UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DE UNA RED SEMÁNTICA DE ARCHIVOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

JUAN ENRIQUE MUÑOZ ZOLOTOOCHIN

PROFESOR GUÍA:
CLAUDIO GUTIERREZ GALLARDO

MIEMBROS DE LA COMISIÓN:
BÁRBARA JEANNETTE POBLETE LABRA
GONZALO NAVARRO BADINO

SANTIAGO DE CHILE
NOVIEMBRE 2011

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

IMPLEMENTACIÓN DE UNA RED SEMÁNTICA DE ARCHIVOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JUAN ENRIQUE MUÑOZ ZOLOTOOCHIN

PROFESOR GUIA:
CLAUDIO GUTIÉRREZ GALLARDO

INTEGRANTES DE LA COMISIÓN:
BÁRBARA JEANNETTE POBLETE LABRA
GONZALO NAVARRO BADINO

SANTIAGO DE CHILE
NOVIEMBRE 2011

Resumen

El almacenamiento de contenido en nuestro computador es ineficaz. Frente al problema de encontrar un cierto contenido en el disco duro estamos obligados a recordar nombres de archivos, extensiones y carpetas, los cuales distan mucho de reflejar el verdadero contenido que hay en cada uno de ellos. Una indexación efectiva no sólo debe permitir el uso de metadatos altamente específicos al tipo de contenido que se está almacenando, sino también reducir la ambigüedad de conceptos y explicitar las relaciones entre los elementos indexados.

En esta memoria de título se propone un sistema de indexación semántica de archivos basado en RDF, aprovechando la flexibilidad de esta forma de representación de metadatos para construir un índice colaborativo de acceso público. Cada archivo es indexado mediante un identificador que se construye a partir de una función de *hashing* sobre él mismo, el cual es a la vez la URL donde se despliegan la información del contenido y sus relaciones con otros archivos. Se le da especial énfasis a la accesibilidad de la información, contando con interfaces de usuario gráficas así como programáticas para el despliegue y captura de datos.

Durante el desarrollo de esta memoria se encontraron mucos obstaculos, los cuales radican principalmente en la baja adopción de tecnologías de la Web Semántica sobre las cuales se basa el prototipo implementado. De igual importancia que el sistema de indexación antes mencionado, son los documentos y las herramientas que se desarrollaron en el camino para superar estas dificultades. Se buscó ampliamente que el proyecto realizado sirva como una instancia educativa, no solo para el autor de esta memoria de título, sino para todos quienes se interesen en estas áreas de desarrollo. Todas las herramientas desarrolladas se liberaron de forma abierta bajo licencia MIT.

Agradecimientos

En primer lugar, gracias a mis padres, quienes son los verdaderos responsables de todos mis logros académicos, pues proveyeron la base y el sustento de toda mi educación.

A mi profesor guía, Claudio Gutierrez, quien hace un año dictó el curso a partir del cual surgió el tema de esta memoria de título, y me guió en el proceso de su desarrollo. Fundamentales fueron en este proceso también Alvaro Graves, Daniel Hernandez, Manuel Ortega y Aldo Bucchi, parte de la pequeña comunidad de chilenos involucrados en la iniciativa de *Open Data* y la adopción de RDF como estandar de publicación de datos. Así como la fundación Ciudadano Inteligente, con Felipe Heusser y Montserrat Lobos, donde di mis primeros pasos en ese mundo.

Agradezco a la Universidad de Chile en su conjunto, así como a la Ecole Centrale de Marsella, donde realizé una parte importante de mis estudios. En este contexto, agradezco en particular a Rocío Duque e Isabel Amor por hacer posible mi intercambio en Francia, una de las experiencias más enriquecedoras que he tenido, a nivel tanto personal como profesional. Gracias a quienes me acompañaron en esa aventura, a Enrique Hederra, Nicolás Inostroza, Cristián Prado, Allan Cid, Alexandre Schinazi, Pablo Eduardo Mota, Albert Nissimov, Deborah Flinker, André Bazani y muchos otros.

Gracias a los profesores que marcaron mi aprendizaje: Alejandro Mass, Juan Diego Dávila, Patricio Aceituno, Alejandro Hevia, Gonzalo Navarro, Eric Tanter y Bruno Blaive (Francia). Y a Mario Merino, quien a través del piano me enseñó grandes lecciones de vida.

Este documento representa más que un trabajo de fin de carrera, representa un proceso de más de 6 años de aprendizaje, los mejores de mi vida. Así, no puedo dejar de mencionar a mis amigos de la Sección 5: a Alonso Olate, Sebastián Pizarro, David Rosales, Werner Lange, Ignacio Escobar, Miguel Pavez, Marcela Huerta, Marilia Soto, Sebastián Gamboa, Daniel Herrmann, Rodrigo Diaz, Camila Mosso, Rodrigo Martinez, Marcel Augsburger, Pablo Tapia, Camilo Vergara y Margarita Amaya. Y a mis amigos salseros: Felipe Espinoza, Camilo Lopez, Matthew de los Santos, Paola Veliz, Daniel Aliaga y Andrés Rebolledo, por acompañarme a descubrir un nuevo aspecto de mi vida. Gracias a Larry González y Nicolás Velasquez, grandes amigos a quienes conocí en los últimos períodos de mi carrera.

Finalmente, gracias a aquellas personas que me introdujeron al mundo de la ingeniería:

Luis Aburto y Antonio Diaz, quienes me dieron un adelanto de lo que sería mi vida profesional, y en especial a Sebastián Acuña, de quien aprendí las bases de practicamente todo lo que sé en computación.

Índice general

Índice de códigos

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Diariamente usamos computadores para procesar y consumir información. Ésta se encuentra almacenada muchas veces en forma de archivos en nuestros discos duros y medios extraíbles, los cuales enviamos por e-mail o subimos y bajamos de la Web. No es raro que un comptuador personal contenga cientos de miles de archivos, considerando documentos, música, videos, libros, imágenes, archivos de sistema, etc. los cuales se suelen organizar en carpetas. Los usuarios tienen conocimiento del contenido de una pequeña fracción de estos, normalmente aquellos que acceden regularmente, y suelen tener una noción vaga del contenido general de su computador: son muchos archivos para llevar la cuenta y gran parte de ellos no son del interés del usuario. Este desconocimiento de los contenidos del computador se hace evidente cuando el usuario se ve enfrentado a la pregunta de si tiene o no un determinado archivo o contenido. Por ejemplo:

- "Quicktime no puede encontrar 'qtfc.dll'."
- "¿Tienes el Control 2 de CC40A del año 2008?"
- "¡Mándame la última versión del informe de laboratorio!"

A veces la intepretación del problema es clara, en el primer caso es evidente que el archivo 'qtfc.dll' no se encuentra, pero ¿qué es ese archivo? ¿dónde debería estar? Usuarios más avanzados se dan cuenta por su extensión *.dll* que es una librería de Windows y se

formulan aún más preguntas: ¿hay diferentes versiones de la librería? ¿qué versiones me sirven? A veces el nombre del archivo es desconocido pero el contenido es perfectamente entendible para el usuario. Es el caso del segundo ejemplo, pero no hay una buena forma de darle a 'entender' eso al computador con el fin de encontrar el archivo deseado. Al considerar temas de versionamiento (ejemplo 3) el problema se vuelve aún más complejo.

Para intentar resolver estas dificultades, los sistemas operativos y formatos de archivo exponen metadatos de los mismos, los cuales pueden ser utilizados eventualmente en motores de búsqueda muchas veces integrados en el sistema operativo. No es raro encontrar información detallada de cómo fue capturada una fotografía en un archivo de imágen, incluyendo metadatos de tiempo de exposición, apertura del obturador, etc. De la misma forma, el sistema operativo expone información de la última fecha de modificación del archivo, tamaño y otros. Estos metadatos, al ser autogenerados por los distintos dispositivos que dan origen y manipulan los archivos, son altamente técnicos y genéricos y no dan cuenta del *contenido* del archivo. Existe una excepción notable de lo recién dicho: en muchos formatos de archivos de música se puede almacenar metadatos de la canción contenida y su artista. Este mecanismo, que si bien es mejor que nada, no está exento de problemas, ya que al almacenar simples cadenas de caracteres los metadatos son propensos a ambigüedad y errores. Por ejemplo, no es trivial para un computador reconocer que 'Los Beatles' y 'The Beatles' se refieren al mismo grupo.

En general estos problemas se presentan porque no hay un registro certero y sin ambigüedades del contenido de los archivos. La construcción de este registro no es sencilla, pues requiere establecer un leguaje común para describir aquellos elementos que dan un contexto semántico bien definido a un dato o archivo en particular, como un artista musical, un autor de un libro o una ubicación geográfica donde puede haberse tomado una fotografía. Como veremos más adelante existen esfuerzos por establecer una forma de intercambio de información que cumple con estos requisitos. Adicionalmente, ya que muchas veces los contenidos no son directamente interpretables y entendibles por los computadores, una solución óptima del problema debe funcionar de forma colaborativa, set de fácil acceso y estar disponible para el uso general, evitando duplicar esfuerzos cuando sea posible. A la vez, debido a la enorme cantidad posible de archivos, en aquellos casos en que la identificación de contenido es automatizable, debiera proveer las interfaces necesarias para que sistemas automatizados caracterizen los contenidos de los archivos.

Para efectos de eliminar la ambigüedad en las descripciones de la semántica de los archivos, en el desarrollo de esta memoria de título se utilizaron tecnologías presentes en iniciativas como *Linked Data* y la Web Semántica para el desarrollo de un prototipo de un sistema de indexación de archivos, donde uno de los pilares es la interoperabilidad entre distintas fuentes de datos, estableciendo vocabularios comunes e independidentes del el idioma en que estos se encuentren y fácilmente entendible y procesable por computadores. Se diseñó e implementó así una red semántica de archivos que permite a los usuarios registrar metadatos de estos en una base de datos *online* usando estas tecnologías.

El sistema desarrollado se consideró para archivos de caracter público, puesto que expone a traveés de la Web metadatos de estos, los cuales son aportados de manera colaborativa. No busca mejorar la de indexación local de archivos de uso privado. Sin embargo, si se desarrollaran buenas herramientas de aprovechamiento de metadatos a nivel local, una arquitectura similar podría ser implementada con estos fines.

Dentro de las tecnologías utilizadas en la Web Semántica se hace importante mencionar el estándar RDF (**R**esource **D**escription **F**ramework), que corresponde al lenguaje utilizado para la representación de datos. Con RDF los datos se almacenan en estructuras de grafos y pueden posteriormente ser consultados a través de un *endpoint*, una interfaz de consultas similiar aquella que proveen los motores de bases de datos tradicionales.

Uno de los desafíos surge debido a poca adopción actual de las tecnologías antes mencionadas. Siendo relativamente nuevas, existe una falta de conocimiento entre los ingenieros y desarrolladores sobre éstas. Sin ir más lejos, en nuestra facultad solo el año 2010 se incorporó un curso electivo en que se les hace mención [31]. Es por esto que se ha planteado este trabajo de título además como una instancia educativa, no solo para quien la escribe sino también para alumnos y desarrolladores en general, aportando con documentación y herramientas que faciliten la adopción de estas tecnologías.

Un prototipo del proyecto se encuentra disponible en linea en
`http://www.rdfclip.com/`

1.1. Objetivos

La motivación de esta memoria de título viene de la intención promover el uso de tecnologías de Web Semántica mediante la creación de material que introduzca a desarrolladores en el área y facilite el desarrollo de software. Este trabajo se enmarca en la creación de un prototipo de un sistema de indexación semántica de archivos.

1.1.1. Objetivo General

Diseñar e implementar un prototipo de plataforma que soporte el intercambio de información en forma de RDF vinculada al contenido semántico de archivos, independiente de su nombre, ruta o medio físico que lo almacena, facilitando el acceso y modificación de estos datos mediante una API fácilmente portable a diversos sistemas operativos.

1.1.2. Objetivos Específicos

1. Diseño de la arquitectura, incluyendo especificaciones técnicas y funcionales, explicitando posibles usos y limitaciones.
2. Definición de una ontología base apropiada para almacenar información relevante para un conjunto inicial de archivos (de música, documentos u otros) además de un vocabulario RDF necesario para una interfaz de usuario de fácil uso.
3. Creación de material de lectura que introduzca todos los conceptos básicos necesarios para que un ingeniero en computación pueda comenzar a hacer uso de tecnologías de la Web Semántica.
4. Implementación de un prototipo funcional que permita el ingreso de datos y consulta sobre el contexto semántico de los archivos.
5. Desarrollo de una o más aplicaciones basadas en la arquitectura con casos de uso reales a modo de validación.
6. Desarrollo de herramientas que permitan vincular de forma (semi)automática archivos a su contenido semántico, a partir de coincidencias de nombres y análisis del contenido (ID3 Tags en caso de archivos de música, por ejemplo).

7. Levantamiento de un *endpoint* de acceso público al grafo de metadatos de archivos.

Capítulo 2

Antecedentes

2.1. La Web Semántica

La Web es escencialmente un sistema interconectado de documentos accesibles a través de Internet. Ésta fue ideada por Tim Berners-Lee en un intento de facilitar el intercambio de información entre personas que usan distintos sistemas y documentos en diversos formatos. Esta información sería compartida a través de *hipertexto*, descrita en un lenguaje que todos conocemos como HTML. No hay dudas respecto al éxito que ha tenido esta iniciativa. Sin embargo, fue creada en una época donde nosotros, las personas, éramos los principales consumidores de información, y por tanto diseñada para ser entendida por seres humanos. En la actualidad la automatización del procesamiento de datos se ha vuelto cada vez más importante para la generación de información de valor, y una parte importante de esos datos se encuentran dispersos en esta Red creada para humanos. Se hace indispensable entonces elaborar un sistema que permita a los computadores 'entender' el contenido de los documentos, siendo capaces de establecer las relaciones que existen entre diversas fuentes de datos tal como lo hacemos las personas. Esto se logra entendiendo el significado (o *semántica*) de cada uno de estos datos. Llamamos a esto la *Web Semántica*.

Es bajo este contexto que Tim Berners-Lee promueve hace algunos años un movimiento denominado *Linked Data*, haciendo un llamado a todas las personas a que, tal como hasta ahora han publicado documentos en forma de hipertexto en la Web, comiencen a publicar

datos crudos [3]. También ha mostrado los beneficios que tiene adoptar la postura de dar acceso a otros a los datos de los cuales disponemos [4], en especial cuando estos permiten vincularse entre ellos, formando así una gran red cuyo valor es mucho mayor que la suma de sus partes, dando posibilidades inmensas de generar información valiosa en diversas áreas.

La idea entonces es hacer uso de un lenguaje estándar de representación de datos, así como lo es HTML para el consumo humano en la Web. El lenguaje RDF (Resource Description Framework) se plantea como esa alternativa para la Web Semántica, el cual debe permitir interoperabilidad entre distintas fuentes de datos.

2.2. El Lenguaje RDF

RDF o *Marco de Descripción de Recursos* (Resource Drescription Framework) es un lenguaje para representar información en forma de afirmaciones sobre *recursos*. Un recurso web es cualquier entidad a la cual se le pueda asignar un identificador o URI (Uniform Resource Identifier [1]), de la misma forma como los documentos web pueden ser identificados por su URL (Uniform Resource Locator), cadenas de caracteres de la forma 'http://...'. Estos identificadores se utilizan para construir unidades atómicas de información que se denominan *triples*, compuestos de un *sujeto*, un *predicado* y un *objeto*, de forma similar a la gramática del lenguaje natural. De esta forma, a modo de ejemplo, la información "Einstein nació en Alemania" se puede representar por el triple

```
http://res/AlbertEinstein  http://prop/birthPlace  http://res/Germany
```

Aqui, las cadenas de caracteres `http://res/AlbertEinstein`, `http://prop/birthPlace` y `http://res/Germany` se usaron como identificadores de *Einsten*, *Lugar de Nacimiento* y *Alemania* respectivamente. Idealmente aquellas personas que publiquen información deberían intentar mantener consistencia en el uso de identificadores. Por lo general, se utilizan URIs según la especificación de la W3C [6] para construirlos. Un conjunto de triples representa finalmente un conjunto de datos modelado en forma de grafo cuyos arcos están etiquetados. A modo de ejemplo, la Figura 2.1 representa el siguiente conjunto de triples:

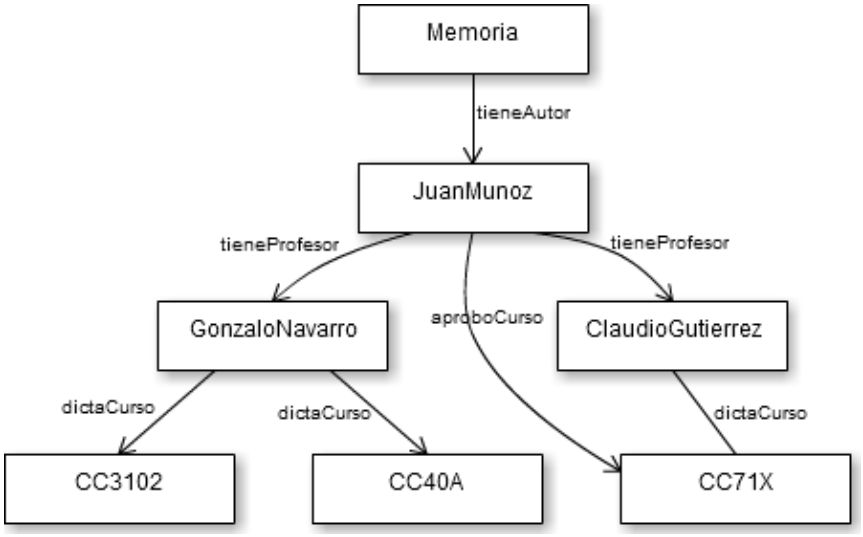

Figura 2.1: Ejemplo de grafo RDF. Los nodos hacen de sujetos y objetos mientras que los arcos corresponden a los predicados.

`Memoria`	`tieneAutor`	`JuanMunoz`
`JuanMunoz`	`tieneProfesor`	`ClaudioGutierrez`
`JuanMunoz`	`tieneProfesor`	`GonzaloNavarro`
`GonzaloNavarro`	`dictaCurso`	`CC3102`
`GonzaloNavarro`	`dictaCurso`	`CC40A`
`ClaudioGutierrez`	`dictaCurso`	`CC71X`
`JuanMunoz`	`aprobóCurso`	`CC71X`

Oontologías

Si bien RDF permite que tres identificadores cualesquiera formen un triple, en la práctica nos encontramos con que ciertos identificadores tienen sentido sólo en ciertos contextos semánticos. Esto nos lleva a agrupar conjuntos de identificadores con su semántica y usos bien definidos utilizables en contextos particulares, veremos que esto es en cierta medida equivalente a los modelos de datos en bases de datos relacionales y se denominan *ontologías*.

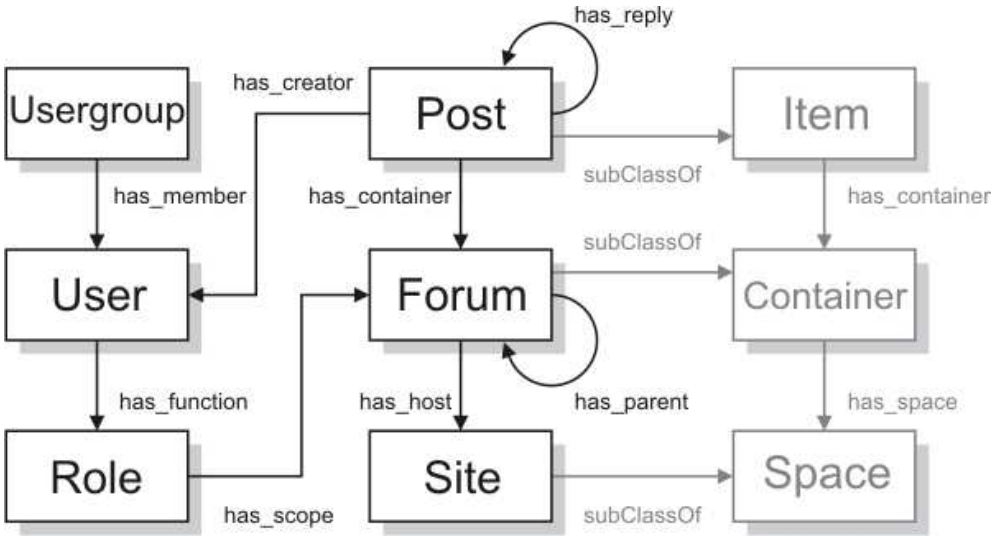

Figura 2.2: Ontolología SIOC. Los cuadros representan clases y los arcos propiedes que apuntan a sus tipos. Al igual que en la figura 2.1 estos se traducen triples RDF.

Éstas son a su vez descritas usando RDF y forman un grafo por sí mismas. La Figura 2.2 representa la ontología **SIOC** que describe comunidades online, ahí los nodos representan 'Clases' y los arcos propiedades de éstas. Existen varias ontologías conocidas, además de esta última, tales como **FOAF**, para representar personas, sus actividades y relaciones, y **Dublin Core**, que describe metadatos de recursos para efectos de publicación y categorización.

El uso de RDF como lenguaje de publicación de datos permite eliminar la ambigüedad de la información normalmente contenida en texto plano. Así, sabemos que al usar el identificador `http://res/AlbertEinstein` nos referimos a Albert Einstein, fisico alemán creador de la teoría de la relatividad general, y no a otra persona del mismo nombre, a quien se le debería asignar un nuevo identificador. Otra ventaja importante es la facilidad para integrar datos: si se mantiene una consistencia con el uso de identificadores entre distintas fuentes de datos, basta con considerar el conjunto completo de triples de diversas fuentes para obtener la unión de los grafos.

Cuadro 2.1: Espacios de nombre.

Nombre	**Prefijo**	Descripción
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	Sintaxis básica RDF.
rdfs	`http://www.w3.org/2000/01/rdf-schema#`	Permite describir modelos de datos.
nfo	`http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#`	Ontología de metadatos archivos de NEPOMUK.
clip	`http://www.rdfclip.com/data#`	Recursos dentro del grafo de archivos del sistema RDFClip.
clips	`http://www.rdfclip.com/schema#`	Extensiones del modelo de datos propias de RDFClip.
uchile	`http://www.rdfclip.com/schema/uchile#`	Describe elementos de la Universidad de Chile.
xmls	`http://www.w3.org/2001/XMLSchema#`	Tipos de datos XML.

2.3. Espacios de nombre y notación

Producto del uso de URIs basadas en el protocolo HTTP como identificadores, estos suelen ser largos, engorrosos y difíciles de memorizar. De la misma forma como en una base de datos relacional un campo `nombre` en una tabla `cliente` probablemente se refiere al nombre de un cliente, en RDF para referirnos al mismo concepto usamos el identificador `http://xmlns.com/foaf/0.1/name`. La solución aceptada para facilitar la escritura de estos identificadores es abreviar prefijos comunes dentro de espacios de nombre mucho más cortos. El cuadro 2.1 muestra los espacios de nombre utilizados a lo largo de este documento.

De esta forma, al referirnos al recurso `nfo:Video`, éste corresponde a la concatenación del prefijo asociado al espacio de nombres **nfo** y la cadena de caracteres "Video": `http://www.semanticdesktop.org/ontologies/2007/03/22/nfo#Video`. Esta es la misma sintaxis utilizada en ciertos formatos de archivos para almacenar contenido en forma de RDF y en el lenguaje de consulta (SPARQL).

2.4. Sistemas de almacenamiento

Hemos visto la forma particular de representar datos en el lenguaje RDF a partir de *triples*. En principio, cualquier sistema de almacenamiento que permita guardar una tabla de tres columnas puede ser utilizado como un sistema de almacenamiento de datos RDF, pero en la práctica lo mejor es hacer uso de sistemas optimizados para datos de esta naturaleza, su almacenamiento y recuperación. Llamamos *triplestores* a estos sistemas especializados en el almacenamiento de triples, las cuales además de almacenar, proveen normalmente interfaces de consulta haciendo uso de un lenguaje especializado para grafos RDF, SPARQL, el cual revisaremos en la sección 2.5.

Una de las interfaces más comúnmente utilizadas para la recuperación de datos almacenados en RDF son los *endpoints* SPARQL, que, identificados por una URI y mediante una interfaz HTTP, reciben y resuelven consultas sobre un determinado grafo, respondiendo con un conjunto de resultados. Una de las capacidades más interesantes de los *triplestores* que se espera explotar más en el futuro de la Web Semántica es la ejecución de consultas a través de varios *endpoints* simultaneamente, obteniendo un resultado enriquecido por los datos presentes en todos los grafos consultados, potencialmente almacenados en distintos servidores, esto es lo que se denomina una *consulta federada*.

2.5. Lenguaje de Consulta

Tomemos como ejemplo los motores de base de datos relacionales, los cuales almacenan datos en tablas. El simple almaceniamiento no sirve de nada sin un lenguaje de consulta que permita extraer ciertos pedazos de información, el estandar más utilizado en estos casos es SQL (**S**tructured **Q**uery **L**anguage). De la misma forma, si necesitamos un lenguaje de consultas para recuperar información contenida en grafos podemos utilizar el lenguaje SPARQL (**S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage), el cual permite especificar consultas describiendo partes del grafo y dejando variables que son completadas por el motor que ejecuta las consultas convirtiendose así en resultados. La W3C especifica un estandar del lenguaje SPARQL [28] cuya última versión 1.1 se encuentra en estado borrador [29] y se encuentra parcialmente implementada en varios motores de bases de datos RDF.

La consulta de ejemplo presentada en el código 2.1 obtiene los nombres y los e-mails de todas las personas del grafo haciendo uso de la ontología **FOAF**.

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 SELECT ?name ?email
3 WHERE {
4   ?person a foaf:Person.
5   ?person foaf:name ?name.
6   ?person foaf:mbox ?email.
7 }
```

Código 2.1: Ejemplo de consulta SPARQL. Para todos los nodos de tipo `foaf:Person` (persona) obtiene los arcos salientes etiquetados como `foaf:name` y `foaf:mbox` (nombre y correo electrónico respectivamente) y los valores de sus nodos de destino.

2.5.1. Inferencia en SPARQL

Como ya hemos visto, los datos en RDF están representados en forma de afirmaciones compuestas por un *sujeto*, un *predicado* y un *objeto*. Muchas veces, conociendo la semántica de ciertos predicados es posible inferir nuevos hechos, dando origen a nuevas afirmaciones en forma de triples. Consideremos un grafo RDF que contiene los siguientes triples:

```
Bart    esHermanoDe  Lisa
Homero  esPadreDe    Bart
```

Conociendo lo que significan los predicados *esHermanoDe* y *esPadreDe*, es fácil inferir que:

```
Homero  esPadreDe  Lisa
```

El hecho de que el nuevo triple inferido se integre trivialmente al grafo enriquéciendolo da cuenta de la flexibilidad y adaptabilidad de RDF como lenguaje para describir datos.

Este tipo de inferencia está implementada en ciertos *triplestores* basándose en ontologías estándares como lo son **OWL** [23] y **RDFSchema** [24], y son particularmente útiles para describir herencia en modelos de datos (ontologías en RDF). Por ejemplo, una aplicación que hace uso de datos de rutas de trenes y aviones en RDF podría tener un modelo de datos que se describe como sigue dentro de un determinado espacio de nombres que abreviamos 'ns':

```
ns:Airline        rdf:type         rdfs:Class
ns:Route          rdf:type         rdfs:Class
ns:TrainRoute     rdf:type         rdfs:Class
ns:FlyRoute       rdf:type         rdfs:Class
ns:TrainRoute     rdfs:subClassOf  ns:Route
ns:FlyRoute       rdfs:subClassOf  ns:Route
ns:RouteDistance  rdfs:domain      ns:Route
ns:RouteDistance  rdfs:range       xmls:float
ns:Airline        rdfs:domain      ns:FlyRoute
ns:MainAirline    rdfs:range       ns:Airline
```

La figura 2.3 muestra el diagrama de clases correspondiente, el predicado `rdfs:subClassOf` nos dice que `ns:FlyRoute` es una subclase de `ns:Route`, y por lo tanto debe heredar las propiedades de la clase madre. Normalmente, si uno fuera a consultar al grafo en busca de todas las propiedades de `FlyRoute` como se muestra en el código 2.2, sólamente obtendría `ns:MainAirline` como resultado, ya que en principio el motor de base de datos desconoce el significado de la herencias de clases. Sin embargo, algunos motores de bases de datos en RDF como Virtuoso permiten activar inferencia básica que 'entiende' las implicancias de ciertos predicados [25], incluyendo buena parte de aquellos definidos por **RDFSchema** [24], reconociendo de esta forma que `ns:RouteDistance` es una propiedad heredada de `FlyRoute`.

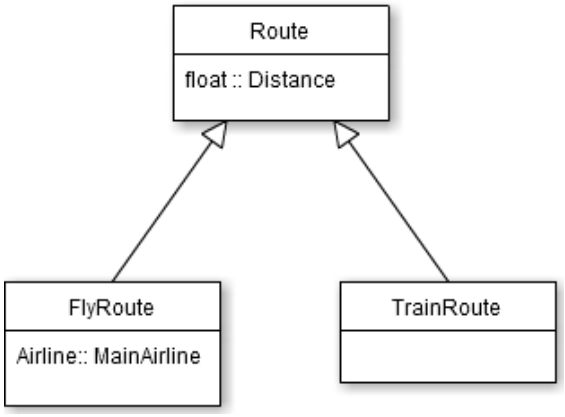

Figura 2.3: Ejemplo de diagrama de clases en bases de datos relacionales. Tanto las propiedades como las relaciones de herencia se pueden representar en forma de triples RDF.

```
1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX ns: <http://...>
3 SELECT DISTINCT ?prop WHERE {
4     ?prop rdfs:domain ns:FlyRoute .
5 }
```

Código 2.2: Consulta de prueba de inferencia. Con un motor que aplique inferencia basada en los predicados de RDFSchema, se obtienen todas las propiedades de la clase `ns:FlyRoute`. Sin inferencia las propiedades heredadas no son identificadas como propias de la clase.

2.6. Desarrollo en Web Semántica

El desarrollo web orientado al intercambio libre de datos es un cambio de paradigma importante. Históricamente los desarrolladores han intentado evitar la automatización de acceso a sus sitios, pero últimamente hemos visto aparecer APIs de acceso a diversos sistemas. La adopción de RDF como lenguaje de representación de datos es llevar esa tendencia un paso más allá. Sin embargo, al hacer uso de tecnologías fundamentalmente distintas a las

tradicionales, la adopción ha sido muy lenta. No se encuentran *Frameworks* de desarrollo apropiados, librerías de conexión a triple stores y otros. Sí se encuentran algunas librerías que facilitan la manipulación de datos, como lo es rdfquery [2], que permite crear, guardar y consultar triples RDF haciendo uso de jQuery.

Un punto importante en el desarrollo de la Web Semántica es la aparición de puntos de consulta o *endpoints* abiertos a través de los cuales los sitios web pueden dar acceso a los datos que generan en formato RDF. Esto permite en principio hacer consultas que cruzan información de diversas fuentes. El proyecto *Linked Data* [17] muestra como ya se han abierto distintos *endpoints* que contienen datos especificos de diversas fuentes. Un ejemplo de esto es DBPedia [14], que publica datos obtenidos de Wikipedia [16] en RDF y provee una interfaz de consulta para obtenerlos y utilizarlos libremente.

Idealmente se debería contar con herramientas para facilitar la creación de estos puntos especializados de datos, para la captura y despliegue de datos, así como el procesamiento, conexión con *triplestores*, despliegue de interfaces de usuario, etc.

2.6.1. RDFa: RDF para la Web

De por si RDF es un lenguaje que existe de forma independiente a la Web, aunque se apoya en sus principales pilares (como el protocolo HTTP) para la interconexión de fuentes de datos. Sin embargo, existen intentos por hacer uso de RDF en el contexto de las páginas web que vemos representadas mediante nuestros navegadores en forma de HTML. Lo que se busca es mantener en el mismo documento, además de la información disponible para el consumo del usuario, los metadatos necesarios para otorgarle semántica al contenido que se está deplegando en pantalla, de forma que sea fácilmente procesable por computadores. Esta es la idea principal de RDFa o *Resource Description Framework - in - attributes*.

RDFa se basa en la idea incrustar metadatos en los atributos de los elementos HTML que conforman los documentos. Aprovechando convenciones comunes en la estructura de documentos, donde estos se ven representados por arboles de elementos que contienen a otros elementos, se estableció una sintaxis que permite indicar, para cualquier elemento, sobre qué recursos se hace referencia al interior de él. El codigo 2.3 en HTML es un ejemplo de esto, muestra la información “Einsten nació en Alemania” con los recursos RDF asociados.

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">
    1879-03-14</span>
  <div rel="dbp:birthPlace"
   resource="http://dbpedia.org/resource/Germany">
    <span property="dbp:conventionalLongName">
        Federal Republic of Germany
    </span>
  </div>
</div>
```

Código 2.3: Ejemplo de uso de RFDa. Los atributos **about**, **property**, **rel**, **resource** y **datatype** son interpretados por los exploradores de RDFa para determinar las relaciones entre los nodos del documento HTML.

Capítulo 3

Especificación del problema

En el marco de esta memoria de título consideramos dos aristas de problemas. Por un lado tenemos la necesidad de indexación semántica de archivos, la cual supone como solución un sistema basado en la Web Semántica que será descrito más adelante. Por el otro, la dificultad de desarrollo en Web Semántica que es necesaria para dicho sistema y para el consumo de datos que hacen uso de estas tecnologias posteriormente. A continuación describimos en más detalle cada uno de estos puntos.

3.1. Indexación Semántica de Archivos

Una de las mayores fuentes de datos que tiene cada persona a su disposición es su computador personal. Éste almacena todo tipo de archivos, incluyendo música, películas, documentos, etc. Normalmente los usuarios tienen cierto grado de conocimiento de la organización de los archivos en su computador, pero nunca es completo, siempre existirán vínculos entre los archivos que tenemos y otras fuentes de datos (incluyendo otros archivos) que ignoramos.

¿Qué hace una persona cuando quiere información adicional sobre lo contenido en un archivo? Normalmente busca en la Web, toma el nombre de aquello que se encuentra en el archivo (la canción, película o lo que sea) y ve los resultados que arroja Google, esto es, si tiene la suerte de saber qué es lo que contiene el archivo. Este método es necesario por una

razón: **el computador no almacena información sobre el contenido semántico de los archivos**. Si lo hiciera, se podría automatizar la búsqueda de información adicional, aun cuando el usuario ignore por completo el contenido de éste. Esta información podría incluir, por ejemplo, una introducción a un libro, una referencia a una película o anotaciones sobre un momento capturado por una fotografía.

Nuestra capacidad de hacer 'buen uso' de los datos contenidos en los archivos en nuestros computadores depende del acceso que tengamos a ellos y de nuestro conocimiento de las relaciones que existen entre estos mismos. Para ello se requiere un sistema de indexación según el contenido de archivos. Sin embargo, los sistemas operativos no son capaces de registrar estos metadatos de forma eficaz, considerando la casi infinita variedad de archivos existentes. Además, dicha semántica es muchas veces imposible para un computador de hacer explícita.

Esto trae como resultado que muchas veces un usuario sencillamente *no sabe* qué es lo que tiene en computador, y dentro de los archivos que sabe que tiene, desconoce muchas veces su utilidad. Dado que los archivos son el elemento básico de información que se usa a nivel de usuario, el problema anterior se traduce en una desaprovechamiento de información, redundancia de datos y mucho tiempo perdido en la búsqueda de soluciones que serían triviales si se contara con los metadatos correctamente indexados y las relaciones entre los archivos bien establecidas. Esto no solo se limita a la administración local de archivos, sino también a cuando un usuario busca un determinado contenido en la Web, mucha veces está buscando en realidad, el archivo que almacena ese contenido con el fin de descargarlo.

Adicionalmente, dado que los archivos son la principal fuente de datos en nuestros computadores y habiendo ya establecido que la publicación de datos de forma abierta trae consigo beneficios y genera un valor de la misma forma en que lo ha hecho la publicación de documentos web a través de Internet, es necesario derribar las barreras de publicación y apertura de datos en forma de archivos, donde la indexación semántica es fundamental.

3.2. Acercamiento a la Web Semántica

Dentro del mismo mundo de los desarrolladores, incluso aquellos que más participan en desarrollo Web, existe un alto grado de desconocimiento de la Web Semántica, tanto de su

escencia y sus objetivos, como de las tecnologías en las que se apoya. Esto se ve reflejado en la falta de herramientas y librerías que permitan el acceso a sistemas de almacenamiento o faciliten la manipulación de datos en RDF. De la misma forma, hay pocos expertos en el tema y faltan fuentes de información para quienes se interesan por adoptar estas tecnologías. Es por esto que la publicación de datos en RDF se lleva a cabo principalmente a través de transformaciones de datos existentes, ya sea de bases de datos relacionales o documentos, pero no existen muchas aplicaciones que hagan aplicación directa de RDF, SPARQL otras tecnologías de Web Semántica. Además, la inmadurez del software que acompaña esta area de estudio lleva consigo problemas de optimización y por tanto escalabilidad que no han sido resueltos completamente. Para el desarrollo de esta memoria de título se hicieron uso de herramientas usadas en proyectos de gran escala de publicación de datos, pero que no resuelven totalmente los problemas de escalabilidad.

Reconociendo que RDF es la buena forma de representar los metadatos de archivos, lo anterior es un problema pues el buen aprovechamiento de estos metadatos, así como la construcción del sistema de indexación en sí, depende de la adopción de estas tecnologías por parte de los desarrolladores. Razón por la cual el desarrallo de herramientas que faciliten esta transición es uno de los objetivos de esta memoria de título, acá se debe incluir la conectividad con *endpoints* así como el desarrollo de interfaces gráficas para la visualizacón de datos y administración de *triplestores*.

Ejemplo de las barreras de entrada para el consumo de datos en RDF es la dificultad para navegar los grafos, el lenguaje de consulta SPARQL es poco conocido y el uso de identificadores basados en URIs suele confundir, por lo que el acceso típico a los datos a través de *endpoints* es poco intuitivo. Además no existen interfaces de fácil uso o sistemas de administración de alto nivel que permitan acceder a los datos.

3.3. Solución deseada

A partir de la descripción previa del problema, reconocemos dos tipos de usuarios: desarrolladores y no-desarrolladores. Los primeros cuentan con un *background* en ciencias de la computación, pero no necesariamente en Web Semántica, y se desea que dispongan de un espacio de indexación archivos el cual puede ser accesado programáticamente y a partir de tareas automáticas, para esto es necesario facilitar la adopción de las tecnologías

involucradas y proveer librerías de acceso a la aplicación. Los no-desarrolladores son principalmente consumidores de datos, pero también se les debe dar la oportunidad de aportar información mediante interfaces de usuario que no requieran ningún tipo de conocimiento adicional. En general, existen ciertas características globales que debe cumplir la solución deseada que revisaremos más adelante en esta sección.

Los usuarios deben poder aportar con conocimiento a la base de metadatos almacenada en el sistema de una forma sencilla e intuitiva. Así mismo, también deben poder recuperar aquellos metadatos ingresados con anterioridad por otros usuarios o ellos mismos, y realizar consultas simples que permitan establecer la relación entre dos o más archivos locales.

Por su parte, los desarrolladores deben contar con interfaces de acceso programáticas a los metadatos almacenados y con las herramientas y documentación necesaria para accesarlos y modificarlos.

3.3.1. Escalabilidad

La escalabilidad es un tema no menor, esto es debido a la gran variabilidad existente en formatos de archivo, su continua evolución y las infinitas posibilidades del contenido de estos, no es difícil darse cuenta que un modelo de datos relacional estándar es impráctico e infactible. En cuanto a la infraestructura de hardware, el tema de escalabilidad escapa del alcance de esta memoria de título, pero es un problema parcialmente resuelto por *wikis* que permiten el acceso de millones de usuarios simultáneos quienes tienen acceso a editar el contenido y por otro lado por fuentes de datos pertenecientes a *Linked Data* (e.g. DBPedia) que almacenan conjuntos de datos muy grandes en formato RDF para acceso público sin restricciones.

Así mismo, la solución debe estar bien acotada al problema en cuestión, esto es, indexación de archivos. Idealmente no se debe almacenar más que metadatos referentes a los archivos mismos, refiriendo al usuario a otras fuentes de datos cuando corresponda. Este punto es importante porque cuando uno considera los metadatos asociados a, por ejemplo, un archivo MP3, puede sin querer llegar muy lejos. Ese MP3 puede contener una canción de un determinado artísta, el cual viene de cierto país que tiene cierta población; todos estos datos, que si bien están relacionados con el archivo, no debiesen ser almacenados en el sistema.

3.3.2. Accesibilidad

Uno de los principales problemas de las alternativas estudiadas en la próxima sección es su baja accesibilidad. Es importante que la solución a desarrollar permita acceder fácilmente a los datos y agregar nuevas informaciones. Debe incluir integración directa con los archivos en Windows (al ser el S.O. más popular) y en lo posible en otros sistemas operativos. De la misma forma, se debe contar con una interfaz de usuario intuitiva y fácil de usar.

Adicionalmente, se deben proveer herramientas para el acceso a datos y desarrollo en tecnologías de Web Semántica para desarrolladores, ayudando así a la masificación de la tecnología utilizada. Esto incluye buscar métodos que permitan realizar consultas SPARQL con mayor facilidad, librerías de código abierto para procesamiento de datos, etc.

3.4. Otras soluciones

Como hemos visto, el sistema propuesto abre un campo de desarrollo de aplicaciones para resolver distintos problemas. Existen trabajos ya realizados que coinciden parcialmente con el proyecto o que apuntan a la resolución de problemas similares. Sin embargo, ninguno de estos ha sido ampliamente adoptado ni abarca el problema de forma suficientemente general como el proyecto que se planteado en esta memoria de título. Estos serán discutidos en las secciones siguientes.

3.4.1. TripFS

Probablemente el trabajo que más se acerca desde el punto de vista técnico. TripFS [7] plantea la idea de vincular archivos con fuentes de datos existentes según su contenido semántico haciendo uso de RDF, sin embargo no parece existir una implentación concreta ni una propuesta de arquitectura. Adicionalmente, el tema de los identificadores universales para archivos no está totalmente resuelto. Es un buen punto de partida y sirvió como una buena base teorica para esta memoria de título.

Ideas positivas rescatables

TripFS presenta alternativas interesantes para la extracción de metadatos haciendo uso del framework Aperture [8]. Y discute la importancia de vincular archivos a otras fuentes ya establecidas de datos, como lo son MusicBrainz [9] en el caso de archivos de música o DBPedia.

Diferencias con este trabajo

Desde un comienzo, TripFS parece plantearse como un sistema de manejo de archivos a nivel local, o cuando mucho, empresarial. No se pone en discusión cómo los metadatos se comparten a un nivel más global. Respecto al uso de identificadores, se usan combinaciones aleatorias, con lo cual pierde un concepto importante que se plantea en esta memoria, que los archivos mismos deben proveer sus indentificadores mediante una transformación única, por ejemplo una función de *hashing*.

3.4.2. NEPOMUK Semantic Desktop

El proyecto NEPOMUK Semantic Desktop [10] se enfoca en el problema de organización local de los archivos, e intenta resolverlo a partir de la indexación en RDF, sin embargo, el proyecto parece estar abandonado, sin avances desde el 2007. Es importante considerar que las tecnología RDF se está recién empezando a masificar, así como la adopción de estándares, por lo que este tipo de proyectos recién empiezan a tener sentido con oportunidades reales de adopción por parte de los usuarios. Afortunadamente, este proyecto definió varias ontologías para la descripción de archivos en RDF, las cuales son perfectamente reutilizables, esto tiene la ventaja adicional discutida anteriormente, permitiendo interoperabilidad en el procesamiento y consumo de datos de las distintas fuentes al mantener la consistencia entre identificadores.

Ideas positivas rescatables

El elemento principal que se rescata del proyecto NEPOMUK son las ontologías desarrolladas. Prácticamente todos los proyectos relacionados hacen referencia a éstas pues

presentan de forma bastante completa la representación RDF de metadatos de archivos. Este punto es fundamental pues, al tener un conjunto de ontologías base reconocido para el tipo de datos que se está indexando, se pueden aprovechar las capacidades de interoperabilidad de RDF para complementar los distintos proyectos entre sí.

Diferencias con este trabajo

Más que en el desarrollo de un sistema de indexación colaborativa, NEPOMUK se centra en la definición de ontologías y vocabularios para la representación de metadatos. No se definen interfaces de acceso, ingreso y recuperación más allá de los estandares tipícos RDF (SPARQL a través de endpoints HTTP).

3.4.3. Tracker

Similar al proyecto NEPOMUK Semantic Desktop, pero más actual es (Meta) Tracker [11], un proyecto de la Fundación GNOME, lo cual es una fortaleza y a la vez una debilidad, tiene la ventaja de ser código abierto, sin embargo al sólo abarcar una fracción pequeña del universo de usuarios (aquellos que usan Linux) se dificulta su masificación. Si lo que se quiere es identificar semánticamente la mayor cantidad de archivos posibles, cualquier proyecto que quiera universalizar esta idea necesita desarrollar software con soporte en las plataformas más populares, incluyendo Windows y MacOS y que presente además una interfaz web para acceso a los datos de forma totalmente independiente al sistema del usuario.

Ideas positivas rescatables

Es uno de los proyecto más activos en el area y valida el uso de RDF como sistema de indexación de archivos, así como la utilización de las ontologías NEPOMUK.

Diferencias con este trabajo

Tracker se enfoca principalmente al aprovechamiento de metadatos en RDF para el manejo de archivos locales. Este es un complemento interesante para este proyecto de titulo, cuyo centro está en la indexación colaborativa y el aprovechamiento social de los metadatos haciendo uso de un sistema centralizado de almacenamiento.

3.4.4. Bitzi

Bitzi [18] es un proyecto que apunta directamente a generar una gran base de datos en RDF con información de archivos, pero no cuenta con una interfaz tal que pueda ser utilizado por la mayoría de las personas. Tampoco tiene un *endpoint* abierto al público, el proyecto se ve además abandonado (sin actividad desde agosto de 2008), pero es una buena fuente de referencia. Es importante notar que Bitzi ya asigna URIs únicas a archivos basados en un hash del mismo, para mantener interoperabilidad con el proyecto de esta memoria y Bitzi se considera indicar de forma explicita la correspondencia de archivos entre ambos grafos. Ciertas ontologías RDF proveen mecanismos para facilitar la interoperabilidad de datos en estos casos (ver sección 4.1.3).

Ideas positivas rescatables

La idea de un repositorio central de metadatos. Bitzi apunta directamente a esto, otorgando identificadores basados en funciones de *hashing* sobre los archivos.

Diferencias con este trabajo

La accesibilidad a los datos es un tema fundamental que está un poco dejado de lado en Bitzi. En este trabajo de título se le dió enfasis importante a la facilidad para acceder a datos y a proveer distintos mecanismos para consultar el *triplestore*. Esto incluye el desarrollo de interfaces de usuario y una API de acceso REST.

Capítulo 4

Descripción de la solución

En este capítulo describimos la solución implementada, aún en desarrallo, y que será accesible en `http://www.rdfclip.com`. A grandes rasgos, corresponde a un protipo funcional de un sistema de indexación de archivos por RDF. El sistema expone los datos de forma abierta a través de diversas interfaces de usuario y programáticas. Las siguientes secciones detallan el funcionamiento interno, así como los recursos para desarrolladores liberados en la implementación de esta memoria de título.

Es importante destacar que este texto es en sí mismo uno de los productos importantes de este trabajo. Los primeros capítulos se han editado en distintos formatos pues constituyen un buen primer acercamiento práctico al desarrollo en RDF y Web Semántica, y se ha planteado como apunte complementario al curso CC71X - La Web de Datos que se dicta en la Facultad.

4.1. Un grafo de archivos

Hemos establecido que es importante que la gente publique datos, y por lo tanto se hace un llamado para que la gente lo haga. Pero hacerlo bien (en un lenguaje estándar, manteniendo la consistencia de identificadores) es muy difícil. Hay que aprender un nuevo lenguaje, transformar datos, etc. El problema radica en que la gente normalmente no trabaja con *datos* de forma abstracta sino con *archivos* que se encuentran en su computador, los

cuales no poseen una semántica bien definida a priori (hasta que se haga explícita de alguna forma). Para efectos de esta memoria de título, se desarrolló un sistema que busca otorgarles un contexto semántico a los archivos, facilitando así las posibilidades de publicar datos en diversos formatos y a la vez aprovechando las ventajas de la publicación de datos en RDF.

Dado que los archivos no son más que una secuencia de bits, se propone utilizar esta secuencia para asignarle un identificador 'único' (o con bajísima probabilidad de colisión) a *cada* archivo (a partir de un *hash* calculado sobre esta secuencia de bits) y vincularlo con fuentes de datos existentes, asignándole así un contexto semántico bien definido. De esta forma, a modo de ejemplo, si a un cierto archivo (una cierta secuencia de bits) se le asigna un identificador `http://file/001` y éste corresponde a un documento Word 2007 con la biografía de Albert Einsten, los triples

```
http://file/001  http://prop/biographyOf  http://res/AlbertEinsten
http://file/001  http://prop/fileFormat   http://res/MSWord2007
```

hacen explicita esta relación. Además, es trivialmente integrable con otros conjuntos de datos que mantengan los mismos identificadores (como los ejemplos de la sección 2.2), de forma que la consulta "*Quiero la lista de archivos con biografías de personas nacidas en alemania*" se puede responder fácilmente.

Lo importante aquí es que basta con que una persona tenga un cierto conocimiento del contenido de sus archivos y manifieste el interes de hacer pública esa información, y el software debería almacenar estos metadatos para ser usados por éste y otros usuarios asignándole así un contexto semántico de conocimiento público a los archivos.

4.1.1. Arquitectura de Software

La Figura 4.1 muestra un diagrama de alto nivel de la arquitectura. Se almacena en un servidor el grafo RDF con la información del contexto semántico de los archivos según determinadas ontologías definidas para ciertos tipos de archivo, estos datos pueden estar vinculados con otras fuentes que almacenen datos en RDF tales como DBPedia [14] o

páginas tradicionales como IMDB y Wikipedia mediante el uso de sus URLs como identificadores de recursos externos (ver sección 4.2.2), acá vemos la importancia de la consistencia de identificadores. Las ontologías y datos son accesibles por aplicaciones través de diversas interfaces y proveen de información semántica a los clientes.

Más al detalle, la figura 4.2 presenta los distintos componentes del servicio y sus interfaces de acceso. El grafo se almacena en un *triplestore* desarrollado por Openlink denominado Virtuoso [19]. Éste se eligió por ser una de las alternativas más populares, y porque además provee un *endpoint* SPARQL. Virtuoso es utilizado por algunos servicios reconocidos que proveeen datos en RDF, tal como DBPedia [15], una de las fuentes de datos más grandes con más de un mil millones de triples [30], lo que da cuenta de su estabilidad y escalabilidad.

Las otras interfaces de acceso se apoyan en un *backend* desarrollado en *Django* [20], un popular Framework MVC (Modelo Vista Controlador [22]) basado en Python. Se tomó esta elección basándose en la disponibilidad de herramientas y librerías de desarrollo en RDF, en particular *rdflib* [21], que permite la manipulación de grafos etiquetados generando un output en RDF fácilmente. La conexión con el *triplestore* desde el *backend* fue desarrollada íntegramente y usa como acceso el *endpoint* que provee Virtuoso. Se cuenta además con una base de datos MySQL de apoyo, se eligió MySQL por simplicidad, de todas formas el Framework (Django) provee una interfaz independidente del motor de base de datos elegido al respecto y permite fácilmente cambiar de uno a otro.

Sobre este *backend* Django se presentan dos interfaces de acceso, una API REST que permite hacer consultas simples e ingresar nuevos triples al grafo, y una interfaz gráfica HTML a través de la cual los usuarios pueden visualizar fichas online de los archivos y consultar el grafo usando SPARQL.

4.1.2. Definición de identificadores

Discutimos anteriormente la importancia de los identificadores en RDF, y el desarrollo de este trabajo de título requirió definir algunos nuevos. Para esto, se registró `rdfclip.com` para ser usado como dominio del espacio de nombres. Este es el nombre que se le asignó al proyecto y la URL donde se encuentra disponible, su origen está en la idea de que de el grafo que se construye relaciona archivos y, análogamente, un 'clip' vincula dos documentos.

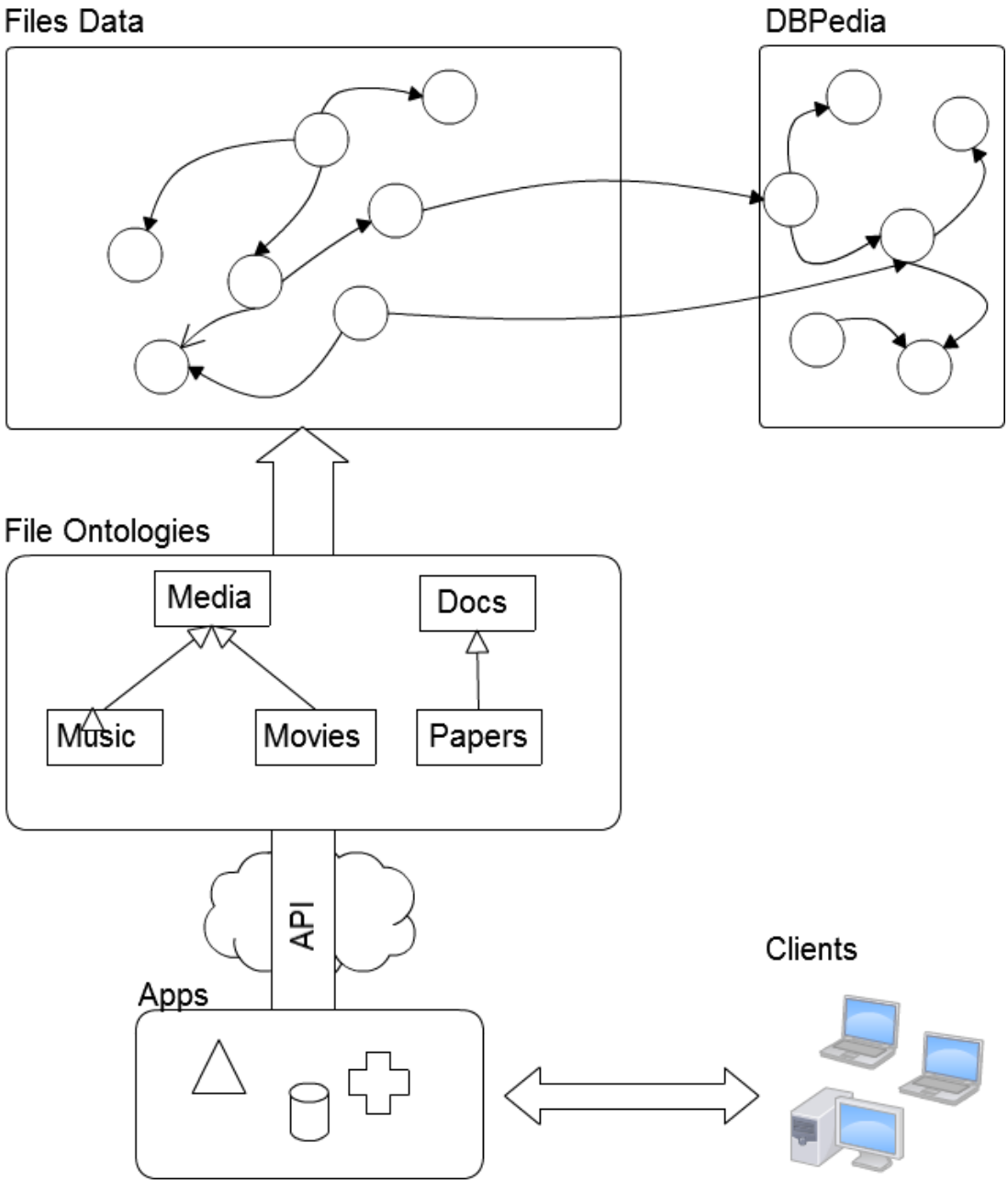

Figura 4.1: Diagrama general de la arquitectura. El servidor almacena dos grafos, para metadatos de archivos y ontologías que se encuentran vinculados a fuentes de datos externas. Se proveen diferentes interfaces de acceso que permiten la creación de aplicaciones que hacen uso de estos metadatos.

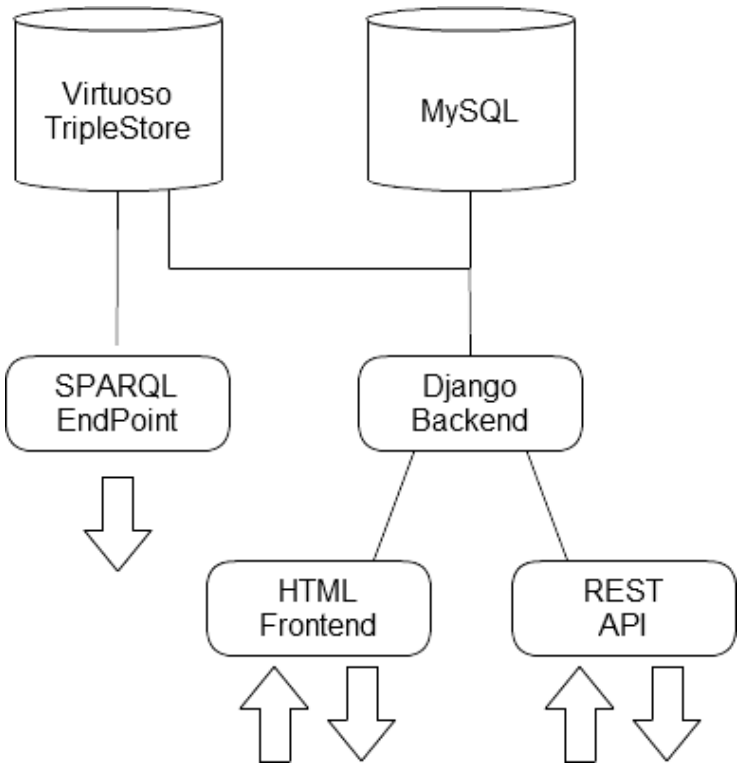

Figura 4.2: Detalle de la Arquitectura. Dos bases de datos, una relacional y un *triplestore*. Se proveen tres interfaces de acceso, el *endpoint* SPARQL es solamente de lectura, mientras que la interfaz web y la API REST permiten el ingreso de metadatos

Cuadro 4.1: Tiempo de cálculo de funciones de *hashing* (en segundos).

Tamaño	CRC32	MD5	SHA1
4KB	0.091	0.004	0.004
1MB	0.098	0.011	0.235
100MB	0.336	0.374	0.952
1GB	27.336	26.898	27.102

Se definieron dos espacios de nombres, uno para los datos: `http://www.rdfclip.com/resource/`, abreviado con el prefijo **CLIP** y otro para el modelo de datos: `http://www.rdfclip.com/schema#` abreviado con el prefijo **CLIPS**. El grafo del modelo de datos extiende aquel definido por las ontologías NEPOMUK [10].

Para identificar cada archivo se optó por el uso de *hashes* MD5, se prefirió por sobre otros *hashes* porque se privilegió la velocidad de cálculo del hash por sobre la improbabilidad de colisión, se realizaron pruebas usando MD5, SHA1 y CRC32, el cuadro 4.1 muestra los tiempos de cálculo para archivos de distintos tamaños haciendo uso de las librerías de mayor disponibilidad en Ubuntu.

De esta forma, a los archivos se les asignan identificadores de la forma `http://www.rdfclip.com/resource/{MD5-HASH}` o equivalentemente `CLIP:MD5-HASH`.

4.1.3. Ontologías e interoperabilidad

Para la puesta en marcha, es necesario definir ciertas ontologías que permitan modelar de forma adecuada el contexto semántico de los archivos. Éste es un elemento fundamental para la masificación de la plataforma pues, como se explicó anteriormente, el buen manejo de identificadores es una de las bases para obtener la extensibilidad y flexibilidad que caracteriza a RDF. Algunas de estas ontologías ya se encuentran parcialmente definidas por los proyectos mencionados en la sección 3.4 y, siguiendo la filosofía RDF, es importante mantener consistencia con ellos. En particular, el proyecto NEPOMUK [10] define ontologías de metadatos de archivos bastante completas que se usarán como base, las cuales se han extendido con vocabularios propios según los requerimientos del sistema. Gracias a la naturaleza de RDF, la introducción de nuevas clases y propiedades en la ontología se traduce en el ingreso de nuevos triples al grafo, dando cuenta de la extensibilidad del lenguaje.

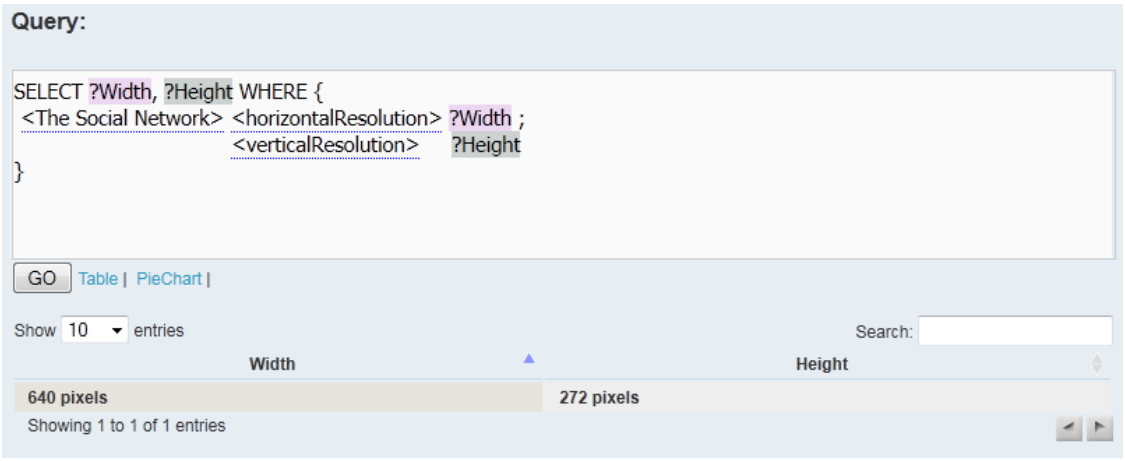

Figura 4.3: Interfaz de consulta web por SPARQL. Esta consulta obtiene el ancho y alto en pixeles de un archivo de video. Se hace uso de etiquetas fácilmente legibles en vez de URIs explicitas lo cual simplifica la construcción de la consulta.

Adicionalmente, al ya existir ciertos identificadores asociados a archivos por otros proyectos tales como Bitzi [18], se debe cosiderar un mecanismo que permita la interoperabilidad de estos, identificando recursos equivalentes entre grafos. Para ello, existe un predicado definido por **OWL** [23], `owl:sameAs`, el cual indica que un determinado identificador semánticamente se refiere al mismo recurso que otro identificador. Así,

```
R1  owl:sameAs  R2
```

indica que `R1` y `R2` son equivalentes. De esta forma se pueden integrar fácilmente datos de dos fuentes distintas que usen ciertos identificadores diferentes luego de explicitar esas equivalencias con este predicado.

4.2. Captura y acceso a los metadatos

La indexación de archivos debe ser un trabajo colaborativo, el grafo que almacene estos metadatos debe ser de acceso público y todo aquel que lo desee debe poder colaborar. Como

metodo básico de captura, es fundamental que todo usuario pueda registrar conocimientos que él posee sobre algún determinado archivo dentro del grafo, el cual será accesible posteriormente por el resto de los usuarios. La interfaz para realizar esta acción es fundamental, debe ser sencilla de tal forma que usuarios sin conocimiento profundo de las tecnologás involucradas puedan publicar metadatos.

El caso de uso básico en que el usuario solicita información sobre un determinado archivo se desarrolla como sigue:

1. El usuario selecciona el archivo en cuestión.
2. Mediante un menú contextual (Click derecho en Windows) o si lo desea mediante linea de comando, manifiesta su interés de navegar a la ficha online del archivo.
3. Se despliega la ficha online en el browser del usuario, la cual provee una interfaz web sencilla para visualizar los metadatos registrados y agregar nuevos.

Se provee además una interfaz de acceso programatico al grafo que permita la automatización del proceso mencionado anteriormente a través de requests HTTP a mediante un webservice REST.

Adicionalmente, se abre el accceso a los datos a través de un *endpoint* SPARQL, y se provee una interfaz de consultas en la misma aplicación, la cual, además de ser accesible directamente por el protocolo HTTP, se puede utilizar mediante una interfaz web especialmente diseñada para hacer la construcción de consultas más sencilla, el desarrollo de dicha interfaz derivó en dos 'subproductos' de este trabajo de título, un administrador de *triplestores* que permite consultar, visualizar y modificar los contenidos de un grafo a través de un *endpoint* y un plug-in de jQuery que permite el procesamiento de texto escrito por el usuario en tiempo real para la interpretación de comandos y reconocimiento de patrones con el fin de identificar y explicitar la semántica de la entrada. Ambos proyectos se liberaron como código abierto y los detalles de la implementación se describen en las secciones 4.4.2 y 4.4.3. La figura 4.3 muestra un ejemplo de esta interfaz, consultando por el ancho y alto de un determinado archivo de video, es importante notar como el concepto de las URIs está abstraído del usuario, simplificando la consulta.

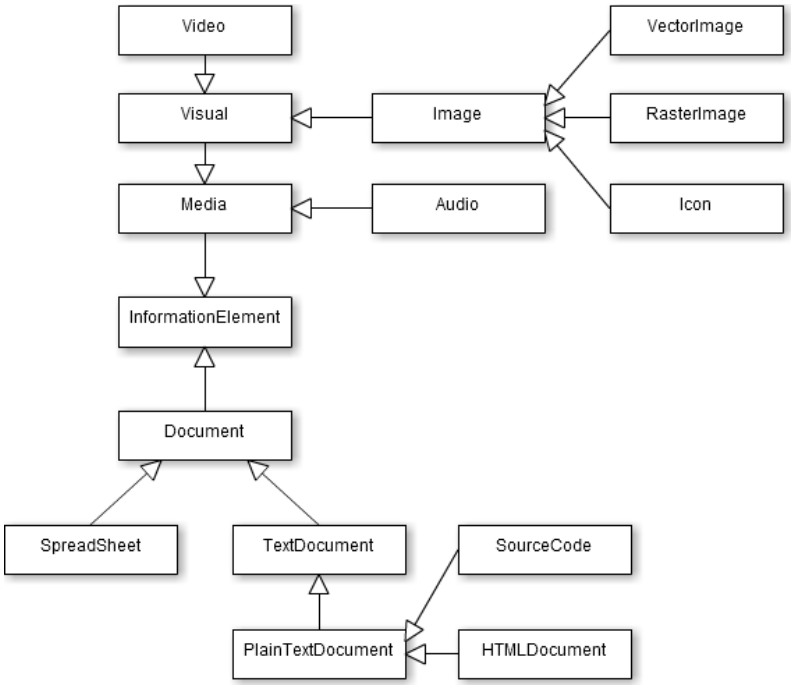

Figura 4.4: Extracto de diagrama de clases NEPOMUK. Las flechas indican herencia, estas clases se usan para indicar el tipo de contenido existente en un archivo.

4.2.1. La ficha de archivo

La aplicación expone públicamente una ficha para cada archivo indexado. Una analogía se podría hacer con los perfiles de Facebook y sus usuarios. Los modelos de datos para los distintos tipos de archivos de basan en las ontologías del proyecto NEPOMUK [10], que describen las propiedades existentes en varios tipos de archivos, incluyendo imágenes, archivos de audio y documentos. La figura 4.4 muestra parte de las clases definidas según los tipos de archivos en estas ontologías.

En cada ficha de archivo se listan las propiedades de éste, las cuales se determinan de la ontología haciendo uso del sistema de inferencia SPARQL que provee Virtuoso. Esto permite que propiedades como "Resolución Vertical", comunes a distintos tipos de archivos (imagenes, videos y otros), sean declaradas una sola vez en una clase madre ("Elemento Visual" en este caso), haciendo efectiva la herencia de clases a partir de la especificación en la ontología. A partir de esto, se les presenta a los usuarios una interfaz para completar la

Figura 4.5: Prototipo de ficha de archivo. Las propiedades desplegadas se determinan según las ontologías de archivos haciendo uso de inferencia SPARQL.

ficha agregando metadatos. Aprovechando que cada propiedad define su *rango* y *dominio*, indicando el tipo de recursos que pueden ser valores de dicha propiedad y la clase a la que pertenece, se desarrolló una librería en *JavaScript* que permite capturar *input* del usuario, así como formatear datos y proveer las interfaces adecuadas para modificar el grafo según los tipos de datos que se están manipulando. La figura 4.5 muestra un prototipo de una ficha de archivo reducida para este documento.

En el apéndice A se encuentran las definición de nuevos identificadores basados sobre la ontología NEPOMUK para efectos de implementar adecuadamente una interfaz de usuario que se adaptara según el conjunto de archivos que se usó de prueba.

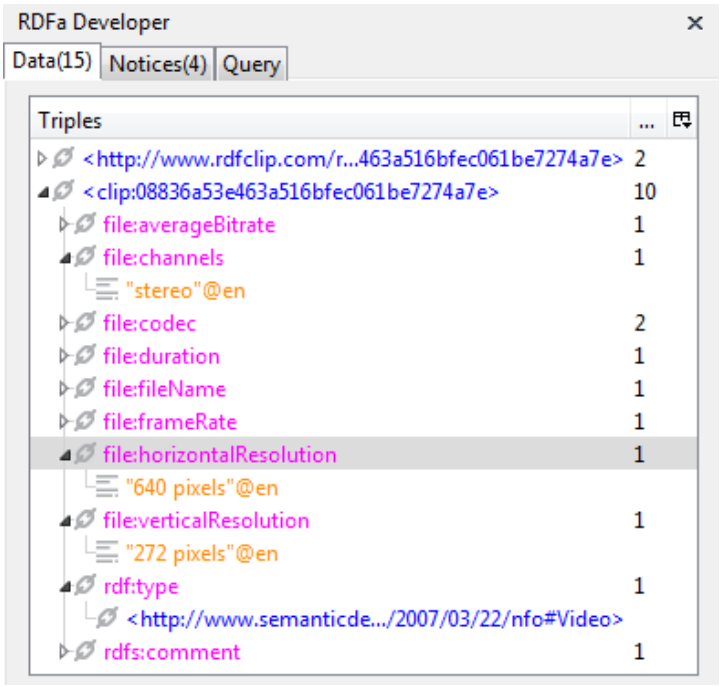

Figura 4.6: Obtención de metadatos mediante RDFa. Los exploradores RDFa procesan el HTML del documento en busca de la semántica de los datos presentados al usuario.

Publicación de datos mediante RDFa

Siendo la ficha de archivo construida íntegramente a partir del grafo RDF, se da la posibilidad de exponer la semántica de los contenidos a partir de los metadatos disponibles, adicionalmente a la representación en HTML. De esta forma la información contenida en las fichas de archivo puede ser "entendida" por el computador si se accede a ella mediante un explorador que lo soporte. Existen extensiones de los navegadores más populares de Internet que permiten acceder a estos metadatos. Esto provee una interfaz más de acceso a la base de datos. La figura 4.6 muestra como se puede navegar esta información correspondiente a una ficha como la de la figura 4.5 a partir de la información semántica incluida en el HTML mediante RDFa haciendo uso de una extensión de Mozilla Firefox.

4.2.2. Enlaces externos (e internos)

RDFClip no pretende ser un repositorio de todo tipo de datos. Basándose justamente en los principios de *Linked Data*, la información almacenada se limita a los metadatos de archivos, y refiere al usuario mediante links a otras fuentes de información cuando es adecuado. Acá se tomó una decisión de diseño importante, pues no hay estándares claros en cuanto a la asignación de URIs a nivel global. ¿Quién determina cuál es la URI que uno debería usar para referirse a Chile? DBPedia por ejemplo identifica a Chile con la URI `http://dbpedia.org/resource/Chile`, y lista al menos 5 URIs más como equivalentes. Sin embargo, es muy improbable que un usuario promedio haga uso de esos identificadores, probablemente preferirían algo como `http://www.chile.cl` o la página de Wikipedia de Chile.

Considerando que el espíritu del proyecto está centrado en la accesibilidad de información al usuario común, se tomó la decisión de usar, cuando sea posible, URLs de sitios de uso masivo, tales como Wikipedia o IMDB (Internet Movie Database) para identificar recursos externos. Esto se traduce en que usuarios pueden colaborar con la indexación semántica copiando y pegando URLs de sitios que ya usan. Se diseñó una interfaz de usuario que apunta específicamente a este fin.

A modo de ejemplo, supongamos que un usuario quiere indicar que un determinado archivo, al cual se le ha asignado el identificador `clip:8ae79e107a`, contiene la película The Matrix. Al usuario se le indica que pegue un link a IMDB que corresponda a la película contenida en el archivo, esta URL es procesada para obtener el título de la película, y se mantiene el link externo. Esta acción del usuario se traduce en 4 triples nuevos en el grafo (las URLs están acortadas por legibilidad):

`clip:8ae79e107a`	`clips:movieContent`	`http://imdb.com/tt0133093`
`http://imdb.com/tt0133093`	`rdf:type`	`clips:Movie`
`http://imdb.com/tt0133093`	`rdfs:label`	`‘‘theMatrix’’`
`http://imdb.com/tt0133093`	`clips:userLabel`	`‘‘The Matrix’’`

De esta forma el grafo obtiene la información mínima para manipular este nuevo recurso

(la película The Matrix) y se mantiene consistencia entre los identificadores. Si dos usuarios agregan metadatos a dos archivos de video distintos haciendo uso de la misma URL de IMDB, se entiende que ambos archivos contienen la misma película.

Este mismo principio se aplica para enlaces internos, por ejemplo, para explicitar una relación entre dos archivos, basta copiar y pegar la URL de la ficha del archivo correspondiente.

La implementación de este comportamiento desde el punto de vista de interfaz de usuario se tradujo en el desarrollo de una librería que hace uso de los tipos RDF para interpretar y desplegar información desde y hacia el usuario según el tipo de dato que se está manipulando, los detalles se pueden encontrar en la sección 4.4.4.

4.3. Pruebas de indexación

Como conjunto principal de datos de prueba se utilizó, previa autorización del Área de Desarrollo de Infotecnologías (ADI) de la Facultad, el material docente de U-Cursos. Éste incluye material de estudio, documentación, diapositivas, apuntes, examenes, controles, etc. de varias carreras de toda la universidad. Se indexaron todos los archivos ingresados desde Marzo de 2003 hasta Septiembre de 2011, con un total de más de 300 mil archivos dando origen a 1 millon y medio de triples RDF. A pesar de la cantidad de datos, las operaciones de consultas al *triplestore* se resuelven de forma fluida. Sin embargo se tomó la decisión de optimizar mediante un índice en una base datos relacional algunas consultas sencillas que se repiten con gran frecuencia, tal como la busqueda de recursos (ya sean archivos, clases, propiedades, etc.) a partir de su etiqueta (`rdfs:label`). Idealmente, para comienzos del semestre de Primavera 2011 se abrirá este servicio de búsqueda a los alumnos.

Adicionalmente, se estan indexando contenidos generados por celulares mediante un acuerdo con WeSync.tv, empresa que recopila contenido multimedia de eventos generado por los asistentes, acá se indexan fotos, imagenes y audio. Se les dá un contexto semántico dado el evento en el cual fueron capturados, y muchas veces se vincula a otras fuentes del contenido en la red (Facebook, Instagram, YouTube). Siendo un proyecto emergente, aún no se recopila un volumen considerable de datos.

Finalmente, se indexaron aproximadamente 500Gb de material audiovisual, series, pelícu-

las y música con el fin de disponer de metadatos para realizar pruebas locales de relaciones entre archivos. Así como 5gb adicionales de material de estudio para cursos de la facultad, para probar la integración de datos con el contenido indexado de U-Cursos.

A partir de estas pruebas se determinó que una buena práctica es mantener grafos separados considerando que muchas veces se busca realizar consultas dentro de un subconjunto de todos los datos. Esto trae consigo importantes beneficios de velocidad, de esta forma, se destinó un grafo etiquetado `http://www.rdfclip.com/data/ucursos` a los metadatos de material docente de U-Cursos, y otro `http://www.rdfclip.com/data/wesync` a los metadatos del contenido de dispositivos móviles de WeSync.tv, además de un grafo genérico `http://www.rdfclip.com/data` disponible para cualquier conjunto de triples que contenga información de archivos. A la fecha no se ha definido una regla de delimitación ni nombres de grafos dada la naturaleza sencilla del prototipo implementado.

4.4. Detalles de implementación

La siguiente sección detalla varios componentes de particular interés en el desarrollo de RDFClip. Muchos de estos componentes se liberaron en forma de código abierto de forma separada al proyecto.

4.4.1. Control de cambios

Dada la naturaleza colaborativa del proyecto, lo cual implica que en potencia cualquier persona puede agregar y eliminar contenido de la base de datos, es importante contar con un mecanismo que permita mantener un registro de triples ingresados y eliminados del *triplestore*, de tal manera de contar con una historia de cambios y poder deshacer algunos cuando sea necesario, de forma similar a como lo hacen otros proyectos que recopilan información de manera colaborativa, tales como Wikipedia.

El hecho de que un grafo RDF se pueda representar simplemente como un conjunto de triples facilita bastante el problema. En principio, se puede contar con toda la historia de la evolución del grafo si cada triple ingresado o eliminado queda registrado con la fecha en que se realizó la acción. Tenemos así una evolución cronológica de la base de datos, donde

Figura 4.7: *Tagging* semántico en Facebook. Este tipo de procesamiento en tiempo real de texto es trivial de implementar haciendo uso de la librería.

el deshacer un cambio se reduce a realizar la acción opuesta correspondiente a éste, basta con quitar el triple que se agregó o bien agregar el triple que se eliminó. Al no existir una estructura con datos segregados por tablas con llaves foráneas que relacionan las filas de éstas como en las bases de datos relacionales, el problema se simplifica enormemente. La base de datos relacional de apoyo a la aplicación registra cada uno de los movimientos.

El costo en tamaño de este mecanismo puede ser bastante alto pues se lleva un historial de cada triple del grafo lo cual al menos significa replicarlo. Una alternativa estudiada consiste en guardar imagenes (*snapshots*) de fragmentos del grafo que se encuentran estables y solo registrar las modificaciones posteriores a la última imagen capturada, sin embargo para el prototipo implementado la primera solución cumple el objetivo y se implementó de esta forma.

4.4.2. Cuadros de texto semánticos

Dentro de los objetivos se plantea la necesidad de introducir a desarrolladores e ingenieros a las tecnologías de la Web Semántica, una de las barreras de entrada se produce en la adopción de SPARQL como lenguaje de consulta. A diferencia de su par de bases de datos relacionales, SQL, la documentación es escasa y los expertos son pocos, esto se suma a los problemas de notación y poca claridad que se produce con el uso de cadenas de caracteres muy largas como identificadores.

Lo anterior motivó el desarrollo de una interfaz que facilitara construcción de consultas SPARQL, la cual tiene como base un cuadro de texto donde se recibe une entrada del usuario

(la consulta), pero que se va construyendo de forma interactiva, reconociendo patrones y sugiriendo formas de completarla. Esto derivó en un plugin de jQuery (librería JavaScript ampliamente utilizada en desarrollo web) que provee estas funcionalidades en contextos más amplios que la construcción de consultas para el que fue creado, este código fue liberado de forma abierta con licencia MIT a través de GitHub.

La librería hace uso de expresiones regulares para identificar patronesa medida que el usuario tipea contenido. El código 4.1 muestra un ejemplo de su uso, el cual identifica los patrones que calzan con la expresión regular `/rainbow(s)?/` y pinta las letras de distintos colores.

```
(new MagicTextArea($('#custom_textarea'))).setParser({
    parseText : function(txt){
        return match(txt, /rainbow(s)?/ig, function(w){
            var span = jQuery('<span>');
            for(var i = 0; i < w.length; i ++){
                span.append($('<span>').html(w[i]));
                span.css('background',getRandomColor());
            }
            return {span : span};
        });
    }
});
```

Código 4.1: Ejemplo uso cuadros de texto semánticos. Se define un procesador de texto que calza la expresión regular `/rainbow(s)?/` y la marca con un patrón multicolor.

Optimización

La implementación se basa en la inclusión de filtros personalizables a un área de *input* de texto los cuales procesan la entrada del usuario y se aplican con cada modificación de la entrada. En pricipio esto puede ser muy costoso, pues cada filtro debe procesar el texto entero. Suponiendo un texto de largo n, y m filtros, esto tiene una complejidad en tiempo de

$O(n \times m)$. Lo cual puede ser aceptable para la primera vez que el usuario carga un texto (al cargar la página por primera vez por ejemplo), pero no lo es para el procesamiento en tiempo real, el cual debe efectuarse varias veces por segundo (después de cada tecla presionada). La solución implementada minimiza este trabajo obligando a los filtros a segmentar la entrada luego de procesarla. Cada vez que se presiona una tecla, se reprocesa el segmento modificado (y en casos especiales los segmentos contiguos), evitando así un reprocesamiento total. Un filtro normalmente segmentará la entrada tantas veces como sea aplicable (según los distintos patrones que reconozca), subdividiendo la entrada en k segmentos, por lo que el reprocesamiento efectuado después de cada tecla presionada tiene un costo de $O(\frac{n \times m}{k})$, donde muchas veces $k > m$ e incluso puede crecer linealmente con n. En la sección 4.4.3 se muestra el uso del plugin para la construcción de consultas SPARQL, la figura 4.9 hace uso de dos filtros, uno que reconoce variables de la consulta (que comienzan con '?'), y otro que sugiere autocompletado de recursos por sus etiquetas, evitando la manipulación de URI's.

4.4.3. Explorador de RDF

En el desarrollo de la aplicación web de RDFClip, se consideró desde un comienzo un sistema que permitiera navegar y consultar los datos RDF de forma sencilla, y que fue siendo adaptado al uso particular del contexto de este trabajo de título. Sin embargo, la base sobre la cual fue desarrollada esta aplicación es mucho más flexible y se utilizó para la confección de un administrador de *triplestores* el cual fue liberado de forma abierta con licencia MIT a través de GitHub [33]. La misma aplicación se ha presentado a otros desarrolladores de la Facultad quienes se han mostrado interesados, incluyendo al ADI (Area de Desarrollo de Infotecnologías de la Facultad) y otros entendidos en RDF.

El explorador RDF provee una interfaz web de consultas a *endpoints* SPARQL haciendo uso de cuadros de entrada de texto semánticos (ver sección 4.4.2), esto permite abstraer identificadores complicados del usuario, exponiendo nombres más legibles y fáciles de manipular. La figuras 4.8 y 4.9 muestran la construcción de consultas SPARQL con y sin el uso de este plugin respectivamente.

Ejemplos de aplicaciones de este sistema se encuentran en redes sociales como Facebook o Twitter, donde los usuarios pueden *taggear* a otros haciendo uso de un caracter específico (normalmente '@'), esto inserta un texto 'especial' en el lugar del cursor el cual referencia

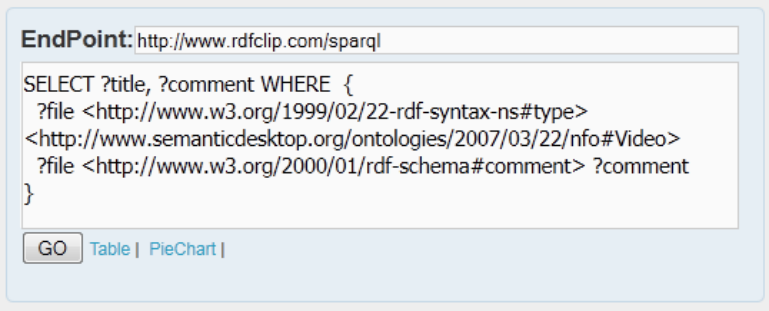

Figura 4.8: Consulta SPARQL regular. Las URIs largas dificultan la lectura y son muy difíciles de recordar. Si bien el uso de prefijos facilita la lectura de la consulta, es muy común copiar y pegar URIs debido a su extensión, lo que resulta incómodo.

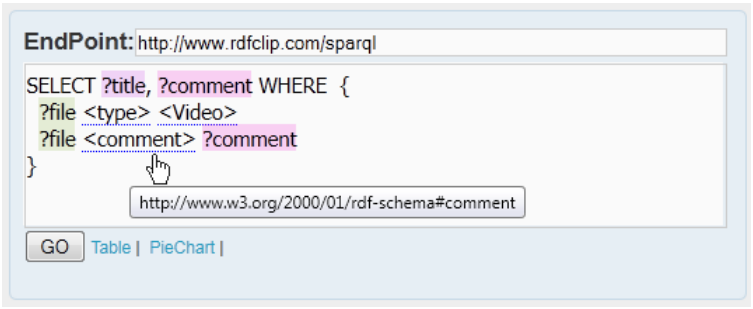

Figura 4.9: Consulta SPARQL con cuadros de texto semántico. Las URIs se han abstraido del usuario y se presentan etiquetas de fácil legibilidad en su lugar.

al otro usuario en cuestión. Esto es interesante en el area en que se desarrolla esta memoria de título, pues justamente se trata de explicitar la semántica de un concepto dentro de un texto ingresado por el usuario (ver figura 4.7).

Funcionalidades principales

Dentro de las funcionalidades principales del explorador RDF encontramos:

- Consultar cualquier *endpoint* SPARQL público, local o remoto.
- Visualizar conjuntos de datos en forma de tablas o gráficos.
- Inspeccionar rápidamente el grafo haciendo uso de consultas prefabricadas para estos

fines.

- Almacenar consultas favoritas.
- Listar los grafos presentes en el *triplestore* y navegarlos.
- Mantener un historial de las consultas realizadas.
- Listar los arcos salientes de un nodo y agregar nuevos, modificando el grafo (lo cual se convertiría en la ficha de archivo, ver 4.2.1).

Esta aplicación es análoga a aplicaciones como phpMyAdmin [26] utilizada para administrar y consultar bases de datos MySQL. Es importante destacar la necesidad de una aplicación similar para *triplestores*, pues al almacenar datos en grafos, su visualización resulta mucho más complicada que en bases de datos relacionales, al estar mucho menos estructurados. No hay datos separados por tablas con estrucutaras bien definidas, solo nodos y arcos.

Siendo la aplicación de RDFClip una versión adaptada de este administrador de base *triplestores*, la arquitectura descrita en 4.1.1 aplica también para este componente.

4.4.4. Librería Javascript para RDF

La aplicación desarrollada usa fuertemente JavaScript para el despliegue, captura y preprocesamiento de metadatos, así como la conexión a a *triplestores*. En esta sección se detalla la implementación de estos componentes.

Interpretación de tipos

El uso de identificadores universales no tiene ningún sentido si no hay aplicaciones que los soporten, esto significa que, ya acordando la semántica de un determinado recurso, este se inteprete apropiadamente según dicha semántica. Ejemplos de esto se pueden ver en el motor de inferencia de Virtuoso, el cuál interpeta correctamente predicados que describen modelos de datos (ver sección 2.5.1). Basado en la misma idea, se construyeron objetos de `tipos` JavaScript asociados a recursos RDF específicos que determinan como se debe

considerar el input del usuario y desplegar la información. A modo de ejemplo, cuando un usuario ingresa el valor '300' en un cuadro de texto para atribuirle un valor a un atributo de un nodo del grafo RDF, este se debe interpretar de forma completamente distinta si se trata de un atributo que espera un número, una cadena de caracteres o el nombre de una película.

De esta forma, se definió una forma estandar en que los `tipos` proveen funciones de validación, procesamiento de entrada, despliegue de interfaces de usuario y otros. Idealmente, si terceros quisieran modificar el grafo externamente, debieran usar estos tipos (y extenderlos de ser necesario) para asegurar la consistencia de datos.

Conexión a *triplestores*

Para efectos de la ejecución de consultas SPARQL en el servidor virtuoso se hizo uso del *endpoint* que éste ofrece, el cual recibe consultas a través de una interfaz HTTP mediante peticiones GET, para esto se usó las capacidades de AJAX de la librería jQuery. Debido a que por seguridad no se pueden realizar peticiones HTTP via Javascript a dominos externos (en este caso, el dominio del servidor Virtuoso), se hizo uso de un proxy desarrollado por *data.gov* [27], uno de los proyectos de *Open Data* más grandes del mundo. También se incluye una librería que hace uso JSONP [36], para peticiones HTTP entre dominios.

La librería hace uso del proxy local para obtener los datos en formato JSON y los encapsula en objetos de fácil manipulación, asociando los identificadores a objetos de tipos cuando es posible. La interfaz programatica se simplificó al máximo, el código 4.2 muestra un ejemplo simple de la ejecución una consulta.

```
var sparql = "SELECT␣*␣WHERE␣{?subject␣?predicate␣?object}";
getProxy().query({
    format : "json",
    query  : sparql,
    callback : function(r){
            var resultSet = RDF.getResultSet(r);
            while(row = resultSet.fetchRow()){
                //row.name contains the ?name variable
            }
    },
    error : function(e){
        notifyUser("An␣error␣ocurred␣:"+e);
    }
});
```

Código 4.2: Ejemplo de consulta SPARQL por JavaScript. La cosulta SPARQL es enviada al proxy quien la resuelve asíncronamente. Se proveen utilidades para iterar fácilmente por el conjunto de resultados.

4.4.5. Automatización de procesamiento de archivos

Para lograr un indice útil de archivos, es indispensable contar con mecanismos que permitan obtener metadatos de archivos de forma automática. Normalmente existen tres fuentes de metadatos, aquellos que son expuestos por el sistema operativo (tamaño, nombre del archivo, etc.), aquellos que el mismo formato de archivo define dentro de un encabezado (como el ID3 tag de los archivos MP3) y aquellos solo inferibles a partir del contenido en sí, 'entendiendolo'. Existen librerías que permiten la extracción de metadatos para cada uno de estos niveles. Debido a la gran complejidad que supone el análisis programático del contenido real de un archivo (mediante, por ejemplo, análisis de Fourier o calculo de correlación de señales para identificar el contenido de un archivo de audio), las herramientas usadas para indexación rápida de archivos en este proyecto se basan principalmente en los metadatos que expone el sistema operativo y los encabezados de archivos.

La librería de python `hachoir-metadata` [32] permite extraer metadatos de archivos multimedia y se utilizó para el desarrollo de herramientas de indexación automática. Esto incluye metadatos técnicos como la resolución de una imagen y, cuando están disponibles, semánticos, como el artista de una canción. Incluye soporte para muchos formatos, incluyendo ID3 Tag de MP3 y EXIF de fotografías.

Adicionalmente, se desarrollaron herramientas de prueba semi-exitosas para obtener metadatos de archivos de la Web cuando es posible a través de la automatización de consultas y procesamiento de respuestas HTML (*web scrapping*). Muchas veces el contenido de un archivo puede ser determinado a partir de bśquedas de su nombre en la Web. Este mismo procedimiento se usa en RDFClip para vincular externamente contenidos a Wikipedia, IMDB y otros sitios, una vez que el usuario hace explicita la relación del archivo con el contenido referenciado. Se usó la librería de Python `BeautifulSoup` para todos los processos de *web scrapping*.

4.5. API REST

De forma adicional a la interfaz de usuario HTML y al endpoint SPARQL, se dispuso de una API REST [34] que permite interactuar con el *triplestore* haciendo uso de los distintos métodos de peticiones HTTP. La motivación viene de la importancia de la retrocompatibilidad en la adopción de nuevas tecnologías; si se quiere que los desarrolladores aporten con conjuntos de datos en formato RDF deben primero empezar a usar el sistema, pero para esto deben estar familiarizados con las tecnologías RDF, pero no hay motivación para hacerlo si es que no se cuentan con datos en el sistema. Joel Polsky, ex desarrollador de Microsoft denomina este patrón como el problema del *huevo y la gallina* [35]. La solución al problema del huevo y la gallina consiste en la retrocompatibilidad con tecnologías ya existentes con el fin de atraer a usuarios familiares con esos sistemas. En el area de publicación de dato las APIs REST han sido la norma en los últimos años, grandes compañias han hecho publicos sus datos de esta forma, incluyendo muchas redes sociales como Twitter, FourSquare y Facebook. Lo cual hace sentido dada la naturaleza colaborativa de las redes sociales, aspecto que está presente también en esta memoria de título.

Cuadro 4.2: API Rest - Metadatos de archivo.

Tipo	Metadatos de archivo.
URL	`http://www.rdfclip.com/api/(hash-md5)/`
Métodos aceptados	POST, GET.
Descripción	Dado un identificador de archivo (`hash-md5`), permite ingresar (POST) y leer (GET) los metadatos de ese archivo.

Cuadro 4.3: API Rest - Recursos vinculados.

Tipo	Recurso vinculado a archivo.
URL	`http://www.rdfclip.com/api/(hash-md5)/(predicado)`
Métodos aceptados	POST, GET.
Descripción	Dado un identificador de archivo (`hash-md5`), y un predicado de la forma `prefijo:sufijo` (e.g. `clips:related`) permite ingresar (POST) y leer (GET) los recursos vinculados al archivo mediante ese prediado.

4.5.1. Puntos de Acceso

Se definieron tres puntos de acceso a los metadatos, éstos se presentan en los cuadros 4.2, 4.3 y 4.4. Ejemplos de uso de estos puntos de acceso se pueden encontrar en `http://www.rdfclip.com/api/`.

Cuadro 4.4: API Rest - Triples genéricos.

Tipo	Triples genéricos.
URL	`Recurso vinculado a archivo`
Métodos aceptados	POST.
Descripción	Permite ingresar multiples triples a un grafo determinado.
Parametros	
data	Arreglo de triples a insertar.
graph:	Nombre del grafo donde ingresar los triples (opcional).

Ejemplo de aplicación

Haciendo uso de la API REST es sencillo desarrollar aplicaciones que interactúan con el *triplestore* de RDFClip. A modo de ejemplo, se desarrolló una herramienta sencilla que permite vincular archivos en el grafo, y luego hacer uso de esos metadatos para buscar archivos locales. De esta forma un usuario puede indicar, por ejemplo, que un determinado archivo `lordOfTheRings.avi` tiene un archivo de subtítulos `lordOfTheRings.srt`.

```
1 $ clip lordOfTheRings.avi clip:hasSubtitle lordOfTheRight.srt
```

Eventualmente otro usuario que tenga los mismos archivos puede hacer una consulta al respecto:

```
1 $ clip --find lordOfTheRings.avi clip:hasSubtitle ?
2 /home/user/lordOfTheRings.srt
```

La aplicación 'clip' obtiene los *hashes* de los archivos y los envía por HTTP junto con el predicado que los relaciona, esta relación es almacenada en el grafo de RDFClip. Posteriormente, la opción `--find` indica que se quiere buscar el objeto de la relación enunciada por los dos parametros siguientes en el directorio actual.

Capítulo 5

Conclusiones

El sistema diseñado (RDFClip) es un primer paso para resolver aquellas interrogantes planteadas en la introducción de esta memoria de título. Contando con un servicio de acceso público de metadatos de archivos se puede hacer un manejo más eficiente de los contenidos de nuestros computadores. El uso de identificadores basados en *hashes* permite asociar unívocamente un archivo a su nodo correspondiente en el grafo RDF, independiente del sistema de almacenamiento o la ubicación del archivo. Sin embargo, dada su naturaleza colaborativa no es aplicable para la indexación de archivos privados.

Al momento de escribir esta memoria de título se han indexado varios GBs de música en formato MP3, archivos de videos con sus subtitulos correspondientes y material de estudio de cursos de ingeniería de la Facultad. Se está discutiendo la posibilidad de indexar y y vincular todo el material docente de acceso (semi)público de U-Cursos a modo de que sea más fácil obtener material de estudio.

Un tema no muy tratado es aquel de los derechos de autor. En principio la legalidad de la indexación de archivos no debiese ser un problema pues no se publican los archivos, sino metadatos de los mismos. Por otro lado, parte de estos metadatos pueden ser enlaces de descarga a material potencialmente protegido por derechos de autor. No es el objetivo del sistema apuntar en esa dirección.

Gran énfasis se le ha dado a la accesibilidad de los datos, tanto a nivel de usuario como de desarrollador. Se desarrollaron sistemas que facilitan la construcción de consultas SPARQL

así como interfaces de usuario de alto nivel que permiten navegar y modificar grafos RDF haciendo uso de diversos métodos de acceso: servicios REST, *endpoint* SPARQL y RDFa.

En cuanto a la escalabilidad, se usaron otros proyectos de *Linked Data* como punto de referencia. La estructura del sistema es sencilla y las consultas SPARQL que se realizan sobre el grafo no parecen verse notablemente afectas por el tamaño de este. Sin tener información sobre la implementación de la resolución de consultas en Virtuoso es imposible hacer un análisis más detallado. Sin embargo pruebas sobre grafos de gran tamaño (como DBPedia, con más de mil millones de triples) con consultas similares a aquellas que se utilizan para el uso estandar de la aplicación responden en tiempos razonables, lo que da cuenta de la escalabilidad de la solución.

Las aplicaciones de manejo de datos RDF se han distribuido a diversos usuarios de estas tecnologías, quienes han recalcado la necesidad de herramientas de este tipo y han provisto de *feedback* para mejorarlas.

Con respecto a la 'evangelización' hacia la Web Semántica que se planteó, la liberación de librerías de desarrollo, así como la disponbilización de guias prácticas de desarrollo debieran, cuando menos, evitar en algunos desarrolladores las frustraciones que se produjeron durante la implementación del prototipo. En el mismo ámbito, se discute el uso de parte de este documento como apunte del curso CC71X - La Web de Datos dictado en la Facultad, único curso en el cual se tratan estos temas.

5.1. Cumplimiento de objetivos

El objetivo general de esta memoria de titulo, que consistía en la elaboración de un prototipo de una plataforma de indexación semántica de archivos, se concretó y se encuentra disponible en `http://www.rdfclip.com/`. Si bien es sólo un prototipo y hay mucho que se puede hacer para mejorarlo, prueba la factibilidad tecnica del sistema e incluso tiene ya aplicaciones prácticas, tales como la indexación de material docente de los cursos de la universidad, servicio que ya se encuentra disponible.

Con respecto a los objetivos específicos planteados:

1. Se buscó un diseño de arquitectura simple que se concretó y presentó en este docu-

mento, basado en una aplicación web con multiples interfaces de acceso.

2. Se utilizaron ontologías predefinidas para la representación de datos de archivos con ciertas modificaciones según fue necesario.

3. La sección de Antecedentes de esta memoria de título sirve como introducción a los conceptos importantes de Web Semántica y se ha editado para ser distribuido con esos fines.

4. La aplicación de RDFClip se encuentra en linea y ya provee interfaces de ingresos de datos y consultas.

5. Se desarrollaron aplicaciones simples que hacen uso de la API REST de RDFClip para la aplicación de búsqueda de archivos locales basandose en metadatos almacenados externamente.

6. Durante el transcurso de esta memoria de título se desarrollaron algunas herramientas que permitieron indexar masivamente archivos de forma semiautomática. Sin embargo no estan lo suficientemente acabados para su distribución. Estos se usaron para indexar grandes cantidades de archivos de música, video, texto y material docente de U-Cursos.

7. Existe un *endpoint* disponible para la realización de consutlas SPARQL en `http://www.rdfclip.com/sparql`.

5.2. Problemas encontrados y limitaciones

El foco principal de este trabajo de título fue desviandose fuertemente desde su concepción original. La razón principal de esto fue el grado de desconocimiento de las tecnologías utilizadas, lo cual se tradujo en demoras en el desarrollo. Es por esto que se ha intentado incorporar lo más posible de lo aprendido en ese proceso para ser transmitido en el producto final. Idealmente esta memoria puede servir como punto de partida para otros proyectos inspirados en la Web Semántica, ya que provee una serie de conceptos y herramientas de base, todas aquellas al menos que hicieron falta durante el desarrollo de este proyecto.

Las limitaciones del proyecto se desprenden de aquellas de RDF como lenguaje de representación de datos, así como de los sistemas sobre los cuales estos se almacenan y

consultan. Un buen aprovechamiento de los datos requiere de un sistema de inferencia más poderoso y eficiente que con el que cuentan los motores de *triplestores* más conocidos, pero esto está sujeto a cambiar conforme avance la aplicación de estandares y evolucionen los motores de bases de datos RDF. En su esecencia, el volumen de datos no parece ser una limitación para el sistema desarrollado, aunque esto está sujeto a las potenciales aplicaciones que se podrían construir sobre él. Sin embargo, se ha visto que para aplicaciones sencillas de aprovechamiento local de metadatos remotos el sistema parece comportarse bien.

Existen por otro lado muchos aspectos del sistema a pulir, en especial en temas de interfaz de usuario, siendo la implementación actual simplemente una prueba de concepto. Algunas funcionalidades descritas en este documento se encuentran, al momento de escribir esto, en un estado que no es suficientemente estable para el uso público y aún bajo desarrollo interno.

5.3. Trabajos futuros

Existen muchas posibles aplicaciones, pequeñas y grandes, que se podrían apoyar en la existencia de un sistema de centralizado indexación semántica de archivos. A continuación se describen a grandes rasgos algunas de éstas.

No más carpetas

Probablemente una de las primeras conclusiones que uno tiene al trabajar con estas tecnologías es que el uso de carpetas (directorios) no es el más adecuado para la organización de archivos. Sería realmente interesante ver un sistema de navegación semántica de archivos locales, ya teniendo los metadatos de los archivos, el resto es un desafío de interfaz y visualización, además de monitoreo permanente de archivos locales.

Ontologías más completas

La construcción de ontologías con reglas de inferencia bien definidas por sobre el grafo de archivos permitiría una navegación más efectiva. A modo de ejemplo, el material de estudio

actualmente indexado cuenta con controles y apuntes de distintos cursos de la Facultad, estos están organizados en un vocabulario sencillo que vincula al archivo con su curso y al curso con su area de conocimiento. Una buena descripción jerarquica de las diversas areas de conocimiento permitiría realizar búsquedas transversales entre diversos cursos de un mismo departamento o Facultad, y eventualmente podría vincularse con material de estudio de otras universidades. Estandarizando esta descripción, un alumno podría, por ejemplo, solicitar material de apoyo para su curso 'CC40A - Algoritmos y estructuras de datos' y obtener el material disponible de 'IIC2132 - Estructuras y representación de datos' de la Universidad Católica.

Publicación de archivos indexados

El prototipo desarrollado solo indexa metadatos de archivos y provee interfaces básicas para vincular a la descarga de estos archivos desde su ficha. Aprovechando la masificación de sistemas de almacenamieto en linea (como Dropbox y Ubuntu One), sería interesante automatizar la indexación de estos archivos al momento de almacenarlos 'en la nube'. Esto facilitaría la publicación de datos crudos en forma de archivos y manteniendo la indexación e interoperabilidad de RDF para la búsqueda. Alternativamente se podrían desarrollar servidores locales de archivos mediante FTP que indentifiquen cambios en los archivos compartidos y actualizen los indices con vínculos de descarga apropiados.

SPARQL de alto nivel

Sin duda uno de los grandes beneficios de almacenar datos en RDF es la posibilidad de consultarlos haciendo uso de SPARQL. Para que los usuarios puedan realmente explotar las relaciones entre los archivos y la disponibilidad de metadatos en el sistema, lo ideal sería contar con una interfaz de alto nivel que permita la construcción de consultas complejas.

Integración local

Teniendo ya un sistema que almacena metadatos online, su verdadera utilidad se ve reflejada en la aplicación de esa información adicional en los archivos locales. Si bien en el desarrollo de esta memoria se hicieron aplicaciones sencillas que pueden responder a

consultas que relacionen archivos locales haciendo uso de los metadatos almacenados en el servidor, una mayor integración con el sistema operativo masificaría y facilitaría su uso.

Mayor automatización

La arquitectura que soporta el índice semántico de archivos es inutil si no se cuenta con archivos indexados. Durante el desarrollo de este proyecto se han hecho pruebas de desarrollo de herramientas de indexación automática, pero esto no es suficiente, el problema en sí es tremendamente complejo. El contar con este tipo de herramientas de buen nivel y de uso masivo permitiría la indexación masiva de datos, lo que enriquece al sistema en su conjunto.

Capítulo 6

Bibliografía

[1] Uniform Resource Identifier
RFC 3986 - January 2005
http://tools.ietf.org/html/rfc3986

[2] rdfquery - jQuery plugin
http://code.google.com/p/rdfquery/wiki/RdfPlugin

[3] Tim Berners-Lee on the next Web
TED Talk - February 2009 http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html

[4] Tim Berners-Lee: The year open data went worldwide
TED Talk - February 2010 http://www.ted.com/talks/tim_berners_lee_the_year_open_data_went_worldwide.html

[5] Resource Description Framework (RDF) - February 10th 2004
http://www.w3.org/RDF/

[6] W3C Naming and Addressing: URIs, URLs, ...
http://www.w3.org/Addressing/

[7] Bernard Schandl, Niko Popitsch.
April 27th, 2010, North Caroline, USA.
Lifting File Systems into the Linked Data Cloud with TripFS

[8] Aperture
A Java framework for getting data and metadata. http://musicbrainz.org/

[9] MusicBrainz
The open music enciclopedia. http://aperture.sourceforge.net/

[10] NEPOMUK - The social Semantic Deskop
http://nepomuk.semanticdesktop.org/

[11] MetaTracker
http://projects.gnome.org/tracker/

[12] The Friend of a Friend (FOAF) project
http://www.foaf-project.org/

[13] SIOC Ontology
http://sioc-project.org/ontology

[14] DBPedia
http://www.dbpedia.org/

[15] DBPedia's Virtuoso SPARQL Endpoing
http://www.dbpedia.org/sparql

[16] Wikipedia
http://www.wikipedia.org/

[17] Linked Data
http://www.linkeddata.org/

[18] Bitzi - Bipedia Digital Media Enciclopedia
http://bitzi.com/

[19] OpenLink Virtuoso
http://virtuoso.openlinksw.com/

[20] Django MVC Framework Project
http://www.djangoproject.com/

[21] RDFLib Python Library
http://www.rdflib.net/

[22] Model View Controller
http://java.sun.com/blueprints/patterns/MVC-detailed.html

[23] Web Ontology Language
W3C Recommendation October 27th 2009
http://www.w3.org/TR/owl2-overview/

[24] RDFSchema
W3C Recommendation February 10th 2004
http://www.w3.org/TR/rdf-schema/

[25] O. Erling, I. Mikhailov. *SPARQL and Scalable Inference on Demand.*
Openlink Software.
http://docs.openlinksw.com/virtuoso/rdfsparqlrule.html

[26] phpMyAdmin
http://www.phpmyadmin.net/

[27] Linking data.gov to Linked Data Cloud - php proxy
http://data-gov.tw.rpi.edu/wiki/Linking_data.gov_to_Linked_Data_Cloud#add_a_php_proxy

[28] SPARQL Query Language for RDF
W3C Recommendation January 15th 2008
http://www.w3.org/TR/rdf-sparql-query/

[29] SPARQL 1.1 Query Language
W3C Working Draft May 12th 2011
http://www.w3.org/TR/sparql11-query/

[30] Official DBpedia Live Release
http://blog.dbpedia.org/category/dataset-releases/

[31] CC71X - La Web de Datos - Agosto 2010
Claudio Gutierrez
http://www.dcc.uchile.cl/~cgutierr/cursos/LOD/

[32] hachoir-metadata
Program to extract metadata using Hachoir library
http://pypi.python.org/pypi/hachoir-metadata/1.2.1

[33] RDFAdmin at GitHub
https://github.com/juanique99/RDFAdmin

[34] RESTful Web services
https://www.ibm.com/developerworks/webservices/library/ws-restful/

[35] Chicken and Egg Problems
Joel Polsky - May 24th, 2000
http://www.joelonsoftware.com/articles/fog0000000054.html

[36] JSON-P
http://www.json-p.org/

Apéndice A

Vocabulario RDF

Este apéndice describe la extensiones al vocabulario NEPOMUK utilizadas en la implementación de la aplicación. El cuadro A.1 lista las nuevas clases definidas y el cuadro A.3 las nuevas propiedades. Más que intentar ser una referencia completa de las modificaciones del vocabulario RDF, este apéndice busca ejemplificar los tipos de modificaciones necesarias para adaptar una ontología establecida (como las de NEPOMUK usadas en este proyecto) a las necesidades particulares de una aplicación. En este caso, los siguientes identificadores se han incorporado a la ontología a partir de la naturaleza de los archivos de prueba utilizados en el desarrollo (música, videos, material de estudio). Se incluye también el vocabulario utilizado para indexar el material docente de U-Cursos.

Cuadro A.1: Nuevas clases - RDFClip Schema

URI	**Descripción**
`clips:IMDBLink`	Links a imdb.com para referenciar películas.
`clips:YouTubeLink`	Links a youtube.com para ver videos en línea.
`clips:WikipediaLink`	Links a Wikipedia para referenciar otros recursos externos.
`clips:Movie`	Películas. Para identificarlas se usan URLs de IMDB como URIs.
`clips:KnowledgeDomain`	Areas de conocimientos (i.e. Teoría de la computación). Para identificarlas se usan URLs de Wikipedia com URIs.
`uchile:MaterialDocente`	Contenido subido a la sección MaterialDocente de U-Cursos.
`uchile:GradoVisibilidad`	Grado de visibilidad de un contenido de U-Cursos.
`uchile:Curso`	Curso de la Universidad de Chile.

Cuadro A.2: Nuevas propiedades - UChile Schema

URI (`uchile:`)	**Dominio** $\rightarrow$ **Rango**	**Descripción**
`codigoCurso`	`uchile:Curso` $\rightarrow$ `rdfs:Literal`	Código del curso.
`annoCurso`	`uchile:Curso` $\rightarrow$ `rdfs:Literal`	Año en que se dictó un curso.
`secionCurso`	`uchile:Curso` $\rightarrow$ `rdfs:Literal`	Sección correspondiente a una instancia de un curso.
`tipoCurso`	`uchile:Curso` $\rightarrow$ `rdfs:Literal`	Tipo de curso (e.g. curso regular, comunidad, etc.)
`cursoOrigen`	`uchile:MaterialDocente` $\rightarrow$ `uchile:Curso`	Curso de origen al cual pertenece un determinado Material Docente.
`visibilidad`	`uchile:GradoVisibilidad` $\rightarrow$ `uchile:Curso`	Grado de visibilidad del material.

Cuadro A.3: Nuevas propiedades - RDFClip Schema

URI (`clips:`)	**Dominio → Rango**	**Descripción**
`movieContent`	`nfo:Video` → `clips:Movie`	Película contenida en un archivo de video.
`userLabel`	`rdfs:Resource` → `rdfs:Resource`	Etiqueta para desplegar el recurso en una interfaz de usuario básica.
`hasSubtitle`	`nfo:Video` → `nfo:Document`	Archivo de subtitulos de un video. Inversa de `clips:subtitleOf`
`subtitleOf`	`nfo:Document` → `nfo:Video`	Video que subtitula un documento de texto. Inversa de `clips:hasSubtitle`
`youtubeUpload`	`nfo:Video` → `clips:YouTubeLink`	Video YouTube correspondiente al arhivo de video.
`related`	`nie:InfoElement` → `nie:InfoElement`	Archivo relacionado.
`differentVersionOf`	`nie:InfoElement` → `nie:InfoElement`	Otra versión del mismo contenido.
`foundWith`	`nie:InfoElement` → `nie:InfoElement`	Archivos 'comúnmente' encontrados juntos.
`downloadLink`	`nie:InfoElement` → `rdf:Resource`	Vínculo de descarga.
`indirectDownloadLink`	`nie:InfoElement` → `rdf:Resource`	Vínculo de descarga indirecta (e.g. RapidShare, U-Cursos). Subpropiedad de `downloadLink`.
`directDownloadLink`	`nie:InfoElement` → `rdf:Resource`	Vínculo de descarga directa. Subpropiedad de `downloadLink`.