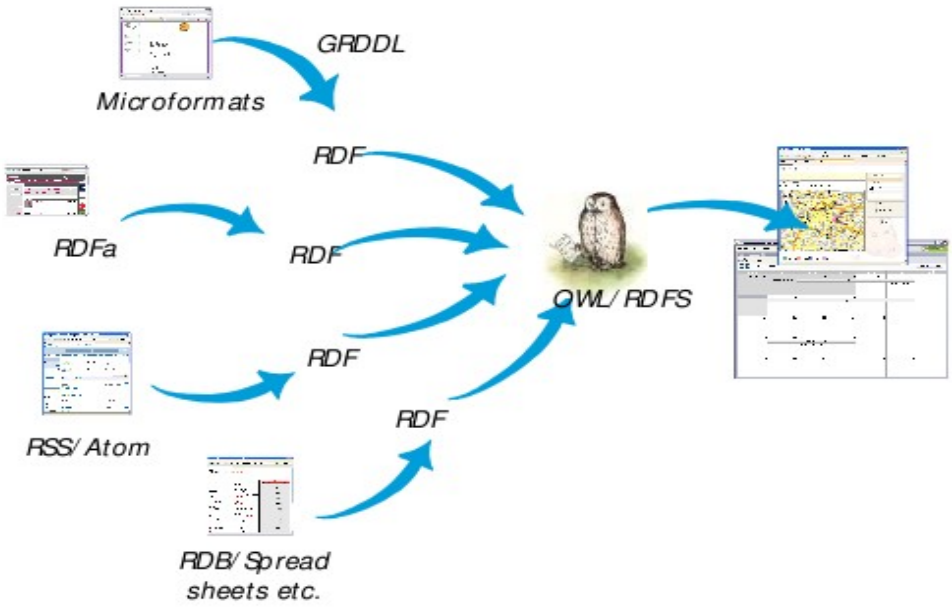

Figura 5.3.2 – Mediación en la Web 1.0/2.0

En este caso, antes de resolver los problemas por las discrepancias terminológicas y conceptuales, primero se tendrán que resolver las discrepancias sintácticas debido al empleo de distintos lenguajes para organizar la información.

Por ejemplo, en el caso práctico, las 6 fuentes con las que trabaja el mashup (YouTube, los servicios de noticias de Google y Yahoo, Amazon, Flickr y Picasa) tienen organizada su información en XML.

Los métodos que se utilicen para realizar el mapeo de las ontologías dependerán, entre otras cosas, de la complejidad de las mismas y el dominio de información que abarquen.

Este proceso estará compuesto por la actividad: *"Mapear ontologías"*.

5.3.1 Mapear ontologías

Mapear ontologías	
Objetivo	
Esta actividad busca mapear las distintas ontologías con las cuales trabaja el mashup semántico tomando la ontología de referencia obtenida en el proceso 5.2 ("*Modelado del dominio*")	
Entrada	*Salida*
La información de las distintas fuentes con las cuales trabajará el mashup semántico, más la ontología que se tomará como referencia para realizar la mediación (es la diseñada en el proceso "*Modelado del dominio*")	La información de las distintas fuentes integrada según la ontología de referencia
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez desarrollada la ontología de referencia	

Tabla 5.3.1.1 – Actividad "*Mapear ontologías*"

5.4 Acceso a las fuentes de información

Una vez que se tienen definidos los mapeos entre las ontologías de las distintas fuentes, el mashup semántico debe extraer la información proporcionada por cada una para combinarla y presentarla en otras formas, para mejorar la presentación visual de la misma o ambas cosas.

Los mapeos entre las distintas ontologías sirven para que:

- Cuando el mashup semántico lea la información de 2 fuentes similares (Flickr y Picasa por ejemplo), sepa qué información de una fuente se corresponde con qué información de la otra.

- Cuando el mashup semántico lea la información de 2 fuentes distintas (Flickr y el servicio de noticias de Google por ejemplo), sepa combinar la información de una fuente con la información de la otra.

Según lo explicado, este proceso estará compuesto por la actividad: *"Acceder a las fuentes de información"*.

5.4.1 Acceder a las fuentes de información

Acceder a las fuentes de información	
Objetivo	
Esta actividad busca obtener la información de las distintas fuentes con las cuales trabaja el mashup semántico. La información que se obtenga de cada una de las fuentes dependerá de los objetivos del mismo	
Entrada	*Salida*
Las fuentes de información sobre las cuales trabajará el mashup semántico y la ontología de referencia, la cual tiene los mapeos entre las ontologías de las distintas fuentes	La información de las distintas fuentes organizada según la ontología de referencia
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Esta actividad se debe realizar una vez definidos los mapeos entre las ontologías de las distintas fuentes de información	

Tabla 5.4.1.1 – Actividad "*Acceder a las fuentes de información*"

5.5 Diseño de la presentación

El objetivo de este proceso es complementar el diseño conceptual obtenido de los procesos anteriores con los detalles necesarios para su implementación. En principio, sería posible generar una implementación a partir del diseño conceptual, pero esto no es realista por varias razones [De

Troyer *et al.*, 2008]. Por ejemplo, la Web está muy orientada a lo visual, y los estándares para presentación se han vuelto muy altos en los últimos años. Los sistemas Web profesionales necesitan tener una apariencia profesional, y los diseñadores gráficos, por lo general, están involucrados para lograr esto. Además, un mashup (semántico o 2.0) no requiere de muchos clics, pero, por otro lado, demasiada información en una sola página sobrecargará la misma y disminuirá la usabilidad.

Un mashup (semántico o 2.0) por lo general no tiene mucha interacción con el usuario. Su función consiste en proporcionar un acceso rápido a la información, proveniente de varias fuentes, sin toda la interacción que implica navegar por páginas Web. Por lo general, esta interacción consta de 2 partes [Feiler, 2008]:

1. Una página de inicio que permite al usuario especificar la información a mostrar

2. Una página donde se muestra la información

A veces se puede omitir la página de inicio en caso que el propio mashup semántico sea quien decida qué información recuperar (por ejemplo, información de las últimas 24 horas).

Por lo tanto, la información se debe agrupar en páginas de tal manera que se logre un buen equilibrio entre cantidad y número de clics necesarios para llegar a la misma.

Durante el diseño de la presentación, se define la apariencia del mashup semántico, así como el diseño de las páginas (es decir, la posición de los elementos dentro de las páginas).

5.6 Implementación

El objetivo de este proceso consiste en desarrollar las páginas de inicio del mashup semántico, a fin de permitirle al usuario ingresar datos, y la página en la cual se muestra la información.

Para la implementación se cuenta con una serie de recursos, como el empleo de formularios, grillas, botones, listas, tablas, menús, secciones, imágenes, controles, etc. Para especificar el estilo, se pueden emplear Hojas de Estilo en Cascada (CSS), las cuales permiten la especificación de estilos para cualquier elemento en particular y tienen el suficiente poder expresivo para describir los estilos más comunes encontrados en los sistemas Web.

5.7 Pruebas

Las pruebas juegan un papel muy importante en cualquier proceso de desarrollo. Sin embargo, en la mayoría de los casos las pruebas y las evaluaciones son un aspecto descuidado del desarrollo. Muchos desarrolladores prueban el sistema sólo después que aparecen fallas en el mismo o después de evidenciar las limitaciones, lo cual se conoce como “pruebas retroactivas”, mientras que lo que se quiere son “pruebas proactivas” en las distintas etapas del ciclo de vida del desarrollo. Las ventajas de las pruebas proactivas incluyen la garantía de buen funcionamiento y niveles de

rendimiento, se evitan correcciones de carácter retroactivo que pueden resultar caras, rendimiento óptimo, menor riesgo, etc [Murugesan, 2008].

Las pruebas no se deberían hacer únicamente al final del proceso de desarrollo. Los desarrolladores y los gerentes deben tener una visión más amplia y seguir un enfoque más holístico: desde el diseño y durante todo el camino hasta la implementación y el mantenimiento.

Las pruebas que se pueden realizar pueden agruparse en las siguientes categorías:

- Compatibilidad del navegador
- Visualización de la página
- Usabilidad
- Disponilibidad
- Internacionalización
- Rendimiento

5.8 Conclusiones

Las aplicaciones sobre la Web Semántica son aplicaciones Web. Por lo tanto, las metodologías de la Ingeniería de Software y los patrones de diseño conocidos para el desarrollo de aplicaciones Web siguen siendo aplicables. Sin embargo, si se tiene en cuenta que las aplicaciones sobre la Web Semántica utilizan ontologías y modelos RDF, se deben hacer cambios a estas metodologías.

En el caso de un mashup semántico, su modelo del dominio es una ontología, la cual permite establecer las correspondencias (mapeos) entre los distintos conceptos y relaciones entre las ontologías de las fuentes de información con las cuales trabaja.

Un mashup semántico al realizar la mediación de estas ontologías, deberá resolver las discrepancias que se presenten debido a heterogeneidades terminológicas y conceptuales (semánticas), y en mucho menor medida las discrepancias sintácticas (en el caso que las distintas ontologías se expresen en un lenguaje distinto al OWL). Pero como en la Web 1.0 y 2.0 la información está organizada de manera que sólo resulta comprensible a las personas (HTML, XML, etc), primero habrá que traducir cada una de las representaciones que empleen las fuentes en su correspondiente representación ontológica, para luego poder realizar la mediación entre las mismas. Es decir, antes de resolver las discrepancias terminológicas y conceptuales, primero se tendrán que resolver las discrepancias sintácticas debido al empleo de distintos lenguajes para organizar la información.

En todos estos casos, la forma de resolver todas estas discrepancias es mediante la ontología que captura el modelo del dominio del mashup semántico, la cual actúa como ontología de referencia para poder integrar las distintas fuentes de información.

CAPÍTULO 6 - CASO DE ESTUDIO

En este capítulo se presenta el diseño de un mashup semántico tomando como base el ejemplo desarrollado en una serie de 6 artículos llamados “The ultimate mashup – Web services and the semantic Web” [Chase, 2007A], [Chase & Peterson, 2006], [Chase, 2007B], [Mitri & Chase, 2006], [Chase & Mitri, 2007], [Chase, 2007C], siguiendo la metodología propuesta en el Capítulo 5.

6.1 Definición de los objetivos del mashup semántico

Según lo explicado en la Sección 5.1, en la tabla 6.1.1 se puede ver la descripción de la actividad “*Definir los objetivos del mashup semántico*”:

<table>
<tr><th colspan="2">Definir los objetivos del mashup semántico</th></tr>
<tr><td colspan="2">Objetivo</td></tr>
<tr><td colspan="2">Establecer los objetivos que debe cumplir el mashup semántico</td></tr>
<tr><td>Entrada</td><td>Salida</td></tr>
<tr><td>Para un determinado concepto, el usuario podrá buscar qué noticias hay relacionadas con el mismo (en los servicios de noticias de Google y Yahoo), qué libros lo tratan (en Amazon), qué videos hay sobre el mismo (en YouTube) y qué imágenes o fotos hay sobre el mismo (en Flickr y Picasa).

El usuario podrá intercambiar las fuentes de información a voluntad (podrá decidir cuál(es) fuente(s) usar). De esta forma, el mashup semántico podrá tratar uniformemente a sus fuentes de información.</td><td>Una declaración con los objetivos del mashup semántico (ver a continuación de esta tabla)</td></tr>
<tr><td colspan="2">Participantes</td></tr>
<tr><td colspan="2">Los desarrolladores del mashup semántico</td></tr>
<tr><td colspan="2">Cuándo</td></tr>
<tr><td colspan="2">Al comienzo, en el proceso “Definición de los objetivos del mashup semántico”</td></tr>
</table>

Tabla 6.1.1 – Actividad “*Definir los objetivos del mashup semántico*”

Objetivos del mashup semántico

- Dado un determinado concepto a buscar, se deberá buscar determinada información sobre el mismo en los servicios de noticias de Google y Yahoo, en Amazon, en YouTube, en Flickr y en Picasa.

- Al interactuar con el mashup semántico, el usuario podrá especificar la(s) fuente(s) que necesite.

- La información que se necesita recuperar de cada fuente es la siguiente:

    - Sobre un libro se necesita el título, ISBN, autor (nombre y/o usuario), editorial, precio, idioma, foto y página Web.
    - Sobre un video se necesita el título, duración, fecha de publicación, descripción, foto, autor (nombre y/o usuario) y URL (para mostrarlo).
    - Sobre una imagen se necesita el título, tamaño, descripción, URLs (una con la imagen en tamaño chico y la otra en tamaño grande) y el autor (nombre y/o usuario).
    - Sobre un artículo de noticias se necesita el título, fecha de publicación, resumen, fuente (nombre) y URL con información detallada.

- La presentación de los resultados de una búsqueda dependerá del tipo de información en particular. Así, al mostrar los libros que traten sobre un determinado concepto, éstos se mostrarán en una lista con la imágen de cada uno, y al lado el título, autor, editorial, ISBN, idioma y precio; al mostrar las imágenes, las mismas estarán organizadas en filas donde se mostrará la imagen, su título y el tamaño; al mostrar los videos, éstos se mostrarán en una lista con la imágen de cada uno, y al lado el título, autor, fecha en que fue subido y la duración; y al mostrar las noticias, éstas se organizarán en una lista donde se mostrará el título, la fuente, fecha y la descripción.

- El formato de presentación de los resultados dependerá de cada fuente, pero en todos los casos, características como el tipo de letra, color, tamaño, etc, serán iguales para todos.

- El acceso a la información de las fuentes se realizará mediante servicios Web del tipo REST.

6.2 Modelado del dominio

Como se dijo antes, para un mashup semántico su modelo del dominio es una ontología, la cual permitirá definir los mapeos entre las ontologías de las distintas fuentes para poder integrar su información. Para modelar la ontología se emplea Protégé[66] 4.1, herramienta de código abierto que permite editar ontologías.

66 http://protege.stanford.edu/

6.2.1 Determinar el dominio y alcance de la ontología

Según lo explicado en la Sección 5.2.1, en la tabla 6.2.1.1 se puede ver la descripción de la actividad "*Determinar el dominio y alcance de la ontología*":

Determinar el dominio y alcance de la ontología	
Objetivo	
Determinar el dominio y alcance de la ontología	
Entrada	*Salida*
La declaración con los objetivos del mashup semántico (ver tabla 6.1.1)	La ontología debe permitir responder a preguntas como: • ¿Qué libros, videos, artículos de noticias y/o imágenes hay con información relacionada con un determinado concepto? • ¿Cuál es la información detallada de los libros, videos artículos de noticias y/o imágenes que tienen información sobre un determinado concepto? No es necesario poder responder a preguntas como: • Dado un autor, ¿cuáles son sus libros que tratan sobre un determinado concepto? De la misma forma para los videos, imágenes y artículos de noticias. Hay que tener en cuenta que los usuarios que empleen el mashup semántico no necesitan conocimientos técnicos para buscar información en las distintas fuentes.
Participantes	
Los desarrolladores del mashup semántico	
Cuándo	
Una vez definidos los objetivos del mashup semántico	

Tabla 6.2.1.1 – Actividad "*Determinar el dominio y alcance de la ontología*"

6.2.2 Reutilizar ontologías existentes

Para este ejemplo se diseña la ontología desde cero, sin buscar otras previas.

6.2.3 Enumerar los términos importantes de la ontología

Según lo explicado en la Sección 5.2.3, en la tabla 6.2.3.1 se puede ver la descripción de la actividad "*Enumerar los términos importantes de la ontología*":

Enumerar los términos importantes de la ontología	
Objetivo	
Obtener una lista con los términos más importantes para la ontología, concentrándose en lo importante y dejando de lado los detalles innecesarios	
Entrada	*Salida*
Salida de la actividad "*Determinar el dominio y alcance de la ontología*" (ver tabla 6.2.1.1)	Libro, ISBN, Título, Idioma, Editorial, Precio, Nombre del autor, Usuario del autor, Imagen, URL, Video, Fecha, Descripción, Duración, Tamaño, Noticia, Resumen, Detalle, Fuente, Servicio, Parámetro
Participantes	
Los desarrolladores del mashup semántico	
Cuándo	
Una vez definidos el dominio y alcance de la ontología que se busca modelar	

Tabla 6.2.3.1 – Actividad "*Enumerar los términos importantes de la ontología*"

6.2.4 Definir las clases y sus relaciones

Según lo explicado en la Sección 5.2.4, en la tabla 6.2.4.1 se puede ver la descripción de la actividad "*Definir las clases y sus relaciones*":

Definir las clases y sus relaciones	
Objetivo	
Obtener una lista con las clases de la ontología y sus relaciones	
Entrada	*Salida*
Salida de la actividad "*Enumerar los términos importantes de la ontología*" (ver tabla 6.2.3.1)	Clases: `Libro`, `Imagen`, `Video`, `Articulo`, `Servicio`, `Parametro`, `Autor`, `Fuente`, `ServicioLibros`, `ServicioImagenes`, `ServicioVideos`, `ServicioNoticias`. Relaciones: `tiene`, `esEscritoPor`, `esInformadoPor`, `esPublicadaPor`, `esSubidoPor`
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Una vez definidos los términos más importantes de la ontología	

Tabla 6.2.4.1 – Actividad "*Definir las clases y sus relaciones*"

Las clases `ServicioLibros`, `ServicioImagenes`, `ServicioVideos` y `ServicioNoticias` se definen como subclases de `Servicio`.

Entre las clases `Servicio` y `Parametro` se define la relación `tiene` (un `Servicio tiene Parametros`).

Entre las clases `Libro` y `Autor` se define la relación `esEscritoPor` (un `Libro` es escrito por un `Autor`). Para este ejemplo, no hay necesidad de definir la relación inversa `escribe` (un `Autor escribe Libros`) ya que no existe el requerimiento de obtener los libros de un determinado autor.

Con los mismos criterios, entre las clases `Articulo` y `Fuente` se define la relación `esInformadoPor`, entre `Imagen` y `Autor` la relación `esPublicadaPor`, y entre `Video` y `Autor` la relación `esSubidoPor`.

En la figura 6.2.4.1 se pueden ver estas clases y las relaciones entre las mismas.

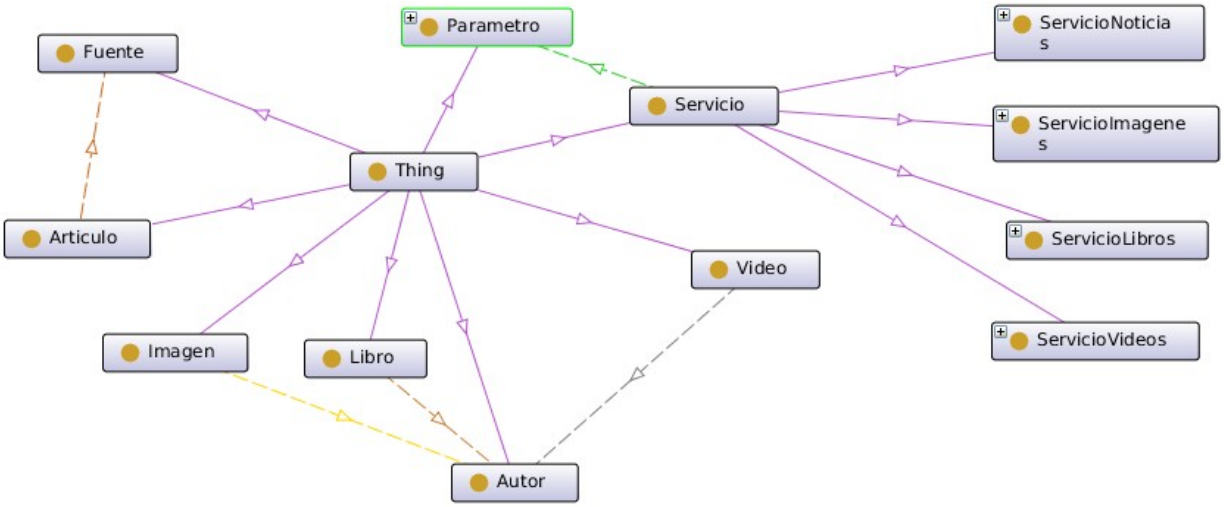

Figura 6.2.4.1 – Clases ontológicas y las relaciones entre las mismas

6.2.5. Definir las propiedades de las clases

Según lo explicado en la Sección 5.2.5, en la tabla 6.2.5.1 se puede ver la descripción de la actividad "*Definir las propiedades de las clases*":

Definir las propiedades de las clases	
Objetivo	
Obtener una lista con las propiedades de las clases de la ontología, las cuales pueden organizarse en una estructura jerárquica del tipo superpropiedad/subpropiedad	
Entrada	*Salida*
Salida de la actividad "*Enumerar los términos importantes de la ontología*" (ver tabla 6.2.3.1)	En las tablas 6.2.5.2, 6.2.5.3, 6.2.5.4, 6.2.5.5, 6.2.5.6, 6.2.5.7, 6.2.5.8 y 6.2.5.9 pueden verse las propiedades para las clases `Libro`, `Imagen`, `Video`, `Articulo`, `Autor`, `Fuente`, `Servicio` y `Parametro` respectivamente
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Una vez definidos los términos más importantes de la ontología y obtenidos a partir de éstos las clases que los representan	

Tabla 6.2.5.1 – Actividad "*Definir las propiedades de las clases*"

Libro		
Propiedad	**Dominio**	**Rango**
tituloLibro	Libro	string
isbn	Libro	string
idiomaLibro	Libro	string
precioLibro	Libro	double
editorialLibro	Libro	string
urlLibro	Libro	anyURI
urlLibroInfo	Libro	anyURI

Tabla 6.2.5.2 – Propiedades para la clase ontológica `Libro`

La propiedad `urlLibro` se emplea para representar la imagen del libro, es decir, el valor de esta propiedad contiene una URL con la imagen.

Imagen		
Propiedad	**Dominio**	**Rango**
tituloImagen	Imagen	string
descripcionImagen	Imagen	string
tamanioImagen	Imagen	string
urlImagen	Imagen	anyURI
urlImagenGrande	Imagen	anyURI

Tabla 6.2.5.3 – Propiedades para la clase ontológica `Imagen`

Las propiedades `urlImagen` y `urlImagenGrande` tienen el mismo significado que para el caso de los libros.

Video		
Propiedad	**Dominio**	**Rango**
tituloVideo	Video	string
descripcionVideo	Video	string
urlVideo	Video	anyURI
urlImagenVideo	Video	anyURI
fechaVideo	Video	date
duracionVideo	Video	nonNegativeInteger

Tabla 6.2.5.4 – Propiedades para la clase ontológica `Video`

Las propiedades `urlVideo` y `urlImagenVideo` tienen las direcciones URL para reproducir el video y la imagen del mismo respectivamente.

Articulo		
Propiedad	**Dominio**	**Rango**
`tituloArticulo`	`Articulo`	`string`
`resumenArticulo`	`Articulo`	`string`
`urlArticulo`	`Articulo`	`anyURI`
`fechaArticulo`	`Articulo`	`date`

Tabla 6.2.5.5 – Propiedades para la clase ontológica `Articulo`

Autor		
Propiedad	**Dominio**	**Rango**
`nombreAutor`	`Autor`	`string`
`usuario`	`Autor`	`string`

Tabla 6.2.5.6 – Propiedades para la clase ontológica `Autor`

Fuente		
Propiedad	**Dominio**	**Rango**
`nombreFuente`	`Fuente`	`string`

Tabla 6.2.5.7 – Propiedades para la clase ontológica `Fuente`

Servicio		
Propiedad	**Dominio**	**Rango**
`nombreServicio`	`Servicio`	`string`
`urlServicio`	`Servicio`	`anyURI`
`xslt`	`Servicio`	`string`

Tabla 6.2.5.8 – Propiedades para la clase ontológica `Servicio`

Cada servicio está formado por una dirección URL base, más una serie de parámetros, los cuales dependen de cada servicio. La propiedad `urlServicio` representa esta dirección base. También, cada servicio es responsable de obtener su propia información, y como cada fuente la organiza de distinta forma, empleando su propia estructura, en la implementación cada servicio será responsable de convertir su representación en una forma ontológica. Esta conversión se hace mediante una

transformación XSL, la cual se guarda en la propiedad `xslt`.

`Parametro`		
Propiedad	**Dominio**	**Rango**
`nombreParametro`	`Parametro`	`string`
`valorParametro`	`Parametro`	`string`

Tabla 6.2.5.9 – Propiedades para la clase ontológica `Parametro`

Debido a los objetivos y alcance del mashup semántico, no hay necesidad de definir propiedades inversas, funcionales, funcionales inversas, transitivas, simétricas, antisimétricas, reflexivas ni irreflexivas.

6.2.6 Definir las restricciones sobre las propiedades

Por las mismas razones anteriores, no es necesario definir ningún tipo de restricción adicional.

6.2.7 Crear instancias (individuos) de las distintas clases

Como se dijo antes, estas instancias funcionan como "valores iniciales" para el mashup semántico, permitiéndole acceder a las distintas fuentes de información. En la tabla 6.2.7.1 se puede ver la descripción de la actividad "*Crear instancias (individuos) de las distintas clases*":

Crear instancias (individuos) de las distintas clases	
Objetivo	
Crear las instancias (necesarias) para las clases	
Entrada	*Salida*
Salida de las actividades "*Definir las clases y sus relaciones*" y "*Definir las propiedades de las clases*" (ver tablas 6.2.4.1 y 6.2.5.1)	Se crearon instancias para las clases `ServicioImagenes`, `ServicioNoticias`, `ServicioLibros`, `ServicioVideos` y `Parametro`
Participantes	
Desarrolladores del mashup semántico	
Cuándo	
Una vez definidas las clases y propiedades	

Tabla 6.2.7.1 – Actividad "*Crear instancias (individuos) de las distintas clases*"

En la tabla 6.2.7.2 se pueden ver los nombres de los individuos que se crean para cada una de las subclases de `Servicio`:

Clase	**Individuo**	**Fuente de datos**
`ServicioImagenes`	`Flickr1`	Flickr
	`Picasa`	Picasa
`ServicioLibros`	`Amazon`	Amazon
`ServicioNoticias`	`ServicioYahoo`	Yahoo
	`Google`	Google
`ServicioVideos`	`ServicioYouTube`	YouTube

Tabla 6.2.7.2 – Individuos para las subclases de `Servicio`

Para determinar la URL base de cada uno de los servicios, junto con sus respectivos parámetros, se puede consultar la documentación de la API correspondiente:

- Flickr[67]
- Picasa[68]
- Amazon[69]
- Yahoo[70]
- Google[71]
- YouTube[72]

A continuación se especifican las direcciones URL base y parámetros de cada uno de los servicios:

- Flickr

    - URL base: `http://api.flickr.com/services/rest/`

    - *Parámetro*: `method`
    - *Valor*: `flickr.photos.search`
    - *Descripción*: devuelve una lista con las que cumplan con algún criterio. Sólo se devuelven las fotos visibles al usuario que hace el pedido. Para que se devuelvan fotos privadas o semi-privadas, quien hace el pedido debe estar autenticado con permisos de lectura, además de tener permiso para ver las fotos. Los pedidos sin autenticar sólo devolverán fotos públicas.

67 http://www.flickr.com/services/api/
68 http://code.google.com/apis/picasaweb/docs/2.0/developers_guide_protocol.html
69 http://docs.amazonwebservices.com/AWSEcommerceService/2005-10-05/
http://apisigning.com/
70 http://developer.yahoo.com/search/news/V1/newsSearch.html
71 http://code.google.com/apis/customsearch/v1/overview.html
72 http://code.google.com/apis/youtube/2.0/developers_guide_protocol.html

- *Parámetro*: `api_key`
- *Valor*: `b355fbc2ca38c86ddee4d19ec3574ddb`
- *Descripción*: para usar la API de Flickr se necesita tener una clave, la cual es usada por Flickr para hacer el seguimiento del uso de la API. Actualmente, el uso comercial de la API sólo está permitido con un permiso previo.

- *Parámetro*: `extras`
- *Valor*: `description,url_t,url_l,owner_name`
- *Descripción*: lista delimitada por comas de información extra para buscar para cada registro devuelto. La lista de valores seleccionados tienen los siguientes significados: `description` (descripción de la foto), `url_t` (URL de la imagen a mostrar en la página de resultados), `url_l` (URL de la imagen grande a mostrar cuando se selecciona la foto), `owner_name` (autor de la foto)

- *Parámetro*: `per_page`
- *Valor*: `10`
- *Descripción*: número de fotos a devolver por página. Si se omite este argumento, el valor por defecto es 100. El valor máximo permitido es de 500.

- *Parámetro*: `sort`
- *Valor*: `date-taken-desc`
- *Descripción*: criterio para ordenar los fotos devueltas. El valor por defecto es `date-posted-desc` (por fecha de publicación, de la más reciente a la más antigua). Los valores posibles son: `date-posted-asc` (por fecha de publicación, de la más antigua a la más reciente), `date-posted-desc`, `date-taken-asc` (por fecha en que fue tomada la foto, de la más antigua a la más reciente), `date-taken-desc` (por fecha en que fue tomada la foto, de la más reciente a la más antigua), `interestingness-desc` (por cuán interesante resulta la foto, de la más interesante a la menos), `interestingness-asc` (por cuán interesante resulta la foto, de la menos interesante a la más) y `relevance` (relevancia).

- *Parámetro*: `text`
- *Valor*: este parámetro toma por valor el texto que se especifique para hacer la búsqueda.
- *Descripción*: implica que se devuelvan aquellas fotos cuyo título, descripción o etiquetas contengan este texto. Se pueden excluir los resultados que coinciden con un determinado término anteponiendo un caracter "–".

- Ejemplo:

  http://api.flickr.com/services/rest/?
  method=flickr.photos.search&api_key=b355fbc2ca38c86ddee4d19ec3574ddb&extras
  =description,url_t,url_l,owner_name&text=Egipto

    - Otros parámetros que no se tuvieron en cuenta para Flickr:

        - `tags` (lista de etiquetas delimitada por comas. Se devuelven las fotos que tengan una o más de estas etiquetas. Se pueden excluir los resultados que coinciden con un término anteponiendo un caracter "–").

        - `min_upload_date` (fecha mínima de subida. Se devuelven las fotos con una fecha de subida mayor o igual a este valor. La fecha puede expresarse mediante un timestamp de Unix o un tipo datetime de mysql).

        - `max_upload_date` (fecha máxima de subida. Se devuelven las fotos con una fecha de subida inferior o igual a este valor. Se puede expresar igual que min_upload_date).

        - `min_taken_date` (fecha mínima en que se tomó la foto. Se devuelven las fotos tomadas con una fecha igual o superior a este valor. Se puede expresar igual que min_upload_date).

        - `max_taken_date` (fecha máxima en que se tomó la foto. Se devuelven las fotos tomadas con una fecha inferior o igual a este valor. Se puede expresar igual que min_upload_date).

- Picasa

    - URL base: `https://picasaweb.google.com/data/feed/api/all`

    - *Parámetro*: `max-results`
    - *Valor*: `10`
    - *Descripción*: número de fotos a devolver.

    - *Parámetro*: `q`
    - *Valor*: este parámetro toma por valor el texto que se especifique para hacer la búsqueda.
    - *Descripción*: especifica el criterio con el cual hacer la búsqueda.

    - Ejemplo:

      https://picasaweb.google.com/data/feed/api/all?q=Egipto&max-results=10

- Amazon

    - URL base: `http://free.apisigning.com/onca/xml`

    - *Parámetro*: `Service`

- *Valor*: `AWSECommerceService`
- *Descripción*: en realidad este parámetro también forma parte de la dirección URL base.

- *Parámetro*: `AWSAccessKeyId`
- *Valor*: `AKIAINTKOBSCZF7HFPQA`
- *Descripción*: este parámetro se requiere en todos los pedidos que se hagan con esta API, por lo cual habrá que registrarse para obtener un ID de clave de acceso.

- *Parámetro*: `Operation`
- *Valor*: `ItemSearch`
- *Descripción*: permite buscar productos por un determinado índice o una combinación de índices. Este tipo de operación consiste de 3 pasos: (1) elegir dónde hacer la búsqueda (en libros, productos de electrónica, música, películas, etc), (2) especificar los parámetros de búsqueda y (3) establecer la salida deseada. Esta operación devuelve hasta 10 registros por consulta, y en caso de necesitar registros adicionales, se puede emplear el parámetro `ItemPage`.

- *Parámetro*: `SearchIndex`
- *Valor*: `Books`
- *Descripción*: especifica el índice según el cual realizar la búsqueda (libros en este caso). Hay algunos índices que en sí mismo son combinaciones, por ejemplo: "juguetes y juegos". También se puede especificar la palabra "`blended`" para hacer una búsqueda por todos los índices (libros, música, DVDs, juguetes, etc). Este parámetro corresponde al paso (1) del parámetro `ItemSearch`.

- *Parámetro*: `ResponseGroup`
- *Valor*: `Large`
- *Descripción*: especifica el grado de detalle de la información devuelta. Se pueden especificar distintos valores según el índice sobre el cual se realiza la búsqueda. Este parámetro corresponde al paso (3) del parámetro `ItemSearch`.

- *Parámetro*: `Keywords`
- *Valor*: este parámetro toma por valor el texto que se especifique para hacer la búsqueda.
- *Descripción*: especifica el criterio con el cual hacer la búsqueda. En este caso se busca en campos como título del producto, autor, artista, descripción, fabricante, etc. Este parámetro corresponde al paso (2) del parámetro `ItemSearch`.

- Ejemplo:

  http://free.apisigning.com/onca/xml?
  Service=AWSECommerceService&AWSAccessKeyId=AKIAINTKOBSCZF7HFPQ
  A&SearchIndex=Books&Operation=ItemSearch&ResponseGroup=Large&Keywords

=Egipto

- Yahoo

    - URL base:

    `http://search.yahooapis.com/NewsSearchService/V1/newsSearch`

    - *Parámetro*: `appid`
    - *Valor*: `mashupid`
    - *Descripción*: ID de la aplicación, el cual es una cadena que identifica unívocamente una aplicación. Si se tienen múltiples aplicaciones, se deben usar IDs diferentes para cada una.

    - *Parámetro*: `type`
    - *Valor*: `all`
    - *Descripción*: tipo de búsqueda a realizar. Los valores posibles son: `all` (valor por defecto. Devuelve los resultados con todos los términos de la cadena que se especifique en la consulta), `any` (devuelve los resultados con uno o más de los términos de la cadena de consulta) o `phrase` (devuelve los resultados que contienen los términos de la consulta en forma de frase)

    - *Parámetro*: `sort`
    - *Valor*: `date`
    - *Descripción*: ordena los artículos por relevancia -`rank`- o desde el más reciente al más antiguo -`date`-. El valor por defecto es `rank`.

    - *Parámetro*: `query`
    - *Valor*: este parámetro toma por valor el texto que se especifique para hacer la búsqueda.
    - *Descripción*: especifica el criterio con el cual hacer la búsqueda.

    - Otros parámetros que no se tuvieron en cuenta para Yahoo:

        - `results` (número de resultados a devolver. El valor por defecto es `10` y el máximo `50`).

        - `start` (el valor desde donde comenzar los resultados. El valor por defecto es `1`).

    - Ejemplo:

    http://search.yahooapis.com/NewsSearchService/V1/newsSearch?
    appid=mashupid&type=all&sort=date&query=Egipto

- Google

    - URL base: https://www.googleapis.com/customsearch/v1

    - *Parámetro*: key
    - *Valor*: AIzaSyAGKE4RbAcN4q8_AGA3AURUYHdJyOhkn6w
    - *Descripción*: esta API requiere una clave, la cual proporciona 100 consultas por día. En caso de necesitar más hay que hacer otro tipo de registro.

    - *Parámetro*: cx
    - *Valor*: 002866492335529227924:oeyohi7lk3s
    - *Descripción*: especifica el ID del motor de búsqueda a emplear

    - *Parámetro*: alt
    - *Valor*: atom
    - *Descripción*: la API puede devolver los resultados en uno de dos formatos posibles: JSON (valor predeterminado) o Atom.

    - *Parámetro*: num
    - *Valor*: 10
    - *Descripción*: especifica el número máximo de resultados a incluirse en el conjunto de resultados.

    - *Parámetro*: q
    - *Valor*: este parámetro toma por valor el texto que se especifique para hacer la búsqueda.
    - *Descripción*: especifica el criterio con el cual hacer la búsqueda.

    - Ejemplo:

      https://www.googleapis.com/customsearch/v1?key=AIzaSyCsUD-AwbMhG_omTsq09WdaXMkCiP2f198&cx=017576662512468239146:omuauf_lfve&alt=atom&num=10&q=Egypt

- YouTube

    - URL base: `http://gdata.youtube.com/feeds/api/videos`

    - *Parámetro*: `orderby`
    - *Valor*: `published`
    - *Descripción*: especifica el método a usar para ordenar las entradas en la respuesta al pedido de la API. Los valores posibles son: `relevance` (las entradas se ordenan por relevancia), `published` (las entradas se ordenan por fecha, de la más reciente a la más antigua. Es el valor por defecto), `viewCount` (las entradas según la cantidad

de visitas, de mayor a menor) o `rating` (las entradas se ordenan según su valoración, de mayor a menor).

- *Parámetro*: `v`
- *Valor*: `2`
- *Descripción*: especifica la versión de la API que debe usar YouTube para responder los pedidos. Si no se especifica una versión de la API, YouTube empleará la versión 1 (la versión actual es la 2). Establecer el valor del parámetro `v` en 2 permite a la aplicación acceder a funciones que no están disponibles en las versiones anteriores. Por ejemplo, los subtítulos sólo se admiten en la versión 2, la API soporta algunos parámetros diferentes en la versión 2 que en la 1, etc.

- *Parámetro*: `max-results`
- *Valor*: `10`
- *Descripción*: especifica el número máximo de resultados a incluirse en el conjunto de resultados. Este parámetro trabaja en conjunto con el parámetro `start-index` para determinar qué resultados devolver. Por ejemplo, para solicitar el segundo grupo de 10 resultados, los resultados del 11 al 20, se establece el parámetro `max-results` en 10 y el parámetro de `start-index` en 11. El valor por defecto es 25, y el máximo 50, pero se recomienda el valor 10 para mostrar una lista de videos.

- *Parámetro*: `q`
- *Valor*: este parámetro toma por valor el texto que se especifique para hacer la búsqueda.
- *Descripción*: especifica el criterio con el cual hacer la búsqueda. YouTube busca en todos los metadatos de un video que coincidan con este término. Los metadatos incluyen títulos, palabras clave, descripciones, nombres de los usuarios autores y las categorías.

- Otros parámetros que no se tuvieron en cuenta para YouTube:

    - `alt` (especifica el formato con el que se devuelven los resultados. Los valores posibles son: `atom` -valor por defecto-, `rss`, `json`, `json-in-script` y `jsonc`).

    - `start-index` (especifica el índice del primer resultado coincidente que se debe incluir en el conjunto de resultados. El primer resultado es 1, el segundo 2 y así sucesivamente. Este parámetro trabaja en conjunto con el parámetro `max-results`).

- Ejemplo:

  http://gdata.youtube.com/feeds/api/videos?orderby=published&v=2&max-results=10&q=Egipto

Para crear por ejemplo el individuo para la fuente de información Flickr, correspondiente a la clase `ServicioImagenes`, se le asigna un nombre cualquiera, en este caso `Flickr1`, se establece a `ServicioImagenes` como su tipo, y se establecen los siguientes valores para sus propiedades:

Propiedad	**Valor**
`urlServicio`	`http://api.flickr.com/services/rest/`
`nombreServicio`	`Flickr.com`

Tabla 6.2.7.3 – Creación del individuo `Flickr1`

Para asignarle a este servicio los parámetros especificados anteriormente, se crean individuos de la clase `Parametro`. De la misma forma que el caso anterior, a estos individuos se les da un nombre y se establecen los valores de sus propiedades. En el caso de un individuo del tipo `Parametro`, éste cuenta con 2 propiedades: `nombreParametro` y `valorParametro`. Así, para el parámetro `method` de Flickr, se crea un individuo del tipo `Parametro`, llamado `ParamMethod`, con los siguientes valores para las propiedades:

Propiedad	**Valor**
`nombreParametro`	`method`
`valorParametro`	`flickr.photos.search`

Tabla 6.2.7.4 – Creación de un individuo tipo `Parametro`

Una vez creados los individuos `Parametro`, mediante la propiedad `tiene` se asocian los individuos `Parametro` con el individuo `Flickr1`.

6.3 Mediación de ontologías

Como se dijo antes, las fuentes con las que trabaja el mashup semántico devuelven su información en XML, empleando cada una su propia estructura. Por lo tanto, además de las discrepancias terminológicas y semánticas, también se presentan las sintácticas.

Para explicar el enfoque adoptado para la mediación, se toma como ejemplo la consulta a YouTube. En la Sección 6.1 ("*Definición de los objetivos del mashup semántico*"), se especifica que sobre un video se necesita el título, duración, fecha de publicación, descripción, foto y URL (para mostrarlo), con lo cual se define la clase ontológica `Video` y las propiedades `tituloVideo`, `duracionVideo`, `fechaVideo`, `descripcionVideo`, `urlImagenVideo` y `urlVideo` respectivamente.

Como también se necesita conocer el autor (nombre y/o usuario) del video, se define la clase ontológica `Autor` y las propiedades `nombreAutor` y `usuario` respectivamente.

En la Sección 4.4.2 (“*Heterogeneidad semántica*”) se presenta la información que se obtiene al hacer la consulta a YouTube (información en XML). En este ejemplo se puede ver que la información de cada video se encuentra dentro de un elemento XML llamado `<entry>`, el cual se encuentra a su vez dentro del elemento raíz `<feed>`. En el elemento `<entry>` se encuentra detallada, en forma de atributos u otros subelementos, la información sobre el video.

En la tabla 6.3.1 se puede ver por cada propiedad ontológica del video su correspondiente ubicación en la estructura XML:

Propiedad ontológica	**Ubicación en la estructura XML**
`tituloVideo`	`/feed/entry/group/title`
`duracionVideo`	`/feed/entry/group/duration/@seconds`
`fechaVideo`	`/feed/entry/published`
`descripcionVideo`	`/feed/entry/group/description`
`urlImagenVideo`	`/feed/entry/group/thumbnail[1]/@url`
`urlVideo`	`/feed/entry/content/@src`

Tabla 6.3.1 – Correspondencia entre propiedades ontológicas e información en XML

Como se puede ver de esta tabla, se presentan discrepancias debido a:

- *Heterogeneidad terminológica*: debido al proceso de nombrar las entidades (empleo de diferentes idiomas, diferentes palabras para nombrar a la misma entidad, etc).

- *Heterogeneidad semántica*: debido a aspectos que tienen que ver con la cobertura (la información proveniente de YouTube cubre partes que se superponen con las de la ontología) y la granularidad (la información proveniente de YouTube contiene una descripción más detallada de las mismas entidades de la ontología).

- *Heterogeneidad sintáctica*: debido al empleo de 2 lenguajes distintos (XML para el caso de YouTube y OWL para el caso de la ontología).

Para resolver los problemas de mediación entre la información proveniente de las distintas fuentes y la ontología que captura el modelo del dominio del mashup semántico se emplean transformaciones XSL como se explica en la Sección 4.4 (“*Resolución del mapeo de ontologías*”).

6.4 Acceso a las fuentes de información

Como se dijo en la Sección 6.1, el acceso a la información de las fuentes se hace mediante servicios Web del tipo REST, empleando Java. Ver el punto 6.6 (“Implementación”).

6.5. Diseño de la presentación

El mashup semántico se organiza en 2 páginas: una inicial, donde se muestra la lista de fuentes de información a partir de las cuales se puede realizar la búsqueda de un concepto, y una página final donde se muestran los resultados. A fin de estandarizar los tipos de letras, colores, tamaño, etc, se trabaja con una hoja de estilos, en la cual se especifican estas características.

Para ambas páginas se trabaja sobre un fondo blanco. En la página inicial, se muetra a modo de encabezado la leyenda "Elija los servicios sobre los cuales hacer la búsqueda", y a continuación a modo de lista con viñetas, cada uno de los servicios (Libros, Imágenes, etc) y sus correspondientes fuentes de información. Finalmente hay un campo de texto donde se ingresa el texto a buscar y un botón para realizar la consulta.

Para la página con los resultados, se muestra el nombre de cada servicio y a continuación sus resultados, cuyo formato de presentación depende del tipo de servicio. Por ejemplo, para los videos, los resultados se muestran como una tabla de N filas y 2 columnas: en la primera columna va la imagen del video sobre la cual se puede seleccionarlo para reproducirlo, y en la segunda columna la información sobre el mismo, como el título, autor, fecha, duración, etc.

El formato de presentación de los resultados depende del tipo de servicio, pero en todos los casos, características como el tipo de letra, color, tamaño, etc, dependen de la hoja de estilos.

6.6 Implementación

En cuanto la Web comenzó a utilizarse para prestar servicios, los proveedores reconocieron la necesidad de generar contenido dinámico. Las Applets fueron uno de los primeros intentos para este fin, centrándose en el uso de la plataforma del cliente, mientras que por el lado del servidor surgieron inicialmente los scripts CGI (*Common Gateway Interface*). Aunque son muy utilizados, los scripts CGI tienen una serie de deficiencias, como ser la dependencia de plataforma y la falta de escalabilidad. Los Servlets Java hacen frente a estas limitaciones: básicamente son programas que se ejecutan en un servidor Web y que construyen páginas Web.

En este caso el mashup semántico se construye mediante un Servlet Java y una serie de JSPs (Java Server Pages). Una JSP es una tecnología que permite mezclar HTML estático con HTML generado dinámicamente. En base a todo esto, se necesita contar con el siguiente software:

- *Apache Tomcat*[73] u otro motor de Servlets. Para este ejemplo se trabaja con la versión 7.0.12.

- *Java*[74]: Apache Tomcat 7.0.12 requiere Java SE 6 o superior. Para este ejemplo se trabaja con la versión 1.6.0_18.

73 http://tomcat.apache.org/

74 http://www.oracle.com/technetwork/java/javase/downloads/index.html

- *Jena*[75]: Jena es un framework Java de código abierto para construir aplicaciones Web semánticas. Proporciona un entorno de programación para RDF, RDFS y OWL.

- *Netbeans*[76]: IDE que corre en Windows, Mac, Linux y Solaris y que permite crear aplicaciones Web, de escritorio y móviles y servicios usando Java, PHP, JavaScript y Ajax, Groovy y Grails y C/C++. Para este ejemplo se trabaja con la versión 6.8.

Todas las funciones de la ontología son manejadas por una clase llamada `LectorOntologia`:

```
public class LectorOntologia {
  public static String uriOnt = "http://www.semanticweb.org/ontologies/tesis";
  public static String archOnt = "file:/root/Documentos/.../Tesis.owl";
  public static String uriNothing = "http://www.w3.org/2002/07/owl#Nothing";

  private OntModel modelo;

  /**
   * Constructor predeterminado
  */
  public LectorOntologia() {
    modelo =
    ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MICRO_RULE_INF, null);

    modelo.getDocumentManager().addAltEntry(uriOnt, archOnt);
    modelo.read(uriOnt);
  }
}
```

En esta clase se definen, en forma de cadenas que pueden ser referenciadas por las otras clases, la URI de la ontología (`uriOnt`), el nombre del archivo donde se encuentra guardada la misma (`archOnt`) y la URI de la clase ontológica `Nothing` (`uriNothing`).

Para poder trabajar con la ontología que se tiene definida, se debe crear un modelo ontológico, el cual está representado dentro de esta clase mediante la variable `modelo`. Para crear este modelo, en el constructor se emplea la clase `ModelFactory`, y como la API Jena también es una API para RDF, da la facilidad de crear distintos tipos de modelos. En este caso se especifica un modelo ontológico que reside en memoria y que usa un razonador, el cual permite inferir que si A es subclase de B, y B es subclase de C, entonces A es subclase de C. Por esta razón, al momento de instanciar a `modelo` se especifica `OntModelSpec.OWL_MEM_MICRO_RULE_INF`.

La API Jena permite crear ontologías desde cero, pero como en este caso ya se tiene una guardada en un archivo, mediante el objeto `DocumentManager` se carga la copia de la misma con el método `addAltEntry()`, al cual se le especifica la URI de la ontología y el archivo donde se encuentra guardada la misma. Finalmente, mediante el método `read()` se lee el documento (mientras no se llame a este método, no se leen las definiciones guardadas en el archivo).

Los distintos ítems que se muestran como resultado para una determinada consulta son libros,

75 http://jena.sourceforge.net/
76 http://netbeans.org/

imágenes, videos y artículos de noticias. Por esta razón se definen las clases `Libro`, `Imagen`, `Video` y `Articulo`. Por ejemplo, a continuación se muestra un fragmento de la clase `Articulo`:

```
public class Articulo {
    private String titulo;
    private String resumen;
    private String url;
    private String fecha;
    private Fuente esInformadoPor;

    ...
}
```

A fin de tratar genéricamente cada uno de estos ítems, se define la clase `Item` como superclase de las clases `Libro`, `Imagen`, `Video` y `Articulo`. Según esto, la definición de la clase `Articulo` anterior queda:

```
public class Articulo extends Item {
    ...
}
```

Para manejar cada uno de los servicios sobre los que se realizan las consultas, se define la superclase `Servicio`, de la cual se muestra un fragmento a continuación:

```
public abstract class Servicio {
  private String nombreServicio;
  private String urlServicio;
  private ArrayList<Parametro> parametros = new ArrayList<Parametro>();
  private String xslt;
  private String consulta;
 public static String uriServicio = LectorOntologia.uriOnt + "#Servicio";
  ...
}
```

Se puede ver que la clase tiene las propiedades para el nombre del servicio, su dirección URL base, su conjunto de parámetros (definidos mediante la clase `Parametro`), la variable donde se guarda la cadena de consulta y la variable donde se guarda la cadena de transformación. Además tiene definida su URI.

La clase `Parametro` simplemente tiene las propiedades para el nombre y valor del mismo:

```
public class Parametro {
  private String nombreParametro;
  private String valorParametro;
  ...
}
```

Una vez definida la superclase `Servicio`, se definen las subclases `ServicioImagenes`,

`ServicioLibros`, `ServicioVideos` y `ServicioNoticias`. Por ejemplo:

```
public class ServicioImagenes extends Servicio {
   ...
}
```

Una de las funcionalidades con las que deberá contar la aplicación será la de obtener todos los individuos de un determinado tipo, por lo cual se define el método `getIndividuals()` en la clase `LectorOntologia`:

```
public ArrayList<Individual> getIndividuos(String tipoIndividuo) {
  ArrayList<Individual> individuos = new ArrayList<Individual>();

  for(Iterator it = modelo.listIndividuals(); it.hasNext(); ) {
    Individual esteIndividuo = (Individual)it.next();
    String tipo = esteIndividuo.getRDFType().getLocalName();
    if(tipo.equals(tipoIndividuo))
      individuos.add(esteIndividuo);
  } //for
  return individuos;
}
```

Este método lo que hace es obtener a partir de la ontología todos los individuos mediante el método `listIndividuals()` y luego va descartando todos aquellos que no correspondan con el tipo que se está buscando. Así, mediante `getIndividuos()` se pueden obtener todos los individuos del tipo `ServicioLibros` por ejemplo.

Otra funcionalidad necesaria de la aplicación es, dada una URI correspondiente a una clase ontológica `Servicio`, obtener el objeto `Servicio` correspondiente a la misma:

```
public Servicio getServicio(String uriServicio) {
  Individual elIndividuo = modelo.getIndividual(uriServicio);
  Servicio servicio = this.esServicio(elIndividuo);
  if (servicio != null) {
    servicio = new Servicio();
    for(StmtIterator it = elIndividuo.listProperties(); it.hasNext(); ) {
      Statement s = it.nextStatement();
     Property p = s.getPredicate();
     if (p.canAs(OntProperty.class)) {
        Resource r;
        Literal l;
       if (p.getURI().equals(uriOnt + "#nombreServicio")) { //cadena
          l = s.getLiteral();
          servicio.setNombreServicio(l.getString());
        }
        else if (p.getURI().equals(uriOnt + "#urlServicio")) { //cadena
          l = s.getLiteral();
          servicio.setURLServicio(l.getString());
        }
        else if (p.getURI().equals(uriOnt + "#tiene")) { //Parametro
          r = s.getResource();
          Parametro parametro = this.getParametro(r.getURI());
          if (parametro != null)
```

```
            servicio.agregarParametro(parametro);
          }
          else if (p.getURI().equals(uriOnt + "#xslt")) { //cadena
            l = s.getLiteral();
            servicio.setXSLT(l.getString());
          }
        }
      } //for
    } //if (esServicio(uriServicio))
    return servicio;
}
```

Este método, lo primero que hace es obtener el individuo correspondiente a la URI especificada, por medio del método `getIndividual()`:

```
Individual elIndividuo = modelo.getIndividual(uriServicio);
```

Luego se comprueba si el individuo que se obtuvo es o no un `Servicio`:

```
Servicio servicio = this.esServicio(elIndividuo);
```

El método `esServicio()` devuelve un tipo de `Servicio`, si el individuo es un `Servicio`, o `null` ni no lo es. Para esto, al individuo primero se le obtiene la clase ontológica a la cual pertenece, luego se obtiene la URI de la misma y por último se comprueba si corresponde a una de las clases ontológicas definidas:

```
public Servicio esServicio(Individual unIndividuo) {
  Servicio servicio = null;
  OntClass claseIndividuo = unIndividuo.getOntClass();
  //Se obtiene la clase ontológica del individuo

  String uriClaseIndividuo = claseIndividuo.getURI();
  //Se obtiene la URI de la clase del individuo

  if (uriClaseIndividuo.endsWith("ServicioImagenes"))
    servicio = new ServicioImagenes();
  else if (uriClaseIndividuo.endsWith("ServicioLibros"))
    servicio = new ServicioLibros();
  else if (uriClaseIndividuo.endsWith("ServicioNoticias"))
    servicio = new ServicioNoticias();
  else if (uriClaseIndividuo.endsWith("ServicioVideos"))
    servicio = new ServicioVideos();
  return servicio;
}
```

Continuando con el método `getServicio()`, cuando se sabe que el individuo sí corresponde a un `Servicio`, se le obtienen sus propiedades. Como no se puede obtener una en particular, se obtienen todas, y según cada una, se va construyendo el objeto tipo `Servicio`:

```
if (servicio != null) {
 for(StmtIterator it = elIndividuo.listProperties(); it.hasNext(); ) {
    Statement s = it.nextStatement();
    Property p = s.getPredicate();
    if (p.canAs(OntProperty.class))  {
      Resource r;
      Literal l;
      if (p.getURI().equals(uriOnt + "#nombe ervicio") ){  //cadena
        l = s.getLiteral();
        servicio.setNombreServicio(l.getString());
      }
      else if (p.getURI().equals(uriOnt + "#urlServicio")) { //cadena
        l = s.getLiteral();
        servicio.setURLServicio(l.getString());
      }
      ...
    }
  } //for
}
return servicio;
```

Para la implementación del servlet en sí mismo se define la clase `TesisServlet`, cuyos 2 métodos principales son `doGet()` y `doPost()`. El método `doGet()` será quien se encargue de mostrar la lista de servicios (fuentes de información) a partir de las cuales se realizará la búsqueda de un determinado concepto:

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
  response.setContentType("text/html;charset=UTF-8");
  PrintWriter out = response.getWriter();
  try {
    LectorOntologia lectorOnt = new LectorOntologia();
    out.print("<html>");
    out.print("<head>");
    out.print("<link rel='stylesheet' type='text/css' href='estilos.css'/>");
    out.print("</head>");
    out.print("<body>");
    out.print("<form action=\"TesisServlet\" method=\"post\">");
   out.print("<h1>Elija los servicios sobre los cuales hacer la búsqueda</h1>");

    OntClass scs =
              lectorOnt.getModeloOntologia().getOntClass(Servicio.uriServicio);
    out.print("<table>");
    int contCheckBox = -1; //para darle nombres únicos a los checkboxes
    for(ExtendedIterator it = scs.listSubClasses(); it.hasNext();) {
      OntClass sc = (OntClass)it.next();
      if(!sc.getURI().equals(LectorOntologia.uriNothing)) {
                                      ArrayList<Individual>     individuos     =
lectorOnt.getIndividuos(sc.getLocalName());
        if (!individuos.isEmpty()) {
                  out.print("<tr><td  colspan='3'><li>"  +  sc.getLocalName()  +
"</li></td></tr>");
          for(int i = 0; i < individuos.size(); i++) {
            Individual unIndividuo = individuos.get(i);
            for(StmtIterator ite = unIndividuo.listProperties(); ite.hasNext();)
            {
```

```
                Statement s = ite.nextStatement();
                Property p = s.getPredicate();
                if (p.canAs(OntProperty.class)) {
                  if (p.getLocalName().equals("nombreServicio")) {
                    String label = s.getLiteral().getString();
                    out.print("<tr>");
                    out.print("<td>");
                    out.print("</td>");
                    out.print("<td>");
                    out.print(label);
                    out.print("</td>");
                    out.print("<td>");
                    String nombreCheckBox = "checkbox" + ++contCheckBox;
                    out.print("<input type='checkbox' name='" + nombreCheckBox +
                     "' value='" + unIndividuo.getURI() + "' align='" + "right" +
                     "'>" + "</input>");
                    out.print("</td>");
                    out.print("</tr>");
                    break;
                  }
                } //if (p.canAs(OntProperty.class))
              }
            } //for(int i = 0; i < individuos.size(); i++)
            out.print("<tr>");
            out.print("<td> </td>");
            out.print("<td> </td>");
            out.print("<td> </td>");
            out.print("</tr>");
          } //if (!individuos.isEmpty())
        }
      }
      out.print("</table>");
      out.print("<input type='hidden' value='" + contCheckBox +
          "' name='cantCheckBox'/>");
      out.print("<p>Ingrese el texto a buscar</p>");
      out.println("<input type='text' name='consulta'/>");
      out.print("<br/>");
      out.print("<input type='submit' />");
      out.print("</form>");
      out.print("</body>");
      out.print("</html>");
    } finally {
      out.close();
    }
}
```

Como se ve, primero se instancia un objeto del tipo `LectorOntologia`, con lo cual se lee y carga la ontología, se especifica una hoja de estilos y se le da un título al formulario HTML:

```
LectorOntologia lectorOnt = new LectorOntologia();
out.print("<html>");
out.print("<head>");
out.print("<link rel='stylesheet' type='text/css' href='estilos.css'/>");
out.print("</head>");
out.print("<body>");
out.print("<form action=\"TesisServlet\" method=\"post\">");
out.print("<h1>Elija los servicios sobre los cuales hacer la búsqueda</h1>");
```

Para poder obtener todos los individuos de los distintos tipos de servicio, primero se comienza obteniendo la superclase ontológica de todos los servicios, es decir, la clase `Servicio`:

```
OntClass sps = lectorOnt.getModeloOntologia().getOntClass(Servicio.uriServicio);
```

Observar que el método `getOntClass()` recibe como parámetro la URI de la clase `Servicio`, es decir, la cadena:

```
"http://www.semanticweb.org/ontologies/tesis#Servicio"
```

Una vez que se tiene la superclase de todos los servicios, se obtienen todas sus subclases:

```
for(ExtendedIterator it = sps.listSubClasses(); it.hasNext();)
```

Dentro de este bloque `for()` se obtienen cada una de las subclases de `Servicio`:

```
OntClass sc = (OntClass)it.next();
```

Como una de las subclases de `Servicio` es la clase ontológica `Nothing`, se la descarta:

```
if(!sc.getURI().equals(LectorOntologia.uriNothing)) {
```

Luego, si la subclase de `Servicio` no es la clase `Nothing`, se obtienen todos los individuos de esa subclase:

```
ArrayList<Individual> individuos = lectorOnt.getIndividuos(sc.getLocalName());
```

Lo que se busca hacer es que por cada subclase, que tenga individuos, se muestre el nombre de la misma y a continuación cada uno de éstos de la siguiente manera:

- Videos
  - Individuo1
  - Individuo2
- Noticias
  - Individuo3
- Libros
  - Individuo4
- Imágenes
  - Individuo5
  - Individuo6

Para que por cada subclase de `Servicio` se muestre "`Videos`", "`Noticias`", etc, en cada una de las subclases ontológicas se define una *anotación* sobre la propiedad `label`. Por ejemplo, la

subclase `ServicioVideos` tiene definida la anotación "`Videos`", la subclase `ServicioNoticias` "`Noticias`" y así sucesivamente:

```
if (!individuos.isEmpty()) {
  out.print("<tr><td colspan='3'><li>" + sc.getLabel(null) + "</li></td></tr>");
```

Luego, por cada individuo de una determinada clase se muestra el valor de su propiedad "`nombreServicio`" (por ejemplo `Flickr.com`) y al lado un control tipo *checkbox*, cuyo nombre será `checkbox0` para el primer control, `checkbox1` para el segundo, y así sucesivamente. El valor que tome cada uno de los controles *checkbox* en caso de estar seleccionado será el de la URI del individuo que representa:

```
for(int i = 0; i < individuos.size(); i++) {
  Individual unIndividuo = individuos.get(i);
  for(StmtIterator ite = unIndividuo.listProperties(); ite.hasNext(); ) {
    Statement s = ite.nextStatement();
    Property p = s.getPredicate();
    if (p.canAs(OntProperty.class)) {
      if (p.getLocalName().equals("nombreServicio")) {
        String label = s.getLiteral().getString();
        String nombreCheckBox = "checkbox" + ++contCheckBox;
        out.print("<input type='checkbox' name='" + nombreCheckBox +
                  "' value='" + unIndividuo.getURI() + "' align='" +
                  "right" + "'>" + "</input>");
```

Como se puede ver en el fragmento de código anterior, no se puede recuperar una propiedad específica, por lo que hay que obtenerlas a todas mediante el método `listProperties()` e ir filtrando la que se necesite.

Como las propiedades se representan como sentencias de la forma sujeto-predicado-objeto, se obtienen cada una de las sentencias y luego los predicados de las mismas:

```
Statement s = ite.nextStatement();
Property p = s.getPredicate();
```

Finalmente, si el nombre de la propiedad es "`nombreServicio`", se obtiene el valor de la misma, y no se tienen en cuenta el resto de las propiedades:

```
if (p.getLocalName().equals("nombreServicio")) {
     String label = s.getLiteral().getString();
```

Como la propiedad "`nombreServicio`" es del tipo *Datatype* y está definida como una cadena, el valor de la misma se puede obtener mediante el método `getLiteral().getString()`.

Luego, cuando se han recorrido todos los individuos de todas las subclases de `Servicio`, se crea en la página un campo oculto cuyo valor es la variable que lleva el control de la cantidad de controles tipo *checkbox* (`contCheckBox`). La razón de guardar este dato es para que cuando se presione el botón para hacer la consulta, se sepa la cantidad total de controles tipo *checkbox*:

```
out.print("<input type='hidden' value='" + contCheckBox +
          "' name='cantCheckBox'/>");
```

Finalmente, se crea en la página una etiqueta con el texto "`Ingrese el texto a buscar`", el campo de texto para escribir el término a buscar y el botón para realizar la consulta:

```
out.print("<p>Ingrese el texto a buscar</p>");
out.println("<input type='text' name='consulta'/>");
out.print("<br/>");
out.print("<input type='submit' />");
```

En estos momentos, se tiene una página como la que se muestra en la figura 6.6.1:

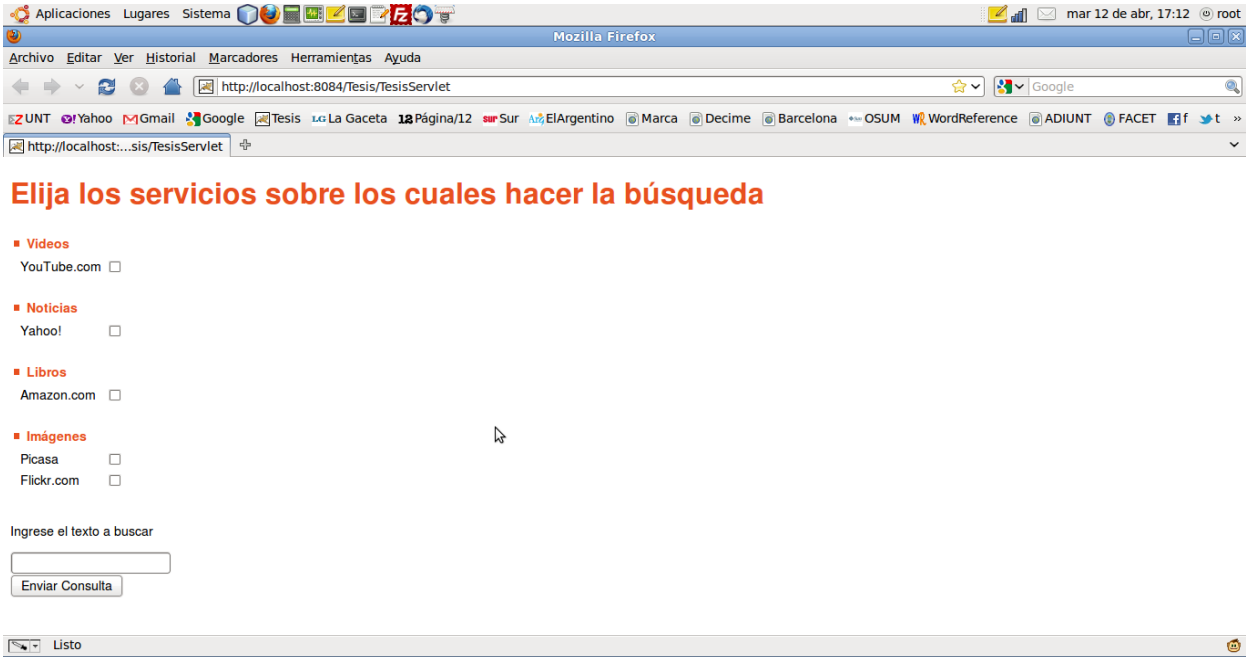

Figura 6.6.1 – Página inicial del mashup

El método `doPost()` es quien se encarga de buscar en cada una de las fuentes seleccionadas los ítems que estén relacionados con el texto ingresado y generar una página con los resultados:

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
  response.setContentType("text/html;charset=UTF-8");
  PrintWriter out = response.getWriter();

  int cantCheckBoxes = Integer.parseInt(request.getParameter("cantCheckBox"));
  String consulta = request.getParameter("consulta");
  String todosLosServicios = null;

  for(int i = 0; i <= cantCheckBoxes; i++) {
    String servicio = request.getParameter("checkbox" + i);
    if (servicio != null) {
```

```
      if (todosLosServicios == null)
        todosLosServicios = servicio;
      else
        todosLosServicios = todosLosServicios + ";" + servicio;
    }
  }
  String[] serviciosSeleccionados = todosLosServicios.split(";");

  try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document docSalida = db.parse(new InputSource(new StringReader("<html/>")));
    Node raiz = docSalida.getDocumentElement(); //<html>
    Element elemHead = docSalida.createElement("head");
    Element elemLink = docSalida.createElement("link");
    elemLink.setAttribute("rel", "stylesheet");
    elemLink.setAttribute("type", "text/css");
    elemLink.setAttribute("href", "estilos.css");
    elemHead.appendChild(elemLink);
    raiz.appendChild(elemHead);

    Element elemBody = docSalida.createElement("body");
    LectorOntologia lectorOnt = new LectorOntologia();

    for(int i = 0; i < serviciosSeleccionados.length; i++) {
      String unServicioURI = serviciosSeleccionados[i];
      Servicio servicio = lectorOnt.getServicio(unServicioURI);
      servicio.setConsulta(consulta);
      Document documento = servicio.obtenerDatos();
      Node nodoTitulo = docSalida.createTextNode(servicio.getNombreServicio());
      Element elementoTitulo = docSalida.createElement("h1");
      elementoTitulo.appendChild(nodoTitulo);
      elemBody.appendChild(elementoTitulo);

      if (documento != null) {
        elemBody.appendChild(servicio.presentarDatos(lectorOnt, docSalida));
      }
      else {
        Node nodoMensaje = docSalida.createTextNode("No se obtuvieron datos");
        elemBody.appendChild(nodoMensaje);
      }
    } //for
    raiz.appendChild(elemBody);

    DOMSource origen = new DOMSource(docSalida);
    StreamResult destino = new StreamResult(out);
    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer t = tf.newTransformer();
    t.transform(origen, destino);
  }
  catch(Exception e) {
    System.out.println("error");
  }
}
```

Lo primero que hace el método es obtener la cantidad total de controles tipo *checkbox*, tanto los seleccionados como lo no seleccionados, y obtener el texto de la consulta ingresada en el campo de texto:

```
int cantCheckBoxes = Integer.parseInt(request.getParameter("cantCheckBox"));
String consulta = request.getParameter("consulta");
```

Luego se arma un vector donde cada elemento es una cadena con la URI de cada servicio seleccionado, para lo cual se recorren todos los controles tipo *checkbox*. Si `checkbox0`, por ejemplo, está seleccionado, su valor valdrá la URI de un individuo de un determinado servicio.

```
String todosLosServicios = null;
for(int i = 0; i <= cantCheckBoxes; i++) {
  String servicio = request.getParameter("checkbox" + i);
  if (servicio != null) {
    if (todosLosServicios == null)
      todosLosServicios = servicio;
    else
      todosLosServicios = todosLosServicios + ";" + servicio;
  }
}
String[] serviciosSeleccionados = todosLosServicios.split(";");
```

Luego, se muestran en la página los resultados que se obtienen de cada una de las fuentes de datos, para lo cual se empieza generando un documento HTML:

```
try {
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder db = dbf.newDocumentBuilder();
  Document docSalida = db.parse(new InputSource(new StringReader("<html/>")));
  Node raiz = docSalida.getDocumentElement(); //<html>
  Element elemHead = docSalida.createElement("head");
  Element elemLink = docSalida.createElement("link");
  elemLink.setAttribute("rel", "stylesheet");
  elemLink.setAttribute("type", "text/css");
  elemLink.setAttribute("href", "estilos.css");
  elemHead.appendChild(elemLink);
  raiz.appendChild(elemHead);

  Element elemBody = docSalida.createElement("body");
```

Para cada fuente de datos seleccionada, se obtiene el objeto tipo `Servicio` que la representa mediante el método `getServicio()`, explicado anteriormente:

```
for(int i = 0; i < serviciosSeleccionados.length; i++) {
  String unServicioURI = serviciosSeleccionados[i];
  Servicio servicio = lectorOnt.getServicio(unServicioURI);
```

Una vez que se tiene el objeto tipo `Servicio`, al mismo se le asigna la cadena de búsqueda y se obtienen los datos de la fuente de información que representa mediante el método `obtenerDatos()`, el cual devuelve un XML con la hoja de estilos aplicada:

```
servicio.setConsulta(consulta);
Document documento = servicio.obtenerDatos();
```

A continuación se muestra la implementación del método `obtenerDatos()`:

```
public Document obtenerDatos() {
  try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setNamespaceAware(true);
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document docEntrada = db.parse(this.armarPedidoREST());
    Document docSalida = null;
    try {
      Source origen = new DOMSource(docEntrada);
      Document docXSL =
                  db.parse(new InputSource(new StringReader(this.getXSLT())));
      DOMSource origenXSL = new DOMSource(docXSL);
      TransformerFactory tf = TransformerFactory.newInstance();
      Transformer t = tf.newTransformer(origenXSL);
      DOMResult resultado = new DOMResult();
      t.transform(origen, resultado);
      docSalida = (Document)resultado.getNode();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    this.documento = docSalida;
    return this.documento;
  }
  catch(Exception e)
  {
    System.out.println(e.getMessage());
    return null;
  }
}
```

Primero, para que el método `obtenerDatos()` pueda hacer el pedido a su fuente de datos, debe armar toda la cadena para hacer el pedido REST mediante el método `armarPedidoREST()`, el cual arma una cadena de la forma:

```
URL_base?parametro1=valor1&parametro2=valor2&...&parametroN=valorN
```

Por ejemplo, para el servicio que representa la fuente de datos de Flickr, la salida de este método sería:

```
https://picasaweb.google.com/data/feed/api/all?q=Egipto&max-results=10
```

```
public String armarPedidoREST() {
  String cadenaREST = this.urlServicio;
  for(int i = 0; i < this.parametros.size(); i++) {
    Parametro parametro = this.parametros.get(i);
    String nombreParametro = parametro.getNombreParametro();
    String valorParametro = parametro.getValorParametro();
    if (valorParametro == null)
      valorParametro = consulta;

    if (cadenaREST.equals(this.urlServicio))
```

```
      cadenaREST = cadenaREST + "?";
    else
      cadenaREST = cadenaREST + "&";
    cadenaREST += nombreParametro + "=" + valorParametro;
  }
  return cadenaREST;
}
```

Continuando con el método `obtenerDatos()`, cuando se tiene la cadena para hacer el pedido, se arma un XML con los resultados de la consulta al servicio REST:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
DocumentBuilder db = dbf.newDocumentBuilder();
Document docEntrada = db.parse(this.armarPedidoREST());
```

Finalmente, al XML obtenido se le aplica la hoja de estilos:

```
Document docSalida = null;
try {
  Source origen = new DOMSource(docEntrada);
  Document docXSL =
  db.parse(new InputSource(new StringReader(this.getXSLT())));

  DOMSource origenXSL = new DOMSource(docXSL);
  TransformerFactory tf = TransformerFactory.newInstance();
  Transformer t = tf.newTransformer(origenXSL);
  DOMResult resultado = new DOMResult();
  t.transform(origen, resultado);
  docSalida = (Document)resultado.getNode();
}
catch(Exception e) {
  e.printStackTrace();
}
```

Continuando con el método `doPost()`, una vez que se tienen los datos en un XML con la hoja de estilos aplicada, en la página se muestra en forma de encabezado el nombre del servicio y los datos:

```
Node nodoTitulo = docSalida.createTextNode(servicio.getNombreServicio());
Element elementoTitulo = docSalida.createElement("h1");
elementoTitulo.appendChild(nodoTitulo);
elemBody.appendChild(elementoTitulo);

if (documento != null) {
  elemBody.appendChild(servicio.presentarDatos(lectorOnt, docSalida));
}
else {
  Node nodoMensaje = docSalida.createTextNode("No se obtuvieron datos");
  elemBody.appendChild(nodoMensaje);
}
```

Como se puede ver en el fragmento de código anterior, los datos se muestran de una determinada

manera dentro de la página dependiendo del servicio (ver el método `presentarDatos()`). Por ejemplo, para los servicios que representan imágenes, las mismas se muestran en forma de tabla sobre la página a razón de 10 imágenes por fila, y por cada imagen se muestra la imagen en sí misma, su título y el autor:

```
public Element presentarDatos(LectorOntologia lo, Document docSalida) {
  try {
    Ontdel mCopia =
    ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MICRO_RULE_INF, null);
    Model modelo =
  lo.getModeloOntologia().getDocumentManager().getModel(LectorOntologia.uriOnt);

    mCopia.add(modelo);

   DOM2Model.createD2M(LectorOntologia.uriOnt,mCopia).load(this.getDocumento());
    ArrayList<Item> nodos = this.obtenerItems(modeloCopia);
    int contadorImagenes = 0;
    Element elemTABLE = docSalida.createElement("table");
    Element elemTR = null;
    Element elemTD = null;
    for(int i = 0; i < nodos.size(); i++) {
      Item item = nodos.get(i);
      contadorImagenes++;
      elemTD = docSalida.createElement("td");
      elemTD.setAttribute("id", "tdimagen");
      elemTD.appendChild(item.formatearSalida(docSalida));
      if (contadorImagenes == 1) {
        elemTR = docSalida.createElement("tr");
        elemTR.setAttribute("id", "trimagen");
        elemTR.appendChild(elemTD);
        if (i == nodos.size() - 1)
          elemTABLE.appendChild(elemTR);
      }
      else if ((contadorImagenes > 1) && (contadorImagenes < IMAGENESPORFILA)) {
        elemTR.appendChild(elemTD);
        if (i == nodos.size() - 1)
          elemTABLE.appendChild(elemTR);
      }
      else if (contadorImagenes == IMAGENESPORFILA) {
        contadorImagenes = 0;
        elemTR.appendChild(elemTD);
        elemTABLE.appendChild(elemTR);
      }
    } //for
    return elemTABLE;
  }
  catch (Exception e) {
    return null;
  }
}
```

Los métodos `presentarDatos()`, redefinidos en cada subclase de Servicio, sólo trabajan sobre sus propios datos, es decir, el método `presentarDatos()` de la clase `ServicioImagenes` sólo muestra los datos que son imágenes, el de la clase `ServicioLibros` sólo muestra los libros, y así sucesivamente. Para esto, se trabaja sobre una copia del modelo ontológico:

```
Ontdel mCopia =
    ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MICRO_RULE_INF, null);
```

```
Model modelo =
 lo.getModeloOntologia().getDocumentManager().getModel(LectorOntologia.uriOnt);
mCopia.add(modelo);
```

Si en este momento se hace `mCopia.write(System.out)`, se muestra toda la ontología, pero sólo su definición, no los datos. La clase `DOM2Model` tiene el método estático `createD2M()` que permite referenciar el modelo para el espacio de nombres especificado. A partir de ahí, se puede cargar el documento DOM que representa los datos devueltos por el servicio y transformarlo en su forma ontológica genérica:

```
DOM2Model.createD2M(LectorOntologia.uriOnt, mCopia).load(this.getDocumento());
```

Si ahora se hace `modelo.write(System.out)`, se muestra toda la ontología, definición y datos.

Luego se obtienen todos los objetos dependiendo del tipo de servicio, que a su vez son `Item`:

```
ArrayList<Item> nodos = this.obtenerItems(mCopia);
```

Luego, una vez que se tienen todos los objetos de un determinado tipo, se muestra en la página mediante el método `formatearSalida()`, el cual decide qué cosas se muestran, y cómo:

```
int contadorImagenes = 0;
Element elemTABLE = docSalida.createElement("table");
Element elemTR = null;
Element elemTD = null;
for(int i = 0; i < nodos.size(); i++) {
  Item item = nodos.get(i);
  contadorImagenes++;
  elemTD = docSalida.createElement("td");
  elemTD.setAttribute("id", "tdimagen");
  elemTD.appendChild(item.formatearSalida(docSalida));
  if (contadorImagenes == 1) {
    elemTR = docSalida.createElement("tr");
    elemTR.setAttribute("id", "trimagen");
    elemTR.appendChild(elemTD);
    if (i == nodos.size() - 1)
      elemTABLE.appendChild(elemTR);
  }
  else if ((contadorImagenes > 1) && (contadorImagenes < IMAGENESPORFILA)) {
    elemTR.appendChild(elemTD);
    if (i == nodos.size() - 1)
      elemTABLE.appendChild(elemTR);
  }
  else if (contadorImagenes == IMAGENESPORFILA) {
    contadorImagenes = 0;
    elemTR.appendChild(elemTD);
    elemTABLE.appendChild(elemTR);
  }
} //for
return elemTABLE;
```

Finalmente, se muestran los resultados en la página:

```
DOMSource origen = new DOMSource(docSalida);
StreamResult destino = new StreamResult(out);
```

```
TransformerFactory tf = TransformerFactory.newInstance();
Transformer t = tf.newTransformer();
t.transform(origen, destino);
```

En estos momentos, se tiene una página como la que se muestra en las figuras 6.6.2, 6.6.3 y 6.6.4.:

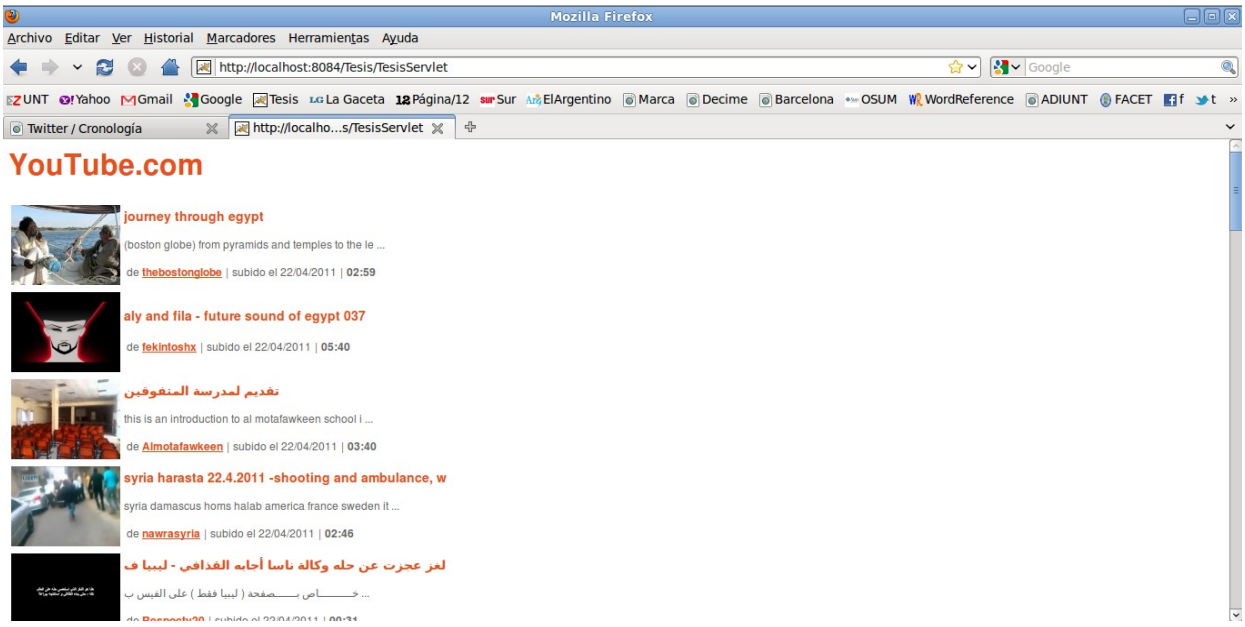

Figura 6.6.2 – Página con los resultados del mashup

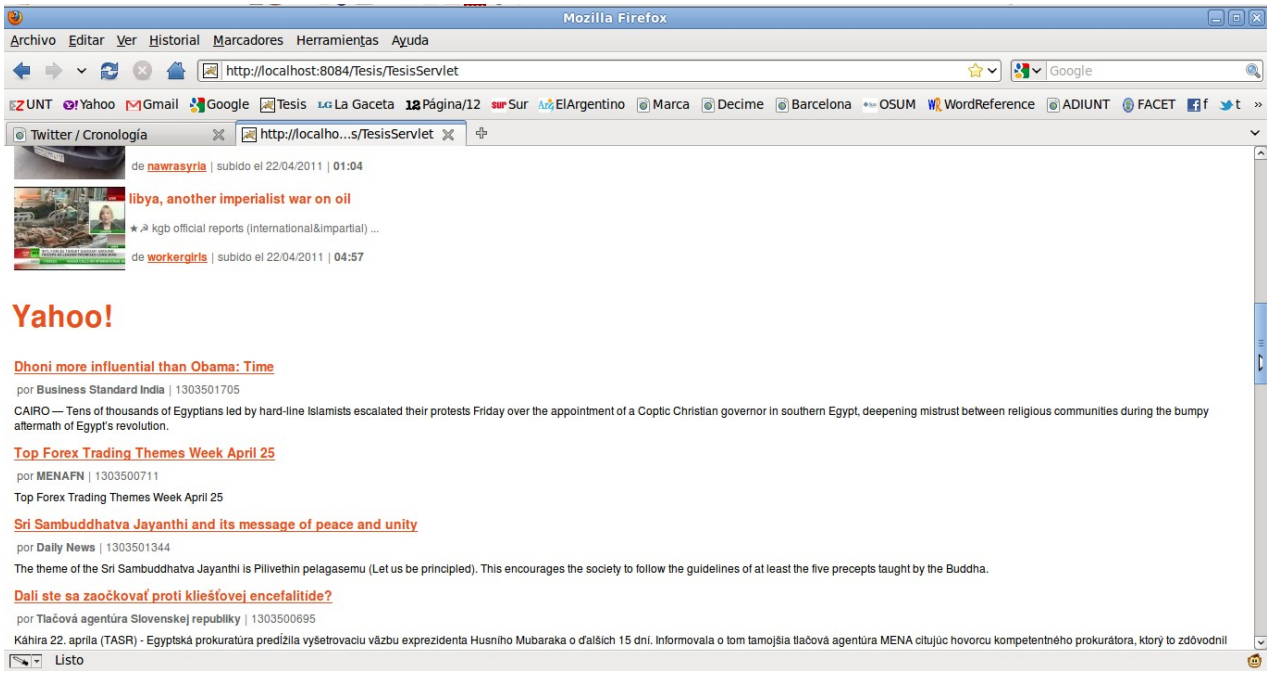

Figura 6.6.3 – Página con los resultados del mashup (continuación)

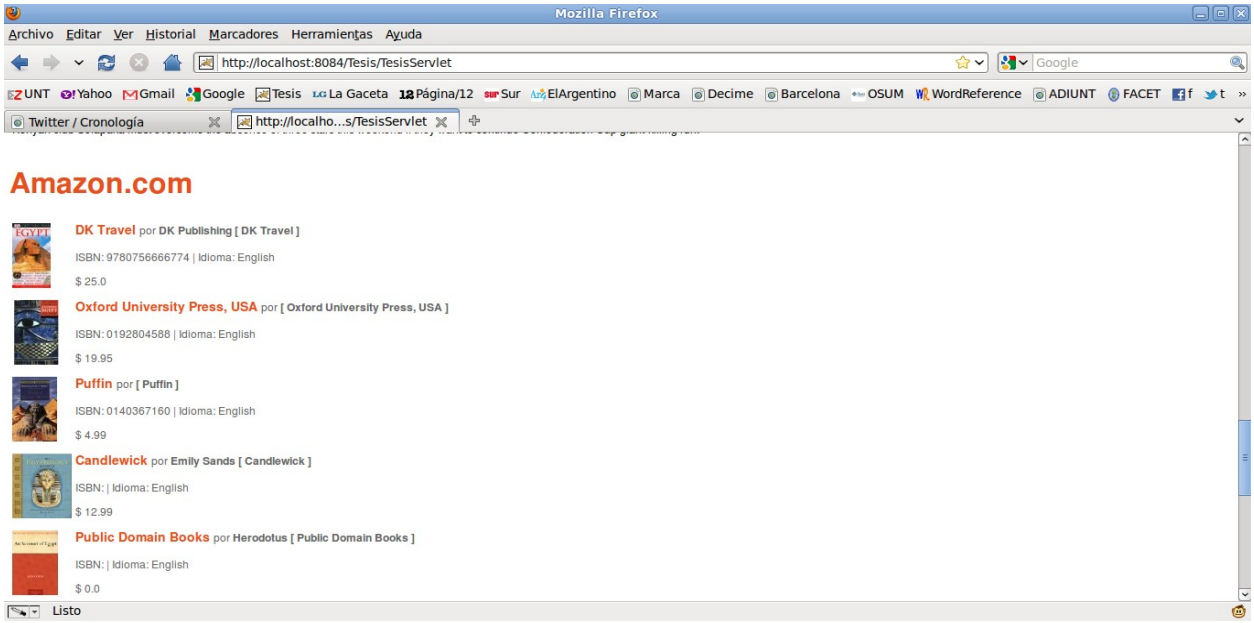

Figura 6.6.4 – Página con los resultados del mashup (continuación)

CAPÍTULO 7 – CONCLUSIONES

En este trabajo se presenta una propuesta de metodología para el diseño de mashups semánticos. Un mashup semántico es una aplicación Web que extrae información de diversas fuentes, la combina (integra) y presenta en otras formas, pero esta información presenta la característica de poder ser "interpretada" por una máquina. Al tratarse de un tipo de aplicación Web, su metodología de diseño en cuanto al aspecto visual, codificación y pruebas es similar a estos tipos de aplicaciones, y difiere en los aspectos propios de las aplicaciones semánticas (el empleo de ontologías).

Para comprender el concepto de mashup semántico, se presenta una introducción a la Web Semántica, mostrando la evolución de la Web, desde sus orígenes hasta la actualidad. Luego se nombran los aspectos propios del desarrollo de mashups semánticos y los desafíos que impone el desarrollo de estas aplicaciones.

Luego se presentan y analizan algunos de los métodos más conocidos para el diseño de aplicaciones Web, como ser OOWS, OOHDM, UWE, IDM, WebML, HERA y WSDM, dando para cada uno una breve descripción de sus características principales, y analizando cuál podría adaptarse al caso de los mashups semánticos. De este análisis se concluye que es necesario contar con una metodología propia para estas aplicaciones.

Como un mashup semántico toma información proveniente de distintas ontologías, se explica este concepto y se muestran algunos lenguajes para su representación, como ser RDF, RDFS y OWL.

Luego se presenta el problema de la mediación de ontologías debido a la necesidad de integrar la información de las mismas. Se presentan las razones por las cuales se producen discrepancias entre las ontologías y se mencionan las características más importantes a tener en cuenta para disminuir este problema. También se muestra el enfoque adoptado en el desarrollo del caso práctico para la mediación.

La metodología que se propone para diseñar mashups semánticos está compuesta de 7 procesos:

1. **Definición de los objetivos del mashup semántico**: se formula la declaración de los objetivos del mashup semántico, para poder tomar decisiones referidas al modelado del dominio o para evaluar la efectividad del mismo. Esta declaración sirve también como base para decidir qué información incluir (o no), cómo estructurarla y cómo presentarla.

2. **Modelado del dominio**: a partir de la declaración de los objetivos del mashup semántico, se convierten los mismos en descripciones formales que se puedan utilizar más tarde para implementar el mashup semántico. Este proceso trata con el modelado del dominio del mashup semántico.

   En la Web Semántica, las aplicaciones utilizan ontologías y modelos RDF, es decir, en cualquier aplicación para la Web Semántica, la ontología es el modelo del dominio, y para

un mashup semántico, esta ontología permite establecer las correspondencias (mapeos) entre los distintos conceptos y relaciones entre las ontologías de las fuentes de información con las cuales trabaja.

La finalidad de este proceso es modelar el dominio del mashup semántico en una ontología, la cual se utilizará como referencia para luego poder realizar la mediación de las distintas ontologías de cada una de las fuentes de información.

3. **Mediación de ontologías**: en la Web Semántica la información está organizada en ontologías, por lo que el mashup semántico al realizar la mediación de las mismas debe resolver los tipos de heterogeneidad debidos a las discrepancias terminológicas y conceptuales (semánticas).

   Pero en la Web 1.0 y 2.0 la información está organizada de manera tal que sólo resulta comprensible a las personas (HTML, XML, etc), por lo que además de resolver las discrepancias terminológicas y conceptuales, también debe resolver las sintácticas.

4. **Acceso a las fuentes de información**: una vez que se tienen definidos los mapeos entre las ontologías de las distintas fuentes, el mashup semántico debe extraer la información proporcionada por cada una para combinarla y presentarla en otras formas, para mejorar la presentación visual de la misma o ambas cosas.

5. **Diseño de la presentación**: se complementa el diseño conceptual con los detalles necesarios para la implementación. Se define la apariencia visual del mashup semántico, así como el diseño de las páginas que lo conforman.

6. **Implementación**: se traduce el modelo conceptual en un modelo con el suficiente grado de detalle como para que pueda ser ejecutado directamente sobre alguna plataforma.

7. **Pruebas**: se realizan las pruebas correspondientes para determinar la conformidad del mashup semántico con los objetivos del mismo.

Finalmente se presenta un caso práctico completo desarrollado siguiendo la metodología propuesta, explicando para los primeros 6 procesos las distintas actividades que se realizaron junto con los problemas que se encontraron.

7.1 Trabajos futuros

Las ontologías buscan capturar genérica y formalmente el conocimiento (consensuado) de un dominio dado, de forma tal que las aplicaciones y las personas puedan reutilizarlo y compartirlo.

Se vio que para un determinado dominio, no existe una única ontología que lo modele (se pueden encontrar varias ontologías que modelen un determinado conocimiento de diferentes maneras). Por ejemplo, en el campo del comercio electrónico, existen varios estándares e iniciativas conjuntas para

la clasificación de productos y servicios (UNSPSC[77], e-cl@ss[78], RosettaNet[79], NAICS[80], etc). Lo mismo se presenta en otros ámbitos, como medicina, derecho, arte, ciencias, etc.

Además, la existencia de diferentes estándares no es la única razón por la que la resolución de un problema requiera la manipulación de diferentes ontologías que modelen un conocimiento similar. Por ejemplo, la traducción de lenguaje de una ontología implica trabajar con 2 ontologías (la de entrada y la de salida), la evolución de una ontología implica varias ontologías (las diferentes versiones de la ontología original), etc. Incluso cuando a priori no hay ontologías dadas, la resolución de un problema puede requerir la manipulación de fuentes de información heterogéneas.

Cualquiera sea la razón por la que se empleen diferentes ontologías para un mismo dominio, generalmente las mismas se vinculan entre sí a través de mapeos, asociando los términos y expresiones definidos en una ontología origen con sus correspondientes en una ontología destino.

Actualmente, estos mapeos se identifican manualmente, presentando las siguientes desventajas:

- La generación de mapeos entre ontologías grandes, o entre una gran cantidad de ontologías diferentes, consume una gran cantidad de recursos.
- Si algunas de las ontologías cambian, la generación tiene que realizarse manualmente de nuevo.

Según Berners-Lee, la Web Semántica es una extensión de la Web actual, donde a la información se le da un significado bien definido, facilitando a las aplicaciones y a la gente trabajar en cooperación. Para lograr la Web Semántica, las páginas Web se anotan con ontologías (diferentes y siempre cambiantes). En consecuencia, la generación automática de mapeos resulta esencial para el futuro de la Web Semántica.

Como extensión a este trabajo, se propone:

- El estudio de la problemática que presenta la generación automática de mapeos de ontologías.
- El análisis de distintas posibilidades para la generación automática de mapeos.
- La búsqueda de enfoques existentes para la generación automática de mapeos y el estudio de posibles mejoras a estos enfoques.
- El análisis de distintas herramientas para la generación automática de mapeos.

77 http://www.unspsc.org/
78 http://www.eclass.de/
79 http://www.rosettanet.org/
80 http://www.census.gov/eos/www/naics/

APÉNDICE A - RDF

Aunque se lo suele llamar lenguaje, RDF es esencialmente un modelo de datos muy simple, con varias representaciones. RDF es una forma, no ambigua, de establecer información sobre algo llamado recurso. Es decir, RDF es una forma de especificar información sobre un recurso.

La información sobre un recurso se especifica mediante las *propiedades* del mismo, las cuales describen las relaciones entre éstos, por ejemplo: “escrito por”, “edad”, “título”, etc [Chase, 2007B]. En conclusión, para describir un recurso en RDF, se lo hace mediante sus propiedades.

Para comprender RDF, se analiza a continuación un ejemplo de distribución de información por la Web [Allemang & Hendler, 2008]. Generalmente, la información suele representarse en forma de tablas relacionadas, donde cada fila representa una entidad que se está describiendo, cada columna representa una propiedad de una entidad, y las celdas son los valores particulares de estas propiedades.

Esta representación en forma relacional, puede verse en la tabla A.1, en la cual se muestra información sobre distintas obras, sus autores y fecha en que fueron creadas.

ID	Título	Autor	Medio	Año
1	As You Like It	Shakespeare	Obra de teatro	1599
2	Hamlet	Shakespeare	Obra de teatro	1604
3	Othello	Shakespeare	Obra de teatro	1603
4	Sonnet 78	Shakespeare	Poema	1609
5	Astrophil and Stella	Sir Phillip Sidney	Poema	1590
6	Edward II	Christopher Marlowe	Obra de teatro	1592
7	Hero and Leander	Christopher Marlowe	Poema	1593
8	Greensleeves	Henry VII Rex	Canción	1525

Tabla A.1 – Representación relacional sobre obras, autores y fecha en que fueron creadas

A continuación se analizan algunas alternativas para distribuir la información de la tabla A.1 por la Web:

- *Alternativa 1*: hacer particiones horizontales sobre la tabla A.1, de forma tal que cada máquina donde se van a distribuir los datos guarde una o más filas completas de la tabla. Con esta alternativa, cualquier consulta sobre una entidad puede ser respondida por la máquina que guarda su fila correspondiente.

  Esta solución ofrece gran flexibilidad, ya que las máquinas pueden compartir la carga de

representar la información sobre varias entidades, pero requiere un poco de coordinación: cada máquina debe compartir información sobre las columnas. Por ejemplo, ¿la segunda columna de una fila en una máquina corresponde a la misma información que la segunda columna de una fila en otra máquina? Esta solución requiere de un identificador global para las columnas de las entidades que se están describiendo.

- *Alternativa 2*: hacer particiones verticales sobre la tabla A.1, de forma tal que cada máquina donde se van a distribuir los datos guarde una o más columnas completas de la tabla original. Esta solución también resulta flexible, de manera diferente, con respecto a la primera solución: permite a cada máquina ser responsable de un tipo de información. Si no se está interesado, por ejemplo, en las fechas de creación, no se necesitará considerar la información de esa máquina. De la misma manera, si se quiere especificar algo nuevo acerca de las entidades (por ejemplo, el número de páginas), se puede agregar una nueva máquina con esta información sin interferir con las otras.

  Esta solución también requiere de coordinación entre las máquinas. En este caso, tiene que ver con las identidades de las entidades que se están describiendo. ¿Cómo se sabe que la fila 3 en una máquina se refiere a la misma entidad que la fila 3 en otra máquina? Esta solución requiere de un identificador global para las entidades que se están describiendo.

- *Alternativa 3*: consiste en una combinación de las 2 alternativas anteriores, es decir, sobre la tabla A.1 original se hacen particiones horizontales y verticales, con lo cual la información no se distribuye fila por fila ni columna por columna, sino celda por celda (cada máquina es responsable de un cierto número de celdas de la tabla).

  Esta solución combina la flexibilidad de las 2 soluciones anteriores: 2 máquinas pueden compartir la descripción de una única entidad (por ejemplo, el año de Hamlet se guarda en una máquina y el título en otra) y pueden compartir el uso de una propiedad en particular (por ejemplo, los valores de la propiedad Medios para las filas 6 y 7 se guardan en diferentes máquinas).

  Esta flexibilidad es necesaria si se quiere respetar una característica de la Web Semántica en la que “cualquiera puede decir lo que sea sobre cualquier cosa”. De esta forma, cualquier máquina debe poder hacer una declaración acerca de cualquier entidad (como la segunda solución), y también debe poder especificar cualquier propiedad de una entidad (como la primera solución). Esta solución tiene ambos beneficios.

  Pero esta solución también combina los costos de las otras 2: ahora se necesita un identificador global para las columnas y otro para las filas. De hecho, cada celda se tiene que representar con 3 valores: un identificador global para la fila, un identificador global para la columna y el valor de la propia celda. Esta tercera solución es la estrategia adoptada por RDF.

Como una celda se representa con 3 valores, el bloque de construcción básico de RDF es una terna:

- El identificador global para la fila es el *sujeto* de la terna (en gramática, el sujeto es la cosa

sobre la cual trata la oración).
- El identificador global para la columna es el *predicado* de la terna (ya que las columnas especifican las propiedades de las entidades en las filas).
- El valor de la celda es el *objeto* de la terna.

La tabla A.2 muestra algunas de las ternas, correspondientes a la tabla A.1, en forma de sujeto, predicado y objeto.

Sujeto	**Predicado**	**Objeto**
Fila 7	Medio	Poema
Fila 2	Título	Hamlet
Fila 2	Año	1604
Fila 4	Autor	Shakespeare
Fila 6	Medio	Obra de teatro

Tabla A.2 – Ternas correspondientes a la tabla A.1

También, más de una terna puede referenciar la misma entidad. En la tabla A.3 se muestra este caso:

Sujeto	**Predicado**	**Objeto**
Shakespeare	escribió	ReyLear
Shakespeare	escribió	Macbeth
Anne Hathaway	seCasóCon	Shakespeare
Shakespeare	vivióEn	Stratford
Stratford	estáEn	Inglaterra
Macbeth	estáAmbientadaEn	Escocia
Inglaterra	esParteDe	ReinoUnido
Escocia	esParteDe	ReinoUnido

Tabla A.3 – Más de una terna referenciando la misma entidad

Para facilitar la comprensión, al conjunto de ternas conviene verlo como un grafo dirigido, como el que se muestra en la figura A.1. En este grafo, los arcos, dirigidos desde el nodo origen (el sujeto) hacia el nodo destino (el objeto), representan las propiedades (predicados). Tanto lo nodos como los arcos se etiquetan.

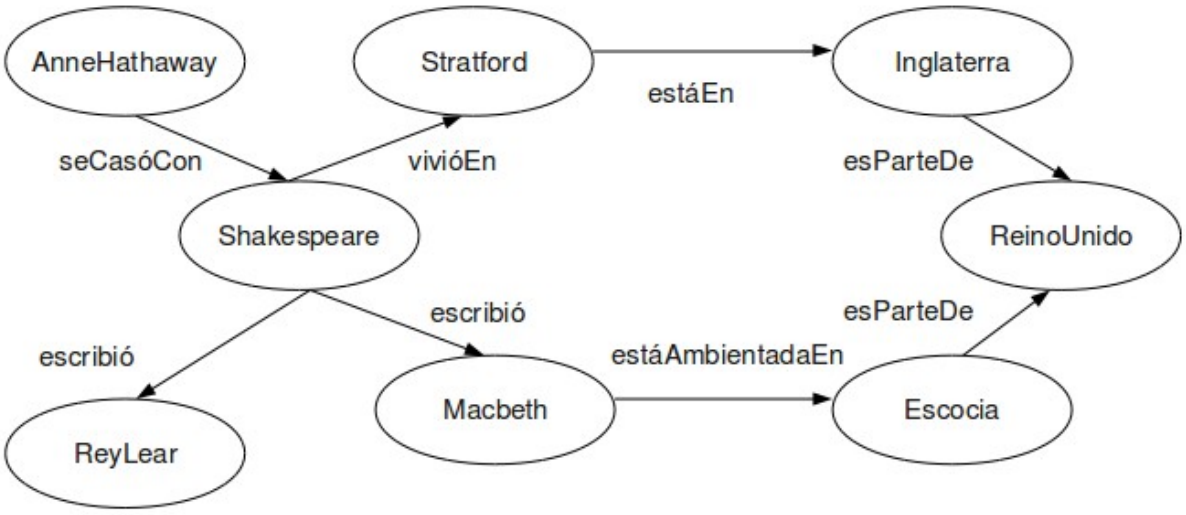

Figura A.1 – Grafo RDF

A.1 Espacios de nombres, URIs e identidad

En el grafo anterior se han etiquetado los nodos y los arcos con nombres simples como "Shakespeare", "Escocia", "escribió", etc. En la Web Semántica, esto no resulta suficiente para identificar algo. Por ejemplo, al hablar sobre Shakespeare, ¿todos saben de lo que se está hablando? En la Web Semántica se tiene que ser más específico: ¿en qué sentido se hace referencia a la palabra "Shakespeare"?

RDF soluciona este problema mediante una tecnología de la Web actual: las URIs[81]. Un tipo especial de URI es la URL[82] u otro tipo de identificador único. Las URIs no sólo han sido definidas para los recursos en la Web sino también para objetos tan diversos como números de teléfono, números de ISBN y ubicaciones geográficas. Es decir, una URI es la identificación global de un recurso Web. Ejemplos de URIs:

- http://www.example.org/file.xml#element(home)
- http://www.example.org/file.html#home
- http://www.example.org/file2.xml#xpath1(//q[@a=b])

Lo importante de todo esto, es que RDF utiliza URIs para identificar los recursos. La forma que tomen las URIs no es importante.

La escritura completa de las URIs suele resultar un poco tedioso. Una alternativa es usar una forma simplificada llamada *qname* [Allemang & Hendler, 2008]. Básicamente, una URI expresada en *qname* tiene 2 partes: un espacio de nombres y un identificador, separados por 2 puntos (:). Por ejemplo, la representación *qname* para el identificador `Inglaterra` en el espacio de nombres

81 Uniform Resource Identifier
82 Uniform Resource Locator

`geo` es `geo:Inglaterra`. En conclusión, cualquier representación de un *qname* debe estar acompañada de una declaración del espacio de nombres correspondiente.

Nota: los *qnames* no son identificadores globales, únicamente las URIs son nombres globales en la Web.

Usando *qnames*, algunas de las ternas anteriores quedarían:

Sujeto	**Predicado**	**Objeto**
lit:Shakespeare	lit:escribió	lit:As You Like It
bio:AnneHathaway	bio:seCasóCon	lit:Shakespeare
lit:Shakespeare	lit:escribió	lit:Love's Labours Lost
lit:Shakespeare	bio:vivióEn	geo:Stratford
lit:Shakespeare	lit:escribió	lit:Hamlet
geo:Stratford	geo:estáEn	geo:Inglaterra
lit:Shakespeare	lit:escribió	lit:Othello
geo:Inglaterra	bio:esParteDe	geo:ReinoUnido

Tabla A.1.1 – Ternas de la tabla A.3 empleando *qnames*

La W3C[83] (Word Wide Web Consortium) ha definido una serie de espacios de nombres estándar, incluyendo *xsd*: para la definición de esquemas XML, *xmlns*: para los espacios de nombres XML, etc:

- *rdf*: espacio de nombres para identificadores utilizados en RDF. El conjunto de identificadores definidos en el estándar es muy pequeño y se utiliza para definir tipos y propiedades en RDF. La URI global para el espacio de nombres *rdf* es http://www.w3.org/1999/02/22-rdf-sintaxis-ns#.

- *rdfs*: espacio de nombres para identificadores utilizados en RDF Esquema o RDFS. La URI global para el espacio de nombres *rdfs* es http://www.w3.org/2000/01/rdf-schema#.

- *owl*: espacio de nombres para identificadores utilizados en OWL. La URI global para el espacio de nombres *owl* es http://www.w3.org/2002/07/owl#.

A.2 Identificadores en el espacio de nombres rdf

Dentro del espacio de nombres `rdf`, RDF define los siguientes identificadores:

83 http://www.w3.org/

- `rdf:type` proporciona un sistema de tipos elemental. Por ejemplo, se puede expresar la relación entre varios escritores empleando información de tipo. El sujeto de `rdf:type` en estas ternas puede ser cualquier identificador, y el objeto se entiende que es un tipo.

Sujeto	**Predicado**	**Objeto**
lit:Shakespeare	rdf:type	lit:Escritor
lit:Miller	rdf:type	lit:Escritor

Tabla A.2.1 – Ejemplo de uso del identificador rdf:type

- `rdf:Property` es un identificador que se usa en RDF para indicar cuándo otro identificador se va a usar como predicado en lugar de sujeto u objeto:

Sujeto	**Predicado**	**Objeto**
lit:escribió	rdf:type	rdf:Property
geo:parteDe	rdf:type	rdf:Property
bio:seCasó	rdf:type	rdf:Property

Tabla A.2.2 – Ejemplo de uso del identificador rdf:Property

A.3 Notaciones para RDF

Para publicar datos RDF en la Web, resulta adecuado la representación en forma de texto más que la representación en forma de grafo. Existen distintas maneras de expresar RDF en forma textual. Algunas de las representaciones textuales más comunes para RDF son:

- **Ternas N**

  Esta notación referencia los recursos mediante sus URIs completas. Cada URI se escribe entre los símbolos de mayor y menor (< y >). Los recursos se expresan de la forma sujeto-predicado-objeto, seguidos por un punto (`.`). Por ejemplo:

```
<http://www.WorkOnt.org/Examples/C3Manufacture.rdf#Producto1>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.WorkOnt.org/Examples/C3Manufacture.rdf#Producto> .
```

  En este ejemplo, por falta de espacio aparece la terna en 3 renglones; sólo el punto al final indica el final de una terna.

- **3 RDF (N3)**

  La notación 3 RDF (o N3) fue desarrollada por Tim Berners-Lee. N3 combina la notación anterior con el uso de *qnames*. N3 es bastante extenso, y sólo se mostrará una parte.

  Como N3 utiliza *qnames*, debe haber una asociación entre los *qnames* (locales) y las URIs (globales). Por lo tanto, N3 comienza con un preámbulo en el que se definen estas asociaciones. Por ejemplo:

  ```
  @prefix mfg:
  <http://www.WorkOnt.com/Examples/C3/Manufacturing.rdf#>
  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  ```

  Una vez que se han definido los *qnames* locales, N3 proporciona una forma muy simple de expresar una terna utilizando las abreviaturas *qnames*, en el orden sujeto-predicado-objecto, seguido de un punto (`.`):

  ```
  mfg:Product1 rdf:type mfg:Product .
  ```

- **RDF-XML**

  Es la forma más popular y conveniente para la Web Semántica [Chase, 2007B]. Un documento RDF se representa mediante un elemento, llamado `rdf:RDF`, dentro del cual hay varios elementos llamados `rdf:Description`, los cuales representan las descripciones de los recursos.

  Toda descripción hace una declaración sobre un recurso, el cual se puede identificar mediante un atributo llamado `about`, para hacer referencia a un recurso existente, un atributo llamado `ID`, para crear un recurso nuevo, o bien el recurso puede no llevar nombre, para crear un recurso anónimo:

  ```
  <rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:ex="http://example.com#"
     xmlns:dc="http://purl.org/dc/elements/1.1/#">
     <rdf:Description rdf:about="http://www.chaosmagnet.com">
        <ex:nombre>Chaos Magnet</ex:name>
        <dc:creador rdf:resource="http://www.nicholaschase.com" />
     </rdf:Description>
  </rdf:RDF>
  ```

RDF describe cómo especificar información sobre un recurso, pero la información en sí misma no significa nada. Para definir los tipos de objetos y sus relaciones, como clases y subclases, es necesario hacer algunos agregados al vocabulario. Estos agregados se encapsulan en RDF Schema, también conocido como RDFS, el cual tiene su propio espacio de nombres y una serie de elementos y atributos predefinidos.

APÉNDICE B - RDFS

El nombre oficial es RDF Vocabulary Description Language, pero por razones históricas se emplea el término “Schema”.

RDF crea simplemente una estructura en forma de grafo para representar los datos, mientras que RDFS proporciona el vocabulario que se utilizará en este grafo [Allemang & Hendler, 2008]. ¿Qué individuos se relacionan entre sí, y cómo? ¿Cómo están relacionadas las propiedades que se usan para definir a los individuos con otros conjuntos de individuos y entre sí? RDFS proporciona una forma de expresar las respuestas a este tipo de preguntas.

Siguiendo con el ejemplo anterior, se dice que hay escritores que escriben determinados libros. Si sólo se quiere hablar sobre el escritor `E` o el libro `L`, se puede utilizar RDF. Pero RDF no tiene la capacidad suficiente para hablar sobre *los* libros, *los* libros de ficción, *los* escritores, etc., es decir, sobre los conjuntos (clases) que definen los tipos de recursos. Tampoco tiene RDF la capacidad de hablar sobre las relaciones entre estos conjuntos, es decir, las relaciones entre los escritores y los libros. Para definir un vocabulario mediante el cual se puedan definir los distintos conceptos (clases) y sus relaciones, es necesario hacer algunos agregados, los cuales se encapsulan en RDFS.

La construcción básica para especificar un conjunto en RDFS es una clase, la cual se define en el espacio de nombres *rfds* (`rdfs:Class`). Como RDFS se expresa en RDF, la forma de expresar que algo es una clase es también mediante una terna, en la cual el predicado es `rdf:type` y el objeto `rdfs:Class`:

Sujeto	**Predicado**	**Objeto**
lit:Escritor	rdf:type	rdfs:Class

Tabla B.1 – Ejemplo de uso del identificador `rdfs:Class`

Al ver este ejemplo, la única manera de saber que se está hablando sobre el conjunto es el empleo de `rdfs:Class`. En RDF, el concepto `rdf:type` se utiliza para especificar que algo *es miembro de* un conjunto. En RDFS se especifica que algo *es* un conjunto.

En RDFS, al igual que en la mayoría de los lenguajes de programación, uno de los usos más importantes de las clases es imponer restricciones sobre qué se puede afirmar sobre un determinado concepto. Por ejemplo, en el ejemplo de los escritores, se quiere evitar declaraciones como las siguientes:

- Libro X es escrito por Libro Y
- El aula A es escrita por el Escritor B

El primer caso no tiene sentido porque se quiere que los libros sean escritos sólo por escritores, lo cual impone una restricción a los valores de la propiedad “*es escrito por*” (se restringe el rango de la

propiedad).

El segundo caso no tiene sentido porque sólo se pueden escribir libros, lo cual impone una restricción a los objetos sobre los cuales se puede aplicar la propiedad (se restringe el dominio de la propiedad).

B.1 Subclases

Una vez definidas las clases, se deberán establecer las relaciones entre las mismas mediante una jerarquía de clase y subclase.

Suponer que se define un vocabulario que dice que "*todo Futbolista es un Deportista, y que X es un Futbolista*". Una persona puede inferir que X es un Deportista.

Uno de los desafíos en el modelado del vocabulario en general (y en particular RDFS) es diferenciar los dos usos de "*es un*" en el ejemplo. Cuando se dice que "*Futbolista es un Deportista*", se están relacionando 2 tipos entre sí, pero cuando se dice "*X es un Futbolista*", se está dando información sobre el tipo de un individuo en particular, es decir, se relaciona un individuo a un tipo. En RDF se puede proporcionar información sobre el tipo de un individuo:

Sujeto	**Predicado**	**Objeto**
bio:X	rdf:type	dep:Futbolista

Tabla B.1.1 – Ejemplo de uso del identificador `rdf:type`

RDFS proporciona un nuevo recurso, `rdfs:subClassOf`, para expresar la relación "*es un*" entre tipos:

Sujeto	**Predicado**	**Objeto**
dep:Futbolista	rdfs:subClassOf	dep:Deportista

Tabla B.1.2 – Ejemplo de uso del identificador `rdfs:subClassOf`

En RDFS no se requiere que las clases formen una jerarquía estricta, con lo cual una clase puede tener varias superclases.

B.2 Subpropiedades

RDFS proporciona un medio por el cual las clases se pueden relacionar entre sí por medio de la relación de subclase y las inferencias que definen el significado de esta construcción. Esto permite a quien modela en RDFS describir (utilizando la noción de conjuntos) las relaciones entre los

elementos que se describen en una colección de ternas RDF. Pero si se quiere dar un significado a los datos, se necesita algo más que hablar sobre los elementos: se tiene que hablar sobre las propiedades que los vinculan, es decir, los predicados de las ternas.

RDFS proporciona un mecanismo sencillo para esto, basado en el recurso `rdfs:subPropertyOf`. Los conceptos de clases, subclases y propiedades tienen sus similitudes y diferencias con los mismos conceptos de la programación orientada a objetos: en esta última, una clase define sus propiedades, y el agregado de nuevas propiedades implica la modificación de la clase. En el caso de RDFS, las propiedades se definen globalmente (no se encapsulan como atributos en las definiciones de las clases). Es posible definir nuevas propiedades que se apliquen a clases existentes sin modificar las mismas.

La intuición dice que esta construcción es análoga a `rdfs:subClassOf`, pero para las propiedades. Por ejemplo la relación "*hermano*" es más específica que la relación "*pariente*": si alguien es mi *hermano*, entonces también es mi *pariente*.

B.3 Tipo y rango de las propiedades

Se puede decir que cuando se utiliza una propiedad, el sujeto de la terna viene de una cierta clase y el objeto proviene de otra. Esto se expresa en RDFS con los recursos `rdfs:domain` y `rdfs:range`, respectivamente.

El dominio se refiere al sujeto de cualquier terna que utiliza P como su predicado, y el rango al objeto de cualquier terna que utiliza a P como predicado. Cuando se afirma que la propiedad P tiene dominio D (o rango R), se está diciendo que cada vez que se use la propiedad P, se puede inferir que el sujeto (o el objeto) de esa terna es un miembro de la clase D (o R).

APÉNDICE C – TÉCNICAS PARA EL MAPEO DE ONTOLOGÍAS

Una posible división de las técnicas para mapear ontologías puede ser [Euzenat & Shvaiko, 2007]:

- *Las que trabajan a nivel elemento*: consideran las entidades de las ontologías en forma aislada de sus relaciones con otras entidades.

- *Las que trabajan a nivel estructura*: consideran a las entidades de las ontologías para comparar sus relaciones con otras entidades, en lugar de, o además de comparar sus nombres o identificadores.

- *Las que trabajan a nivel instancia*: consideran las instancias para encontrar similitudes.

- *Las que se basan en la semántica*: la característica principal de estas técnicas es que usan la semántica de la teoría de modelos para justificar sus resultados.

C.1 Técnicas que trabajan a nivel elemento

- *Técnicas basadas en cadenas de caracteres*

  Se utilizan generalmente para buscar coincidencias entre los nombres y las descripciones de las entidades de las ontologías. Estas técnicas consideran una cadena como una secuencia de letras. Normalmente se basan en la siguiente intuición: cuanto más similares sean las cadenas, más probable es que indiquen los mismos conceptos.

  Hay muchas maneras de comparar cadenas según la forma en que se las considere: como una secuencia exacta de letras, como una secuencia errónea de letras, como un conjunto de letras, como un conjunto de palabras.

  Antes de comparar cadenas de caracteres, existen procedimientos de normalización que pueden ayudar a mejorar los resultados de las comparaciones posteriores. Por ejemplo, se puede hacer una **normalización** de las cadenas (se convierten todos los caracteres a mayúsculas por ejemplo), una **supresión de diacríticos** (se reemplazan los caracteres con signos diacríticos, por ejemplo Montréal por Montreal), **normalización de blancos** (se reemplazan todos los caracteres blancos, como el espacio en blanco, tabulación, etc, en un único caracter blanco), **supresión de dígitos**, **eliminación de puntuación** (se suprimen los signos de puntuación. Por ejemplo “C.D.” se transforma en “CD”), etc.

  Estas operaciones de normalización se deben hacer con cuidado, ya que entre otras cosas, son muy dependientes del lenguaje y del orden, pueden causar pérdida de información, pueden reducir variaciones pero incrementar sinónimos (por ejemplo “solo” y “sólo”), etc.

Una vez aplicada alguna operación de normalización, se puede emplear la función de igualdad de cadenas, la cual devuelve 1 si ambas cadenas son idénticas, y 0 si no lo son. Esta medida no explica cómo difieren 2 cadenas, con lo cual se puede emplear la distancia de Hamming, la cual es una función que cuenta el número de posiciones en las cuales difieren las cadenas.

También, según el resultado obtenido de la función de igualdad de cadenas, se pueden obtener diferentes variaciones, como considerar que las cadenas son muy similares cuando una es subcadena de la otra. De esta forma, si una cadena es subcadena de otra, se puede analizar la parte común de ambas.

También se puede emplear la similitud del tipo "n-gramo" para comparar cadenas. Esta similitud calcula el número de n-gramos comunes, es decir, secuencias de n caracteres. Por ejemplo, los trigramos para la cadena "artículo" serían: "art", "rtí", "tíc", "ícu", "cul" y "ulo". Esta función resulta eficiente cuando hay diferencia en unos cuantos caracteres.

- *Técnicas basadas en el lenguaje*

  Las técnicas basadas en cadenas de caracteres consideran las cadenas como secuencias de caracteres. Con las técnicas basadas en el lenguaje, las cadenas son vistas como textos (por ejemplo, "artículo de revista teórico revisado por pares"). Los textos se pueden segmentar en palabras: secuencias de letras fácilmente identificables ("teórico", "pares", "revisado", "revista", "artículo"). Estas palabras no están sueltas sino en una secuencia con una estructura gramatical. Frecuentemente palabras como "pares" tienen un significado y se corresponden con algunos conceptos, pero los conceptos más útiles para ser manipulados en un texto son los términos, tales como "revisado por pares".

  Los términos son frases que identifican conceptos. Así, frecuentemente son utilizados para etiquetar los conceptos en las ontologías. Como consecuencia, reconocer e identificar los términos en las cadenas puede ayudar mucho en el mapeo de ontologías.

  Las técnicas basadas en el lenguaje se basan en las técnicas del procesamiento del lenguaje natural para ayudar a extraer los términos significativos de un texto. Comparar estos términos y sus relaciones ayuda a evaluar la similitud de las entidades de las ontologías. Aunque estas técnicas se basan en algunos conocimientos lingüísticos, hay técnicas que se basan sólo en algoritmos y técnicas que hacen uso de recursos externos, como diccionarios.

- *Técnicas basadas en restricciones*

  Son algoritmos que se ocupan de las restricciones internas que se aplican a las definiciones de las entidades, como tipo de dato, cardinalidad, clave, etc.

- *Recursos lingüísticos*

  Los recursos lingüísticos, como diccionarios o tesauros de dominio específico, se utilizan

con el fin de buscar coincidencias entre las palabras según las relaciones lingüísticas entre las mismas, por ejemplo, sinónimos, hipónimos (un hipónimo es una palabra cuyo significado es más específico que el de otra en la que está englobada).

- *Reutilización de alineaciones*

  Representan una forma alternativa de explotar recursos externos, los cuales tienen el registro de las alineaciones de ontologías a las cuales ya se les buscaron coincidencias (por ejemplo, cuando se tienen que mapear las ontologías O2 y O3, teniendo en cuenta las alineaciones entre O1 y O2, y entre O1 y O3, disponibles en el recurso externo). La reutilización de alineaciones está motivada por la intuición de que muchas de las ontologías a mapear son similares a las ya mapeadas, sobre todo si describen el mismo dominio de aplicación. Estas técnicas son especialmente prometedoras cuando se trata con ontologías grandes consistentes en cientos y miles de entidades.

- *Ontologías de nivel superior y de dominio específico*

  Las ontologías de nivel superior también se pueden utilizar como fuentes externas de conocimiento. Ejemplos son la ontología Cyc [Lenat & Guha, 1990], SUMO (Suggested Upper Merged Ontology) [Niles & Pease, 2001] y DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [Gangemi *et al.*, 2003]. La característica fundamental de estas ontologías es que son sistemas basados en la lógica, y por lo tanto, las técnicas de comparación que las explotan están basadas en la semántica.

C.2 Técnicas que trabajan a nivel estructura

Las técnicas que trabajan a nivel estructura se pueden subdividir en:

- *Técnicas de estructura interna*: comparan la estructura interna de una entidad, es decir, además de su nombre, sus propiedades (la estructura interna es la definición de entidades sin referencia a otras entidades). Estas técnicas usan criterios como el conjunto de propiedades de una entidad, rango (atributos y relaciones), cardinalidad, y transitividad o simetría de estas propiedades para calcular la similitud entre las entidades.

  Las entidades con estructuras internas comparables o propiedades con dominios y rangos similares en 2 ontologías pueden ser muchas. Por esta razón, estos tipos de técnicas se utilizan para crear grupos de correspondencia en lugar de descubrir las correspondencias exactas entre entidades.

  Por lo general se combinan con técnicas a nivel elemento y son responsables de reducir el número de correspondencias candidatas. Se pueden utilizar con otros enfoques a modo de paso previo para eliminar la mayor parte de las propiedades que son claramente incompatibles.

- *Técnicas de estructura relacional*: comparan la entidad con otras entidades con las que se relaciona. La estructura relacional es el conjunto de relaciones que tiene una entidad con otras entidades.

  Estas técnicas consideran una ontología como un grafo cuyas aristas están etiquetadas por los nombres de las relaciones. Por lo general, la comparación de similitud entre un par de nodos de 2 ontologías se basa en el análisis de sus posiciones dentro de los grafos. La intuición detrás de esto es que, si 2 nodos de 2 ontologías son similares, sus vecinos también deben ser similares de alguna forma.

C.3 Técnicas que trabajan a nivel instancia

Cuando 2 ontologías comparten el mismo conjunto de individuos (instancias), hay muchas probabilidades que exista una correspondencia entre ambas.

Estas técnicas aprovechan una muestra representativa de una población con el fin de encontrar regularidades y discrepancias. Esto ayuda a agrupar elementos o calcular distancias entre ellos.

Entre estas técnicas están las basadas en la distancia, el análisis de conceptos formales, el análisis de correspondencias, el de distribución de frecuencias, etc.

C.4 Técnicas basadas en la semántica

Estas técnicas emplean la interpretación semántica: si 2 entidades son las mismas, comparten las mismas interpretaciones. En consecuencia, se trata de métodos deductivos.

Los métodos puramente deductivos no funcionan muy bien por si solos en cuanto a la concordancia de ontologías, y por lo tanto necesitan de una fase de preprocesamiento que brinde las “anclas”, es decir, entidades que se declaren, por ejemplo, como equivalentes (según la identidad de sus nombres por ejemplo).

BIBLIOGRAFÍA

[Allemang, 2007] Allemang D. (2007) *Mashups and Semantic Mashups*. Java Developer's Journal. Disponible en http://java.sys-con.com/node/361294 (accedido el 9 de Agosto de 2011)

[Allemang & Hendler, 2008] Allemang D. & Hendler J. (2008) *Semantic Web for the Working Ontologist. Effective Modeling in RDFS and OWL*. Morgan Kaufmann.

[Antoniou & van Harmelen, 2008] Antoniou G. & van Harmelen F. (2008) *A Semantic Web Primer (Second Edition).* The MIT Press.

[Berners-Lee *et al.*, 2001] Berners-Lee T., Hendler J. & Lassila O. (2001) *The Semantic Web*. Scientific American.

[Bolchini & Garzotto, 2008] Bolchini D. & Garzotto F. (2008) *Designing Multichannel Web Applications as "Dialogue Systems": The IDM Model*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Bouquet *et al.*, 2004] Bouquet P., Ehrig M., Euzenat J., Franconi E., Hitzler P., Krötzsch M., Serafini L., Stamou G., Sure Y., and Tessaris S. (2004) *Specification of a common framework for characterizing alignment*. Deliverable D2.2.1, Knowledge web NoE, 2004.

[Brambilla *et al.*, 2008] Brambilla M., Comai S., Fraternali P. & Matera M. (2008) *Designing Web Applications with WebML and WebRatio*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Browne & Jermey, 2004] Browne G. & Jermey J. (2004) *Website Indexing (2nd Edition)*. Auslib Press.

[Cardoso, 2007] Cardosi J. (2007) *Semantic Web Services: Theory, Tools and Applications*. Information Science Reference

[Chase, 2007A] Chase N. (2007A) *The ultimate mashup – Web services and the semantic Web, Part 1: Use and combine Web services*. IBM Developer Works.

[Chase, 2007B] Chase N. (2007B) *The ultimate mashup – Web services and the semantic Web, Part 3: Understand RDF and RDFs*. IBM Developer Works.

[Chase, 2007C] Chase N. (2007C) *The ultimate mashup – Web services and the semantic Web, Part 6: Give the user control*. IBM Developer Works.

[Chase & Mitri, 2007] Chase N. & Mitri M. (2007) *The ultimate mashup – Web services and the semantic Web, Part 5: Change out Web services*. IBM Developer Works.

[Chase & Peterson, 2006] Chase N. & Peterson T. (2006) *The ultimate mashup – Web services and the semantic Web, Part 2: Manage a mashup data cache*. IBM Developer Works.

[Chatti *et al.*, 2009] Chatti M.A., Jarke M., Wang Z. & Specht M. (2009) *SMashup Personal Learning Environments*.

[Daconta *et al.*, 2003] Daconta M.C., Obrst L.J. & Smith K.T. (2003) *The Semantic Web. A Guide to the Future of XML, Web Services and Knowledge Management*. Wiley.

[Davies *et al.*, 2006] Davies J., Studer R. & Warren P. (2006) *Semantic Web Technologies. Trends and research in ontology-based systems*. Wiley.

[De Troyer & Leune, 1998] De Troyer O. & Leune C. (1998) *WSDM: A user-centered design method for Web sites*. Computer Networks and ISDN Systems, Proceedings 7th International World Wide Web Conference, Elsevier, Brisbane, Australia, pp. 85–94.

[De Troyer *et al.*, 2008] De Troyer O., Casteleyn S. & Plessers P. (2008) *WSDM: Web Semantics Design Method*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Doran, 2006] Doran P. (2006) *Ontology reuse via ontology modularisation*. Department of Computer Science, University of Liverpool.

[Euzenat, 2001] Euzenat J. (2001) *Towards a principled approach to semantic interoperability*. In Proc. IJCAI Workshop on Ontologies and Information Sharing, pages 19–25, Seattle (WAUS).

[Euzenat & Shvaiko, 2007] Euzenat J. & Shvaiko P. (2007) *Ontology Matching*. Springer.

[Euzenat & Stuckenschmidt, 2003] Euzenat J. and Stuckenschmidt H. (2003) *The 'family of languages' approach to semantic interoperability*. In Borys Omelayenko and Michel Klein, editors, Knowledge transformation for the semantic web, pages 49–63. IOS press, Amsterdam (NL)

[Feiler, 2008] Feiler J. (2008) *How To Do Everything with Web 2.0 Mashups*. Mc Graw Hill.

[Fensel *et al.*, 2006] Fensel D., Lausen H., Polleres A., Bruijn J., Stollberg M., Roman D. & Domingue J. (2006) *Enabling Semantic Web Services. The Web Service Modelling Ontology*. Springer.

[Fischer *et al.*, 2009] Fischer T., Bakalov F. & Nauerz A. (2009) *An Overview of Current Approaches to Mashup Generation*. University of Jena & IBM Research and Development

[Fons *et al.*, 2008] Fons J., Pelechano V., Pastor O., Valderas P. & Torres V. (2008) *Applying the OOWS Model-Driven Approach for Developing Web Applications. The Internet Movie Database Case Study*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Gangemi *et al.*, 2003] Gangemi A., Guarino N., Masolo C. & Oltramari A. (2003) *Sweetening WordNet with DOLCE*. AI Magazine.

[García-Castro *et al.*, 2009] García-Castro R., Gómez-Pérez A. & Sini M. (2009) *NeOn methodology for the development of large-scale semantic applications*. NeOn Project.

[Gruber, 1993] Gruber T. (1993) *A Translation Approach to Portable Ontology Specifications*. Knowledge Acquisition 5, 199-220

[Hart *et al.*, 2004] Hart L., Emery P., Colomb B., Raymond K., Taraporewalla S., Chang D., Ye Y., Kendall E. & Dutra M. (2004) *OWL full and UML 2.0 compared*. Disponible en http://www.itee.uq.edu.au/~colomb/Papers/UML-OWLont04.03.01.pdf.

[Herman, 2009] Herman I. (2009) *Introduction to the Semantic Web (tutorial)*. W3C.

[Horridge, 2009] Horridge M. (2009) *A Practical Guide to Building OWL Ontologies Using Protégé 4 and CO-ODE Tools (Edition 1.2)*. The University of Manchester.

[Houben *et al.*, 2008] Houben G., van der Sluijs K., Barna P., Broekstra J., Casteleyn S., Fiala Z. & Frasincar F. (2008) *Hera*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[IEEE, 1997] IEEE (1997) *IEEE Standard for Developing Software Life Cycle Processes*. IEEE Standard 1074-1997, IEEE Computer Society.

[Kanehisa *et al.*, 2004] Kanehisa M., Goto S., Kawashima S., Okuno Y. & Hattori M. (2004) *The KEGG resources for deciphering the genome*. Nucleic Acids Res. 2004;32:D277-D280.

[Klein, 2001] Klein M. (2001) *Combining and relating ontologies: an analysis of problems and solutions*. Vrije Universiteit Amsterdam.

[Koch *et al.*, 2008] Koch N., Knapp, A., Zhang, G. & Baumeister H. (2008) *UML-Based Web Engineering. An Approach Based on Standards*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Kruchten, 2000] Kruchten P. (2000) *The Rational Unified Process: An Introduction (Second Edition)*. Addison-Wesley.

[Lenat & Guha, 1990] Lenat D. & Guha R. (1990) *Building large knowledge-based systems*. Addison Wesley, Reading (MA US).

[Maglott *et al.*, 2005] Maglott D., Ostell J., Pruitt K. D., & Tatusova T. (2005) *Entrez Gene: gene-centered information at NCBI*. Nucleic Acids Res. 2005 January 1;33((Database Issue)):D54–D58.

[Makki, 2008] Makki S. K. & Sangtani J. (2008) *Data Mashups & Their Applications in Enterprises*. Internet and Web Applications and Services, 2008. ICIW '08. Third International

Conference Athens: IEEE, 2008, pp. 445-450.

[McGuinness *et al.*, 2000] McGuinness D. L., Fikes R., Rice J. & Wilder S. (2000) *An Environment for Merging and Testing Large Ontologies*. Principles of Knowledge Representation and Reasoning, Proceedings of the Seventh International Conference (KR2000). A. G. Cohn, F. Giunchiglia & B. Selman, editors. San Francisco, CA, Morgan Kaufmann Publishers.

[McIlraith *et al.*, 2001] McIlraith S., Son T. C. & Zeng H. (2001) *Semantic web services. IEEE Intelligent Systems, Special Issue on the Semantic Web*. IEEE

[Mitri & Chase, 2006] Mitri M. & Chase N. (2006) *The ultimate mashup – Web services and the semantic Web, Part 4: Create an ontology*. IBM Developer Works.

[Murugesan, 2008] Murugesan S. (2008) *Web Application Development: Challenges and the Role of Web Engineering*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Musen, 1992] Musen M. A. (1992) *Dimensions of knowledge sharing and reuse*. Computers and Biomedical Research 25: 435-467.

[Muthaiyah, 2008] Muthaiyah S. (2008) *A framework and methodology for ontology mediation through semantic and syntactic mapping*. Trabajo presentado para cumplir con el Grado de Doctor en Filosofía en Tecnología de la Información en la Universidad George Mason. Disponible en http://digilib.gmu.edu:8080/handle/1920/3070.

[Niles & Pease, 2001] Niles I. & Pease A. (2001) *Towards a standard upper ontology*. In Proc. 2nd International Conference on Formal Ontology in Information Systems (FOIS), Ogunquit (ME US).

[Noy & McGuinness, 2001] Noy N. F & McGuinness D. L. (2001) *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880

[Paolucci *et al.*, 2002] Paolucci M., Kawamura T., Payne T. R. & Sycara K. (2002) *Importing the Semantic Web in UDDI*. Proceedings of the Web Services, E-Business and Semantic Web Workshop

[Preist, 2004] Preist C. (2004) *A conceptual architecture for semantic web services*. En: McIlraith S. A., Plexousakis D. & van Harmelen F. (2004) *The Semantic Web – ISWC 2004: Third International Semantic Web Conference*. Springer.

[Raza *et al.*, 2008] Raza M., Hussain F. k. & Chang E. (2008) *A methodology for quality-based mashup of data sources*. Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services.

[Rebollo Pedruelo, 2008] Rebollo Pedruelo M. (2008) *Servicios Web Semánticos*. Departamento Sistemas Informáticos y Computación - Universidad Politécnica de Valencia. Disponible en:

http://www.slideshare.net/mrebollo/serviciosweb

[Rossi *et al.*, 2008] Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Rossi & Schwabe, 2008] Rossi G. & Schwabe D. (2008) *Modeling and Implementing Web Applications with OOHDM*. En: Rossi G., Pastor O., Schwabe D. & Olsina L. (2008) *Web Engineering: Modeling and Implementing Web Applications*. Springer.

[Sahoo *et al.*, 2008] Sahoo S. S, Bodenreider O., Rutter J. L., Skinner K. J. & Sheth A. P. (2008) *An ontology-driven semantic mashup of gene and biological pathway information: Application to the domain of nicotine dependence*. Journal of Biomedical Informatics, Volume 41, Issue 5, Pages 752–765

[Sommerville, 2007] Sommerville I. (2007) *Software Engineering (Eighth edition)*. Wesley.

[Staab *et al.*, 2004] Staab S., Gomez-Perez A., Daelemana W., Reinberger M. L., & Noy N. F. (2004) *Why evaluate ontology technologies? because it works!*. IEEE Intelligent Systems, 19, 74–81.

[Taylor *et al.*, 2008] Taylor J., Evans C. & Segaran T. (2008) *Creating Semantic Mashups: Bridging Web 2.0 and the Semantic Web*. freebase.

[Thiran *et al.*, 2005] Thiran P., Hainaut J.L. & Houben G.J. (2005) *Database wrappers development: Towards automatic generation*. Ninth European Conference on Software Maintenance and Reengineering, CSMR'05, Manchester, UK, IEEE CS Press, pp. 207–216.

[Udrea *et al.*, 2007] Udrea O., Getoor L. & Miller R. J. (2007) *Leveraging Data and Structure in Ontology Integration*. SIGMOD '07, Beijing, China

[Yu, 2011] Yu L. (2011) *A developer's guide to the Semantic Web*. Springer

[Walton, 2007] Walton C. D. (2007) *Agency and the Semantic Web*. Oxford University Press.