UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Facultad de Ingeniería

Instituto de Computación

Carrera: Ingeniería en Computación

Proyecto de Grado

Proveedor RDF

Diana Fornara *Harold Selvaggi*

Tutor
Msc. Fernando Carpani

Montevideo, Uruguay
9 de diciembre de 2015

Resumen

Uno de los desafíos en el área de los Sistemas de Información es el acceso, publicación e integración de datos de diversos orígenes y estructuras también diversas. Muchos de los datos se encuentran en diferentes formatos y plataformas, pueden estar en Bases de Datos Relacionales, en archivos XML o en la propia Web.

Para resolver el problema de la integración, se establecen correspondencias (mapeos) entre las diversas estructuras de origen hacia un modelo común. Por otro lado, el acceso a orígenes diferentes introduce el problema de distribución de los datos.

La Web Semántica provee un modelo común para representar los datos que a su vez es distribuido y mecanismos accesorios orientados al mapeo que permiten convertir los datos a ese modelo. También provee herramientas que trabajando con este modelo, permiten el acceso y la publicación de datos.

El presente trabajo presenta un panorama de los mecanismos de transformación de datos relacionales al modelo RDF y un prototipo que permite el acceso a fuentes heterogéneas de datos. El prototipo se construyó con la idea de proveer un mecanismo de acceso común a todos los datos con independencia del proveedor y con la posibilidad de agregar nuevos proveedores.

Palabras clave: Web Semántica, RDF, R2RML, R2RML processor, LDP, RDF Mapping, Linked Data Platform, Bases de Datos Relacionales

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Descripción del proyecto

1.1. Introducción

Vivimos en la era de la información y estamos inmersos en un universo de datos. En este universo conviven datos con diversos formatos y estructuras que residen en diversas fuentes como la Web o sistemas corporativos.

Entre los desafíos en el área de los Sistemas de Información está el acceso, la publicación y la integración de datos de diverso origen.

En este sentido, el presente proyecto pretende colaborar implementando un dispositivo de acceso homogéneo a un conjunto heterogéneo de datos.

Este dispositivo se basa en dos mecanismos: un mecanismo para la transformación de las estructuras de datos a un modelo común y un mecanismo de acceso a los datos transformados.

Estos mecanismos de transformación y acceso a datos, se implementan en un artefacto de software denominado *wrapper* o adaptador.

En el resto del documento se emplearán indistintamente los términos adaptador o wrapper para referirse al citado artefacto.

Para realizar esta propuesta se emplearán tecnologías de la Web Semántica.

La Web Semántica provee un modelo común para representar los datos que a su vez es distribuido y mecanismos accesorios orientados al mapeo que permiten convertir los datos a ese modelo. También provee herramientas que trabajando con este modelo, permiten el acceso y la publicación de datos.

A continuación se presentan algunos conceptos referidos a estas tecnologías.

Sigue un ejemplo que se utilizará para presentar algunos de los problemas que se pueden resolver con un adaptador (*wrapper*) construido utilizando las herramientas de la Web Semántica.

Al final de este capítulo se presenta la definición y los objetivos del proyecto Proveedor RDF.

1.1.1. Web Semántica: algunos conceptos

La Web Semántica según Tim Berners-Lee es una web extendida que permitirá a humanos y máquinas trabajar en cooperación mutua.

Según W3C [1]el éxito de la Web, también ha originado sus principales problemas: sobrecarga de información y heterogeneidad de fuentes de información con el consiguiente problema de interoperabilidad. La Web Semántica ayuda a resolver estos problemas permitiendo a los usuarios delegar tareas en el software. Gracias a la semántica en la Web, el software es capaz de procesar su contenido, razonar con este, combinarlo y realizar deducciones lógicas para resolver problemas cotidianos automáticamente..

Para que los datos puedan ser procesados automáticamente es necesario describirlos y asociarles significado. Para posibilitar esto, es necesario asociar metadatos a los datos. Una forma de definir metadatos es utilizar vocabularios que permiten asociar un mismo concepto a los datos con la misma semántica.

Los objetos que se quieren describir para que las máquinas comprendan su significado se denominan recursos. Para identificar cada recurso se utiliza una URI (Uniform Resource Identifier) que es un identificador de recursos único.

El modelo común para representar datos utilizado es RDF (Resource Description Framework) que es un modelo basado en grafos. Con RDF los recursos son descriptos por ternas: $< Sujeto >< Propiedad >< Valor >$.

Un formato común de los datos y el uso de identificadores únicos para los objetos que se describen, permiten llegar a enlazar datos de distintas fuentes y poder consultarlos para obtener nuevo conocimiento.

Linked Data (Datos enlazados) es una política de publicación de datos utilizando las ideas de la Web Semántica. En esta forma de publicación, los datos deben accederse como grafos RDF.

En particular, existen mecanismos que permiten la publicación de Bases Relacionales como grafos RDF. Entre las estrategias existentes para hacer esto se destaca R2RML que es una recomendación de W3C que provee un lenguaje para especificar la correspondencia entre datos relacionales y RDF.

Estos estándares que se acaban de mencionar se describen más detalladamente en el Capítulo 2.

1.2. Ejemplo: Unificación de datos de clientes de una empresa

En esta sección se presenta un caso hipotético como ejemplo para ilustrar un problema de integración de datos. Se describe una solución basada en el modelo relacional y una solución basada en tecnologías de la Web Semántica con una implementación específica utilizando un adaptador RDF.

El objetivo de este ejemplo es fijar ideas mostrando la utilización de las herramientas de la Web Semántica y más concretamente de un adaptador que realiza la transformación de datos desde bases de datos relacionales al modelo RDF y los publica permitiendo el acceso de lectura y escritura.

[1]El World Wide Web Consortium, abreviado W3C, es un consorcio internacional que produce recomendaciones y estándares que aseguran el crecimiento de la Web (World Wide Web) a largo plazo. En http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica 5/12/2015

1.2.1. Descripción del problema

Una empresa organizada por unidades de negocio desea unificar su base de clientes. Debido a su organización, cada unidad de negocio se maneja en forma independiente y utiliza sus propias aplicaciones con distintas implementaciones de hardware y software.

Se necesita unificar la información de clientes que está separada en distintas bases de datos de la empresa. En cada unidad de negocios existe una base de clientes que tiene su propia implementación y estructura de tablas sin un modelo estandarizado.

Los clientes de la Unidad A están en tablas DB2 for z/OS, los clientes de la Unidad B están en DB2 for AS/400 y los clientes de la Unidad C están en una base Oracle.

En las tres plataformas existen datos de tipo de cliente y número de cliente. Por ejemplo, el documento de identidad para cliente tipo persona o el número de empresa para cliente tipo empresa.

Se necesita mantener estos sistemas funcionado, lo que implica que las bases de datos originales se siguen actualizando pero a su vez se quiere mantener una base unificada de clientes que facilitará la atención comercial, servicios de soporte y generación de estudios de marketing.

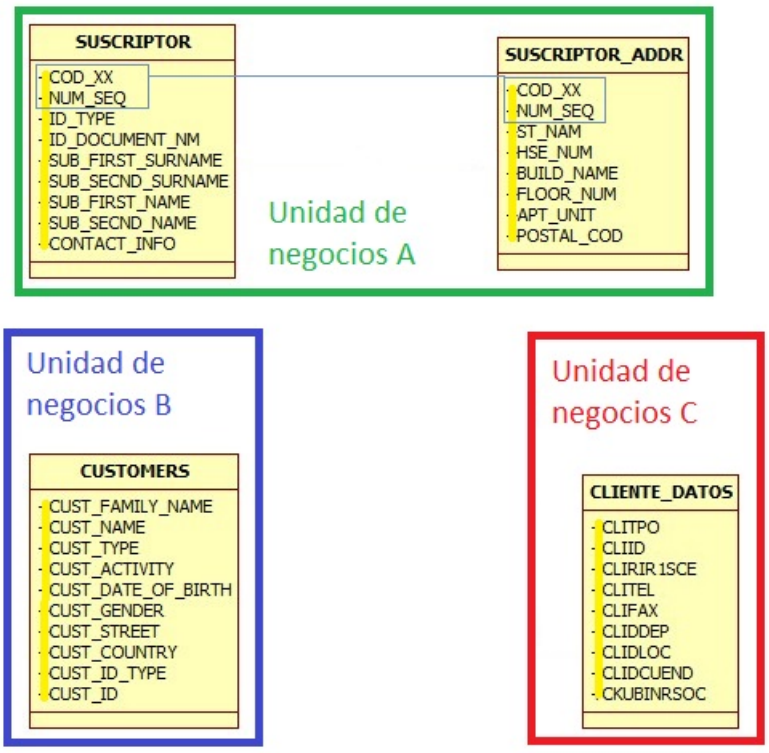

Figura 1.1: Tablas con datos de Clientes en las distintas bases de datos

1.2.2. Solución basada en el Modelo Relacional

Un enfoque basado en el modelo relacional para este problema sería definir un repositorio para contener la información de CLIENTE_UNIFICADO en una base de datos relacional. Esta base de datos se alimentará con los datos de las tres implementaciones de clientes.

Para integrar esta información, una de las dificultades será el acceso a las bases de datos que residen en diferente hardware y sistema operativo, podría emplearse un área de trabajo (staging) donde se descargarían los datos de las tres plataformas y se procesarían para luego cargarlos en el repositorio unificado.

Al definir una estructura común para toda la información de clientes, se deberán determinar los atributos que contendrá. Estos podrán ser: atributos comunes a las tres implementaciones, atributos relevantes definidos en alguna de las bases originales y nuevos atributos que se asignarán directamente en el repositorio a través una nueva aplicación de gestión de cliente único. Para generar un repositorio de datos unificados en este ejemplo se cuenta con

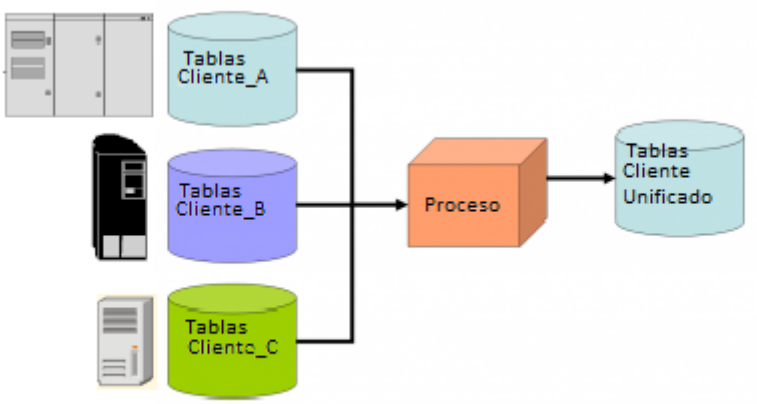

Figura 1.2: Esquema de interacción entre las bases de datos

la ventaja de una identificación común para el cliente que permite obtener los datos para un mismo cliente en cualquiera de las tres bases. Estos atributos son compatibles por tipo de dato y por semántica :

- Unidad_A Suscriptor: ID_TYPE y ID_DOCUMENT_NM
- Unidad_B Customer: CUST_ID_TYPE y CUST_ID
- Unidad_C Cliente_Datos: CLITPO y CLIID

Si bien en las tres Unidades de negocio, existen tablas con datos de clientes, hay diferencias semánticas a nivel de las tablas entre las tablas de la Unidad_A que representan servicios contra las de la Unidad_B y la Unidad_C que representan clientes.

En las tablas de la Unidad_A, la tabla que contiene los datos de cliente en realidad representa los servicios, el cliente aparece como un atributo del servicio. La identificación del servicio es COD_ XX , NUM_SEQ y tienen como atributos no nulos ID_TYPE tipo de documento y número de documento ID_DOCUMENT_NM. Pueden existir registros repetidos para un mismo tipo y nro. de documento que son los servicios a los que un cliente está suscripto.

Los datos de la dirección del servicio aparecen en otra tabla relacionada por COD_XX , NUM_SEQ, pueden existir varias direcciones para un mismo cliente ya que la dirección es del

servicio. Se puede apreciar además que los datos de dirección están muy detallados en el caso de la Unidad_A, mientras que en las otras dos unidades son muy genéricos, sin embargo también representan atributos de objetos distintos ya que en el caso de la Unidad_A son atributos del servicio y en las otras dos unidades lo son del cliente.

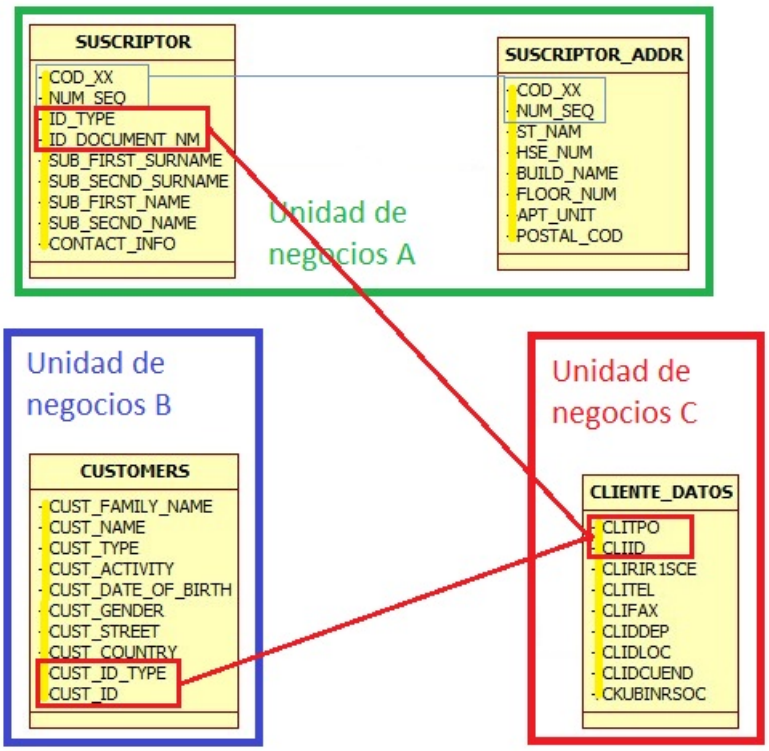

Figura 1.3: El mismo identificador de cliente en las tres implementaciones de base de datos

Examinando los campos en las tablas correspondientes a las tres unidades de negocio se pueden encontrar:

- Campos con el mismo nombre y distinta semántica
- Campos con distinto nombre y la misma semántica
- Campos con distinto nombre y distinta semántica

En este ejemplo no existen campos que cumplan con el primer caso ya que no hay nombres de campo iguales en las distintas implementaciones de cliente.

Los campos con datos de nombre del cliente en las tres implementaciones son un ejemplo del segundo caso: En la Unidad_A los campos SUB_FIRST_SURNAME, SUB_SECND_SURNAME, SUB_FIRST_NAME y SUB_SECND_NAME que contienen información sobre el nombre del cliente tienen la misma semántica que CLINOMRSOC en la Unidad_C y los campos CUST_FAMILY_NAME apellido y CUST_NAME nombre en la Unidad_B.

Como ejemplo del tercer caso se tiene que además de los atributos de cliente que se puede encontrar en las tablas de la Unidad_A, existen otros atributos de cliente en las tablas utilizadas por las otras dos Unidades de negocio tales como CLITEL teléfono del cliente y CLIFAX fax del cliente en la Unidad_C y en la Unidad_B CUST_ACTIVITY actividad, CUST_DATE_OF_BIRTH

fecha de nacimiento y CUST_GENDER Sexo.

Deberán definirse los criterios para elegir los datos que se utilizarán al Cliente Unificado en el caso de los campos de distintas bases con la misma semántica. Por ejemplo, ¿qué datos se utilizarán para el nombre del cliente? ¿qué datos para su dirección? Por ejemplo, se pueden utilizar criterios de calidad de los datos para elegir qué campo se utilizará en cada caso. De acuerdo al criterio elegido, será necesario un programa o proceso dependiente de la estructura y de las características de implementación de la base de datos que se toma como fuente.

Una vez definidos los datos que tendrá el Cliente Unificado, se deberá definir el proceso de carga inicial y cómo se realizarán las actualizaciones. Que las actualizaciones se procesen sincrónica o asincrónicamente dependerá de las necesidades del negocio y en base a ello se definirá su implementación.

1.2.3. Solución utilizando tecnologías de la Web Semántica

En lugar de una solución basada en el modelo relacional, es posible utilizar herramientas de la Web Semántica para lograr la integración de datos. Utilizando estas tecnologías la

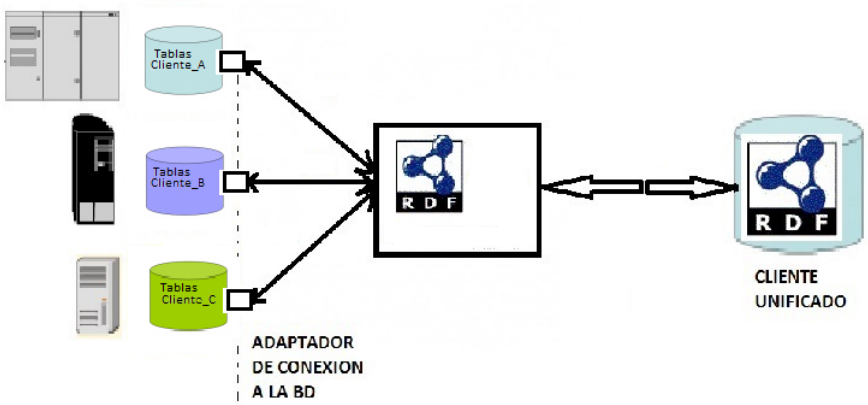

Figura 1.4: Solución con tecnologías de la Web Semántica

solución consiste en transformar los datos de las tablas relacionales a RDF.

Para cada cliente se generará un grafo. Como las identificaciones de cliente son las mismas para las tres implementaciones, es posible generar en cada grafo una única URI normalizada que será la identificación común para el cliente.

De esta forma, los datos de un mismo cliente, aunque provengan de distintas bases, quedarán unidos bajo una única URI.

Tal como se describió en la solución basada en el modelo relacional, los atributos son compatibles por tipo de dato y por semántica,

- Unidad_A Suscriptor: ID_TYPE y ID_DOCUMENT_NM
- Unidad_B Customer: CUST_ID_TYPE y CUST_ID
- Unidad_C Cliente_Datos: CLITPO y CLIID

Con estos datos se puede generar una URI normalizada que será la misma en los tres grafos generados a partir de cada una de las tres implementaciones. En el grafo de la figura 1.8 se

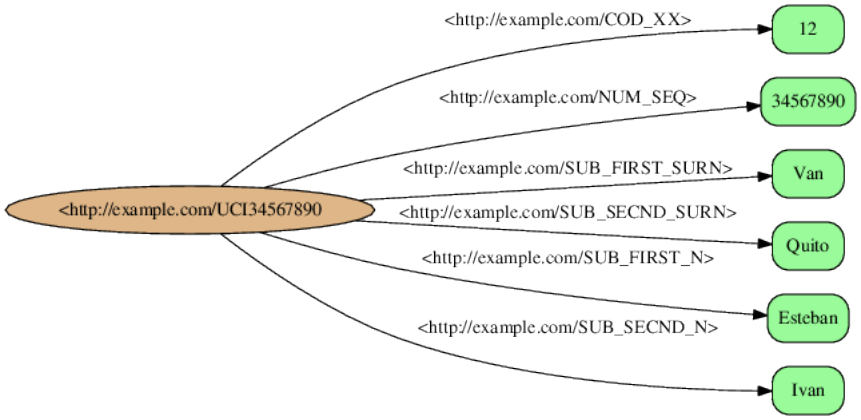

Figura 1.5: Grafo de ejemplo con datos de un cliente de la Unidad A

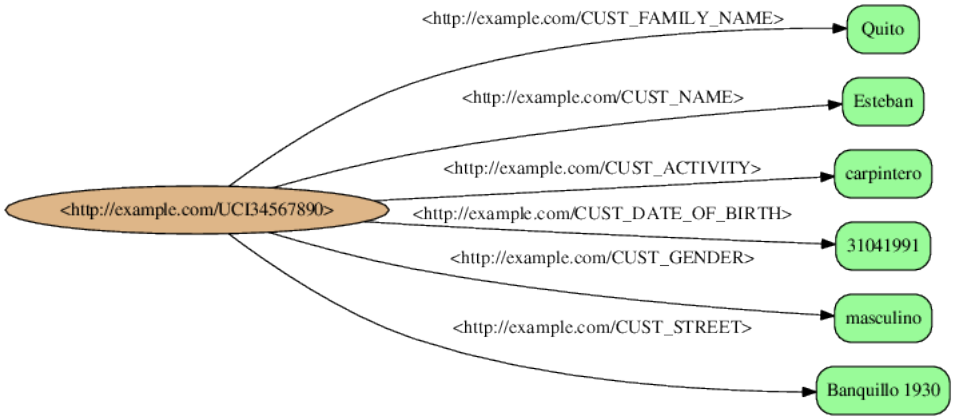

Figura 1.6: Grafo de ejemplo con datos de un cliente de la Unidad B

presenta un ejemplo de grafo unificado que contiene datos de las tres unidades correspondientes a un mismo cliente. Se puede observar que utiliza la misma URI de cliente y posee todos los atributos conocidos.

En este caso, dado que los grafos simplemente se concatenan, se duplican los datos con la misma semántica por ejemplo CUST_NAME y SUB_FIRST_N. En el caso de campos con la misma semántica puede elegirse con cuál de ellos trabajar.

Si se avanzara en el modelo y se agregaran metadatos, se podría indicar que estas propiedades tienen la misma semántica mediante el uso de un vocabulario estándar.

Una vez que los datos de cliente estén en formato RDF es posible establecer nuevas conexiones a través de Linked Data, como por ejemplo sería posible cruzar datos de la base de clientes con los datos publicados en formato RDF.

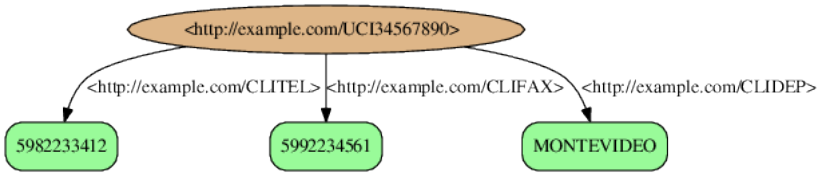

Figura 1.7: Grafo de ejemplo con datos de un cliente de la Unidad C

Implementación de la solución mediante un adaptador (*wrapper*) RDF

En esta implementación se utilizará RDF como modelo de datos, los principios de Linked Data para conectar la información y el protocolo LDP que permitirá leer y grabar datos. Para la conversión de los datos en bases de datos relacionales, se utilizará la especificación R2RML que provee un lenguaje para especificar la transformación desde tablas relacionales al formato RDF.

La propuesta consiste en acceder a cada base de datos utilizando un adaptador RDF y pasar cada estructura de tablas con datos de clientes a grafos RDF.

Se trata de una solución genérica que se puede emplear para transformar datos desde bases de datos relacionales o datos en otros formatos.

El wrapper RDF en este caso, es un artefacto de software que accede a las bases relacionales utilizando consultas SQL y transforma los datos a formato RDF para facilitar su integración.

Los datos de los clientes ya transformados en grafos RDF podrán ser accedidos utilizando tecnologías de la Web Semántica.

En particular utilizando el protocolo LDP que se explicará más adelante, será posible leer, grabar y actualizar estos datos. Es decir que se utilizará SQL para acceder a los datos de cada base de datos origen y LDP para realizar la interacción con el repositorio de Cliente Unificado que contiene los datos integrados de cliente en forma de grafos RDF.

El wrapper genera un grafo RDF a partir de la implementación de las tablas de Cliente en cada base de datos utilizando una especificación R2RML adecuada a cada implementación.

El acceso al cliente unificado se realizará mediante la plataforma LDP. Como LDP permite leer y grabar datos, cada alta, baja o modificación en las bases originales generará a través de una aplicación el comando LDP correspondiente para mantener actualizado el grafo original.

También es posible cargar los datos del grafo de cliente unificado en una base de datos relacional utilizando el wrapper en la salida para transformar de RDF a relacional y trabajar con estos datos en una base relacional, todas las actualizaciones se podrán reflejar en el grafo RDF a través de LDP.

1.2.4. Ventajas de la solución utilizando tecnologías de la Web Semántica

Entre las ventajas del uso de RDF está la facilidad con que se pueden integrar los datos una vez que se han generado los grafos.

Como la solución propuesta está basada en estándares, se hace posible la integración futura con nuevas aplicaciones. Además como se utiliza Linked Data, se podrán descubrir e integrar nuevos datos.

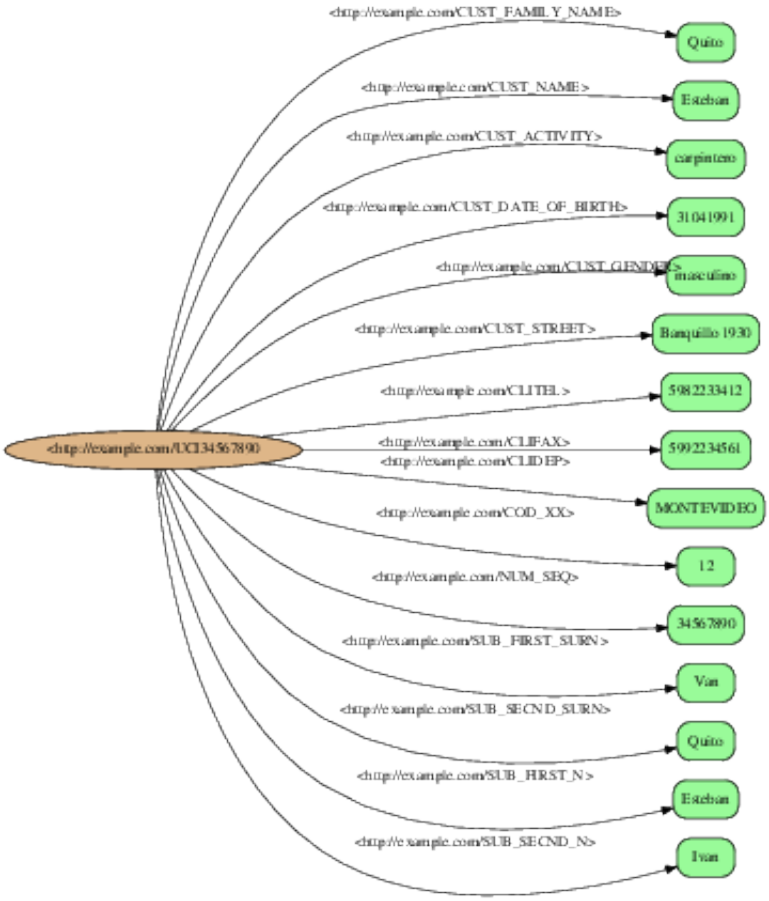

Figura 1.8: Grafo de ejemplo con datos de las tres unidades correspondientes a un mismo cliente

En cambio, en una solución tradicional, sería necesario desarrollar nuevos programas de integración por cada aplicación o fuente de datos a integrar.

La integración en el modelo relacional consiste en uno o varios procesos y un cambio en las estructuras de origen (que es algo propio del ciclo de vida de los sistemas) generará la necesidad de modificar parte de los programas de integración.

En la implementación presentada utilizando el adaptador RDF, un cambio en las estructuras originales simplemente implica una modificación en el archivo que contiene especificación R2RML.

Por ejemplo, si se agregara un campo en la tabla Unidad_C Cliente_Datos, en la solución relacional sería necesario modificar los programas de integración que trabajan con la tabla para incorporar el nuevo campo.

En el caso del wrapper solamente hay que modificar el archivo de especificación R2RML agregando el nuevo nombre del campo a la especificación si se desea agregar el campo al grafo. Todos los cambios en la estructura de la base de datos original se manejan desde este único archivo.

Otra ventaja del empleo del adaptador es que como LDP permite la lectura y grabación

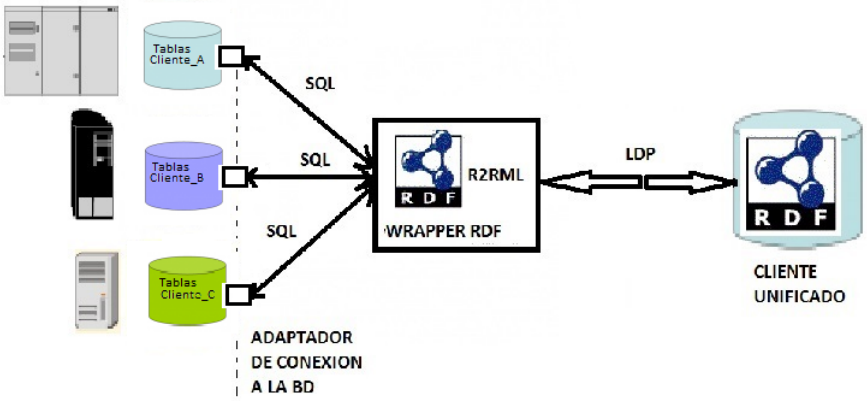

Figura 1.9: Interacción con el Wrapper RDF

de Linked Data, los cambios que se originen en el cliente unificado pueden reflejarse en las bases originales a través del adaptador, utilizando la misma especificación R2RML para el camino inverso.

En la solución relacional reflejar un cambio de este tipo requiere definir para cada campo que se actualiza, cuáles tablas de cuáles bases de origen deben actualizarse.

1.2.5. Resumen de las ventajas de la implementación de un adaptador RDF

Simple: la solución consiste en generar un grafo RDF con los datos de un cliente a partir de cada base de datos, como los grafos generados tienen la misma URI para un mismo cliente, los datos de las tres unidades se integran automáticamente mediante Linked Data.

Estándar: la solución está completamente basada en estándares lo que facilita su mantenimiento evolutivo y la integración de los datos a través de Linked Data.

Flexible: un cambio en la estructura de cualquiera de las bases de datos originales solamente requiere la modificación de la especificación R2RML correspondiente.

Adaptable: posibilidad de actualización de datos desde y hacia las bases de datos originales empleando LDP que permite lectura/escritura de Linked Data.

1.2.6. Conclusiones

- La información accesible a través de un adaptador RDF ofrece una interfase uniforme y basada en estándares, lo que simplifica el acceso.
- Los datos son accesibles a través de LDP, una recomendación de la W3C que utiliza el estándar HTTP y métodos de localización estándar como URIs.
- El resultado de las solicitudes se entrega en el formato RDF. El usuario puede acceder al grafo RDF a través de una URI.

- Debido a la forma en la que son publicados los datos, éstos son accesibles simplemente utilizando un cliente REST en cualquier navegador de Internet. Esta forma de publicación permite que el usuario que desea acceder a la información no necesite utilizar una aplicación especializada para poder acceder a la misma.

Figura 1.10: Esquema de la solución propuesta

1.3. Definición y objetivos del proyecto

El presente trabajo es el resultado de la realización del Proyecto de Grado de la carrera Ingeniería en Computación de la Facultad de Ingeniería de la Universidad de la República (UdelaR).

La propuesta es construir un wrapper RDF para poder satisfacer la necesidad de acceso, publicación e integración entre diversas fuentes de datos. Este adaptador debe funcionar como un servidor web (proveer URL's) a grafos cuyos datos están almacenados en otras fuentes.

En particular, se implementará una prueba de concepto que contempla el caso de fuentes relacionales, usando una especificación en R2RML. La intención es que el wrapper soporte alguna forma de mecanismo de extensión y que permita la inclusión de nuevas fuentes.

Los objetivos del proyecto son :

- Obtener un panorama de los mecanismos de wrapper RDF existentes
- Investigar, evaluar y proponer formas de especificar los mapeos entre diferentes fuentes y RDF. En particular, utilizar R2RML para el caso relacional
- Se implementará una prueba de concepto de un wrapper que soporte mecanismos de extensión y en particular, soporte la especificación R2RML.

Resultados esperados del proyecto:

- Un documento que presente un panorama de los mecanismos de mapeo y wrappers RDF desde fuentes relacionales.
- Un documento que presente una propuesta de diseño de un wrapper con su mecanismo de definición de mapeos.
- Un prototipo a nivel de prueba de concepto de las ideas expuestas en los documentos anteriores.

En los capítulos siguientes se presentan los resultados obtenidos:

En el Capítulo 2 se presentan conceptos, tecnologías y estándares utilizados en este proyecto.

El capítulo 3 contiene un panorama de trabajos y distintos encares para definir la transformación desde bases relacionales a RDF con énfasis en el estándar R2RML y sus perspectivas de futuro. Se realiza la descripción del relevamiento realizado sobre los mecanismos de mapeo y una clasificación de artículos y software relevado.

En el capítulo 4 se presenta el diseño de un prototipo de un adaptador que permite la transformación de datos desde varias fuentes a RDF y su publicación. En particular, permite la transformación y publicación de datos residentes en bases de datos relacionales.

Al final del capítulo 4 se presentan las conclusiones del trabajo y el trabajo a futuro.

Capítulo 2

Conceptos y Tecnologías aplicadas

A continuación se presenta una breve descripción de conceptos y tecnologías de la Web Semántica aplicadas en este proyecto .

Se describen con mayor detalle SPARQL, Linked Data Platform (LDP) y R2RML que son componentes fundamentales del adaptador RDF.

2.1. Descripción de conceptos y tecnologías aplicadas

Web Semántica: La web semántica implica una infraestructura común mediante la cual se pueda compartir, procesar y transferir información Según Tim Berners Lee: la web semántica es una red de datos que pueden ser procesados por máquinas, es una web extendida que permitirá a humanos y máquinas trabajar en cooperación mutua.

Recursos: Tanto los objetos de los que se habla como lo que se dice de esos objetos (propiedades) son recursos [1]. Pueden ser elementos digitales como por ejemplo: páginas html, imágenes, videos, archivos en diversos formatos, objetos físicos o personas del mundo real.

URI: Para identificar los recursos se utiliza una URI (Uniform Resource Identifiers) que es un identificador de recursos únicos, sin posibilidad de ambigüedad. Puede ser una localización (URL), un nombre (URN) o ambos.

RDF (Resource Description Framework): Es el lenguaje utilizado para describir recursos y permitir su tratamiento (consulta, clasificación y acceso). RDF [2] es el estándar para publicación de datos en la Web. Los recursos son descriptos por ternas. $< Sujeto >< Propiedad >< Valor >$ Un grafo RDF es [1]: un modelo donde cada recurso se representa por un grafo donde cada arco representa alguna forma de relación entre recursos o datos

- un multigrafo: que puede tener muchas aristas, lazos y ciclos sobre sus nodos
- un grafo dirigido: las aristas están orientadas

- un grafo etiquetado: cada nodo o arista tiene una etiqueta

Cualquier grafo RDF se puede ver como un conjunto de ternas

Triplestore: es una Base de Datos que utiliza el modelo de datos RDF como nativo. Un Triplestore [1] almacena grafos RDF

Linked Data: Tim Berners-Lee definió cuatro principios que caracterizan los datos enlazados (Linked Data [3]) en su ponencia de presentación para W3C y realizó la primera presentación pública del concepto en TED 2009. Los principios de Linked Data promueven la publicación de datos en un formato legible por computadores usando estándares de la Web e interconectándolos. Los cuatro principios de Linked Data son:

- Utilizar URIs para identificar los recursos publicados en la Web
- Utilizar URIs HTTP para permitir acceder a esos links
- Ofrecer información sobre los recursos usando estándares como RDF o SPARQL
- Incluír enlaces a otras URIs para posibilitar el descubrimiento de otros recursos

SPARQL (Simple Protocol and RDF Query Language) es un lenguaje de consulta sobre grafos RDF, que permite hacer búsquedas sobre los recursos de la Web Semántica utilizando distintas fuentes de datos en formato RDF. SPARQL [4] es un lenguaje basado en la coincidencia de patrones (pattern matching) sobre los grafos. La idea general consiste obtener las ternas de los grafos RDF consultados que correspondan a los patrones que están expresados en la consulta.

SPARQL endpoint Es un servicio que acepta consultas SPARQL y retorna sus resultados.

R2RML (RDB to RDF Mapping Language) provee un lenguaje que permite especificar la transformación desde tablas relacionales al formato RDF. Esta recomendación [5] especifica cómo realizar un mapeo desde tablas o vistas relacionales a RDF. Permite la transformación al formato RDF de datos residentes en una base de datos relacional.

R2RML processor es un sistema que dada una base de datos relacional, una conexión a dicha base y un documento de mapeo R2RML, genera una salida con datos de la base en formato RDF a un archivo o dump.

LDP Linked Data Platform (LDP) es una especificación reciente de W3C que define un conjunto de patrones para la construcción de servicios RESTful HTTP que son capaces de leer y grabar Linked Data. [6]

2.2. Componentes del wrapper RDF

Integrando todos los conceptos descriptos en la sección anterior en el wrapper, se presenta una herramienta que permite: dado un acceso a una base de datos, una especificación de mapeo R2RML y una url, generar un grafo RDF con la url proporcionada, a partir de datos residentes en una base de datos relacional, utilizando para la selección de datos, el mapeo especificado.

El wrapper RDF posee un mecanismo para la transformación de datos a RDF y un mecanismo de acceso a los datos transformados.

- Para poder realizar la transformación de datos en bases de datos relacionales a partir de una especificación en R2RML debe implementar un *R2RML processor*.
- Para el acceso a los datos transformados, se utilizó Linked Data Platform (LDP). LDP define operaciones HTTP sobre recursos web y permite leer y grabar Linked Data.

A continuación se presentan con más de detalle los estándares fundamentales en la construcción del Wrapper RDF.

2.3. SPARQL

SPARQL (Simple Protocol and RDF Query Language) [4] [1] permite hacer búsquedas sobre los recursos de la Web Semántica utilizando distintas fuentes de datos en formato RDF.

Es un lenguaje basado en concidencias de patrones (pattern matching) sobre grafos distribuidos.

La idea general consiste obtener las ternas de los grafos RDF consultados que correspondan a los patrones que están expresados en la consulta.

Por ejemplo, el patrón más simple: $?s?p?o.$ coincide con cualquier terna

El siguiente ejemplo [1] muestra una consulta simple.

Esta consulta retorna nombres y direcciones de correo de todas las personas en el dataset

```
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 }
```

La consulta retorna todas las ternas de sujetos donde el tipo de predicado *a* es una persona (foaf:Person) y la persona tiene uno o más nombres (foaf:name) y una o más direcciones de correo (foaf:mbox).

Una consulta sparql permite realizar definiciones adicionales que permiten realizar un join con otros tipos de sujetos.

Si se consideran por ejemplo otro dataset conteniendo datos RDF de automóviles con sus características y la URI de sus propietarios, sería posible realizar una consulta que retorne la lista de nombres y direcciones de correo que personas que tienen autos con ciertas características.

SPARQL posibilita [7] :

- Construir un nuevo grafo RDF usando datos del grafo RDF consultado
- Retornar las descripciones de los recursos que coinciden con el patrón de la consulta.
- Especificar patrones opcionales para filtrar casos.
- Probar la ausencia o no existencia de tuplas.
- Extraer subgrafos

[1] En https://en.wikipedia.org/wiki/SPARQL 5/12/2015

2.4. LDP Linked Data Platform 1.0

LDP especifica la forma en que las aplicaciones pueden publicar nuevos recursos, editar y borrar recursos existentes en servidores que exponen recursos como Linked Data usando el protocolo HTTP.

Linked Data Platform 1.0 [6] define una arquitectura para leer-grabar Linked Data basada en accesos HTTP a recursos web descriptos en RDF.

Es decir que propone una forma de trabajar con recursos RDF puros casi como si fueran páginas web.

Se utiliza del estándar HTTP y RDF para construir clientes y servidores que puedan crear, leer y grabar Linked Data Platform Resources (LDPRs).

Posee dos aspectos básicos [8]:

- Un vocabulario específico para describir cómo están compuestos los recursos que puede verse como un modelo de datos.
- El comportamiento de cada recurso frente a los métodos (o comandos) HTTP.

Un Recurso, como ya se vio al principio del capítulo, es algo que tiene una URI con la que se puede referenciar. En LDP hay dos tipos de recursos:

- RDF: Tiene una descripción RDF que es mantenida por la plataforma.
- NO RDF: No tiene una descripción RDF. En general se trata de un recurso externo que suele ser multimedia o texto.

Los recursos RDF pueden ser de dos tipos: Contenedores y miembros de los contenedores

Un Contenedor (LDP Container) es un recurso que contiene a otros. Por ejemplo, una colección de fotos de una persona. En general contiene una colección de recursos relacionados por algún criterio.

En LDP hay tres tipos de contenedores:

- Basic Containers
- Direct Containers
- Indirect Containers.

También se especifican algunas estrategias para el tratamiento de recursos de gran volumen. Una extensión de la especificación LDP brinda la posibilidad de fraccionar la recuperación de recursos de gran volumen en varias páginas cada una con una dirección URL (LDP-PAGING).

El protocolo LDP define las interacciones de lectura y escritura de recursos.

Los recursos pueden ser creados, modificados, borrados y leídos usando los métodos estándar de HTTP (como POST, PUT/PATCH, DELETE, GET).

La lectura se realiza mediante GET, la escritura incluye la creación de recursos utilizando POST o PUT, la actualización utilizando PUT o PATCH y el borrado con DELETE.

2.4.1. Componentes de LDP

Un *Contenedor LDP* contiene links a recursos. Básicamente representa una colección de recursos.

Un contenedor responde a los requerimientos de un cliente para creación, modificación, y/o enumeración de sus miembros.

La URI que identifica a un contenedor es un punto de interacción HTTP que permite manejar a sus miembros y a la relación de membresía.

El tipo de contenedor más simple definido en el protocolo LDP es el *Basic Container*. Un Basic Container define la relación con sus miembros únicamente mediante la relación *ldp:contains*. En la figura 2.1 se muestra que un contenedor puede contener además otros

Figura 2.1: Basic Container

contenedores.

En los siguientes ejemplos extraídos de [9] [10]se ve el resultado de operaciones sobre un Basic Container empleando sentencias GET/POST para leer y grabar Linked Data.

Ejemplo: GET para listar los recursos existentes en un contenedor. En este ejemplo se muestra como leer datos. (Se quitaron los headers HTTP para contribuir a la claridad del ejemplo)

Pedido:

```
1  GET /container1 HTTP/1.1
2  Host: example.org
3  Accept: text/turtle
```

Respuesta:

```
1  @prefix dcterms: <http://purl.org/dc/terms/>.
2  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
3  @prefix ldp: <http://www.w3.org/ns/ldp#>.
4  <http://example.org/container1>
5  a ldp:BasicContainer;
6  dcterms:title "A very simple container";
7  ldp:contains
8  <http://example.org/container1/member1>,
9  <http://example.org/container1/member2>,
10 <http://example.org/container1/member3>.
```

En este ejemplo container1 está en: http://example.org y contiene los recursos: member1 member2 member3

Ejemplo: POST para crear un nuevo recurso. En este ejemplo se muestra cómo es posible grabar Linked Data.

Pedido:

```
POST /container1 HTTP/1.1
Host: example.org
Content-type: text/turtle
Content-length: 324
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix o: <http://example.org/ontology/>.
<>
a o:Stock; dcterms:title ACME Co.; o:value 100.00.
```

Respuesta:

```
HTTP/1.1 201 CREATED
Content-Location: http://example.org/container1/member4
```

En este ejemplo container1 está en: http://example.org contiene : member1 member2 member3 y member4.

Una descripción más detallada y ejemplos de utilización se encuentran en el Anexo C.

2.5. R2RML

R2RML (RDB to RDF Mapping Language) [5] es una recomendación de W3C que define un lenguaje para especificar la correspondencia entre datos relacionales y RDF.

- R2RML permite especificar una transformación personalizada donde es posible especificar el vocabulario utilizado en el grafo RDF.
- Es posible generar vistas sobre los datos e incluso no transformar datos que no sean de interés.
- Se especifica mediante un grafo RDF, generalmente en formato turtle, cómo se quiere mapear las diferentes tablas, campos y relaciones a RDF.
- Agrega flexibilidad ya que permite que la correspondencia entre el contenido de la base de datos y el grafo RDF sea definida por el usuario

Un mapeo R2RML define *tablas lógicas* (logical tables) para recuperar datos desde la base de datos de entrada. Una *tabla lógica* puede ser uno de los siguientes objetos:

- Una tabla de la base de datos de entrada,
- Una vista definida en la base de datos de entrada
- Una consulta SQL (denominada “R2RML view” porque emula una vista SQL sin alterar la base de datos de entrada).

Cada tabla lógica se mapea a RDF utilizando una regla *triples map*.

Triples map es una regla que mapea cada fila de la tabla lógica a ternas RDF. La regla tiene dos partes:

- Un subject map que genera el sujeto de todas las ternas RDF que van a generarse a partir de la fila de la tabla lógica. Generalmente los sujetos son IRIs que se generan a partir de la primary key de la tabla.
- Múltiples predicate-object maps que consisten respectivamente en predicados y objetos (o referencing object maps).

Las ternas se producen combinando el subject map con un predicate map y object map, y aplicándolos a cada fila de la tabla lógica. Por ejemplo:

- Sujeto: Se usa un template http://data.example.com/empleado/{empno} para generar las IRIs del sujeto a partir de la columna empno.
- Predicado: Se usa una IRI constante del vocabulario ex:name .
- Objeto: Se usa el valor de la columna ename para producir un literal RDF .

A continuación se muestra un ejemplo extraído de [11] Sea una tabla en una base relacional con datos de empleados:

Tabla EMP

EMPNO (PK)	ENAME	EJOB
1111	Harold	consultor

Se utilizará la siguiente especificación R2RML:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://example.com/ns#>.
<#TriplesMap1>
rr:logicalTable [ rr:tableName "EMP" ];
rr:subjectMap [
    rr:template "http://data.example.com/empleado/{EMPNO}";
    rr:class ex:Empleado; ];
rr:predicateObjectMap [
    rr:predicate ex:name;
rr:objectMap [
    rr:column "ENAME" ]; ].
```

Aplicando la especificación R2RML a los datos de la tabla EMP se obtiene el siguiente grafo RDF:

```
<http://data.example.com/empleado/1111> rdf:type ex:Empleado.
<http://data.example.com/empleado/1111> ex:name "Harold".
```

Por defecto, todas las ternas RDF generadas a partir del mapeo están en un grafo en el archivo de salida (*default graph*). Adicionalmente, en *triples map* se pueden agregar definiciones *graph maps* que permiten colocar todas o algunas de las ternas generadas en *grafos nominados (named graphs)*.

Capítulo 3

Panorama de los mecanismos de transformación de datos relacionales a RDF

Datos de todas las áreas del conocimiento residen en bases de datos relacionales, por ello importa proveer un mecanismo que permita compartir estos datos y hacerlos accesibles a través de Linked Data.

En la siguiente sección se describen las estrategias utilizadas para transformar los datos relacionales a RDF.

Seguidamente se presenta el método utilizado para realizar el relevamiento de artículos y software.

A continuación se brinda una visión general de los artículos relevados y una clasificación de los artículos y el software relevados.

El resultado permite establecer un panorama de la situación de los mecanismos de transformación de bases de datos relacionales a RDF al mes de setiembre de 2015, fecha de finalización del relevamiento.

Un breve resumen de cada uno de los artículos relevados se presenta en el Anexo B.

3.1. Estrategias de transformación de datos relacionales a RDF

3.1.1. Mecanismos de acceso a los datos relacionales

De acuerdo al artículo: Transient and persistent RDF views over relational databases in the context of digital repositories[12] , los mecanismos utilizados para proveer acceso a las bases relacionales tienen dos estrategias básicas:

1. Acceder a los datos en las bases relacionales a través de un lenguaje de consulta
2. Transformar los datos para que puedan ser accedidos por herramientas de la Web Semántica.

Para el primer caso, se transforma una consulta SPARQL en una o varias consultas SQL que se ejecutan contra los datos de la base relacional.

En el segundo caso, a partir de los datos relacionales se pueden materializar grafos RDF o definir grafos virtuales que pueden ser consultados utilizando SPARQL o recorridos con una API para grafos RDF.

3.1.2. Mecanismos de mapeo de datos relacionales a RDF

Existen diferentes mecanismos para poder realizar un mapeo de datos de una base relacional a RDF.

Entre estos mapeos hay algoritmos y especificaciones que son propietarias y también existen algunos estándares propuestas por la W3C. Simplificando la clasificación presentada en el artículo : A survey of RDB to RDF translation approaches and tools [13] se definen tres enfoques:

1. Direct Mapping [14] es una recomendación de W3C que define una transformación simple y directa. DirectMapping se caracteriza por su simplicidad, la estructura del grafo resultante y el vocabulario reflejan los nombres de las tablas, base de datos y columnas que se están mapeando. Esta transformación no ofrece ninguna forma de modificar el vocabulario o el grafo generado. Es el mapeo por defecto considerado por W3C. Se considera un punto de partida para la definición de otras transformaciones más sofisticadas.

2. R2RML[5] es una recomendación de W3C, define un lenguaje para especificar la correspondencia entre datos relacionales y RDF. R2RML permite especificar una transformación mucho más personalizada donde es posible especificar el vocabulario utilizado en el grafo RDF, también es posible generar vistas sobre los datos e incluso no transformar datos que no sean de interés. Se especifica mediante un grafo RDF, generalmente en formato turtle, cómo se quieren mapear las diferentes tablas, campos y relaciones a RDF. Agrega flexibilidad ya que permite que la correspondencia sea definida por el usuario.

3. El tercer enfoque es propietario, basado en las propiedades de un dominio en particular. Esta correspondencia es particular para cada aplicación, no cumple estándares ni recomendaciones. A los efectos de este trabajo este enfoque se denominará Domain Mapping.

El hecho de tener especificaciones estándar como DM y R2RML propuestas por W3C, permite intercambiar los mapeos utilizados entre diferentes herramientas que los implementen permitiendo reutilizar las transformaciones de la base relacional a RDF.

Utilizar estos estándares también facilita la integración de datos. En trabajos anteriores a la especificación de estándares se utilizaban implementaciones propietarias que en general no son compatibles entre sí y dificultan la integración.

En lo que sigue de este capítulo, se presenta:

En la sección Relevamiento, la metodología utilizada para el relevamiento de artículos y software.

En la sección Resultados, un resumen del contenido de los artículos considerados y un listado del software relevado.

En la sección Clasificación, se presenta una clasificación de los artículos y del software que permite ordenar los trabajos relevados.

Para finalizar este capítulo en la sección Conclusiones y Consideraciones se presentan las consideraciones derivadas del relevamiento realizado y las conclusiones que se tendrán en para la construcción del prototipo del wrapper.

3.2. Relevamiento

Como primer paso se llevó a cabo un relevamiento en dos etapas:

- Relevamiento de artículos sobre mecanismos de mapeo desde bases de datos relacionales a RDF
- Relevamiento de software que implemente mecanismos de mapeo.

3.2.1. Metodología de relevamiento de artículos

El proceso de relevamiento y selección de artículos se basa en un método estructurado para el relevamiento de literatura científica utilizado en [15].

En este caso el método fue aplicado de la siguiente manera:

1. Se buscan artículos en las colecciones de las más relevantes conferencias y publicaciones provistas por el Portal Timbó.
2. En una primera instancia se buscó con la palabra clave: R2RML
3. En siguientes instancias se decide ampliar la búsqueda utilizando otras palabras claves: RDB2RDF, RDB to RDF, relational to RDF mapping.
4. Los artículos encontrados se seleccionan manualmente de acuerdo a la relevancia con respecto al objeto estudiado, en este caso los mecanismos de mapeo y wrappers.
5. Se consideraron los artículos más recientes privilegiando aquellos publicados a partir de 2012, también se consideraron aquellos trabajos que son más citados por otros.
6. Se relevaron más de 30 artículos de los cuales se reseñaron 25 y se seleccionaron 18 para clasificar.

3.2.2. Metodología de relevamiento de software

El proceso de relevamiento y selección de software se realizó siguiendo la siguiente estrategia:

1. Se revisó el software provisto en los artículos relevados.
2. Se buscó software a nivel de internet realizando la búsqueda a nivel de palabras clave: R2RML, RDB2RDF, RDB to RDF,relational to RDF mapping
3. Se consideraron aquellas herramientas más recientes privilegiando aquellos publicados a partir de 2012.

4. Se privilegió el software libre sobre el comercial.
5. Se seleccionaron aquellas herramientas que implementan R2RML para realizar el mapeo de relacional a RDF por considerar que esta especificación más recientele y permite mayor flexibilidad al definir los datos que se transformarán a RDF
6. Se relevaron 14 herramientas, de las cuales se instalaron y probaron 5.

3.3. Resultados del relevamiento de mecanismos de transformación a RDF para bases de datos relacionales

3.3.1. Artículos relevados

Como complemento del estudio de la especificación R2RML[5], El artículo R2RML Processor for Materializing RDF View of Relational Data: Algorithms and Experiments [16] describe los algoritmos utilizados para definir un R2RML processor. La lectura de este trabajo contribuyó a la comprensión de las funciones de un R2RML processor.

Se trata de un trabajo de noviembre de 2013 por lo que fue posible contar con datos actualizados que pueden ser tomados como referencia para el relevamiento de artículos y software.

En la introducción realiza recuento de herramientas que implementan R2RML, esta lista fue utilizada como base para comenzar el trabajo de evaluación de herramientas que implementan R2RML. Señala que solamente dos de las herramientas referenciadas (XSPARQL y Ultrawrap) cumplen con la totalidad de los test cases de R2RML definidos por W3C[17].

Utilización de Direct Mapping

En cuanto a la utilización de Direct Mapping [14] se relevó el artículo:

- RDOTE - Publishing Relational Databases into the Semantic Web [18] RDOTE es un framework para convertir una base de datos relacional a RDF. RDOTE permite mapear múltiples bases de datos relacionales a diferentes ontologías e integrarlas en un único OWL. Ofrece un mapeo automático dada una base de datos que se consigue implementando Direct Mapping según la especificación de la W3C.

Soluciones a medida

Entre los artículos que presentan mecanismos de tranformación no estándar dirigidos a solucionar la transformación para un dominio específico están:

- An Extensible Approach for Mapping Relational DB to RDF [19] que describe una extensión de DB2RDF que convierte DB a RDF incluyendo información adicional que resulta de gran utilidad para motores de búsqueda.
- Semantic querying of relational data for clinical intelligence: a semantic web services-based approach [20] propone la conversión de datos de las bases de datos relacionales a RDF se realiza mediante la ejecución de consultas sql y la conversión de esos resultados a RDF. Para esto utiliza servicios SADI con una interfase SPARQL aunque reconoce que falta más experimentación y madurez.

- Algorithms for Mapping RDB Schema to RDF for Facilitating Access to Deep Web [21] define una metodología formal basada en heurísticas para mapear una base de datos relacional a una base de conocimiento semántico, comprendiendo no sólo los datos sino también el conocimiento específico del dominio. Los algoritmos presentados utilizan metadatos del diccionario de datos de la base relacional para construir la versión inicial del repositorio semántico al que se agrega conocimiento específico de dominio en etapa de procesamiento.

Soluciones genéricas que no utilizan los estándares propuestos por W3C

Entre estas soluciones se encuentran:

- Transformation of Relational Model to RDF Model [22] que presenta un algoritmo genérico para transformar una base de datos relacional a RDF preservando la semántica. De acuerdo al artículo, la ventaja del algoritmo propuesto radica en que realiza el mapeo de forma 100 % automática sin intervención del usuario.

- A Semantic Query Method for Deep Web [23] propone la idea de un acceso a datos relacionales centrado en la funcionalidad en vez de centrado en los datos como considera que son los métodos utilizados hasta ahora.Consideran que el enfoque centrado en datos tiene falencias ya que en entornos de negocios las organizaciones son más proclives a publicar funcionalidades que datos. El enfoque centrado en funcionalidades se basa en una combinación de Semantic Web Services y tecnología de acceso a datos RDF.

- Creating Semantic Data from Relational Database [24] presenta OntoURI que convierte datos desde RDBMS a instancias OWL.

- Accessing relational data on the web with sparqlmap [25] presenta SparqlMap un software que reescribe SPARQL a SQL. El objetivo es permitir la ejecucuón de consultas SPARQL en una base de datos relacional.

- RDB2RDF: completed transformation from relational database into RDF ontology [16] presenta RDB2RDF, un método de transformación de todas las tablas de una base de datos relacional a una ontología RDF. La transformación es reversible y permite volver desde RDF a tablas relacionales. Todos los pasos se realizan automáticamente sin necesidad de intervención de los usuarios. Se utiliza el formato RDF/XML para almacenar los resultados con el argumento de que este formato posibilita utilizar las herramientas que manejan XML.

- StdTrip+ K: Design Rationale in the RDB - to - RDF process [26] StdTrip+K es un proceso que integra DR (design rationale) junto a la estrategia de triplificación.El proceso StdTrip+K recibe como entrada, la base de datos, los metamodelos (p.ej. ER y RDF son metamodelos) y el vocabulario Kuaba+W. En cada fase del proceso se registran las decisiones de diseño con el vocabulario Kuaba+W. Como resultado final se obtiene el archivo de mapeo RDB-to-RDF, una ontología OWL y el DR resultante de todo el proceso.

- Mapping Relational Databases into OWL Ontology [27] presenta un prototipo para crear una ontología desde una base de datos relacional. En el proceso se extrae metadata del esquema relacional, se transforma en un modelo canónico para facilitar la migración, se

genera la estructura de la ontología en OWL, se valida y se refina la ontología. El grafo RDF resultante se guarda en un documento OWL.

- An integrated framework for enhancing the semantic transformation, editing and querying of relational databases [28] presenta Iconomy, una suite que integra todas las tecnologías de la Web Semántica. Describe Iconomy como un mecanismo para la transformación de datos relacionales en entidades semánticas y la provisión de una interfase de usuario amigable para ver, manejar, editar y realizar querys sobre la ontología y sus instancias. La impresión luego de leer este artículo es que la idea planteada de integrar herramientas es buena y tal como dice el artículo evita instalar herramientas particulares para cada paso de la incorporación de datos existentes en otros formatos a la Web Semántica y Linked Data. No hay noticias de versiones actuales de Iconomy por lo que se desconoce si se continuó el desarrollo.

Propuestas que utilizan R2RML

Dentro del grupo de artículos donde se describe la utilización del estándar R2RML se encuentran:

- RDB2RDF: A relational to RDF plug-in for Eclipse [29] plugin para Eclipse que soporta la conversión de datos relacionales a RDF. Este plugin tiene una arquitectura modular construída teniendo en cuenta las dos etapas del proceso de conversión: extracción de esquema y conversión de datos, de forma de encapsular por un lado el proceso estable y bien conocido como la extracción de esquema de otro proceso más cambiante y en continua evolución como la conversión de datos. La herramienta permite generar mapeos en R2RML.

- Ultrawrap: SPARQL execution on relational data [30] Ultrawrap genera una vista lógica de una base de datos relacional como un grafo RDF definido utilizando vistas SQL. Las consultas se ejecutan sobre esas vistas lógicas en la base de datos original transformando SPARQL en SQL. De esta forma puede utilizar todos los algoritmos y optimizaciones de las bases de datos comerciales para ejecutar consultas SPARQL. Este enfoque permite trabajar con bases de datos instaladas sin tener que duplicar los datos en RDF, esto permite que los datos estén siempre actualizados y resguardados en la base de datos original, evitando la duplicación y la caducidad de los datos.

- Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation using Morph [31] Tomando como base el artículo anterior [30] en este trabajo se presenta la posibilidad de ejecutar querys SPARQL contra datasets virtuales. Las consultas SPARQL se traducen a SQL considerando mapeos en R2RML para que puedan ser evaluadas contra un motor de base de datos relacional sin necesidad de tener datos en formato RDF.

- morph-LDP: An R2RML-based Linked Data Platform implementation [32] En este trabajo, en base a morph [31] se presenta morph-LDP un sistema que combinando la recomendación Linked Data Platform (LDP) y R2RML, permite exponer datos relacionales como Linked Data y posibilita su lectura y escritura por parte de aplicaciones que utilicen el protocolo LDP.

Investigaciones para mejorar la eficiencia en las consultas que utilizan R2RML

Una de las estrategias que permite a las herramientas de la Web Semántica acceder a bases de datos relacionales es a través de la transformación de consultas SPARQL en querys SQL utilizando R2RML. Sobre estas técnicas se desarrolló una línea de investigación sobre la eficiencia de la traducción de SPARQL a SQL. Entre los trabajos relevados se encuentran :

- Efficient SPARQL-to-SQL with R2RML mappings [11];
- Algorithms for Mapping RDB Schema to RDF for Facilitating Access to Deep Web [21];
- Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph [31] ;
- Accessing relational data on the web with sparqlmap [25]

En todos estos trabajos se evidencia la preocupación para mejorar el desempeño del sql generado a partir de sparql de forma de poder competir en eficiencia con las bases de datos tradicionales.

Aplicaciones que utilizan R2RML

Entre mediados de 2013 y la actualidad comenzaron a aparecer trabajos sobre aplicaciones prácticas de los mapeos R2RML para el acceso y la incorporación de datos residentes en bases relacionales en la generación de ontologías y en la publicación de datos en Linked Data. Entre estos trabajos se encuentran.

- Exposing bibliographic information as linked open data using standards-based mappings: methodology and results [33] describe la generación de Linked Open Data a partir de un repositorio digital conteniendo información bibliográfica .Utiliza mapeos R2RML en el proceso de transformación. Observa dificultades técnicas tales como la necesidad de codificar manualmente utilizando un editor de texto común los mapeos R2RML. Este proceso necesitó un especialista ya que no encontró una forma de asegurar la correctitud de los mapeos sino hasta que se produce el grafo resultante de la transformación.
- Constitute: the world’s constitutions to read, search and compare [34] provee acceso para búsquedas a las constituciones de los países del mundo. En la implementación los textos se convierten a RDF utilizando mapeos R2RML y Ultrawrap (Capesenta).
- LODHub A platform for sharing and integrated processing of linked open data [35] plantea la necesidad de una plataforma que combine las soluciones existentes para publicar y compartir Linked Open Data con la infraestructura de servicios para explorar, procesar y analizar datos provenientes de múltiples fuentes. Entre las tecnologías utilizadas se incluye a R2RML para las transformaciones de RDB a RDF.
- Engineering ontology-based access to real-world data sources [36] describe la utilización de un método basado en Ingeniería de Software para la construcción de un procedimiento para la transformación de datos del mundo real a ontologías de calidad. En este caso la adopción de R2RML está planeada para el futuro.

Propuestas de automatización de la generación de mapeos R2RML

Una vez asentado el estándar R2RML, surgen propuestas de automatización entre las que se encuentran:

- R2BA - Rationalizing R2RML Mapping by Assertion. [37] Propone un método semi-automático para definir mapeos R2RML mappings que incluye DR (*design rationale*)
- MIRROR: Automatic R2RML Mapping Generation from Relational Databases. [38] Describe el sistema MIRROR que genera automáticamente dos conjuntos de especificaciones R2RML, un primer conjunto en DM (Direct Mapping) y un segundo conjunto de especificaciones que utilizan conocimiento implícito en el esquema de la base relacional que se está utilizando para mejorar la calidad del mapeo.

Iniciativas de extensión y generalización de R2RML

En años posteriores a la publicación de la especificación R2RML surgen iniciativas de extensión del estándar basadas en la necesidad de una especificación genérica que cubra otros tipos de datos además de las bases de datos relacionales. Entre ellas se destacan:

- RDF Mapping Language (RML) [39] En 2014 se publica una iniciativa de extensión de R2RML denominada RML que generaliza la especificación a fuentes de datos heterogéneas tales como archivos en xml o csv.
- KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources [40] Este trabajo de 2015 presenta una interpretación alternativa de R2RML y lo complementa con un procesador R2RML extendido para cualquier tipo de fuente (*source-agnostic R2RML processor*) y que además soporta procesos de limpieza y transformación de datos.

3.3.2. Listado de software relevado

- Morph-LDP, `https://github.com/fpriyatna/morph-LDP`
- Morph-RDB, `https://github.com/fpriyatna/morph`
- OpenLink Virtuoso, `http://virtuoso.openlinksw.com`
- RDB2RDF: a relational to RDF plug-in for Eclipse `http://lod2.inf.puc-rio.br/site/download/index.php?page=4`
- RDF-RDB2RDF `https://metacpan.org/release/RDF-RDB2RDF`
- XSPARQL,`http://xsparql.deri.org`
- Ultrawrap, `http://www.capsenta.com/`
- db2triples, `https://github.com/antidot/db2triples`
- RDOTE -publishing relational databases into the semantic web `http://sourceforge.net/projects/rdote/`
- Apache Marmotta, `https://marmotta.apache.org`

- Revelytix Spyder, http://www.revelytix.com/content/spyder
- D2RQ, http://d2rq.org/
- Triplify http://triplify.org/Documentation
- Ontop 1.12, http://ontop.inf.unibz.it/

3.4. Clasificaciones

3.4.1. Clasificación de mapeos

Tomando como base las clasificaciones empleadas en el artículo A survey of RDB to RDF translation approaches and tools [13], se han clasificado los mapeos en diferentes categorías de acuerdo a la forma en que se especifica el mapeo y el resultado obtenido al aplicarlo.

Clasificación de mapeos según su especificación

- Mapeos semi automáticos: Este tipo de mapeos generan un mapeo inicial entre la base de datos relacional y RDF. En general vienen dados por una generación automática utilizando el estándar Direct Mapping para generar el mapeo inicial. Luego se permite al usuario personalizar el mapeo generado para obtener lo que quiere. Lo bueno de este enfoque es que tiene la flexibilidad de permitir obtener lo que se requiere sin tener que especificar todo cuando la correspondencia generada por defecto sirve.

- Mapeos automáticos: Este tipo de implementaciones realiza un mapeo de la base relacional a RDF de forma totalmente automática. Los algoritmos intentan mantener la semántica de la base de datos antes de realizar el mapeo. Al hablar de la semántica se hace referencia a las relaciones entre entidades que pueden ser asociaciones, herencia, etc.

- Mapeos totalmente manuales: Un mapeo se clasifica como totalmente manual cuando es el usuario quien genera la correspondencia desde cero sin ningún tipo de asistencia.

Clasificación de mapeos según su resultado

- Materializan el RDF: Los algoritmos de mapeo a RDF pueden materializar o no el grafo RDF final. Cuando el RDF se materializa, es posible dar una base de datos como entrada y obtener un grafo RDF como salida. El grafo RDF resultante puede estar vez materializado en un archivo físico (dump) o ser un grafo virtual.

- No materializan el RDF: En estos casos en general se permite ejecutar consultas en SPARQL como si se tuviera el grafo RDF y estas son traducidas a SQL normal consultando una base de datos relacional. No se genera el grafo RDF final sino que solamente se transforman las consultas.

3.4.2. Clasificación de los artículos sobre mapeo a RDF

Criterios de clasificación

1. Fecha de publicación del artículo.
2. Utiliza o no utiliza el estándar R2RML para realizar el mapeo de RDB a RDF.
3. Forma de especificar el mapeo entre entidades : el mapeo se realiza por DM Direct Mapping, R2RML o se utiliza un algoritmo propietario dependiente del caso Domain Mapping.
4. Características del resultado o endpoint provisto SQL, SPARQL, dump.
5. Si permite obtener la URI del grafo RDF como resultado

La clasificación de artículos relevados se puede ver en el cuadro 3.1.

Cuadro 3.1: Clasificación de artículos relevados

Articulos	Fecha	R2RML	Mapeo	Resultado	URI
Morph-LDP [32]	2014	si	R2RML	HTTP	si
Plugin for Eclipse [29]	2013	si	R2RML, domain	RDF dump	
R2RML Proc [41]	2013	si	R2RML, DM	RDF dump	
Transient [12]	2013	si	R2RML	materializado	
Formalization [31]	2014	si	R2RML	sparql	
SPARQLMAP [25]	2013	si (limitado)	R2RML (limitado)	sparql	
RDOTE [18]	2013	compatible	DM y domain	RDF dump	
Ultrawrap [30]	2013	no	domain	sparql	
Semantic qu[20]	2013	no	domain	sparql	
OntoUri [24]	2013	no	domain	OWL	si
StdTrip+K [26]	2013	no	domain	OWL	
Mapping [27]	2013	no	domain	OWL	
Semantic query [20]	2013	no	domain	servicio	
Algorithms [21]	2013	no	domain	repositorio semantico	
RDB2RDF [16]	2014	no	domain	RDFS/XML	
An Extensible [19]	2013	no	domain		
An integrated [28]	2011	no	domain	sparql	
Transformation[22]	2008	no	domain		

3.4.3. Clasificación del software relevado

Criterios de clasificación

1. Año de la versión más reciente, se descartan aquellas herramientas de software cuya última versión es anterior a 2013

2. Utiliza el estándar R2RML para realizar el mapeo de RDB a RDF.
3. Tipo de licenciamiento, se privilegia el software libre ya que permite instalar y probar las funcionalidades ofrecidas.
4. Tipo de resultado obtenido al utilizar la herramienta para convertir a RDF: endpoint SPARQL, RDF dump o traduce SQL.

La clasificación del software relevado se puede ver en el cuadro 3.2.

Cuadro 3.2: Clasificación de software relevado

NOMBRE	Año	Admite R2RML	LICENCIA	Resultado	Observaciones
Morph-LDP	2014	si	Open Source licencia Apache	expone RDBMS como Linked Data con LDP	Se basa en Morph-RD y agrega LDP para lectura y grabación de datos desde HTTP. Se intentó instalar para ser evaluado pero no se pudo hacer funcionar
Morph-RDB	2014	si	Open Source licencia Apache	materializa RDF / traduce sparql	Esta herramienta está incluída en Morph-LDP
Virtuoso	2014	si	Open Source Edition	base de datos híbrida almacena tablas relacionales, objetos, grafos y linked data. Servidor de Aplicaciones	Virtuoso desarrolló previamente su propio lenguaje equivalente a R2RML llamado Linked Data Views, que usa el Meta Schema Mapping Language de Virtuoso para el mapeo desde relacional a RDF. El soporte a R2RML se realiza incluyendo un traductor que genera la sintaxis de Linked Data Views a partir de la especificación de R2RML
RDB2RDF	2013	si		RDF dump	Es un plug-in para Eclipse que convierte desde relacional a RDF. No se pudo evaluar ya que los fuentes no estaban disponibles en el sitio indicado en el artículo [29]
RDF-RDB2RDF	2013	si	Perl-5	RDF dump	Mapeo declarativo desde una base relacional a RDF

Cuadro 3.2: Clasificación de software relevado

NOMBRE	Año	Admite R2RML	LICENCIA	Resultado	Observaciones
RDOTE	2014	compa- tible	GNU GPL	RDF dump	Entorno para transportar datos residentes en RDBMS a la Web Semántica. Provee una GUI, y suficiente expresividad para crear RDF dumps a partir de datos relacionales
Ultrawrap	2013	no	Comercial propieta-rio Capsenta	materializa RDF traduce SPARQL	Ultrawrap tiene dos partes: Compile y Server. Ultrawrap Compile procesa el mapeo y genera RDF y OWL. Ultrawrap Server ejecuta querys SPARQL agregando optimizationes para el motor SQL del RDBMS
db2triples	2013	si	LGPL v2.1	RDF dump	Db2triples es una librería de herramientas para extraer datos de bases de datos relacionales y cargarlas en un triplestore RDF. Implementa R2RML y Direct Mapping
Apache Marmotta	2014		Apache		Provee lectura-escritura de Linked Data, RDF triple store con transacciones, versionado y razonamiento basado en reglas, SPARQL, LDP y LDPath query
Revelytix Spyder		si	Comercial		Spyder es un software empresarial para exponer datos como RDF
D2RQ	2014	no	Freeware		D2RQ es una versión académica de Spyder, provee una alternativa freeware para exponer datos como RDF. Tiene menos funcionalidades que Spyder, en particular no soporta R2RML

Cuadro 3.2: Clasificación de software relevado

NOMBRE	Año	Admite R2RML	LICENCIA	Resultado	Observaciones
Triplify	2013	no	GNU Lesser General Public License	RDF dump	Triplify es un plugin para aplicaciones Web. Está basado en la definición de querys sobre bases de datos relacionales para una aplicación Web específica y convierte los resultados a RDF, JSON. Después de generar vistas en la base de datos el software Triplify software puede ser utilizado para convertir esas vistas a RDF, JSON, etc.
Ontop 1.12	2014	no	Apache 2.0 License	sparql	ontop- es una plataforma para realizar querys sobre bases de datos como grafos virtuales RDF utilizando SPARQL. Tiene un lenguaje de mapeo propietario. Integrado con Protege 4.x.

3.5. Conclusiones y consideraciones

3.5.1. Consideraciones a partir del relevamiento de artículos y software

- Se identifica un problema con el tiempo de procesamiento empleado en la transformación de una base de datos de tamaño medio, es decir una base de 10 o más GB. Este problema cobra importancia en el caso en que se decida transformar a RDF toda una base de datos. Por otra parte, si se trata de solamente de transformar una vista de la base a RDF, el volumen de los datos puede ser menor y los tiempos de la transformación dejarían de ser un problema.

- Otro problema es la vigencia de los datos en el RDF generado. Si los datos en la base de origen son estables y se actualizan poco, puede ser más eficiente generar el grafo ya que se trata de un proceso que va a correr relativamente poco comparado con transformar consultas cada vez que se necesita acceder al grafo RDF. Si interesa que el grafo RDF se mantenga lo más actualizado posible en comparación a la base de datos, la transformación tendría que aplicarse de forma periódica o encontrar algún mecanismo que permita solamente trabajar con las diferencias generadas por actualizaciones en la base de datos con el fin de optimizar los tiempos.

- Teniendo en cuenta el volumen y vigencia de los datos de origen, algunas de las herramientas evaluadas no materializan el grafo RDF sino que traducen una consulta SPARQL a SQL y la ejecutan contra la base de datos, resultando en una ejecución más óptima y evitando problemas como la desincronización entre los datos del grafo RDF y la base de datos. Al mismo tiempo también disminuyen los grandes tiempos del proceso de transformación. Por otra parte, la ventaja de materializar el RDF es que permite manipularlo, consultarlo con SPARQL, agregar restricciones, propiedades, aplicar razonamiento o exponerlo para que sea descubierto e incorporado a otros conocimientos con Linked Data.

- De la revisión de artículos se desprende que al momento del estudio el uso de R2RML no estaba suficientemente generalizado y la mayoría de los artículos relevados presentaba enfoques propios, algunos muy innovadores pero por fuera de las recomendaciones propuestas por W3C. Esta característica los hace inviables para el proyecto ya que la falta de estándares genera dificultades en el uso y dificulta su integración, mantenimiento y evolución.

- Del relevamiento realizado surge que la mayor parte de los productos existentes, proveen una interfase en SPARQL y la mayoría no provee el acceso al grafo vía una url. Además, algunos productos sólo permiten la conexión con su propia base relacional propietaria.

- Direct Mapping aparece mencionado solamente en dos de los artículos relevados. Esto puede ser debido a que si bien es un buen punto de partida para una transformación RDB2RDF, carece de la flexibilidad necesaria en entornos complejos.

- R2RML al momento del relevamiento apareció en una propuesta de plugin para Eclipse RDB2RDF y en los artículos del grupo que desarrolla Ultrawrap, Morph y MorphLDP. Como ya se mencionó anteriormente posiblemente esto se debe a que se trata de un estándar reciente al momento del relevamiento.

- En cuanto al software relevado, se encontraron problemas de compatibilidad y posibilidad de integración:

- Algunas herramientas solamente funcionan contra su propio motor de base de datos como es el caso de Virtuoso.

- Herramientas prometedoras que no pudieron probarse, como un plugin para Eclipse RDB2RDF, que no pudo instalarse porque su código no pudo ser encontrado.

- Apache Marmotta funciona como servidor web, pero debe complementarse programando o adaptando la consulta a la base de datos y solucionando la generación del grafo RDF a partir de la especificación R2RML, es decir implementando un R2RML Processor.

- De todo el software relevado, Morph-LDP parecía cumplir con todas lo necesario para comportarse como wrapper RDF, pero no se logró hacerlo funcionar a pesar de que se probaron varias alternativas en la instalación.

3.5.2. Conclusiones a partir del relevamiento

Se define como una necesidad poder ejecutar consultas sobre la base de datos para filtrar datos al generar el grafo RDF resultante de aplicar la transformación R2RDF. De esta forma, se podrían cubrir todos los requerimientos manteniendo una performance mucho más cercana a lo que seria trabajar directamente con una base de datos relacional. La parte adicional del trabajo seria hacer la traducción de las consultas y/o filtros hacia SQL, una vez hecho esto, el resto seria responsabilidad de la base de datos.

Si además se agregan condiciones de para permitir integración y compatibilidad ,se obtienen una serie de características deseables que un wrapper RDF debería cumplir:

1. Debe permitir definir qué parte de la base de datos se desea transformar a RDF. Por ejemplo, que pueda especificarse la transformación del resultado de una consulta sql, de una o varias tablas o de una vista definida en la base. Esto permite poder manejar el volumen de datos generado permitiendo acotar tiempo de procesamiento y espacio utilizado.

2. Interfase estándar para facilitar su aplicación y la integración al resto de la infraestructura de software existente

3. Independencia del proveedor de RDBMS, la herramienta debería permitir la conexión con la mayor cantidad posible de motores de base de datos existentes en el mercado.

Como consecuencia de no haber encontrado entre el software relevado uno que cumpla con estas características, se decide realizar un prototipo propio cuya descripción se verá en el próximo capítulo de este documento.

Capítulo 4

Prototipo de wrapper RDF

4.1. Objetivos del prototipo

El prototipo implementado como parte del proyecto de grado pretende publicar recursos en formato RDF que pueden proceder de diferentes fuentes de datos no necesariamente RDF. Dichos recursos son publicados a través de un endpoint que trata de cumplir la especificación más reciente de LDP a través del cual se pueden publicar nuevos recursos así como consultar o modificar recursos existentes.

La implementación actual ofrece acceso a recursos provenientes de tres fuentes de datos diferentes.

- Una implementación R2RML que permite especificar como se transforman datos desde el modelo relacional a RDF.
- Otra que ofrece datos que vienen desde la Web obtenidos mediante *crawling* desde la página de alquileres de apartamentos de *El Gallito*.
- Finalmente también se ofrecen recursos que vienen desde un endpoint SPARQL configurado.

El prototipo presenta la posibilidad de agregar nuevas fuentes de datos que pueden publicarse como recursos. Esta es una de las razones por las que se implementaron tres orígenes de datos diferentes de forma de demostrar que se soporta la inclusión de nuevas fuentes de datos de una forma relativamente simple.

4.2. Arquitectura general

El prototipo está implementado utilizando una arquitectura de tuberías y filtros. Se pueden observar 2 capas bien diferenciadas:

- La capa de LDP se encarga estrictamente de los detalles del protocolo LDP y de iniciar el procesamiento de una solicitud a través del sistema de filtros así como de esperar el resultado y convertirlo en una respuesta LDP válida.

- El sistema de filtros trabaja sobre mensajes que son publicados en canales, cada filtro toma los mensajes a procesar de un canal y publica su resultado a otro canal. Los canales se identifican por su nombre que es una cadena de texto. Existen dos canales especiales que son el canal de entrada y el canal de salida cuyos nombres son “input” y “” respectivamente.

El procesamiento de los mensajes se realiza en etapas:

- Primero todos los filtros procesan los mensajes que sean para ellos dejando sus salidas en una estructura temporal.
- Una vez que todos los filtros procesaron los mensajes esta estructura temporal se convierte en la nueva entrada para la siguiente etapa, descartando los mensajes anteriores.
- Este proceso continúa hasta que la salida de todos los filtros termina en el canal de salida, que no es entrada de ningún filtro.

La capa correspondiente al sistema de filtros está compuesta por la clase **QueueManager** que permite publicar, procesar, agregar filtros y obtener el resultado del procesamiento.

- Al iniciar el sistema el **QueueManager** debe inicializarse agregándole todos los filtros que van a estar disponibles según la configuración para procesar las solicitudes.
- Al momento en que llega una solicitud se llama al método *process* que se encarga de procesar los mensajes haciéndolos pasar por los filtros hasta que se termina todo el procesamiento necesario para la solicitud.
- En este momento se genera una notificación a través de una *Promesa* que contiene como parámetro el resultado del procesamiento.

La capa de LDP esta compuesta por 2 clases, **LDP** y **LDPController**.

- La clase **LDP** es la encargada de traducir las solicitudes LDP en invocaciones a **LDPController** y de convertir las respuestas devueltas por LDPController a una respuesta LDP válida.
- La clase **LDPController** se encarga de inicializar el sistema de colas, instanciar los filtros así como iniciar la ejecución de los filtros para servir una solicitud y esperar la respuesta final de los mismos.

4.2.1. Transformaciones R2RML

El prototipo utiliza R2RML para especificar las transformaciones desde el modelo relacional a RDF. Estas transformaciones se guardan en algún tipo de almacenamiento que debe poder ser consultado a través de un endpoint SPARQL. El prototipo fue desarrollado utilizando Apache Marmotta que luego fue remplazado por Virtuoso por problemas de estabilidad.

Las transformaciones R2RML siguen la especificación publicada en [42]. Como parte del prototipo y por razones de performance se agrega una pequeña extensión a R2RML agregando la propiedad *http://www.w3.org/ns/r2rml#database* con el objetivo de poder especificar sobre que base de datos se espera que se ejecute una consulta. Esto se debe a que pueden estar utilizándose múltiples bases de datos incluso para un único recurso y se quiere lograr que las consultas se ejecuten en las bases correctas.

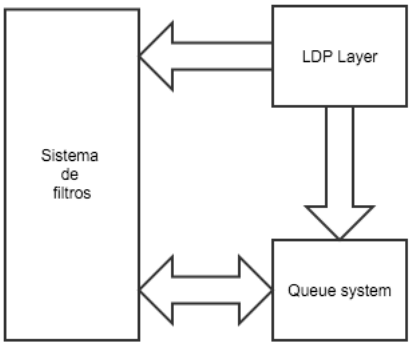

Figura 4.1: Esquema de capas del prototipo

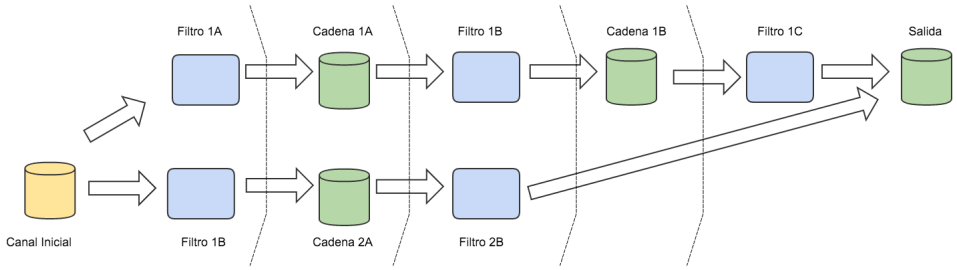

Figura 4.2: primer paso del proceso

4.2.2. Procesamiento de una solicitud

En la implementación específica se tienen colas de mensajes con un canal inicial y un canal final.

1. Las solicitudes LDP se encolan en el canal inicial
2. Los filtros tomarán los mensajes de los diferentes canales para procesarlos y publicarán sus salidas en otros canales.
3. Este proceso se repetirá hasta que solamente haya mensajes en el canal de salida desde donde serán tomados para generar la respuesta a los clientes del sistema.

En la secuencia de imágenes se ejemplifica el funcionamiento de este proceso simulando una ejecución a través del sistema.

1. En el primer paso del proceso (ver figura 4.2) se puede apreciar que se cuenta con dos cadenas de filtros diferentes a través de las cuales se procesará la solicitud.
2. En el segundo paso (ver figura 4.3) se puede ver que el mensaje de entrada ha sido procesado por el primer filtro en cada una de las cadenas y un mensaje de salida de cada filtro ha sido publicado en el canal de salida.

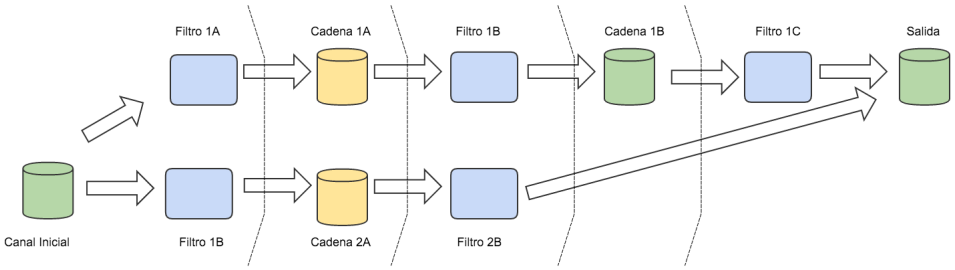

Figura 4.3: segundo paso del proceso

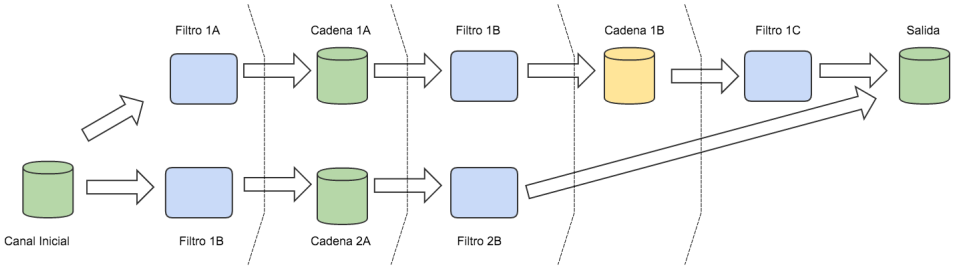

Figura 4.4: tercer paso del proceso

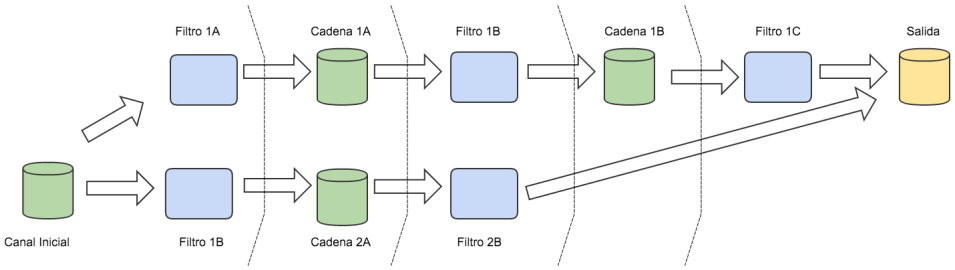

Figura 4.5: cuarto paso del proceso

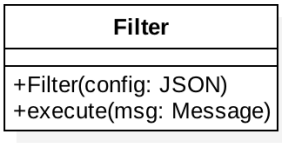

Figura 4.6: Representación de un filtro

3. En el tercer paso (ver figura 4.4) los mensajes de la segunda etapa son procesados nuevamente. Esta vez el mensaje de la segunda cadena llegó al final de su procesamiento puesto que no hay más filtros y está a la espera de ser tomado para generar la respuesta a la solicitud. Mientras tanto el mensaje en la cadena 1 ha pasado al último canal antes de terminar su procesamiento.

4. En el cuarto y último paso (ver figura 4.5) todos los mensajes han llegado al canal de salida y están esperando a ser tomados para generar la respuesta al cliente. En este momento se considera terminado el procesamiento a través de la cadena de filtros y se pasa a generar la respuesta gracias al controlador de la aplicación que se explicará más adelante.

El diseño basado en esta arquitectura permite la reutilización de los filtros ya existentes para obtener nuevas formas de generar una respuesta. A modo de ejemplo, en la cadena que transforma de relacional a RDF podrían utilizarse todos los filtros del camino y simplemente remplazar el filtro que genera y ejecuta la consulta contra la base de datos por otro logrando obtener soporte para una base de datos relacional aun no soportada. Al trabajar de esta forma, quien quiera extender las bases de datos soportadas no tiene ninguna necesidad de acceder al código fuente de los filtros anteriores ya que solamente le importa la salida que este filtro en particular genera y la salida que el filtro que desea implementar debe generar.

4.2.3. Estructura de un filtro

Un filtro (ver figura 4.6) consiste de una clase con un constructor que recibe la configuración y un método *execute*. Este método recibirá como único parámetro el mensaje de entrada y devolverá una *promesa*. Al finalizar la ejecución, el filtro debería resolver la promesa dando como resultado un objeto con propiedades *message* y *to*, donde *message* es el mensaje de salida que será pasado al siguiente filtro en la cadena y *to* que representa el canal en el cual se publicará la respuesta. La función constructor que creará el filtro recibe un único parámetro que es la configuración del filtro. No existe ninguna regla respecto a qué datos deben estar presentes en la configuración del filtro. Generalmente un filtro debe saber dónde publicará sus mensajes de salida por lo que generalmente alguna de las propiedades en la configuración especifica dónde publicar la salida.

En la ejecución de un filtro se espera que se devuelva una Promesa debido que muchas de las operaciones que estos pueden llegar a realizar pueden tener una naturaleza asincronica.

La configuración de todos los filtros en el sistema se hace a través de una lista de objetos JSON, donde cada uno tiene la siguiente estructura:

```
1 {
2   "name": <Nombre del filtro>,
```

```
  "from": <De donde toma sus mensajes de entrada>,
  "config": { <Objeto JSON con la configuracion especifica para el filtro> }
  }
```

4.3. Implementación del prototipo

El prototipo fue implementado utilizando JavaScript que ejecuta sobre NodeJS. También se utilizaron las librerias *MySQL*, *Node-Spider*, *Promise*, *Sparql-client* y *N3*. Se utilizó una libreria para LDP que luego fue removida como dependencia y tomada como parte del proyecto ya que no implementaba correctamente las especificaciones del protocolo o al menos las especificaciones actuales por lo que hubo que hacerle numerosas actualizaciones y mejoras a los reportes de errores.

4.3.1. Librerías utilizadas

- N3: Fue utilizada como una librería de serialización para poder leer y escribir datos en RDF. Es utilizada al momento de escribir una respuesta RDF para un request, así como para parsear una solicitud RDF cuando se requiere crear o actualizar un recurso existente.
- Node-Spider: Esta librería se utilizó para implementar un filtro de ejemplo que publica como recursos los avisos de alquileres de apartamentos publicados en los avisos clasificados (El Gallito). La utilidad de esta librería es principalmente manejar la descarga de las páginas y poder seguir los links desde la página principal a las publicaciones.
- Promise: Es una implementación de promesas en JavaScript para NodeJS.
- Sparql-client: Permite realizar solicitudes a un endpoint sparql.
- MySQL: Es un cliente de MySQL para NodeJS.

4.3.2. Filtros de la cadena R2RML

En la cadena de filtros para publicar datos R2RML uno de los mas complejos es el ARTranslator (de Abstract Representation Translator). El objetivo de este filtro es dada una transformación R2RML dar una representación intermedia de la transformación mas apropiada para procesar la solicitud entrante.

El problema con R2RML es que no dice de una forma directa (sin navegar el grafo) que columnas son las que nos interesa proyectar, que tablas, etc. Por eso el trabajo de generar la consulta se divide en varias etapas. La representación intermedia generada por este filtro tiene la forma:

```
{NombreTabla:
  { mapping:
    {Columna1: PropiedadRDF1, . . . ., ColumnaN: PropiedadRDFN},
  database: NombreBaseDeDatos,
  template: <url del template R2RML>
  }
```

```
}
```

A partir de esta representación es sencillo generar una sentencia del estilo SELECT.

```
SELECT Columna1, . . . , ColumnaN FROM NombreTabla
```

Pero en caso de que nos pidan un recurso particular falta ver que se deberia utilizar como condición en la sentencia WHERE. De esto se encarga el segundo filtro de la cadena RelToRDF que toma la entrada de ARTranslator e incrementa la información para luego poder generar una consulta. Este filtro se encarga de extraer de la URL en la solicitud los parametrso que coinciden con los nombres de columnas para agregarlos a las clausulas de WHERE. Con lo que da como salida una estructura de la siguiente forma.

```
{NombreTable:
  { mapping:
    {Columna1: PropiedadRDF1, . . . ., ColumnaN: PropiedadRDFN},
  database: NombreBaseDeDatos,
  candidateKeys: { ColumnaX: valorX, ColumnaY: valorY, . . .}
  }
}
```

Para poder generar esta salida se toma el tempalte del filtro anterior y la URI de la solicitud. A modo de ejemplo con los siguientes datos de entrada se termina deduciendo que se tiene un filtro donde la columna id se iguala al valor 1.

URL de la solicitud http://localhost/resources/Persona/1 tempalte http://localhost/resources/Persona/{id}

La salida de este filtro pasa luego al filtro que genera la consulta SQL (QueryGenerator) que termina generando una consulta de la forma

```
SELECT Columna1, . . . , ColumnaN FROM NombreTabla WHERE ColumnaX = valorX and ColumnaY = valorY
    , . . . .
```

Esta consulta junto con la información anterior se convierte en la entrada para el siguiente filtro DbExecutor que termina ejecutando la consulta contra la base de datos a partir de la siguiente información.

```
{ mapping:
  [NombreTable:
    { mapping: {Columna1: PropiedadRDF1, . . . ., ColumnaN: PropiedadRDFN},
    candidateKeys: { ColumnaX: valorX, ColumnaY: valorY, . . .}
    }
  ],
  query: [ SELECT Columna1, . . . , ColumnaN FROM NombreTabla WHERE ColumnaX
      = valorX and ColumnaY = valorY, . . . ]
}
```

El resultado de esta consulta es acumulado al mensaje de salida dando como resultado

```
{ mapping:
  [NombreTable:
    { mapping:
      {Columna1: PropiedadRDF1, . . . ., ColumnaN: PropiedadRDFN},
    candidateKeys: { ColumnaX: valorX, ColumnaY: valorY, . . .}
    }
```

```
  ],
 data: [ {NombreTabla: { Columna1: valor1, . . ., ColumnaN: valorN} }  ]
 }
```

Esto es tomado como entrada por el ultimo filtro de la cadena (RelToRDF) que se encarga de generar las ternas RDF.

```
[ { subject: Sujeto, predicate: PropiedadRDF1, object: valor1}, . . ., {
    subject: Sujeto, predicate: PropiedadRDFN, object: valorN} ]
```

4.4. Inclusión de nuevas fuentes de datos

Incluir una nueva fuente de datos implica crear el conjunto de filtros necesarios para poder manipular la nueva fuente. En muchos casos no será necesario escribir todos los filtros de la nueva cadena ya que idealmente se deberían poder reutilizar filtros que se están utilizando para otras fuentes de datos.

Como regla general aunque no es realmente necesario se codifica un filtro por archivo en JavaScript. El código fuente de un filtro siempre tiene la siguiente estructura.

```
exports.createTranslator = function(config) {
  var execute = function(message) {
    // TODO: Here is the code for your filter.
  }

  return {
    execute: execute
  };
};
```

La función execute es donde el filtro procesara el mensaje de entrada y también devolvera una promesa la cual se resolvera con el mensaje de salida del filtro.

```
var execute = function(message) {
    return new Promise(function(resolve, reject) {
      // TODO: Some procesing code

      resolve({"message": {}, "to": config.to});
    });
  }
```

El mensaje de respuesta siempre contiene una propiedad message que es la salida del filtro y la propiedad to indicando en que canal se desea publicar la respuesta. Como se puede ver en el ejemplo de código el valor para to siempre se toma de la configuración debido a que es la única forma de permitir que configurar el filtro para que pueda interactuar con filtros que no conoce.

Una vez creados los filtros necesarios el ultimo paso implica configurar como se encadenan los filtros correctamente para producir la salida final.

4.4.1. Consideraciones sobre las tecnologías utilizadas

Resultó interesante realizar el desarrollo con NodeJS debido a la gran productividad que se puede llegar a lograr con el mismo. Al mismo tiempo es posible encontrar una cantidad de librerías muy grande para realizar diferentes tareas.

Con respecto a las librerías utilizadas, se encontraró que hay librerías que están bien establecidas y tienen un buen soporte y confiabilidad, mientras que existe una gran cantidad de librerías que no pueden ni ser instaladas correctamente y no se brinda información sobre qué tanto se las está utilizando y qué soporte tienen.

Si bien Javascript mostró tener un nivel de productividad muy alto y permite generar prototipos increíblemente rápido comparado con otras tecnologías, también mostró algunos problemas generalmente asociados a lenguajes débilmente tipados. En un sistema complejo donde los datos se van transformando a diferentes representaciones resultó muy común olvidar la estructura de los objetos que se mueven entre funciones lo que solamente es detectable en tiempo de ejecución.

4.5. Conclusiones finales y trabajo futuro

4.5.1. Conclusiones

El wrapper presentado cumple con los objetivos planteados en el proyecto ya que soporta mecanismos de extensión y en particular, soporte las extensiones a R2RML.

Permite la integración, compatibilidad y las características deseables definidas en la sección Conclusiones de la segunda parte de este documento para un wrapper RDF:

1. Permite definir qué vistas o tablas de la base de datos se desea transformar a rdf. Esto permite poder manejar el volumen de datos generado permitiendo acotar el tiempo de procesamiento y el espacio utilizado. Esta funcionalidad está provista por la utilización del estándar R2RML que permite definir vistas o tablas a transformar.

2. Interfase estándar para facilitar su aplicación y la integración al resto de la infraestructura de software existente. Los requerimientos se realizan a través de un cliente REST instalados en un browser de internet y el grafo RDF resultante de la transformación se recupera a través de la URI provista.

3. Independencia del proveedor de RDBMS, el prototipo permite la integración con cualquier manejador de base de datos que soporte SQL estándar.

4.5.2. Trabajo futuro

- Una vez sentadas las bases de funcionamiento del prototipo y sus características básicas, es posible seguir incorporando módulos para el tratamiento de distintos tipos de datos lo que permitirá construir una herramienta que realice la transformación a RDF de distintos formatos.

- Será necesario realizar pruebas de performance para comprobar el comportamiento del wrapper RDF presentado en relación con el resto de las herramientas estudiadas.

- Se podrá comprobar el funcionamiento del wrapper como solución a un problema de integración empresarial realizando la implementación de la solución propuesta en el caso de ejemplo presentado en la primera parte del documento.

- Deberia ser posible realizar diferentes optimizaciones a la implementación agregando diferentes niveles de cache. A modo de ejemplo se puede pensar que si alguien realiza una consulta para un tipo de recurso particular podria llegar a hacer una consulta para un mismo tipo de recurso en un periodo corto de tiempo. Esto implica que si la representación intermedia generada para dicho recurso es almacenada en un cache seria posible evitar todas las consultas SPARQL que son ejecutadas contra el endpoint SPARQL. De esta forma incrementando mucho el tiempo de respuesta de la aplicación.
- También sería necesario poder escribir los datos a medida que se van generando al stream de respuesta o a un archivo en disco debido a que si la cantidad de datos generados para un recurso en particular es muy grande esto podría causar que el servidor se quedara sin memoria y esto provocaría una falla del prototipo de wrapper RDF .
- En cuanto al relevamiento de artículos y software, se trabajó en un entono muy dinámico con estándares muy recientes. Tan es así que durante 2014 se pudo presenciar el trabajo del grupo LDP que renovaba semanalmente los documentos publicados en la Web. Se estima que al momento de presentar este trabajo ya existen nuevas implementaciones de R2RML que no fueron relevadas y en cuanto a LDP recientemente alcanzó el status de recomendación de la W3C por lo que es de esperarse que comiencen a publicarse nuevas herramientas que lo implementan.

Bibliografía

[1] F. Carpani, “Lenguajes y tecnologias de la web semantica,” CPAP Cursos y Postgrados de Actualizacion Profesional, Facultad de Ingenieria de la Universidad de la Republica Oriental del Uruguay, 2014.

[2] *Resource Description Framework (RDF) Model and Syntax Specification*, 1998, available at http://www.w3.org/TR/WD-rdf-syntax/. [Online]. Available: http://www.w3.org/TR/WD-rdf-syntax/

[3] A. Hogan, “Linked data & the semantic web standards,” in *Linked Data Management*, A. Harth, K. Hose, and R. Schenkel, Eds. Chapman and Hall/CRC, 2014, pp. 3–48. [Online]. Available: http://www.crcnetbase.com/isbn/9781466582415

[4] E. Prud’hommeaux and A. Seaborne, “SPARQL query language for RDF,” W3C, W3C Recommendation, Jan. 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. [Online]. Available: http://www.w3.org/2001/sw/wiki/SPARQL

[5] S. Das, R. Cyganiak, and S. Sundara, “R2RML: RDB to RDF mapping language,” W3C, W3C Recommendation, Sep. 2012, http://www.w3.org/TR/2012/REC-r2rml-20120927/. [Online]. Available: http://www.w3.org/TR/2012/REC-r2rml-20120927/

[6] S. Speicher, J. Arwe, and A. Malhotra, “Linked data platform 1.0,” World Wide Web Consortium, Recommendation REC-ldp-20150226, Tech. Rep., 2015.

[7] L. Liu and M. T. Zsu, *Encyclopedia of database systems*. Springer Publishing Company, Incorporated, 2009. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-39940-9

[8] F. Carpani, “Linked data y linked data platform taller de lenguajes y tecnolog?as de la web sem,” CSI InCo Facultad de Ingenieria de la Universidad de la Republica Oriental del Uruguay, 2015.

[9] A. J. Le Hors and S. Speicher, “The linked data platform (ldp),” in *Proceedings of the 22nd international conference on World Wide Web companion*. International World Wide Web Conferences Steering Committee, 2013, pp. 1–2.

[10] A. Harth, K. Hose, and R. Schenkel, *Linked Data Management*. Chapman and Hall/CRC, 2014. [Online]. Available: http://www.crcnetbase.com/isbn/9781466582415

[11] M. Rodriguez-Muro and M. Rezk, “Efficient sparql-to-sql with r2rml mappings,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 33, pp. 141–169, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2015.03.001

[12] N. Konstantinou, D. Spanos, and N. Mitrou, “Transient and persistent RDF views over relational databases in the context of digital repositories,” in *Metadata and Semantics Research - 7th Research Conference, MTSR 2013, Thessaloniki, Greece, November 19-22, 2013. Proceedings*, ser. Communications in Computer and Information Science, E. Garoufallou and J. Greenberg, Eds., vol. 390. Springer, 2013, pp. 342–354. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-03437-9_33

[13] F. Michel, J. Montagnat, and C. Faron-Zucker, “A survey of RDB to RDF translation approaches and tools,” I3S, Research Report, May 2014, iSRN I3S/RR 2013-04-FR 24 pages. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00903568

[14] M. Arenas, J. Sequeda, E. Prud’hommeaux, and A. Bertails, “A direct mapping of relational data to RDF,” W3C, W3C Recommendation, Sep. 2012, http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/. [Online]. Available: http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/

[15] E. Blomqvist, “The use of semantic web technologies for decision support-a survey.” *Semantic Web*, vol. 5, no. 3, pp. 177–201, 2014.

[16] P. T. T. Thuy, N. D. Thuan, Y. Han, K. Park, and Y. Lee, “RDB2RDF: completed transformation from relational database into RDF ontology,” in *The 8th International Conference on Ubiquitous Information Management and Communication, ICUIMC ’14, Siem Reap, Cambodia - January 09 - 11, 2014*, S. Lee, S. Kim, L. Hanzo, and R. Ismail, Eds. ACM, 2014, pp. 88:1–88:7. [Online]. Available: http://doi.acm.org/10.1145/2557977.2558083

[17] M. Hausenblas and B. Villazón-Terrazas, “R2RML and direct mapping test cases,” W3C, W3C Note, Aug. 2012, http://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/.

[18] K. N. Vavliakis, T. K. Grollios, and P. A. Mitkas, “RDOTE - publishing relational databases into the semantic web,” *Journal of Systems and Software*, vol. 86, no. 1, pp. 89–99, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.07.018

[19] M. Farouk and M. Ishizuka, “An extensible approach for mapping relational DB to RDF,” in *Electronics, Communications and Computers (JEC-ECC), 2012 Japan-Egypt Conference on*. IEEE, 2012, p. 163 to 167.

[20] A. Riazanov, A. Klein, A. Shaban-Nejad, G. W. Rose, A. J. Forster, D. L. Buckeridge, and C. J. O. Baker, “Semantic querying of relational data for clinical intelligence: a semantic web services-based approach,” *J. Biomedical Semantics*, vol. 4, p. 9, 2013. [Online]. Available: http://dx.doi.org/10.1186/2041-1480-4-9

[21] W. Y. Mallede, F. Marir, and V. Vassilev, “Algorithms for mapping RDB schema to RDF for facilitating access to deep web,” in *WEB 2013, The First International Conference on Building and Exploring Web Based Environments*, 2013, p. 32 to 41.

[22] Y. Lv and Z. Ma, “Transformation of relational model to RDF model,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Singapore, 12-15 October 2008*. IEEE, 2008, pp. 506–511. [Online]. Available: http://dx.doi.org/10.1109/ICSMC.2008.4811327

[23] H. Jiang, W. J. Liu, and L. L. Lu, “A semantic query method for deep web,” in *Applied Mechanics and Materials*, vol. 347. Trans Tech Publ, 2013, pp. 2559–2563. [Online]. Available: http://www.atlantis-press.com/php/download_paper.php?id=4938

[24] C. Jeong, S. Choi, S. Shin, S. Lee, H. Jung, S. Kim, and P. Kim, “Creating semantic data from relational database,” in *International Conference on Social Computing, SocialCom 2013, SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013*. IEEE Computer Society, 2013, pp. 1081–1086. [Online]. Available: http://dx.doi.org/10.1109/SocialCom.2013.174

[25] J. Unbehauen, C. Stadler, and S. Auer, “Accessing relational data on the web with sparqlmap,” in *Semantic Technology, Second Joint International Conference, JIST 2012, Nara, Japan, December 2-4, 2012. Proceedings*, ser. Lecture Notes in Computer Science, H. Takeda, Y. Qu, R. Mizoguchi, and Y. Kitamura, Eds., vol. 7774. Springer, 2012, pp. 65–80. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37996-3_5

[26] R. Berardi, K. Breitman, M. A. Casanova, G. R. Lopes, and A. P. de Medeiros, “Stdtrip+k: Design rationale in the rdb-to-rdf process,” in *Database and Expert Systems Applications - 24th International Conference, DEXA 2013, Prague, Czech Republic, August 26-29, 2013. Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Decker, L. Lhotská, S. Link, J. Basl, and A. M. Tjoa, Eds., vol. 8055. Springer, 2013, pp. 303–310. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40285-2_26

[27] H. Ling and S. Zhou, “Mapping relational databases into OWL ontology,” *International Journal of Engineering and Technology*, vol. 5, no. 6, pp. 4735–4740, 2013.

[28] K. N. Vavliakis, A. L. Symeonidis, G. T. Karagiannis, and P. A. Mitkas, “An integrated framework for enhancing the semantic transformation, editing and querying of relational databases,” *Expert Syst. Appl.*, vol. 38, no. 4, pp. 3844–3856, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2010.09.045

[29] E. Marx, P. E. Salas, K. Breitman, J. V. Filho, and M. A. Casanova, “RDB2RDF: A relational to RDF plug-in for eclipse,” *Software: Practice and Experience*, vol. 43, no. 4, pp. 435–447, 2013. [Online]. Available: http://dx.doi.org/10.1002/spe.2145

[30] J. Sequeda and D. P. Miranker, “Ultrawrap: SPARQL execution on relational data,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 22, pp. 19–39, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2013.08.002

[31] F. Priyatna, Ó. Corcho, and J. Sequeda, “Formalisation and experiences of r2rml-based SPARQL to SQL query translation using morph,” in *23rd International World Wide Web Conference, WWW ’14, Seoul, Republic of Korea, April 7-11, 2014*, C. Chung, A. Z. Broder, K. Shim, and T. Suel, Eds. ACM, 2014, pp. 479–490. [Online]. Available: http://doi.acm.org/10.1145/2566486.2567981

[32] N. Mihindukulasooriya, F. Priyatna, Ó. Corcho, R. García-Castro, and M. E. Gutiérrez, “morph-ldp: An r2rml-based linked data platform implementation,” in *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis,

and A. Tordai, Eds., vol. 8798. Springer, 2014, pp. 418–423. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11955-7_59

[33] N. Konstantinou, N. Houssos, and A. Manta, "Exposing bibliographic information as linked open data using standards-based mappings: methodology and results," *Procedia-Social and Behavioral Sciences*, vol. 147, pp. 260–267, 2014.

[34] Z. Elkins, T. Ginsburg, J. Melton, R. Shaffer, J. F. Sequeda, and D. P. Miranker, "Constitute: The world s constitutions to read, search, and compare," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 27, p. 10 to 18, 2014.

[35] S. Hagedorn and K.-U. Sattler, "Lodhub a platform for sharing and integrated processing of linked open data," in *2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2014, pp. 260–262.

[36] M. G. Skjæveland, M. Giese, D. Hovland, E. H. Lian, and A. Waaler, "Engineering ontology-based access to real-world data sources," *J. Web Sem.*, vol. 33, pp. 112–140, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2015.03.002

[37] R. Berardi, V. M. P. Vidal, and M. A. Casanova, "R^2ba - rationalizing R2RML mapping by assertion," in *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 2, Barcelona, Spain, 27-30 April, 2015*, S. Hammoudi, L. A. Maciaszek, and E. Teniente, Eds. SciTePress, 2015, pp. 5–14. [Online]. Available: http://dx.doi.org/10.5220/0005337700050014

[38] L. F. de Medeiros, F. Priyatna, and Ó. Corcho, "MIRROR: automatic R2RML mapping generation from relational databases," in *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, ser. Lecture Notes in Computer Science, P. Cimiano, F. Frasincar, G. Houben, and D. Schwabe, Eds., vol. 9114. Springer, 2015, pp. 326–343. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-19890-3_21

[39] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle, "Rml: a generic language for integrated rdf mappings of heterogeneous data," in *Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014), Seoul, Korea*, 2014.

[40] J. Slepicka, C. Yin, P. A. Szekely, and C. A. Knoblock, "KR2RML: an alternative interpretation of R2RML for heterogenous sources," in *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, US, October 12th, 2015.*, ser. CEUR Workshop Proceedings, O. Hartig, J. Sequeda, and A. Hogan, Eds., vol. 1426. CEUR-WS.org, 2015. [Online]. Available: http://ceur-ws.org/Vol-1426/paper-08.pdf

[41] S. Zhou, Z. Xu, Y. Ni, and H. Zhang, "R2rml processor for materializing RDF view of relational data: Algorithms and experiments," in *Web Information System and Application Conference (WISA), 2013 10th*, 2013, p. 450 to 455.

[42] S. Das, S. Sundara, and R. Cyganiak, "R2RML: RDB to RDF mapping language," World Wide Web Consortium, Recommendation REC-r2rml-20120927, Tech. Rep., 2012.

[43] D. Calvanese, W. Fischl, R. Pichler, E. Sallinger, and M. Simkus, “Capturing relational schemas and functional dependencies in RDFS,” in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, C. E. Brodley and P. Stone, Eds. AAAI Press, 2014, pp. 1003–1011. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8487

[44] A. G. Hernández and M. N. M. García, “Restful writable apis for the web of linked data using relational storage solutions,” in *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*, ser. CEUR Workshop Proceedings, C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., vol. 813. CEUR-WS.org, 2011. [Online]. Available: http://ceur-ws.org/Vol-813/ldow2011-paper04.pdf

[45] N. Mihindukulasooriya, R. Garcia-Castro, and M. E. Gutiérrez, “Linked data platform as a novel approach for enterprise application integration,” in *Proceedings of the Fourth International Workshop on Consuming Linked Data, COLD 2013, Sydney, Australia, October 22, 2013*, ser. CEUR Workshop Proceedings, O. Hartig, J. Sequeda, A. Hogan, and T. Matsutsuka, Eds., vol. 1034. CEUR-WS.org, 2013. [Online]. Available: http://ceur-ws.org/Vol-1034/MihindukulasooriyaEtAl_COLD2013.pdf

[46] M. H. Roshdy, K. M. Fadel, and H. F. ElYamany, “Developing a RDB-RDF management framework for interoperable web environments,” in *Proceedings of Eurocon 2013, International Conference on Computer as a Tool, Zagreb, Croatia, July 1-4, 2013*. IEEE, 2013, pp. 307–313. [Online]. Available: http://dx.doi.org/10.1109/EUROCON.2013.6625001

[47] D. Allemang and J. A. Hendler, *Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL, Second Edition*. Morgan Kaufmann, 2011. [Online]. Available: http://www.elsevierdirect.com/product.jsp?isbn=9780123859655

[48] M. Arenas, J. Sequeda, E. Prud’hommeaux, and A. Bertails, “A direct mapping of relational data to RDF,” W3C, W3C Recommendation, Sep. 2012, http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/.

[49] A. Chebotko, S. Lu, and F. Fotouhi, “Semantics preserving sparql-to-sql translation,” *Data & Knowledge Engineering*, vol. 68, no. 10, pp. 973–1000, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.datak.2009.04.001

[50] S. Das, R. Cyganiak, and S. Sundara, “R2RML: RDB to RDF mapping language,” W3C, W3C Recommendation, Sep. 2012, http://www.w3.org/TR/2012/REC-r2rml-20120927/.

[51] J. Domingue, D. Fensel, and J. A. Hendler, *Handbook of Semantic Web Technologies*, J. Domingue, D. Fensel, and J. A. Hendler, Eds. Springer Science & Business Media, 2011, vol. 1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92913-0

[52] U. Eco, *Como se hace una tesis*. Editorial Gedisa, 2014.

[53] S. Fernandez, “Introduction to ldp in apache marmotta,” 2014. [Online]. Available: http://marmotta.apache.org/events/iswc2014

[54] M. Hausenblas and B. Villazón-Terrazas, “R2RML and direct mapping test cases,” W3C, W3C Note, Aug. 2012, http://www.w3.org/TR/2012/NOTE-rdb2rdf-test-cases-20120814/.

[55] ——, "RDB2RDF implementation report," W3C, W3C Note, Aug. 2012, http://www.w3.org/TR/2012/NOTE-rdb2rdf-implementations-20120814/.

[56] O. Lassila, "Resource description framework (RDF) model and syntax specification," W3C, W3C Recommendation, Feb. 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[57] Y. Lu, C. Li, R. Sun, W. Lihua, and G. Zhu, "Improved automatic conversion rules and algorithm from relational schema to ontology," *Journal of Computational and Theoretical Nanoscience*, vol. 11, no. 6, pp. 1537–1541, 2014.

[58] N. Mihindukulasooriya, "Learning w3c linked data platform with examples," Center for Open Middleware, 2014. [Online]. Available: http://www.ldp4j.org/tutorials/eswc2015/

[59] D. L. Phuoc, H. Q. Nguyen-Mau, J. X. Parreira, and M. Hauswirth, "A middleware framework for scalable management of linked streams," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 16, pp. 42–51, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2012.06.003

[60] J. J. W. Presbrey, "Linked data platform for web applications," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[61] E. Prud'Hommeaux and A. Seaborne, "SPARQL query language for RDF," World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115, Jan. 2008.

[62] F. Song, G. Zacharewicz, and D. Chen, "An ontology-driven framework towards building enterprise semantic information layer," *Advanced Engineering Informatics*, vol. 27, no. 1, pp. 38–50, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.aei.2012.11.003

[63] M. D. Wilkinson, B. P. Vandervalk, and E. L. McCarthy, "The semantic automated discovery and integration (SADI) web service design-pattern, API and reference implementation," *J. Biomedical Semantics*, vol. 2, p. 8, 2011. [Online]. Available: http://dx.doi.org/10.1186/2041-1480-2-8

[64] C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., *WWW2011 Workshop on Linked Data on the Web, Hyderabad, India, March 29, 2011*, ser. CEUR Workshop Proceedings, vol. 813. CEUR-WS.org, 2011. [Online]. Available: http://ceur-ws.org/Vol-813

[65] C. E. Brodley and P. Stone, Eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. AAAI Press, 2014. [Online]. Available: http://www.aaai.org/Library/AAAI/aaai14contents.php

[66] C. Chung, A. Z. Broder, K. Shim, and T. Suel, Eds., *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*. ACM, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2566486

[67] P. Cimiano, F. Frasincar, G. Houben, and D. Schwabe, Eds., *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, ser. Lecture Notes in Computer Science, vol. 9114. Springer, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-19890-3

[68] H. Decker, L. Lhotská, S. Link, J. Basl, and A. M. Tjoa, Eds., *Database and Expert Systems Applications - 24th International Conference, DEXA 2013, Prague, Czech Republic, August 26-29, 2013. Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 8055. Springer, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40285-2

[69] E. Garoufallou and J. Greenberg, Eds., *Metadata and Semantics Research - 7th Research Conference, MTSR 2013, Thessaloniki, Greece, November 19-22, 2013. Proceedings*, ser. Communications in Computer and Information Science, vol. 390. Springer, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-03437-9

[70] S. Hammoudi, L. A. Maciaszek, and E. Teniente, Eds., *ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 2, Barcelona, Spain, 27-30 April, 2015*. SciTePress, 2015.

[71] O. Hartig, J. Sequeda, and A. Hogan, Eds., *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, US, October 12th, 2015*, ser. CEUR Workshop Proceedings, vol. 1426. CEUR-WS.org, 2015. [Online]. Available: http://ceur-ws.org/Vol-1426

[72] O. Hartig, J. Sequeda, A. Hogan, and T. Matsutsuka, Eds., *Proceedings of the Fourth International Workshop on Consuming Linked Data, COLD 2013, Sydney, Australia, October 22, 2013*, ser. CEUR Workshop Proceedings, vol. 1034. CEUR-WS.org, 2013. [Online]. Available: http://ceur-ws.org/Vol-1034

[73] S. Lee, S. Kim, L. Hanzo, and R. Ismail, Eds., *The 8th International Conference on Ubiquitous Information Management and Communication, ICUIMC '14, Siem Reap, Cambodia - January 09 - 11, 2014*. ACM, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2557977

[74] V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, Eds., *The Semantic Web: ESWC 2014 Satellite Events - ESWC 2014 Satellite Events, Anissaras, Crete, Greece, May 25-29, 2014, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 8798. Springer, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11955-7

[75] H. Takeda, Y. Qu, R. Mizoguchi, and Y. Kitamura, Eds., *Semantic Technology, Second Joint International Conference, JIST 2012, Nara, Japan, December 2-4, 2012. Proceedings*, ser. Lecture Notes in Computer Science, vol. 7774. Springer, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37996-3

[76] *Proceedings of Eurocon 2013, International Conference on Computer as a Tool, Zagreb, Croatia, July 1-4, 2013*. IEEE, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6596534

[77] *International Conference on Social Computing, SocialCom 2013, SocialCom/-PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013*. IEEE Computer Society, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6693161

[78] *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Singapore, 12-15 October 2008*. IEEE, 2008. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4803719

Apéndice A

Anexos

A.1. Instructivo para la instalación del wrapperRDF

Prerequsitos:

1. Instalar NodeJS, esta aplicación se construyó utilizando NodeJS v0.10.25. No debería haber problemas en ejecutarlo con versiones más nuevas pero dado que se esta yendo a una implementación de ECMA 6 podría surgir algún problema de compatibilidad. En caso de encontrar cualquier problema, se recomienda utilizar la versión con la que el software fue escrito.

2. Instalar una base de datos relacional, como por ejemplo MySQL, en caso de no tener un RDBMS ya instalado. Se puede bajar MySQL desde *http://www.mysql.com/downloads/*

3. Se necesita tener instalado algún software que ofrezca un endpoint SPARQL 1.1 Se puede bajar Marmotta desde *http://marmotta.apache.org/download.html*

4. Para utilizar la aplicación debe instalarse en el navegador internet (browser) un cliente REST desde donde se ingresarán los comandos

Preparando el software:

1. Bajar los paquetes del repositorio *https://gitlab.fing.edu.uy/ldp/proveedor-rdf/tree/master*

2. Ejecutar el comando *npm install* en el mismo directorio donde quedaron los paquetes. Esto instalará todas las dependencias de librerías.

A.1.1. Configuración para R2RML

La configuración del sistema se encuentra en el archivo configuration.json que es donde se declaran todos los filtros y como se conectan entre ellos.

Para R2RML se necesitan 5 filtros:

- El primer filtro es un traductor de R2RML a una representación intermedia con la que es más fácil trabajar. Este filtro toma los mensajes desde el canal de entrada (input)

y los publica para ser procesados por un segundo filtro que toma su entrada del canal fromRDF.

- El parámetro endpoint índica un endpoint SPARQL en el que se buscaran las transformaciones R2RML.
- El parámetro log simplemente indica si se desea logear información de depuración.

```
{
  "name": "artranslator",
  "from": "input",
  "config": {
    "endpoint": "http://localhost:8890/sparql/",
    "to": "fromRDF",
    "log": true
  }
  }
```

- El segundo filtro toma la representación intermedia más la solicitud y genera una estructura de datos que tiene toda la información para luego poder generar una consulta a al base de datos. Esto lo logra especificando identificadores y/o valores que se obtienen generalmente de la URL junto con las columnas que deben actualizarse o proyectarse y nombres de tablas que se toman desde la transformación R2RML. La configuración para este filtro solamente necesita especificar dónde se publica la salida y el parámetro log para indicar si se desea loguear información de depuración.

```
{
  "name": "rdfToRel",
  "from": "fromRDF",
  "config": {
    "log": true,
    "to": "databasequery"
  }
  }
```

- El tercer filtro genera una consulta SQL que es compatible con el dialecto de MySQL. Este filtro tampoco requiere configuración. Se puede especificar si se desea loguear información y dónde se publica la salida.

```
 {
  "name": "queryGenerator",
  "from": "databasequery",
  "config": {
    "logsql": true,
    "to": "executor"
  }
```

- El cuarto filtro ejecuta la consulta SQL contra la base de datos y por lo tanto necesita una configuración para poder acceder a la base de datos.

```
{
  "name": "dbExecutor",
  "from": "executor",
  "config": {
    "name"   : <Nombre que se toma de la transformacion para saber si la
        consulta debe ejecutarse en esta base de datos>,
    "host"     : <Host donde esta la base de datos>,
    "port"   : <Puerto en el que escucha la base de datos>,
    "user"     : <Usuario de base de datos con acceso a la base
       especificada en name>,
    "password" : <Clave para el usuario>,
    "database" : <Nombre de la base de datos con la que se trabaja>,
    "to": "relToRDF"
  }
  }
```

- Finalmente, el último filtro toma el resultado de la consulta anterior y la representación intermedia de la transformación R2RML como entrada y genera un conjunto de ternas RDF como salida que se publican en el canal de salida de todo el proceso. Este filtro no necesita ninguna configuración ya que no interactúa con otros componentes.

```
{
  "name": "relToRDF",
  "from": "relToRDF",
  "config": {
    "to": ""
  }
  }
```

Para comenzar a trabajar:

1. Crear una base de datos sobre la que se quiera utilizar la aplicación. También puede utilizar una base de datos ya existente.

2. Editar el archivo *configuration.json*. Este archivo es autodescriptivo, es necesario editar configuraciones relacionadas a la base de datos y el endpoint donde enviar las consultas SPARQL.

3. Para ejecutar la aplicación:

   a) En Linux: dependiendo de la version de linux ejecutar *nodejs index.js* o *node index.js*

   b) En Windows ejecutar *node index.js*

4. Una vez hecho esto el servidor estara ejecutando y podra enviar solicitudes LDP desde el cliente REST del browser.

Para introducir comandos:

1. El servidor queda escuchando en *localhost:4000*.
2. Para poder interactuar lo primero que se debe hacer es crear algún juego de tablas con datos en la base de datos que piensa utilizar.
3. Definir los mapeos R2RML correspondientes a los datos de las tablas que se deseen transformar a RDF.
4. Levantar los mapeos R2RML para que sean accesibles en el endpoint SPARQL. En Marmotta este paso se realiza mediante la importación del archivo que contiene el mapeo.
5. Los comandos para interactuar con el wrapper se dan desde el cliente REST instalado en un browser

A.2. Resumen de los Artículos considerados en el Relevamiento de Mecanismos de Transformación a RDF para Bases de Datos Relacionales

Transient and persistent RDF views over relational databases in the context of digital repositories En este artículo [12] de 2013, se discuten los beneficios y desventajas de las dos formas de generar grafos RDF a partir de bases de datos relacionales: la materialización de grafos RDF y la traducción de consultas SPARQL a SQL para obtener datos directamente desde las bases relacionales. Este tema es examinado en el contexto de los repositorios digitales y se publican las conclusiones luego de evaluar un conjunto de pruebas. Se argumenta que en el contexto de los repositorios digitales donde las actualizaciones no son frecuentes, la generación de archivos RDF es más eficiente que la traducción en tiempo real de SPARQL a SQL para consultar bases relacionales. Para los experimentos se utilizó D2RQ y un archivo de mapeo R2RML para el caso de traducción en tiempo real y para el caso de materialización se utilizó D2RQ y un parser R2RML con el mismo archivo de mapeo que el caso anterior y el grafo generado se cargó en una instancia de Virtuoso. Como trabajo futuro se sugiere ampliar las mediciones teniendo en cuenta otras herramientas tales como Ultrawrap y otros repositorios como Eprints y triple stores como Sesame.

A survey of RDB to RDF translation approaches and tools Realiza [13] un recuento detallado de métodos y herramientas tanto académicas como comerciales para implementar RDB-to-RDF.

Es un trabajo reciente (noviembre 2013) por lo que contiene datos actualizados que pueden tomarse como referencia. En la introducción señala que si bien la definición de R2RML en setiembre de 2012 ha marcado un hito ya que permite desacoplar la integración de datos en RDB de herramientas específicas, tiene algunas limitaciones sobre los tipos de mapeo que se pueden expresar, el reuso de vocabularios o la forma en que los datos son accedidos.

Se presentan tres motivaciones para el pasaje RDB-to-RDF:

(i) Acceder datos residentes en RDB. Como los RDB ya proveen soluciones para asegurar transaccionalidad, seguridad, rendimiento, se necesitan técnicas para accederlos y convertirlos a ternas RDF.

(ii) Publicar datos como parte de la iniciativa Linking Open Data.

(iii) Integrar datos desde fuentes heterogéneas.

Este artículo presenta una clasificación de las soluciones RDB-to-RDF en cuatro aspectos:

1 - Motivación de la herramienta

1.1 Extracción de ontologías a partir del esquema relacional

1.2 Definición de un lenguaje de mapeo de propósito general

1.3 Implementación de un motor de transformación de RDB-to-RDF

2 - Tipo de mapeo

2.1 Mapeo directo (Direct Mapping)

2.2 Mapeo manual o transformador dependiente de la semántica de los datos. Dentro de esta modalidad hay dos estrategias:

2.2.1 El mapeo depende de SQL para presentar los datos

2.2.2 El mapeo se realiza a través de un lenguaje dedicado.

En particular R2RML y D2RQ usan ambas estrategias simultáneamente: complementan consultas SQL con un lenguaje de mapeo específico.

3 - Implementación del mapeo

3.1 Materialización de los datos: consiste en la transformación estática de la base de datos fuente a una representación en RDF. Este tipo de soluciones no admite grandes volúmenes de datos y debe tener en cuenta la caducidad de los datos. Para esto último debe correr procesos de extracción periódicos por lo que se trata de una solución de compromiso entre el costo de la extracción y la tolerancia a trabajar con datos desactualizados.

3.2 Mapeo a demanda: consiste en la corrida de consultas contra la base de datos relacional. En este modelo los datos permanecen en la base de datos. Los querys sobre RDF deben ser convertidos a SQL durante la ejecución de las consultas. Este enfoque es adecuado para grandes volúmenes de datos , provee datos actualizados y permite forzar el cumplimiento de políticas se seguridad de acceso a la información. Como desventaja está el riesgo de mal rendimiento en el procesamiento de las consultas, para minimizarlo se limitan las posibilidades expresivas de SPARQL.

4 - Método de recuperación de datos

4.1 Basado en querys: los datos en RDF se obtienen como resultado de una query generalmente en SPARQL. Unas pocas herramientas implementan su propio lenguaje de consulta.La consulta SPARQL puede correr contra datos en RDF si se utiliza materialización de datos o contra el RDB si se utiliza mapeo a demanda.

4.2 Linked Data: a cada entidadel cliente realiza un requerimiento HTTP GET a la URI de la base de datos y obtiene todo el grafo RDF.

Después de definir los criterios de clasificación, el artículo realiza una descripción y una tabla comparativa de las herramientas relevadas según la clasificación vista anteriormente.

Se describen 7 herramientas compatibles con R2RML: DB2Triples, OpenLink Virtuoso, Morph, RDF-RDB2RDF, Ultrawrap, XSPARQL y Oracle Spatial and Graph 12c.

y 11 herramientas que no lo implementan: Asio Semantic Bridge for Relational Databases and Automapper, D2R Server y el lenguaje D2RQ, Datalift, DB2OWL, METAmorphoses, R20 y ODEMapster, RDBToOnto,Relational.OWL, SPASQL, SquirrelRDF, Triplify.

Para terminar realiza una síntesis de los aspectos comunes y los enfoques más utilizados por las herramientas estudiadas.

El artículo finaliza con dos preguntas abiertas: la primera es que una vez que se haya poblado la web de datos, será necesario habilitar la incorporación y actualización de datos desde RDF a las bases de datos relacionales utilizando los protocolos de la web semántica.

De esta primera pregunta se desprende la segunda ya que se convierte en crítica la necesidad de explicar los resultados de las consultas y poder rastrear el origen de los datos.

La conclusión presentada es que cualquiera sea el método empleado para la transformación, el resultado es una Web de datos de solo lectura. Pronostica que esta web de datos deberá evolucionar a un modelo que permita la escritura. De la misma forma, en el caso de las bases relacionales a pesar que los mecanismos de mapeo son expresivos, se trata de una transformación unidireccional que no contempla el pasaje desde RDF a las bases originales. Finalmente observa que será necesario encontrar un punto de compromiso entre la expresividad de los mapeos de RDB a RDF y la necesidad de actualizar datos relacionales utilizando protocolos de la web semántica.

Transformation of Relational Model to RDF Model Presenta [22] un algoritmo genérico para transformar una base de datos relacional a RDF preservando la semántica. La ventaja del algoritmo propuesto radica en que el algoritmo realiza el mapeo de forma 100 % automática

sin intervención del usuario. Para esto el trabajo describe 3 algoritmos con los que identifican entidades, relaciones y finalmente la cardinalidad entre las relaciones. La generación del mapeo se realiza siguiendo los pasos de ingeniería inversa: identificación de entidades, identificación de relaciones e identificación de las cardinalidades. En la identificación de entidades se consideran como entidades a todas las tablas que contengan una clave primaria formada por un único atributo o la clave primaria está formada por más de una columna de las cuales ninguna es una clave foránea. En la identificación de relaciones distingue (i) asociaciones: si todos las columnas son parte de la clave primaria y son todas claves foráneas entonces se trata de una relación; (ii) herencia: dos tablas tienen una relación de herencia cuando tienen la misma clave primaria y en una de ellas esta clave primaria es también una clave foránea; (iii) 1 a 1: si las tablas tienen más de una clave candidata y la clave foránea de la otra tabla referencia a una de estas claves; (iv) 1 a muchos: s54re detecta cuando solamente una de las tablas está referenciando a la otra tabla.

An Extensible Approach for Mapping Relational DB to RDF Describe [19] una extensión de DB2RDF que convierte DB a RDF incluyendo información adicional que resulta de gran utilidad para motores de búsqueda. El enfoque es comparado con SPIN que permite agregar reglas a la traducción pero éstas se mezclan con la definición de la ontología haciendo complicado agregar nuevas reglas. Este enfoque simplifica la tarea de agregar nuevas reglas y también permite que diferentes usuarios definan diferentes reglas para calcular el mismo campo en base a los datos que cada uno de ellos tienen disponible. Las reglas que permite definir son de tipo inferencias, un ejemplo seria una base de datos que especifica autores de papers, si dos personas son autores del mismo paper se puede definir una regla que permita mapear que estas dos personas se conocen entre si.

R2RML Processor for Materializing RDF View of Relational Data: Algorithms and Experiments Describe [16] los algoritmos utilizados para definir un R2RML processor. Es decir, un sistema que dada una base de datos relacional, una conexión a dicha base y un documento de mapeo R2RML, genera una salida con datos de la base en formato RDF a un archivo o dump. Se trata de un trabajo relativamente reciente (noviembre 2013) por lo que se pudo contar con datos actuales que pueden tomarse como referencia para el relevamiento de artículos y software que se realizó. En la introducción realiza un buen recuento de herramientas que implementan R2RML, esta lista puede ser utilizada como base para comenzar el trabajo de evaluación de herramientas que implementan R2RML. Señala que solamente dos de las herramientas referenciadas (XSPARQL y Ultrawrap) cumplen con la totalidad de los test cases de R2RML definidos por W3C[17]. En tanto, el prototipo construido utilizando java JSE 1.7.0 y Jena tuvo un buen desempeño en todos los test. Describe los algoritmos utilizados lo que puede resultar muy útil para implementar el prototipo de R2RML processor. En cuanto a la implementación del prototipo, solamente soporta conexiones a MySQL y Microsoft SQL Server. Si bien no se menciona específicamente, se asume que cumple con la posibilidad de generar named graphs habiendo pasado los test R2RMLTC0007*, R2RMLTC0008* y R2RMLTC0009*. No se adjuntan los fuentes del prototipo que permitan instalarlo y realizar pruebas.

Ultrawrap: SPARQL execution on relational data Ultrawrap [30] genera una vista lógica de una base de datos relacional como un grafo RDF definido utilizando vistas SQL. Las

consultas se ejecutan sobre esas vistas lógicas en la base de datos original transformando SPARQL en SQL. De esta forma puede utilizar todos los algoritmos y optimizaciones de las bases de datos comerciales para ejecutar consultas SPARQL. Este enfoque permite trabajar con bases de datos instaladas sin tener que duplicar los datos en RDF, esto permite que los datos estén siempre actualizados y resguardados en la base de datos original, evitando la duplicación y la caducidad de los datos.

Ultrawrap está formado por cuatro componentes:

(i) traductor de un esquema SQL incluyendo constraints a una ontología OWL; (ii) la creación de una vista lógica representando una tabla de ternas formada por una o más vistas SQL de la base de datos; (iii) traducción de SPARQL a SQL equivalente utilizando una estructura similar a la de un compilador (parser, AST, expression tree), (iv) utilización del optimizador de la base de datos de origen para ejecución de la consulta

En el artículo se explican los componentes del software y se muestra el seudocódigo para la creación de las vistas. El desempeño de las consultas en SPARQL es comparable al de SQL corriendo sobre la representación relacional de los datos. Este desempeño no había sido logrado hasta el momento por otros sistemas RDB2RDF. Un análisis más detallado del rendimiento de las consultas revela que no son suficientes las optimizaciones implementadas al transformar SPARQL a SQL y que se requiere un diseño que complemente al optimizador de consultas de la base de datos de origen, esto último queda abierto a nuevos desarrollos. Las pruebas fueron realizadas contra las principales bases comerciales: Microsoft SQL Server 2008, IBM DB2 9.2, Oracle 11gR2 y se utilizaron los benchmarks Berlin SPARQL Benchmark (BSBM) y Barton Benchmark modificado donde los datos RDF son obtenidos a partir de bases de datos.En el apéndice se adjuntan las consultas reaalizadas. Este artículo es de agosto de 2013.

RDOTE - Publishing Relational Databases into the Semantic Web Presenta RDOTE [18] un framework para convertir una base de datos relacional a RDF. RDOTE permite mapear múltiples bases de datos relacionales a diferentes ontologías e integrarlas en un único OWL. Ofrece un mapeo automático dada una base de datos que se consigue implementando Direct Mapping según la especificación de la W3C. También es posible especificar los mapeos de forma manual mediante la definición de una ontología. El artículo explica los algoritmos utilizados para las diferentes funcionalidades ofrecidas (mapeo directo, manual, validación de ontologías, etc). La herramienta tiene soporte para un número limitado de bases de datos entre las que se encuentran MySQL y Oracle. Ofrece una interfase gráfica desde la que se pueden definir o modificar los mapeos (incluyendo los generados de forma automática). Esta herramienta se encuentra disponible de forma libre en http://sourceforge.net/projects/rdote/ Presenta comparaciones de performance contra D2RQ donde RDOTE es un poco menos performante en general pero con una diferencia constante.

An integrated framework for enhancing the semantic transformation, editing and querying of relational databases Presenta Iconomy [28], una suite que integra todas las facetas de tecnologías de la Web Semántica. Describe la principal característica de Iconomy como un mecanismo para la transformación de datos relacionales en entidades semánticas y la provisión de una interfase de usuario amigable para ver, manejar, editar y realizar querys sobre la ontología y sus instancias, soporta inferencias y tiene una interfase para incorporar restricciones y controles de consistencia. Provee un entorno gráfico que permite a usuarios

instanciar ontologías y probarlas fácil y rápidamente . El artículo es de 2010 y revisa las herramientas más comunes utilizadas en cada paso del trabajo en la Web Semántica, observa que si bien para cada fase existen herramientas maduras, hace falta un entorno integrado que cubra todas las operaciones. Iconomy soporta la incorporación de Jena o Sesame y emplea el razonador Pellet y una API de Protege para especificación de restricciones. Iconomy presenta las siguientes funcionalidades : carga de una ontología; encriptación y seguridad de las ontologías generadas por el mecanismo de transformación; desencriptación para ver y editar archivos owl; listar, recorrer y editar clases y propiedades de objetos de la ontología y de instancias específicas; listar,agregar y editar restricciones en las clases de la ontología; chequeo de consistencia de la ontología; crear querys SPARQL a través de un menú; explicación en lenguje natural de la ejecución del SPARQL y edición del contenido del código SPARQL; habilitar y deshabilitar el razonador; manejar cuentas de usuario y sus derechos sobre las ontologías. Con respecto al pasaje de datos desde RDB a una ontología, el mismo se realiza directamente a partir de una herramienta gráfica. Se implementa con APIs de Jena o Sesame y fue probado con bases de datos Oracle y MySql. La idea planteada de integrar herramientas es muy buena y tal como dice el artículo evita instalar herramientas particulares para cada paso de la incorporación de datos existentes en otros formatos a la Web Semántica y Linked Data. No hay noticias de versiones actuales de Iconomy por lo que se desconoce si se continuó el desarrollo.

Semantic querying of relational data for clinical intelligence: a semantic web services-based approach Este artículo [20] plantea un caso específico en el campo de la bioinformática, donde la mejor manera de consultar datos clínicos es mediante querys ad-hoc. Para ser económicamente viable, estas consultas deben proveerse en un formato de autoservicio. Se plantea que para que esto sea posible para usuarios no técnicos sin ayuda de programadores, la utilización de querys semánticas basadas en la aplicación de ontologías, axiomas y reglas. Las forma más básica son las consultas SPARQL a datos RDF.

Se utiliza el entorno SADI (Semantic Automated Discovery and Integration) que consiste en un conjunto de prácticas de diseño para la implementación de Web Services Semánticos. Los servicios SADI consumen datos en RDF y producen datos en RDF. Esto permite utilizar endpoints SAPRQL y razonadores OWL para posibilitar o mejorar la interacción entre los servicios. Estos se invocan pasando datos RDF al endpoint correspondiente al servicio utilizando HTTP POST y proveen datos utilizando HTTP GET. SADI impone una sola restricción al comportamiento del servicio: la URI de salida debe ser la misma que la URI de entrada. Los servicios pueden ser descubiertos automáticamente y orquestrados. Los servicios SADI operan agregando nuevas propiedades a una URI de entrada descripta en el documento RDF de entrada y esas propiedades son fijas para cada servicio, utiliza OWL para especificar condiciones en la entrada y declarar predicados en la salida. En la práctica esto tiene como resultado que los servicios puedan encadenarse unos con otros formando un flujo de transformaciones obteniendo como resultado un grafo RDF transformado a través de los servicios que utilizó.

Es posible tener un motor de SPARQL que puede responder querys descubriendo automáticamente los servicios SADI e invocándolos. En un escenario típico el usuario selecciona los predicados que necesita para su consulta en la lista de predicados provistos por los servicios SADI, así como clases y predicados en las ontologías referenciadas y utiliza estos conceptos para elaborar una consulta SPARQL y la ejecuta en un endpoint SHARE que es una implementación de SADI.

La conversión de datos de las bases de datos relacionales a RDF se realiza mediante la ejecución de consultas sql y la conversión de esos resultados a RDF.

La conclusión es que el uso de servicios SADI con una interfase SPARQL es viable pero falta más experimentación y madurez. Como desventaja, para implementar SADI se necesita programación especializada en Java o Perl lo que lo transforma en una herramienta más cara que las alternativas de mapeo declarativo como la materialización o el query rewriting.

RDB2RDF: A relational to RDF plug-in for Eclipse En este trabajo [29] de 2012 se presenta un plugin para Eclipse que soporta la conversión de datos relacionales a RDF. Este plugin tiene una arquitectura modular construida teniendo en cuenta las dos etapas del proceso de conversión: extracción de esquema y conversión de datos, de forma de encapsular por un lado el proceso estable y bien conocido como la extracción de esquema de otro proceso más cambiante y en continua evolución como la conversión de datos. Una arquitectura plugin tiene como ventaja el soporte de modularidad de grano fino, facilita la reusabilidad del código y el desarrollo de extensiones. La herramienta permite generar mapeos en R2RML y es extensible lo que permite la generación de mapeos automáticos para otras recomendaciones que puedan surgir. La arquitectura del plugin consiste en dos módulos separados. El primero es el Schema Generator que es responsable por comunicarse con la BD usando drivers de dominio público, proveer un mecanismo para definir un esquema externo que define los datos que se convertirán a RDF e importar los elementos seleccionados desde la base de datos. El segundo es el Mapper que implementa uno de varios algoritmos de mapeo, la idea detrás de esta modularización es que un cambio en el algoritmo de mapeo no afecte al resto del plugin.La implementación está dividida en cuatro paquetes principales: Mapper, Schema, Preferences y Handler.El paquete Mapper contiene clases que definen el mapeo relacional contra RDF, en particular implementa una clase R2RML. El paquete Schema tiene clases modelo que se utilizan para representar el esquema extraído del RDB, se utiliza para generar el mapeo y se pasa como parámetro a Mapper. Preferences tiene clases para la ventana de preferencias donde el usuario puede entre otras cosas elegir el algoritmo de mapeo. Handler dispara la ejecución del algoritmo seleccionado.Este plugin está disponible bajo Academic Free License.

A Semantic Query Method for Deep Web Este artículo [23] de 2013 denomina Deep Web a los datos que no son directamente accesibles por la Web, estos datos en su mayoría residen en bases de datos relacionales. Propone la idea de un acceso a datos relacionales centrado en la funcionalidad en vez de centrado en los datos como considera que son los métodos utilizados hasta ahora.Consideran que el enfoque centrado en datos tiene falencias ya que en entornos de negocios las organizaciones son más proclives a publicar funcionalidades que datos. El enfoque centrado en funcionalidades se basa en una combinación de Semantic Web Services y tecnología de acceso a datos RDF.Se implementa un Deep web semantic query service que consiste en: (1) los usuarios ingresan una consulta SPARQL a partir de una aplicación (2) la sentencia SPARQL se reescribe en SQL basado en una vista RDF (3) se accede a la base de datos con SQL (4) el resultado de la consulta se procesa para transformarlo a RDF. La vista RDF utilizada para reescribir la consulta es un descripción semántica de la parte de los datos que un usuario puede ver a través de cierta interfase denominada limited RDF view, esta vista utiliza RDF schema y se define en base a dos elementos: (1) RDB schema abstraction (2) reglas para mapear RDB a RDF. No utiliza R2RML ni ninguna otra recomendación

de la W3C para realizar el mapeo.

Algorithms for Mapping RDB Schema to RDF for Facilitating Access to Deep Web El objetivo de este trabajo [21] de 2013 es definir una metodología formal basada en heurísticas para mapear una base de datos relacional a una base de conocimiento semántico, comprendiendo no sólo los datos sino también el conocimiento específico del dominio. Los algoritmos presentados utilizan metadatos del diccionario de datos de la base relacional para construir la versión inicial del repositorio semántico al que se agrega conocimiento específico de dominio en etapa de procesamiento. Se realiza una revisión de los trabajos previos sobre este tema basándose en un trabajo de W3C del año 2009, observándose que la implementación de mapeos se puede realizar de dos maneras: estática utilizando ETL o dinámica a través de querys. La implementación estática tiene como desventaja la desactualización de los datos y la implementación dinámica tiene costo elevado en tiempo de proceso. Por otra con el mapeo automático de RDB a RDF se pierde casi toda la semántica de los datos, por lo que se concluye que los mapeos automáticos son útiles para generar un punto de partida pero debe continuar el trabajo de análisis. refinado y proceso de los datos semánticos. Considera que a los lenguajes de mapeo existentes les falta considerar el conocimiento que no siempre esta representado explícitamente, le faltan reglas y lógica para que el conocimiento pueda ser derivado, comprobado usando predicados específicos de la aplicación o generado usando procedimientos y funciones dependientes de la aplicación. Se categorizan las deficiencias de los lenguajes de mapeo existentes en 4 categorías de acuerdo a los niveles de conocimiento (1) Deficiencia en el uso de todo el conocimiento en el área de bases de datos relacionales y sus metadatos (2) Deficiencia en el uso de conocimiento del dominio de los datos como patrones de simetría, transitividad, disjunción etc. (3) Deficiencia en el uso de conocimiento específico de la aplicación como predicados, procedimientos y funciones (4) Deficiencia en el uso de reglas y lógica para emplear distintos predicados específicos de la aplicación. En este trabajo se presenta un proceso de conversión en tres etapas (1) pre-proceso donde se crea el repositorio semántico (2) proceso donde se realiza el mapeo incremental de los datos relacionales y (3) post proceso donde se modifica el repositorio semántico agregándole conocimiento específico del dominio. Después del mapeo de los datos relacionales a un repositorio semántico con RDF, se aplican reglas semánticas para seguir trabajando con los datos del dominio. El área de conocimiento de bases de datos relacionales se utiliza para definir tablas, columnas, restricciones y tipos de datos. El conocimiento del dominio de los datos se usa para representar los datos en el esquema RDF, se buscan relaciones transitivas, simétricas, de estrella, conjuntos disjuntos, etc. El conocimiento de la aplicación específica se usa para evaluar y refinar el mapeo. Estos tres niveles de conocimiento se usan para construir los algoritmos de mapeo de RDB a RDF. Los algoritmos se implementaron para un caso específico de datos espaciales. Se desarrolló una aplicación interactiva en Java para ejecutar los procedimientos de mapeo y el repositorio resultante se cargó en Protégé para continuar refinando la ontología.

Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation using Morph Este trabajo [31] propone una extensión de la propuesta de Chebotko [49] agregando la posibilidad de incorporar mapeos en R2RML al proceso de reescritura de querys SPARQL a SQL. Chebotko presentó en 2009 un trabajo muy detallado sobre reescritura de querys para trabajar con triplestores residentes en RDBMS. En este trabajo de 2014 se pre-

senta la posibilidad de que las querys SPARQL contra datasets virtuales, se traduzcan a SQL considerando mapeos en R2RML para que puedan ser evaluadas contra un motor de base de datos relacional sin necesidad de tener datos en RDF. En el documento se observa que la especificación R2RML no provee una formalización del proceso de transformación de SPARQL a SQL y describe diversos trabajos que intentaron realizar esta formalización, pero ninguno logró cubrir la totalidad de posibles casos que permite la especificación, además de esto la performance de las querys resultantes no ha sido satisfactoria en general. Explican que el principal motivo para la ineficiencia de las querys es que no han sido suficientemente optimizadas para trabajar con el motor de base de datos de destino. Se describe brevemente la formalización de Chebotko y la extensión propuesta para que todas las funciones y mapeos usados en los algoritmos presentados tomen en cuenta los mapeos R2RML definidos. Las querys traducidas además se optimizan eliminando non-correlated subquerys y self-joins que pueden generarse en la transformación. Estas transformaciones se evaluaron contra benchmarks(BSBM SQL 4 y BSM SQL 5) y contra casos reales tales como los proyectos BizkaiSense, proyecto para medir variables ambientales por medio de sensores, REPENER proyecto que provee acceso a información sobre energía usando tecnología semántica llamada SEiS y el proyecto Integrate para compartir e integrar datos clínicos usando HL7-RIM, todos estos proyectos utilizan comúnmente querys SPARQL que fueron evaluadas. En todas las pruebas el desempeño de las querys fue similar al de las querys en SQL nativo y mejor al de todos los traductores conocidos. Como futura mejora se propone la utilización de Semantic Query Optimization (SQO) que puede introducir mejoras en las querys de tal forma que sean más eficientes que las querys SQL nativas.

morph-LDP: An R2RML-based Linked Data Platform implementation Se presenta morph-LDP [32] un sistema que combinando la recomendación Linked Data Platform (LDP) y R2RML, permite exponer datos relacionales como Linked Data y posibilita su lectura y escritura por parte de aplicaciones que utilicen el protocolo LDP. En la demo se muestra como se puede utilizar R2RML para posibilitar que bases de datos relacionales puedan ser expuestas como Linked Data con posibilidad de lectura y escritura de datos. Estos datos son desreferenciables y quedan disponibles a través de HTTP GET, además pueden ser actualizados utilizando HTTP PUT, POST, PATCH y DELETE. Si bien existen trabajos previos (por ejemplo Garrote [44] los datos generados para Linked Data no pueden ser desreferenciados usando HTTP GET y solamente pueden ser accedidos a través de un endpoint SPARQL. En contraste los datos Linked DATA generados por morph-LDP pueden ser derreferenciados y actualizados usando URIs conforme al protocolo LDP y no se requiere la utilización de SPARQL por los clientes que acceden a los datos. Se explica como los conceptos de LDP se mapean a R2RML: las URI especificadas como SubjectMaps en R2RML se mapean a recursos LDP (LDPR) permitiendo acceso de lectura y escritura a través del protocolo LDP este proceso cumple con el segundo principio de Linked Data. PredicateObjetctMaps permite generar la información específica sobre el recurso de forma que cuando se accede a la URI, morph-LDP provee información usando RDF, cumpliendo el tercer principio de Linked Data. En R2RML se utiliza TripleMaps para generar las ternas RDF a partir de las filas de tablas lógicas, esas tablas y sus correspondientes TripleMaps se mapean a Containers LDP. Para la implementación se utilizan dos resultados de proyectos anteriores morph-RDB y LDP4j. morph-RDB es un motor RDB2RDF basado en Scala, es compatible con R2RML y realiza una traducción optimizada de querys SPARQL a SQL. LDP4j es un middleware basado en Java para desarrollo de aplicaciones compatibles con

LDP. morph-LDP extiende morph-RDB con una capa LDP provista por LDP4j. Funciona de la siguiente forma: la capa LDP extrae los metadatos del requerimiento HTTP y lo envía como entrada a morph-RDB que tiene dos modos de operación, como API o query SPARQL y realiza la transformación entre RDF y los datos relacionales. El video conteniendo la demostración y las sentencias y respuestas HTTP junto con las consultas SPARQL y SQL utilizadas se pueden ver en la página de morph-LDP.

Creating Semantic Data from Relational Database Este trabajo de 2013 presenta OntoURI [24] que convierte datos desde RDBMS a instancias OWL. OntoURI toma como entrada (o asigna si no viene especificada) una URI para la base relacional a y que se utiliza en la identificación de entidades.El proceso utiliza una regla de mapeo entre el esquema de base de datos y el esquema de la ontología, una regla de identificación para identificar entidades, text mining para extraer las ternas semánticas y los datos correspondientes a authority de la URI para soportar la creación automática de instancias. También permite extraer datos desde campos BLOB y transformarlos en datos RDF. Este software tiene una GUI que soporta que un operador genere el mapeo. Se pone énfasis en la asignación de URIs para evitar por ejemplo, que a un mismo objeto se le asignen diferentes URIs durante el proceso de conversión de datos. No utiliza las recomendaciones de W3C, DM ni R2RML.

Accessing relational data on the web with sparqlmap Se presenta SparqlMap [25] un software que reescribe SPARQL a SQL. El objetivo es permitir la ejecucuón de consultas SPARQL en una base de datos relacional. El mapeo se realiza de tal manera que se genere una única query SQL correspondiente a la consulta original. El mapeo se basa en R2RML pero con limitaciones, que por ejemplo R2RML permite múltiples predicado-objeto para un mismo sujeto y esa especificación no está soportada. El proceso de reescritura de la consulta consta de 3 pasos (1) Selección de candidatos para el mapeo, es el proceso de identificar las partes del grafo mapeado que intervienen en la consulta (2) traducción de la consulta, genera la consulta SQL usando las correspondencias detectadas en el paso anterior (3) ejecución de la consulta sql en la base de datos y el resultado se mapea al resultado de la query SPARQL origina. Este trabajo es de 2013. Para la implementación se utiliza ARQ para parseo de SPARQL y Java. La implementación está disponible bajo licencia open-source en la página del proyecto.

Capturing Relational Schemas and Functional Dependencies in RDFS Este artículo [43] de 2014 propone una extensión de Direct Mapping para transferir información semántica desde el esquema relacional a RDF, por ejemplo dependencias funcionales. Se utiliza lógica descriptiva para realizar la extensión utilizando una extensión RDFS de DL-Lite. Se trata de un trabajo fundamentalmente teórico que excede el alcance del proyecto Proveedor RDF.

RDB2RDF: completed transformation from relational database into RDF ontology En este trabajo [16] de 2014 se presenta RDB2RDF, un método de transformación de todas las tablas de una base de datos relacional a una ontología RDF. La transformación es reversible y permite volver desde RDF a tablas relacionales. Todos los pasos se realizan automáticamente sin necesidad de intervención de los usuarios. Se utiliza el formato RDF/XML para almacenar los resultados con el argumento de que este formato posibilita utilizar las herramientas que manejan XML. El principal objetivo del método es permitir mapeos flexibes de estructuras relacionales complejas a RDF sin cambiar la base de datos existente. La flexibilidad se logra

empleando sentencias SQL directamente en los pasos de la transformación, luego se agrupan los resultados y los datos se mapean a ternas RDF. La transformación lleva registro de los atributos clave en las tablas relacionales por lo que el algoritmo puede extenderse para revertir la transformación. Se menciona entre otros trabajos relacionados a RDB2RDF: A relational to RDF plug-in for Eclipse [29] se argumenta que en ese trabajo al utilizar R2RML y DM no se considera la semántica de RDFS ignorando propiedades como rdfs:subClassof, rdfs:subPropertyof, rdfs:domain, rdfs:range. Pasos del proceso: (1) describir todos los atributos en la base de datos, la descripción se envía a un archivo de texto (2) utilizando SELECT se extraen los datos que describen un recurso, esto se realiza partiendo de una tabla generada a partir de una entidad del modelo de datos, debe tener por lo menos un atributo clave que se usa para generar la URI del recurso (3) generar un archivo RDFS basado en la descripción del paso 1 y los atributos del paso 2, cada atributo tendrá un dominio: nombre de la tabla y un rango que puede ser un tipo de datos en XML Schema para un atributo normal, el nombre de la tabla para la clave primaria, el nombre de la tabla referenciada para fk (4) ejecutar una consulta con SELECT para extraer las instancias desde la base y almacenarlas en formato XML (5) generar un dataset RDF a partir de RDFS y el archivo XML con los datos. RDB2RDF se implemetó utilizando C y .Net2.0 y se probó con SQLServer 2012. Genera la transformación automática de toda la base de datos o permite especificar la información que se necesita a través de SQL. La salida genera dos archivos un archivo RDFS que describe las clases y las relaciones entre clases y propiedades y un archivo RDF con los datos en formato XML. Como trabajo futuro se plantea generar la transformación a OWL.

StdTrip+ K: Design Rationale in the RDB - to - RDF process Este artículo de 2013 presenta StdTrip+K [26] un proceso que integra DR (design rationale) junto a la estrategia de triplificación. Se denomina design rationale (DR) a las desiciones que se toman durante el proceso de diseño, las opciones que se rechazan y las que se aceptan y los criterios utilizados. En ese caso, por analogía, el artículo se refiere a triplification rationale. Los datos de diseño son valiosos para soportar el diseño de nuevas ontologías y ayudan al reuso de datasets. Los autores no han encontrado ningún trabajo previo que utlice DR en el dominio de Linked Data. En StdTrip+K se realiza el registro de la toma de desiciones de diseño con el enfoque Kuaba a través de la utilización del vocabulario Kuaba+W en el proceso de triplificación. El proceso StdTrip+K recibe como entrada, la base de datos, los metamodelos (p.ej. ER y RDF son metamodelos) y el vocabulario Kuaba+W. En cada fase del proceso se registran las desiciones de diseño con el vocabulario Kuaba+W. Como resultado final se obtiene el archivo de mapeo RDB-to-RDF, una ontología OWL y el DR resultante de todo el proceso. El vocabulario Kuaba+W extiende el enfoque Kuaba agregando elementos tales como Description relacionado con Justification y contiene información de las razones por las que un experto de dominio ha aceptado o rechazado una idea. Las fases del proceso son (1) RDB-to-ER (2) ER-to-OWL (3) mapeo entre la ontología intermedia obtenida en el paso anterior y vocabularios RDF estándar (4) selección de los términos de las fases previas para construir la ontología final, para ejecutar esta fase se requiere intervención de los usuarios con conocimiento del dominio de la aplicación

Mapping Relational Databases into OWL Ontology En este artículo [27] de diciembre de 2013 se presenta un prototipo para crear una ontología desde una base de datos relacional. En el proceso se extrae metadata del esquema relacional, se transforma en un modelo

canónico para facilitar la migración, se genera la estructura de la ontología en OWL, se valida y se refina la ontología, el grafo RDF resultante se guarda en un documento OWL. La ontología se contruye de acuerdo a un conjunto de reglas, como por ejemplo:(1)Cada relación en la base relacional se mapea a una clase si la relación no es binaria (2) Para cada relacion los atributos se mapean a la propiedad data-type (3) las tuplas de una relación se trasladan a instancias aplicando las reglas de mapeo etc. Para demostrar la validez de la propuesta se realizó un prototipo que genera un grafo RDF a partir de una base relacional automáticamente. Este prototipo se desarrolló en Java 1.6 con API Jena y MySQL para la parte relacional.

Engineering ontology-based access to real-world data sources Este trabajo [36] describe la utilización de un método basado en Ingeniería de Software para la construcción de un procedimiento para la transformación de datos del mundo real a ontologías de calidad. Para los datos transformados se utiliza el formato RDF. El artículo da cuenta de la complejidad de la transformación y la necesidad de utilizar diversas herramientas para las distintas etapas de la transformación. Propone una arquitectura que ordena los componentes y las configuraciones utilizadas a lo largo del proceso de transformación. Se presenta un caso de estudio utilizan distintas metodologías dependiendo de los datos. Por ejemplo, se materializan algunos datos generados en tripestores mientras otros se generan dinámicamente. La adopción de R2RML para este caso está planeada para el futuro.

Exposing bibliographic information as linked open data using standards-based mappings: methodology and results En este artículo [33] se describe la generación de Linked Open Data a partir de un repositorio digital conteniendo información bibliográfica .Los principales beneficios resultantes de la exposición de los datos han sido: (i) la facilidad de integración con datos de terceras partes; (ii) la expresividad ganada en la descripción de la información y (iii) la facilidad de consulta. La información se exporta periódicamente a RDF a partir de repositorios. Se utilizan mapeos R2RML para esta transformación. Se documentan algunas dificultades técnicas encontradas tales como la necesidad de codificar manualmente utilizando un editor de texto común los mapeos R2RML. Este proceso necesitó un especialista ya que no hay forma de asegurar la correctitud de los mapeos sino hasta que se produce el grafo resultante de la transformación.

Constitute: the world's constitutions to read, search and compare Presenta Constitute [34] que provee acceso para búsquedas a las constituciones de los países del mundo. Esta iniciativa utiliza tecnologías de la Web Semántica para posibilitar la búsqueda, lectura y comparación entre constituciones de distintos países.En la implementación los textos se convierten a RDF utilizando mapeos R2RML y Ultrawrap (Capesenta). Los textos se pasan primero a una planilla excel y desde allí a una base MySQL. A partir de la representación tabular de las constituciones estas se convierten a formato RDF utilizando una combinación de DM, R2RML y Ultrawrap para integrarse a una ontología OWL. Esta ontología permite navegación, búsqueda de texto y aplicación de razonamiento. Entre las opciones de diseño se eligió la representación RDF sobre XML, por tres razones básicas: (i) la consistencia en la sintaxis debido a que la libertad en el formato XML se vió como un factor complicaba la integración;, (ii)la flexibilidad de RDF y la facilidad para enriquecer los datos; (iii) la posibilidad de conectarse con otros datos abiertos a partir de Linked Data.

Efficient SPARQL-to-SQL with R2RML mappings Este artículo [11] publicado en 2015 presenta una formalización de la traducción de SPARQL a SQL utitizando mapeos R2RML. Formaliza la generación de querys SQL eficientes a través de la adaptación de técnicas de optimización de programación lógica combinadas con técnicas de optimización de SQL. También presenta una formalización basada en reglas para los mapeos R2RML que se puede integrar para soportar mapeos a esquemas arbitrarios de base de datos. En el artículo se discuten aquellos aspectos de SQL que deben tenerse en cuenta en la traducción de SPARQL a SQL para evitar problemas en la performance de las querys generadas.Finalmente se compara el software ONTOP con otros sistemas RDB2RDF, para mostrar las mejoras en el desempeño de ONTOP basados en las técnicas presentadas.

LODHub - A platform for sharing and integrated processing of linked open data En este artículo [35] se discute la necesidad de una plataforma que combine las soluciones existentes para publicar y compartir Linked Open Data con la infraestructura de servicios para explorar, procesar y analizar datos provenientes de múltiples fuentes. La idea de la plataforma es combinar las funcionalidades de proyectos existentes como Datahub y SPARQL endpoints. Entre las tecnologías utilizadas se incluye a R2RML para las transformaciones de RDB a RDF. Se busca que los usuarios puedan manejar sus datos abiertos sin tener que comprar, configurar y mantener la infraestructura y las herramientas necesarias. El objetivo es que los usuarios puedan publicar datos abiertos y hacerlos accesibles.

R2BA - Rationalizing R2RML Mapping by Assertion Este trabajo [37] propone un método semi-automático para definir mapeos R2RML que incluyan DR (*design rationale*). Identifica un problema en la dificultad de los usuarios para comprender la transformación que sufren los datos que se publicarán en RDF a partir de la especificación de los mapeos R2RML. Para mejorar esta situación, propone un método semi-automático que extiende la especificación de mapeos R2RML para incluír DR *Design Rationale* . Con esta modificación se pretende ayudar a quienes publican datos a documentar el proceso de diseño y a los usuarios finales a comprender mejor los datos publicados. También propone utilizar el DR capturado para enriquecer la representación en RDF de los datos originales, así mediante uso de algoritmos se puedan encontrar otros vocabularios existentes de forma de promover la interoperabilidad.

MIRROR: Automatic R2RML Mapping Generation from Relational Databases Describe el sistema MIRROR [38] que genera automáticamente dos conjuntos de especificaciones R2RML. Un primer grupo de especificaciones genera un conjunto de ternas R2RML con el mismo formato que hubiera generado un procesador de la especificación DM (*Direct Mapping*) pero generando diferentes URIs. Este formato permite que el procesador R2RML se comporte como un procesador DM. El segundo grupo de especificaciones se generan utilizando conocimiento implícito en el esquema de la base relacional que se está utilizando para mejorar la calidad del mapeo. En este trabajo se demuestra el comportamiento de MIRROR utilizando los casos de prueba de *W3C DM Test Case* y una versión extendida de una de las bases de datos de prueba.

RDF Mapping Language (RML) Esta iniciativa [39] de 2014 propone una extensión de R2RML denominada RML que generaliza la especificación a fuentes de datos heterogéneas tales como archivos en xml, json o csv. RML (*RDF Mapping language*) es un lenguaje definido

para expresar reglas de mapeo entre estructuras de datos heterogéneas y serializaciones de RDF. RML contiene al estándar R2RML y extiende su aplicación agregando soporte a otros formatos estructurados. RML respeta exactamente la misma sintaxis que R2RML; por lo tanto los mapeos RML son también grafos RDF. Como funcionalidad interesante permite integrar fuentes generando ternas RDF a partir de varias fuentes combinadas. Esta propuesta es un borrador público de trabajo, no es oficial y no ha sido adoptada aún como posible estándar por los organismos competentes como la W3C.

KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources La motivación para este trabajo de 2015 [40] es que la integración de datos es difícil sin una limpieza previa de los datos y sin el agregado de anotaciones semánticas. Declara que R2RML y RML no son fácilmente extensibles ni escalables y no proveen facilidades para realizar una limpieza de los datos de las fuentes. Presenta una interpretación alternativa de R2RML y lo complementa con un procesador R2RML extendido para cualquier tipo de fuente (*source-agnostic R2RML processor*) y que además soporta procesos de limpieza y transformación de datos. Permite múltiples formatos de entrada y de salida. Para construir esta herramienta se utilizó el *Nested Relational Model* (NRM) como representación intermedia de los datos. El NRM permitió abstraerse de los distintos formatos, así que una vez que se traducen los datos de entrada a la representación intermedia todo el resto del proceso: limpieza, transformación, generación de RDF es el mismo independientemente de la fuente. Agregar un nuevo formato de entrada consiste en definir como parsear y traducir los datos a NRM. La implementación se realizó utilizando Karma, una herramienta de código abierto. KR2RML está disponible bajo licencia Apache 2.0 y ha sido embebida en Apache Hadoop y Apache Storm para generar millones de ternas RDF y millones de documentos JSON procesando por lotes (batch) y en modo streaming y puede extenderse para procesar cualquier formato jerárquico.

A.3. Ejemplos LDPR y LDP Containers

Esta sección es un resumen basado en [9][10]

A.3.1. Generalidades

Linked Data Platform 1.0 [6] define una arquitectura para leer-grabar Linked Data basada en accesos HTTP a recursos web descriptos en RDF. Es decir que propone una forma de trabajar con recursos RDF puros casi como si fueran páginas web.

LDP define un conjunto estándar de técnicas usadas para crear clientes y servidores que trabajan con recursos RDF usando HTTP. El alcance de la especificación LDP 1.0 es intencionalmente acotado, se espera definir nuevas especificaciones a partir de esta base. Por ejemplo, el Linked Data Platform Working Group siguió trabajando sobre técnicas de paginado para grandes grupos de recursos y también trabajan sobre control de acceso.

Un LDP es cualquier cliente, servidor o combinación de ambos que cumplen completamente o en gran parte la especificación LDP. LDP permite que los recursos se manejen usando métodos HTTP (GET, POST, etc.)

Un recurso es algo que puede ser representado en RDF o no. Por ejemplo, un archivo binario que no tiene representación en RDF. Los recursos LDP deben tener una URI que los identifica. En el estándar LDP sección 4.1 dice: Linked Data Platform Resources (LDPRs) son recursos HTTP que cumplen las convenciones definidas en el protocolo LDPR, entonces es un subconjunto de los recursos HTTP. En la rfc 2616 (sec 1.3) se encuentra la definición de recurso HTTP: es un objeto (dato o servicio) de la red que puede ser identificado mediante una URI.

Cuando los recursos son manejados por un LDP, se denominan Linked Data Platform Resource (LDPR), pero se distingue entre Linked Data Platform RDF Source (LDP-RS) y Linked Data Platform Non-RDF Source (LDP-NR). Se define un tipo especial de recurso llamado Linked Data Platform Container (LDPC).

LDPC es una especialización de LDP-RS representando una colección de links a LDPRs o recursos de información (information resources) que responden a los requerimientos de clientes para creación, modificación o enumeración de los miembros y documentos linkeados.

En esta especificación se define cómo trabajar con contenedores. Los recursos pueden agregarse a los contenedores usando operaciones HTTP como por por ejemplo POST. Durante la creación de un recurso, éste se agrega a un contenedor y se crea un link (containment link) entre el nuevo recurso y su correspondiente contenedor.

A.3.2. LDP Containers

Los contenedores (containers) son útiles para trabajar con conjuntos de recursos con las mismas características. Por ejemplo, un cliente puede acceder a la URI de un contenedor en un LDP de alguna universidad para listar los cursos de una carrera y puede acceder a otro contenedor para listar los docentes y a su vez cada docente puede ser un contenedor de los cursos que dicta. De esta forma los contenedores actúan como un pegamento para armar el modelo y también como puntos de acceso para manejar el modelo a través de HTTP. Aquellos familiarizados con sitios web estáticos con carpetas y archivos HTML pueden imaginar que un contenedor sería equivalente a una carpeta.

Un Contenedor LDP contiene links a recursos. Básicamente representa una colección de recursos. Un contenedor responde a los requerimientos de un cliente para creación, modificación, y/o enumeración de sus miembros.

La URI que identifica a un contenedor es un punto de interacción HTTP que permite manejar a sus miembros y a la membresía. Un contenedor es también un Linked Data Platform RDF Resource, por lo que además de tener una función específica como controlador de membresía, contiene datos para los agentes que lo acceden tal como sus propiedades específicas como recurso.

Basic Containers

Un *BasicContainer* es de tipo *rdf:type http://www.w3.org/ns/ldp#Container* y más específicamente tiene el tipo *http://www.w3.org/ns/ldp#BasicContainer* (a ldp:Container, ldp:BasicContainer).

La propiedad *ldp:contains*, define el conjunto de ternas contenidas en el contenedor, que son documentos creados por el contenedor. Según la terminología LDP se denominan *containment triples* y tienen la siguiente forma: $< LDPCURI >< ldp : contains >< document - URI >$

Direct Containers

Un *Direct Container* es de tipo *rdf:type http://www.w3.org/ns/ldp#Container* y más específicamente tiene el tipo *http://www.w3.org/ns/ldp#DirectContainer* (a ldp:Container, ldp:DirectContainer).

Es más flexible que el *Basic Container* porque sus *membership triples* pueden referir a otros recursos además de sí mismo como contenedor. Un *Direct Container* puede actuar como un punto de interacción HTTP para manejar la membresía sobre otro recurso además del propio contenedor.

Un *Direct Container* se define con dos propiedades adicionales del 'ldp' namespace:

ldp:membershipResource – es el recurso cuyos miembros maneja el contenedor. El contenedor maneja la membresía a este recurso.

ldp:hasMemberRelation – define el predicado de membresía que es la propiedad de un *membership resource* que se usará como predicado de todas las triples correspondientes a los miembros del contenedor.

Uno de los usos más comunes de *Direct Containers* es cuando se necesita manejar distintas facetas de un recurso utilizando múltiples contenedores.

Un *Direct Container* va a tener solamente *containment triples* de aquellos recursos que fueron creados por un POST hacia el.

Indirect Containers

Un *Indirect Container* tiene tipo *rdf:type of http://www.w3.org/ns/ldp#Container* y más específicamente, es de tipo *http://www.w3.org/ns/ldp#IndirectContainer* (a ldp:Container, ldp:IndirectContainer).

Los *Indirect Containers* son similares a los *Direct Containers*, excepto porque los *Indirect Containers* tienen la capacidad de tener miembros cuyas URIs se basan en el contenido de los documentos miembros en vez de en las URIs de esos documentos.

Esto se logra agregando otra propiedad al contenedor *ldp:insertedContentRelation*, define un predicado que puede existir en recursos posteados al contenedor. Cuando el predicado

definido existe en el contenido posteado, el objeto de esa terna se usa para definir al miembro en lugar de la URI que se utiliza en general al agregar nuevos recursos.

A.3.3. Resumen

1. LDPRs son recursos HTTP que pueden ser creados, modificados, borrados y leídos usando los métodos estándar de HTTP (como POST, PUT/PATCH, DELETE, GET).
2. LDPRs usan RDF para definir su estado.
3. Se puede pedir la representación en Turtle de un LDPR y posiblemente otras (ej.., XML/RDF).
4. Los clientes clients usan Optimistic Collision Detection en el update (etags).
5. LDPCs son LDPRs.
6. Los clientes pueden recuperar la lista de recursos miembros de un LDPC usando GET.
7. Los recursos nuevos se crean POSTeando a un LDPC.
8. Cualquier recurso puede ser POSTeado a un LDPC, no tiene porque ser LDPR (por ejemlo, LDP-NR y LDPCs).
9. Después de POSTear un nuevo recurso a un LDPC, el nuevo recurso aparecerá como miembro hasta que sea dado de baja.
10. Los clientes pueden recuperar información sobre un LDPC sin tener que recuperar todo su contenido, incluído sus miembros.
11. Cuando se deletea un LDPC el servidor puede borrar todos los recursos que son miembros.

A.4. Ejemplos

A.4.1. LDP BasicContainer

GET lista los recursos existentes en el contenedor (No se incluyen los headers HTTP para aportar mayor claridad del ejemplo)

Pedido:

```
GET /container1 HTTP/1.1
Host: example.org
Accept: text/turtle
```

Respuesta:

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
<http://example.org/container1>
a ldp:BasicContainer;
dcterms:title "A very simple container";
ldp:contains
```

```
<http://example.org/container1/member1>,
<http://example.org/container1/member2>,
<http://example.org/container1/member3>.
```

En este ejemplo container1 está en: http://example.org y contiene : member1 member2 member3

POST crea un nuevo recurso

Pedido:

```
POST /container1 HTTP/1.1
Host: example.org
Content-type: text/turtle
Content-length: 324
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix o: <http://example.org/ontology/>.
<>
a o:Stock; dcterms:title ACME Co.; o:value 100.00.
```

Respuesta:

```
HTTP/1.1 201 CREATED
Content-Location: http://example.org/container1/member4
```

En este ejemplo container1 está en: http://example.org contiene : member1 member2 member3 + member4

GET devuelve la lista de miembros actualizada

Pedido:

```
GET /container1 HTTP/1.1
Host: example.org Accept: text/turtle
```

Respuesta:

```
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix ldp: <http://www.w3.org/ns/ldp#>.
<http://example.org/container1>
a ldp:BasicContainer;
dcterms:title "A very simple container";
ldp:contains
<http://example.org/container1/member1>,
<http://example.org/container1/member2>,
<http://example.org/container1/member3>,
<http://example.org/container1/member4>.
```

A.4.2. LDP DirectContainer

```
<http://example.org/netWorth/nw1>
a o:NetWorth;
o:asset
<http://example.org/netWorth/nw1/assetContainer/a1>,
<http://example.org/netWorth/nw1/assetContainer/a2>.
```

```
<http://example.org/netWorth/nw1/assetContainer>
a ldp:DirectContainer;
dcterms:title "The assets of JohnZSmith";
ldp:membershipResource <http://example.org/netWorth/nw1>;
ldp:hasMemberRelation o:asset.
```

- Los miembros están asociados con un recurso que no es el contenedor.
- El predicado de membresía es específico del dominio.
- Las relaciones pueden tomar dos formas diferentes:

```
<membershipResource> <ldp:hasMemberRelation> <member>
o
<member> <ldp:isMemberOfRelation> <membershipResource>
```

```
<http://example.org/netWorth/nw1>
a o:NetWorth;
o:asset
<http://example.org/netWorth/nw1/assetContainer/a1>,
<http://example.org/netWorth/nw1/assetContainer/a2>,
o:liability
<http://example.org/netWorth/nw1/liabilityContainer/l1>.
```

```
<http://example.org/netWorth/nw1/assetContainer>
a ldp:DirectContainer;
dcterms:title "The assets of JohnZSmith";
ldp:membershipResource <http://example.org/netWorth/nw1>;
ldp:hasMemberRelation o:asset.
```

```
<http://example.org/netWorth/nw1/liabilityContainer>
a ldp:DirectContainer;
dcterms:title "The liabilities of JohnZSmith";
ldp:membershipResource <http://example.org/netWorth/nw1>;
ldp:hasMemberRelation o:liability.
```

Se pueden definir varios contenedores con el mismo recurso (ej., assets, liabilities)

A.4.3. IndirectContainer

```
<http://example.org/netWorth/nw1>
a o:NetWorth;
o:advisor
<advisorContainer/bob#me>,
<advisorContainer/marsha#me>.
```

```
<http://example.org/advisors/>
a ldp:IndirectContainer;
dcterms:title "The asset advisors of JohnZSmith";
ldp:membershipResource <http://example.org/netWorth/nw1>;
ldp:hasMemberRelation o:advisor;
ldp:insertedContentRelation foaf:primaryTopic;
ldp:contains
<advisorContainer/bob>,
<advisorContainer/marsha>.
```

```
<http://example.org/advisorContainer/bob>
foaf:primaryTopic #me.
#me a foaf:Person,
foaf:name "Bob Marlow".
```

Soporta listar recursos *non-information* como miembros

A.4.4. LDP Non-RDF Sources (Binary Resources)

```
POST /attachments/ HTTP/1.1
Host: example.org
Content-type: image/png
Content-length: 1048
Slug: myimage
binary content not displayed]
```

Respuesta:

```
HTTP/1.1 201 CREATED
Content-Location: http://example.org/attachments/myimage
Link: <http://example.org/mycontainer/myimage-info>;
rel=describedby
```

Un recurso No-RDF se crea al POSTear a un contenedor.

Como resultado el servidor puede crear dos recursos:

- Un LDP Non-RDF Source que se agrega como miembro
- Un LDP RDF Source que describe el LDP-NR

En este ejemplo attachments está en: http://example.org y contiene : member1 member2 member3 + myimage

Ejemplos extraídos de:

W3C Linked Data Platform Arnaud J Le Hors, IBM Linked Data Standards Lead lehors@us.ibm.com Linked Data Platform 10 April 2014 © 2014 IBM Corporation

http://es.slideshare.net/alehors/www2014-linked-data-platform-20140410-33367843

A.5. Prototipo y prueba de concepto de procesador R2RML en Python

A.5.1. Prototipo R2RML

El objetivo de construir este prototipo ha sido mejorar la comprensión del estándar R2RML.

Algunos puntos a destacar sobre la construcción de este prototipo son la complejidad del algoritmo, las herramientas ofrecidas en este momento y la complejidad de los estándares utilizados.

La complejidad de implementar un algoritmo para realizar este tipo de conversiones tiene una dependencia muy fuerte con las posibilidades que ofrecen las librerías para trabajar con RDF. Existe una amplia variedad de librerías para trabajar con RDF que se encuentran disponibles para la mayoría de los lenguajes. Se decide utilizar RDFLib debido a que es una de las librerías más mencionadas en la web para utilizar con Python, lenguaje que se decide utilizar por su simplicidad y gran productividad. Las funcionalidades ofrecidas por esta libreria fueron de gran ayuda a la hora de implementar debido a que no solamente permite trabajar con RDF sino que es capaz de interpretar consultas en SPARQL.

Diseño

Uno de los objetivos es que el diseño sea lo más simple posible, pero al mismo tiempo suficientemente flexible como para permitir trabajar con diferentes bases de datos.

Para mejorar la calidad del código y la flexibilidad de la implementación se realizó una división de responsabilidades en acceso a datos por una parte y la transformación totalmente abstraída de la capa de datos por otra.

Se logró independencia del motor de base de datos mediante la aplicación de los patrones Strategy y DAO. Por un lado se tiene una jerarquía de DAOs los cuales siguen una interfaz definida. La transformación R2RML puede ser configurada con cualquier DAO al momento de ser creada lo que le permite cambiar la base de datos a la que esta accediendo sin verse afectada. Para lograr una independencia total a la implementación de base de datos, cada uno de estos DAO ofrece los datos necesarios para cualquier paso de la transformación y el tipo de retorno es un RecordSet cuyos elementos son n-tuplas de Python.

Para los casos de prueba se utilizó una base de datos en memoria lo que permitió realizar las pruebas sin necesidad de crear o instalar una base de datos real.

Resultados

Se logró desarrollar una implementación de la transformación R2RML. La implementación es capaz de mapear tablas y relaciones 1:N de una base de datos a RDF siempre que se cuente con una implementación del DAO para el motor de base de datos especifico.

Las pruebas realizadas para probar la correctitud del algoritmo se realizaron en base a los ejemplos presentados en la especificación en la W3C. Los resultados obtenidos de la transformación fueron comparados con los resultados esperados en la W3C obteniendo los resultados esperados.

Trabajo a futuro

Este prototipo no cuenta con la capacidad para realizar transformaciones de relaciones M:N las cuales deberían ser agregadas.

Actualmente la implementación mantiene el grafo RDF que est siendo generado totalmente en memoria hasta la finalización del algoritmo. Esto implica que la implementación actual no es capaz de realizar la transformación de una base de datos de un tamaño cercano o mayor a la memoria disponible del equipo donde ejecuta. Esta limitación podría eliminarse realizando una serialización parcial de los datos que van siendo generados.

Conclusiones

La implementación de R2RML permitió tener una buena idea de la complejidad del algoritmo así como lo beneficioso de utilizar las librerías apropiadas para su desarrollo. Un punto muy importante a destacar es lo mucho que simplificó la implementación contar con la posibilidad de ejecutar consultas SPARQL sobre la especificación de la transformación. Esto permitió que el algoritmo fuera tan simple como iterar sobre las n-tuplas resultantes de la consulta donde cada una de estas se asocia a la generación de una nueva terna en RDF.