
Enriquecimiento automático de un Data Lake con metadatos

Trabajo Final de Grado

Autor:
Enrique González García

Director:
Oscar Romero Moral

Co-director:
Víctor Herrero Ojal

Ponente:
Alberto Abelló Gamazo

21 de enero de 2016

Grado en Ingeniería Informática
Tecnologías de la Información

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Resumen

La importancia de la información crece a cada día que pasa. Aquellos que poseen una gran cantidad de información quieren obtener beneficio de ella utilizando las técnicas apropiadas.

Este proyecto se desarrolló como parte de otro mayor, a petición de un cliente que posee una gran cantidad de información. Cliente que demandó una plataforma que pudiera gestionar esta gran cantidad de datos y sacar partido de su información. El objetivo de este proyecto es dotar de cierta independencia a esta plataforma en base a información sobre los datos del cliente.

Se ha puesto en marcha un repositorio de metadatos semántico que contiene una ontología con toda la información relevante del cliente y sus datos. Esta ontología se ha generado e insertado de forma automática en el repositorio durante la migración de los datos del cliente al nuevo sistema y mediante otros procesos que se encargaban de recabar información. El principal propósito de este repositorio es el de proveer a procesos de extracción de datos de la información necesaria sobre los datos que han de extraer, facilitando así la automatización de estos. De manera que este repositorio se convierte en una pieza clave del sistema de automatización de extracción de datos de esta plataforma.

Resum

La importància de la informació creix a cada dia que passa. Aquells que posseeixen una gran quantitat d'informació en volen treure benefici mitjançant les tècniques adients.

Aquest projecte va ser desenvolupat dintre d'un altre més gran, a petició d'un client que posseeix una gran quantitat d'informació. Client que va demanar una plataforma capaç de gestionar aquesta gran quantitat de dades i treure profit de la seva informació. L'objectiu d'aquest projecte és dotar de certa independència a aquesta plataforma a partir de la informació sobre les dades del client.

S'ha posat en marxa un repositori de metadades semàntic que conté una ontologia amb tota la informació rellevant del client i les seves dades. Aquesta ontologia s'ha generat i inserit automàticament al repositori mentre es duia a terme la migració de les dades del client al nou sistema i mitjançant uns altres processos que s'encarregaven de recavar informació. El principal propòsit d'aquest repositori és el de proveir a processos d'extracció de dades de la informació necessària sobre les dades que han d'extreure, facilitant així l'extracció d'aquests. De manera que aquest repositori es converteix en una peça clau del sistema d'automatització d'extracció de dades d'aquesta plataforma.

Abstract

Information power grows every day that goes by. Those who hold a huge amount of information want to take advantage of it by using the appropriate techniques.

This project has been developed as part of a bigger corporate project, in response to a client's request. This client needed a platform capable of both managing the large amount of data they had, and getting the most out of their information. The aim of this project is to make this platform less dependent on the user through information about client's data.

A metadata repository with an ontology holding important information about the client and its data has been launched. This ontology has been generated and inserted automatically in the repository throughout the migration process of the client's data to the new system and some other metadata ingestion processes. The main purpose of the repository is to make easier the automatization of the access to data stored in the new platform, by providing information about the data to data extraction processes. Thus, this repository becomes a key element of the data extraction's automatization system of this platform.

Me gustaría agradecer a Víctor Herrero por su ayuda y supervisión durante el desarrollo del proyecto, así como a todo el equipo de Big Data Analytics Lab, Víctor, Albert Obiols, Oscar Romero, Alberto Abelló y Juan Salmerón. Por su ayuda y soporte. También me gustaría agradecer a inLab FIB por haberme dado la posibilidad de ser partícipe de este proyecto, y a todo su equipo por este fantástico tiempo que hemos pasado juntos compartiendo un mismo propósito, marcar la diferencia en cada proyecto dando lo mejor de nosotros mismos.

Por supuesto, agradecer también el apoyo de mi familia y de todos mis amigos, no solo durante el desarrollo de este trabajo, sino durante toda la carrera.

Índice general

1. Introducción y contextualización	1
1.1. Introducción	1
1.2. Contexto	3
1.2.1. Empresa cliente	3
1.2.2. inLab FIB	3
1.2.3. Director, co-director y ponente	4
1.2.4. Autor	4
1.3. Formulación del problema	5
1.4. Estado del arte	7
2. Gestión del proyecto	10
2.1. Alcance	10
2.2. Metodología	12
2.2.1. Iteraciones (Sprints)	12
2.2.2. Tablero de tareas (Task board)	13
2.3. Planificación	14
2.3.1. Descripción de las tareas	15
2.3.2. Recursos temporales	17
2.3.3. Valoración de alternativas i plan de acción	17
2.3.4. Diagrama de Gantt	18
2.3.5. Dimensión del proyecto	20
2.3.6. Desviación de la planificación	20
2.4. Identificación de los costes	21
2.4.1. Costes directos	21
2.4.2. Costes indirectos	24
2.4.3. Coste total	24
2.4.4. Coste a nivel de actividades de Gantt	25
2.4.5. Control de gestión	25
2.5. Informe de sostenibilidad	26
2.5.1. Sostenibilidad económica	26
2.5.2. Sostenibilidad social	26
2.5.3. Sostenibilidad ambiental	26
2.5.4. Matriz de sostenibilidad	27
2.6. Identificación de leyes y regulaciones	28
3. Arquitectura del sistema	29
3.1. Data Lake	29
3.2. Repositorio de metadatos	30
3.3. Quarry	31

3.4. Visión global	31
4. Desarrollo del proyecto	33
4.1. Ontología	33
4.1.1. Composición de la ontología	33
4.1.2. Modelo básico	34
4.1.3. Modelo avanzado	38
4.1.4. Modelo final	42
4.2. Repositorio de metadatos	45
4.2.1. Instalación y configuración del entorno	45
4.2.2. Adaptación de la API para el uso de namespaces	45
4.2.3. Implementación de la conexión con Quarry en la API	47
4.2.4. Extensión de la búsqueda de tripletas	52
4.3. Ingestión de metadatos	54
4.3.1. Inserción de las tripletas básicas	54
4.3.2. Generación de las tripletas del modelo básico	54
4.3.3. Generación de las tripletas del modelo avanzado	55
4.3.4. Inserción de las tripletas del modelo final durante los procesos ETL	55
5. Conclusiones	57
5.1. Objetivos alcanzados	57
5.2. Trabajo futuro	57
A. Manual de instalación y configuración de Virtuoso Opensource	59
B. Manual de instalación y configuración de la API	60
Bibliografía	62

Índice de figuras

1.1. Evolución de las referencias al término "big data" en publicaciones científicas y académicas	7
1.2. Prueba de rendimiento utilizando diferente información estadística y volumen de datos	8
2.1. <i>Task Board</i> del equipo de Big Data Analytics Lab.	13
2.2. Diagrama de Gantt de la planificación del proyecto	19
2.3. Dimensión del proyecto: representados los diferentes workflows y fases	20
3.1. Esquema de los diferentes componentes del Data Lake.	29
3.2. Esquema de los diferentes componentes del repositorio de metadatos.	30
3.3. Esquema de funcionamiento del Quarry.	32
3.4. Esquema de las diferentes partes del sistema Big Data en conjunto.	32
4.1. Esquema de los diferentes namespaces de la ontología y su contenido.	34
4.2. Esquema básico de ejemplo de conceptos de la ontología.	35
4.3. Relación entre conceptos principales y características de la ontología.	35
4.4. Ejemplo del modelo básico de la ontología.	36
4.5. Modelo básico de la ontología.	37
4.6. Esquema de las tablas cremallera.	38
4.7. Modelo avanzado de la ontología.	41
4.8. Esquema de las novedades del modelo final de la ontología.	42
4.9. Modelo final de la ontología.	44
4.10. Extracción de las Características específicas del modelo.	51
4.11. Extracción de los valores de un elemento del Mapping.	52

Índice de tablas

2.1. Salario estimado asociado a cada rol involucrado en el proyecto	21
2.2. Distribución de las horas de cada <i>sprint</i> entre los roles	21
2.3. Coste estimado de los recursos humanos por cada rol	22
2.4. Coste del hardware utilizado durante los 12 <i>sprints</i>	22
2.5. Coste del hardware adicional utilizado durante 4 <i>sprints</i>	23
2.6. Coste del software utilizado durante el desarrollo del proyecto	23
2.7. Costes directos del proyecto	24
2.8. Coste total estimado del proyecto	24
2.9. Matriz de sostenibilidad del trabajo de final de grado	27

Glosario

API: Del inglés, Application Programming Interface. Conjunto de funciones y métodos que se ponen a disposición del usuario u otro software, como una capa de abstracción, para facilitar la interacción con un determinado software o tecnología.

Big Data: Sistema que trabaja con una gran cantidad de datos con el fin de analizarlos para extraer conclusiones y predicciones en base a los datos. Se trata de una cantidad de datos tan grande que resulta imposible tratarlos con las herramientas de bases de datos y analíticas convencionales.

Data Lake: Sistema de almacenamiento, normalmente distribuido en un conjunto de máquinas, que alberga una gran cantidad de datos en su formato nativo. Acostumbra a ser el sistema de almacenamiento en los sistemas Big Data.

Data Warehouse: Base de datos corporativa que integra información de una o más fuentes distintas, para ser procesada con el fin de realizar análisis que ayuden a la toma de decisiones en la entidad en la que se utiliza.

ETL: Del inglés, Extract, Transform and Load. Proceso de extracción de información de diferentes fuentes, para su posterior transformación para ser almacenada en el formato idóneo en el sistema de almacenamiento de información deseado.

HTTP: Del inglés, HyperText Transfer Protocol. Protocolo de comunicación que permite la transferencia de información en un sistema de información distribuido y colaborativo, como la World Wide Web (WWW), mediante una serie de métodos para solicitar y enviar información, entre otras utilidades.

JSON: Del inglés, JavaScript Object Notation. Formato ligero de intercambio de datos basado en el lenguaje de programación JavaScript, fácil de comprender y escribir para humanos y simple de generar e interpretar por las máquinas.

Mapping: Definición del lugar concreto donde se guarda cierta información en un sistema de almacenamiento.

Ontología: Descripción de conceptos, propiedades y relaciones entre entidades en un determinado contexto que son representados con un propósito. Puede ser representada, por ejemplo, mediante RDF.

Open Source: Modelo de desarrollo de software que fomenta el acceso universal y desarrollo colaborativo de un código fuente, a través de licencias que permiten el uso, modificación y redistribución de este código.

RDF: Del inglés, Resource Description Framework. Modelo de datos estándar para la representación de metadatos en forma de grafo. Extiende el sistema de enlaces de la Web utilizando URIs para definir las relaciones entre los dos extremos de un enlace (comúnmente llamadas tripletas).

SPARQL: Del inglés, SPARQL Protocol And RDF Query Language. Lenguaje de consulta semántico para bases de datos, capaz de consultar y manipular datos guardados en formato RDF.

Triplestore: Base de datos específica que almacena tripletas, consultadas mediante un lenguaje de consultas semántico.

Tripleta: Entidad compuesta por sujeto, predicado y objeto, todos representados mediante URIs. El predicado define la relación existente entre sujeto y objeto.

URI: Del inglés, Uniform Resource Identifier. Cadena de caracteres utilizada para identificar un recurso. Esta identificación permite interactuar con el propio recurso a través de una red, típicamente la World Wide Web, utilizando protocolos específicos. A diferencia de la URL (del inglés, Uniform Resource Locator), a veces confundida con la URI, la URI únicamente identifica un recurso, mientras que la URL, además de esto, especifica como interactuar con el recurso y su localización en la red. Por ejemplo la URL *http://ejemplo.org/wiki/URI* hace referencia al recurso identificado como */wiki/URI*, cuya representación se puede obtener mediante HTTP desde el dominio *ejemplo.org*.

Capítulo 1

Introducción y contextualización

1.1. Introducción

Desde hace unos años estamos viendo que la información va tomando cada vez más importancia. Las empresas tecnológicas más influyentes del sector no han llegado dónde están simplemente por los productos o servicios que ofrecen, sino por que poseen una gran cantidad de información sobre los usuarios que utilizan sus servicios.

Pero no basta con acumular datos y más datos, para sacarles provecho hay que saber como utilizarlos y cuán relevante puede ser la información. De esto trata el concepto de Big Data del que tanto oímos hablar hoy en día, de sacar el máximo partido de los datos de los que se dispone. No tan solo haciendo análisis retrospectivos y obteniendo conocimiento e información sobre aquello que ya ha sucedido, como hacemos en Business Intelligence, con el objetivo de facilitarnos la toma de decisiones en base a los resultados de estos análisis, sino de ser capaces de hacer predicciones basándonos en conclusiones que obtengamos. No únicamente mediante la explotación de los datos que poseemos, sino estudiando nuestros datos conjuntamente con datos de distintas fuentes de información a las que tengamos acceso para hacer que nuestras conclusiones y predicciones sean mucho más acertadas.

Una de las aplicaciones más comunes se da en sistemas de toma de decisiones, como el DDDM (Data-Driven Decision Making), donde en lugar de basar las decisiones en lo que a uno le parece o unos cuantos asesores le aconsejen, un sistema de decisión, que se basa en el resultado del análisis exhaustivo de unos datos (y cuando hablamos de Big Data estamos hablando de muchos datos) que pueden ser verificados, es el encargado de tomar las decisiones.

Así nace un proyecto ambicioso, de la voluntad de una empresa de dar un gran paso para mejorar notablemente su modelo de negocio y de la necesidad de controlar unos datos que cada vez son generados a mayor escala. Este importante cambio en el motor del sistema de predicciones y toma de decisiones de esta empresa está en manos del proyecto Big Data Analytics Lab, del cual forma parte este trabajo de final de grado.

El proyecto Big Data Analytics Lab consiste en la creación de una plataforma experimental de Big Data basada en herramientas Open Source. Este proyecto consistirá en la construcción de un Data Lake mediante una selección de herramientas de Big Data que serán más útiles para el caso particular del cliente, y en la creación de un repositorio de metadatos semántico que se nutrirá automáticamente mediante procesos de volcado de

metadatos y durante procesos de extracción, transformación y carga de datos (ETL).

En concreto, este trabajo de final de grado se centra en la puesta en marcha del repositorio de metadatos y en la automatización de la ingesta de metadatos durante estos procesos, para la posterior explotación de estos metadatos con el fin de optimizar la interacción de los usuarios de esta plataforma con los datos.

1.2. Contexto

Este trabajo de final de grado forma parte del proyecto Big Data Analytics Lab y, por lo tanto, analizaremos el contexto tomando como referencia todo el proyecto Big Data Analytics Lab.

En este proyecto hay muchas partes involucradas, a continuación se verá que rol tiene cada una dentro del Big Data Analytics Lab.

1.2.1. Empresa cliente

Esta empresa es una multinacional con sede en Barcelona que trabaja con una gran cantidad de datos cada día. Los analistas de datos de esta empresa trabajan con matrices de datos, por las que han de esperar varios días desde que ven la necesidad de extraer esta matriz hasta que finalmente la obtienen. Esto es debido a la cantidad de datos que tratan, además estos datos están almacenados y estructurados de una manera que dificulta mucho la extracción de estos de forma eficiente.

Por lo tanto, en Big Data Analytics Lab, esta empresa ocupa el rol de cliente, ya que es la principal interesada en que este proyecto salga adelante y poder trabajar con esta plataforma e implantarla en su empresa, facilitándole mucho el trabajo a los analistas, haciendo que puedan ser mucho más productivos.

1.2.2. inLab FIB

inLab FIB es un laboratorio de innovación e investigación de la Facultad de Informática de Barcelona (FIB) de la Universitat Politècnica de Catalunya · Barcelona Tech (UPC) que integra profesorado de diferentes departamentos de la UPC y su propio personal técnico para ofrecer soluciones en diferentes áreas.

inLab FIB es el encargado de desarrollar este proyecto. Para ello ha constituido un equipo técnico formado por expertos en Big Data y minería de datos y otros miembros de inLab FIB. Este equipo está formado por el director y el ponente de este trabajo de final de grado, de los que se hablará más adelante, y por el responsable del Área de Sistemas de Información y Gestión de inLab FIB, Albert Obiols, como responsables del proyecto; por otra parte el equipo desarrollador en Big Data Analytics Lab está formado por tres miembros de inLab FIB.

En Big Data Analytics Lab, inLab FIB se encarga de desarrollar el producto en el período establecido para finalizar el proyecto con éxito y mantener una buena relación con el cliente.

1.2.3. Director, co-director y ponente

El director y el ponente de este trabajo de final de grado, Dr. Oscar Romero y Dr. Alberto Abelló respectivamente, son coordinadores del *Information Technologies for Business Intelligence (IT4BI) Erasmus Mundus Master*^I en la UPC y del postgrado *Big Data Management and Analytics*^{II} en la UPC School. Además son profesores del departamento de Ingeniería de Servicios y Sistemas de Información de la UPC y profesores colaboradores en inLab FIB, ambos responsables del proyecto Big Data Analytics Lab. El co-director, Víctor Herrero, es miembro de inLab FIB y lidera el equipo desarrollador en Big Data Analytics Lab.

Como miembros y colaboradores de inLab FIB, su objetivo es el mismo que el descrito en el apartado anterior.

1.2.4. Autor

El autor de este trabajo de final de grado es miembro de inLab FIB y tiene el rol de desarrollador en Big Data Analytics Lab. En el proyecto tendrá el rol de analista, desarrollador y *tester*. Es una de las partes más interesadas en finalizar el proyecto a tiempo, ya que su objetivo es entregar el trabajo dentro del período establecido para finalizar el Grado en Ingeniería Informática que está cursando.

Como miembro de inLab FIB, su objetivo es el mismo que se ha descrito anteriormente, además de los objetivos concretos del trabajo de final de grado.

^I<http://http://it4bi.univ-tours.fr>

^{II}<http://www.talent.upc.edu/ing/professionals/presentacio/codi/331100/big-data-management-analytics/>

1.3. Formulación del problema

Las empresas que albergan una gran cantidad de datos, que además cada vez crece a mayor velocidad, en un Data Warehouse en el que cada vez encuentran más carencias, están viendo en las nuevas tecnologías Big Data la solución a sus problemas. Esto es lo que ha manifestado el cliente del proyecto Big Data Analytics Lab, hace un tiempo que el Data Warehouse con el que trabajan diariamente se les ha quedado pequeño, ya que este les impide trabajar productivamente debido a sus carencias.

Los empleados de esta empresa encargados de analizar todos estos datos, han de extraer una matriz con los datos que necesitan para hacer un posterior estudio de estos datos y sacar sus conclusiones. El proceso de obtención de esta matriz es de lo más engorroso, en especial para unos empleados expertos en el análisis de datos pero inexpertos en el mundo de las bases de datos y las tecnologías que se usan en este Data Warehouse. Estos analistas de datos están dedicando mucho tiempo a un proceso de extracción, el cual incluye llevar a cabo tareas de extracción, aplicar reglas de limpieza de datos, y finalmente generar la matriz en base a estos datos. Cada una de estas tareas necesita mucho tiempo para ser completada debido a la ineficiencia del sistema actual con una cantidad tan grande de datos.

La voluntad del cliente es de solucionar este problema haciendo que el sistema, en el que guardan y del cual obtienen los datos, sea lo más eficiente posible para su caso concreto y abstraiga a los analistas de datos de toda tarea que les haga interactuar con los entresijos del sistema para obtener la matriz.

Atendiendo a estas necesidades, el equipo de inLab FIB ha puesto en marcha un Data Lake, compuesto por diferentes herramientas Big Data del ecosistema Apache Hadoop^{III}, que además está respaldado por un repositorio de metadatos semántico con información y estadísticas sobre los datos. Datos que han sido migrados desde la base de datos operacional de la empresa.

Todos estos metadatos son representados mediante una ontología de los conceptos de negocio del cliente. Para cada concepto de negocio disponemos de toda la información relevante para el cliente: procedencia (características y localización en la base de datos operacional), mapeo en el Data Lake y datos estadísticos. Las estadísticas que almacenamos sobre cada concepto de negocio son: mínimo, máximo, media y distribución de los valores.

Todos los metadatos de los que se dispone han sido generados automáticamente durante los procesos ETL encargados de llevar a cabo la migración de los datos al Data Lake o obtenidos a partir de información proporcionada por el cliente. Estos metadatos son guardados en el repositorio mediante una API (Application Programming Interface) que interactúa con Virtuoso Open-Source^{IV}, que es la herramienta donde almacenamos los metadatos en formato RDF (Resource Description Framework)^V. Además esta ontología es representada visualmente mediante una herramienta llamada Quarry [1], que se ha adaptado para este proyecto para que represente ontologías.

^{III}<https://hadoop.apache.org/>

^{IV}<http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/>

^V<https://www.w3.org/RDF/>

Los analistas de datos (usuarios de este sistema) podrán seleccionar exactamente los datos que quieren obtener mediante esta herramienta de visualización. Esta herramienta obtendrá la información demandada mediante procesos automatizados que Apache Spark^{VI} ejecutará, interactuando con el repositorio de metadatos y el Data Lake para obtener los datos necesarios eficientemente. Finalmente, el usuario obtendrá directamente la matriz de datos que necesita para llevar a cabo sus estudios.

Los objetivos que se han de cumplir, vistos más al detalle en el apartado Descripción de las tareas de la sección Planificación de esta memoria, son:

- Modelar la ontología por fases, añadiendo nueva información a medida que avanza el proyecto, hasta llegar a un modelo final.
- Tener el repositorio de metadatos funcionando junto a la API que se encarga de gestionar (insertar/eliminar contenido y lanzar consultas) el contenido de este repositorio.
- Tener listos los procesos de automatización encargados de hacer la ingesta de metadatos en el repositorio a partir de la información aportada por el cliente y la extraída durante los procesos ETL.
- Finalizar el proyecto con un alto nivel de satisfacción del cliente.

De esta manera se evita que los usuarios, sin un extenso conocimiento del sistema, tengan que dedicar gran parte de su tiempo a extraer estos datos ineficientemente: obteniendo los datos de una forma muy básica del Data Warehouse, aplicando reglas de limpieza de datos, y finalmente generando la matriz en base a estos datos. Por lo tanto, dotamos a estos usuarios de un sistema totalmente transparente para ellos, a través del cual pueden obtener la información que necesitan fácilmente, ahorrándoles mucho tiempo y, por consiguiente, aumentando mucho su productividad.

^{VI}<http://spark.apache.org/>

1.4. Estado del arte

Big Data es un concepto que llevamos escuchando cada vez más desde hace unos años, y no es ningún secreto que en los próximos años lo vamos a escuchar todavía más. Se trata de un término relativamente nuevo en la informática, pero cada vez adquiere mayor importancia.

En un estudio que hicieron para el *2015 IEEE International Congress on Big Data*^{vii}, titulado *MetaData: BigData Research Evolving Across Disciplines, Players, and Topics* [2], podemos observar como en una búsqueda del término "big data" en una base de datos bibliográfica llamada *Web of Science*^{viii}, una de las fuentes más importantes a la hora de buscar información científica y académica, ha crecido drásticamente el número de referencias que se hace al término en este tipo de publicaciones los últimos años. Como podemos ver en la Figura 1.1, extraída de este mismo estudio, podríamos hablar de un crecimiento exponencial en los últimos años, dado que la información del año 2014 está incompleta.

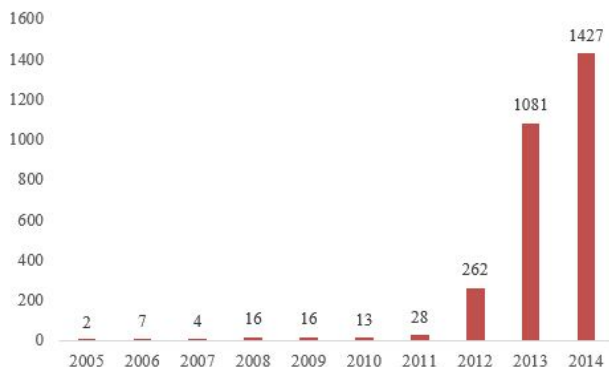


Figura 1.1: Evolución de las referencias al término "big data" en publicaciones científicas y académicas

El uso de sistemas Big Data se ha llegado a convertir en una necesidad en algunas áreas de negocio y, pese a que la informática sigue siendo la disciplina en la que predomina, como podemos ver en la misma publicación [2], cada vez son más disciplinas las que deciden hacer uso de estos sistemas.

Típicamente en las plataformas de Big Data se almacenan los datos en un lugar llamado Data Lake, en el cual se almacenan los datos en bruto, es decir, conservando el formato inicial de los mismos, pudiendo tener la información estructurada, semi-estructurada o no estructurada. Esto permite extraer datos mediante una gran variedad de consultas e incluso de diferentes fuentes de datos. A diferencia de los sistemas tradicionales (como explican en el artículo *Embedding AI and Crowdsourcing in the Big Data Lake* [3]), como el Data Warehouse, que generalmente están diseñados para que se lancen un tipo de consultas limitadas de forma óptima, estructurando los datos de forma adecuada a estas consultas, y están basados en una única fuente de datos.

Normalmente en los sistemas de decisión basados en Data Warehouses no se dispone de la información inmediatamente, ya que los datos de este se generan mediante procesos ETL que se lanzan sobre la base de datos operacional, donde se realizan las operaciones y transacciones del día a día. En cambio en un Data Lake sí que podemos disponer de la

^{vii}<http://www.ieeebigdata.org/2015/>

^{viii}<http://wokinfo.com/>

información en tiempo real.

Además, el volumen de datos cada día que pasa se genera a mayor escala, y este crecimiento aumentará drásticamente a medida que el concepto de *Internet of Things*^{IX} arraigue en nuestra sociedad. Por lo tanto, pese a tener lugares donde poder almacenar todos estos datos, como los Data Lakes, también se debe de optimizar tanto como sea posible el acceso y análisis de los datos para procesar rápidamente un volumen tan grande de información.

Se han hecho estudios para mejorar el rendimiento de los análisis de datos en este tipo de sistemas, donde disponemos de una cantidad de datos considerable. Un ejemplo es el trabajo presentado en el artículo titulado *Improving Data Analysis Performance for High-Performance Computing with Integrating Statistical Metadata in Scientific Datasets* [4]. La idea principal de este estudio es dividir los conjuntos de datos en subconjuntos, siguiendo diferentes criterios, y añadir una pequeña cantidad de metadatos estadísticos en cada subconjunto para optimizar el acceso a los datos. Esto es posible ya que en muchas aplicaciones en las que se trata una cantidad de datos tan grande, como aplicaciones científicas en este caso, el patrón de acceso a los datos suele ser el conocido como *WORM* (Write Once and Read Many). En este patrón de acceso los datos una vez son almacenados no son actualizados, pero sí que se realizan múltiples accesos de lectura sobre estos.

En este estudio cuando los conjuntos de datos son almacenados en el sistema, previamente son procesados para generar los subconjuntos y calcular las estadísticas, para almacenarlas como metadatos junto a cada subconjunto. Por lo tanto, para aprovechar las estadísticas integradas en estos subconjuntos de datos se ha cambiado el patrón de acceso de la función de lectura del sistema, de manera que se obviarán los subconjuntos de datos que no sean relevantes únicamente basándose en las estadísticas que contienen los metadatos. En la Figura 1.2, extraída de este mismo artículo, se aprecia una notable mejora del rendimiento aplicando esta solución sobre conjuntos de datos a partir de cierto volumen de datos, almacenando diferentes datos estadísticos en los metadatos.

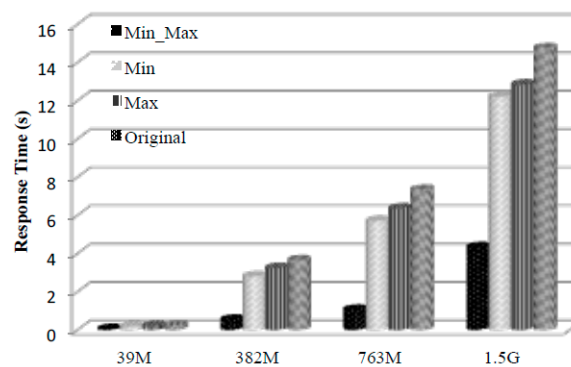


Figura 1.2: Prueba de rendimiento utilizando diferente información estadística y volumen de datos

De esta manera, vemos que almacenar metadatos que contengan estadísticas sobre los propios datos es una solución muy interesante a la hora de optimizar el acceso a los datos en sistemas Big Data, donde un mal patrón de acceso a los datos puede hacernos perder mucho tiempo. Y esta pérdida de tiempo en una empresa típicamente comporta una pérdida de dinero y/o beneficios.

^{IX}'That 'Internet of Things' Thing' article - <http://www.rfidjournal.com/articles/view?4986>

La solución que se propone en Big Data Analytics Lab, vista en Formulación del problema, está en sintonía con este estudio. Se ha optado por crear una solución basada en herramientas Open Source y proyectos académicos existentes, con ligeras adaptaciones que hacen que la solución satisfaga las necesidades del cliente. Pese a utilizar un gran número de herramientas Open Source existentes, tampoco se podría decir que ya existe una solución para el problema de nuestro cliente, ya que este problema necesita una solución a medida, que las herramientas actuales, de por sí solas, no pueden satisfacer.

Por lo tanto, no es necesario crear herramientas Big Data desde cero, las cuáles necesitarían un largo tiempo de desarrollo y no tendrían una comunidad de usuarios que haga aportaciones (facilitando así la constante evolución del producto) como las herramientas que hemos seleccionado.

Capítulo 2

Gestión del proyecto

2.1. Alcance

Como se ha dicho anteriormente, este proyecto se centra en la puesta en marcha del repositorio de metadatos, mediante una serie de herramientas Open Source, y en la automatización de la ingesta de metadatos durante la migración de los datos al Data Lake.

Como resultado de este trabajo obtendremos un repositorio de metadatos semántico con el que podremos interactuar mediante una API, que servirá para optimizar el acceso a los datos almacenados en un Data Lake. Este repositorio se actualizará automáticamente durante los procesos ETL encargados de migrar todos los datos del Data Warehouse, actual sistema de la empresa, al Data Lake.

Finalmente todos estos metadatos, guardados en el repositorio de metadatos semántico, estarán listos para que procesos de explotación del sistema Big Data de soporte al usuario, hecho a medida para el cliente, puedan utilizarlos para optimizar los procesos y el acceso a los datos. De manera que el repositorio de metadatos tiene un papel muy importante en la automatización de estos procesos de extracción de datos.

Además se pondrá en funcionamiento una herramienta de visualización, mediante la cual el usuario podrá interactuar con el sistema Big Data, resultado de todo el proyecto Big Data Analytics Lab, de forma totalmente transparente.

Para ello se han de cumplir una serie de objetivos objetivos, definidos en la sección Formulación del problema de esta memoria, con Diciembre de 2015 como *deadline*, que es el periodo establecido para la finalización de Big Data Analytics Lab, el proyecto del que forma parte este trabajo.

No se ha previsto una larga lista de problemas o obstáculos que puedan surgir durante el desarrollo, más allá de los problemas habituales que puedan surgir al intentar integrar diferentes herramientas. Pese a que esta lista no es muy larga, seguidamente se detallan estos posibles obstáculos que se podrían encontrar durante el desarrollo del proyecto.

- Complicaciones a la hora de modificar algunas de las herramientas, resultando más difícil la implementación de las modificaciones necesarias en el código.
- Mayor dificultad a la hora de integrar algunas herramientas, como Virtuoso junto a la API o la comunicación entre API y Quarry.

Estos obstáculos provocarían una desviación en el tiempo planificado para la implementación del proyecto, cuyos costes se estudian en el apartado Control de gestión de la sección Identificación de los costes de esta memoria.

2.2. Metodología

Para el desarrollo de Big Data Analytics Lab se está utilizando una metodología ágil, en concreto la metodología *Scrum*¹. Se trata de una metodología flexible y ágil que es una de las claves para el éxito del proyecto.

Scrum se basa en iteraciones durante el proceso de desarrollo, llamadas *sprints*. Un *sprint* está compuesto por un conjunto de tareas que deben ser completadas para cumplir los objetivos de esa iteración. Normalmente estas iteraciones duran entre una semana y un máximo de un mes. Los objetivos de cada *sprint* se fijan en reuniones que se llevan a cabo antes de cada iteración; además al finalizar un *sprint* se hace otra reunión para revisar el trabajo hecho durante esta iteración.

Estas iteraciones cortas permiten obtener *feedback* continuamente del cliente, de manera que permite que el producto cambie en función de sus necesidades, ya sea por la inclusión de nuevos requisitos o por la modificación de los requisitos existentes.

Esta metodología permite hacer los cambios necesarios para adaptarla al proyecto y al equipo. En este proyecto no se ha seguido la metodología al pie de la letra, sino que se ha adaptado a las necesidades del proyecto. A continuación se detalla como se han aplicado algunos conceptos importantes de esta metodología al proyecto.

2.2.1. Iteraciones (Sprints)

Como se ha visto previamente, el desarrollo se divide en *sprints* que, en el caso del proyecto de Big Data Analytics Lab, son de dos semanas y el completo desarrollo del proyecto esta planificado para llevarse a cabo en un total de 20 *sprints*. De esta manera se pueden detectar errores rápidamente y se pueden modificar los requisitos para maximizar las garantías de éxito. Esta filosofía es ideal para proyectos en los que se estén desarrollando productos innovadores, ya que la tecnología evoluciona rápidamente, y de esta manera se permite una rápida adaptación a estos cambios.

Al inicio de cada *sprint* se hace una reunión, llamada *sprint planning*, en la que participan todos los miembros involucrados en el proyecto (clientes, responsables y equipo desarrollador). En esta reunión se deciden las tareas que se realizarán durante el *sprint*, que serán colocadas en el *task board* en la columna de *sprint backlog*.

Estas tareas normalmente han sido definidas previamente y forman parte del *product backlog*; estas son las tareas a completar para tener un producto viable de acuerdo con los requisitos del cliente.

El *sprint* finaliza con una reunión, llamada *sprint review*, en la que vuelven a participar todos los miembros involucrados en el proyecto y se revisan todas las tareas completadas y el estado de las tareas que restan por completar mediante el uso del *task board*. Por cada tarea en la columna de *A validar* el equipo desarrollador demuestra que se ha realizado y que se cumplen los criterios de satisfacción.

En el proyecto de Big Data Analytics Lab se celebra una única reunión entre *sprints*

¹<http://scrummethodology.com/>

en la que se lleva a cabo tanto el *sprint review* como el *sprint planning*.

2.2.2. Tablero de tareas (Task board)

El tablero de tareas o *task board* refleja el estado actual de todas las tareas del *sprint*. Este tablero permite a los miembros del equipo visualizar el estado del proyecto en cualquier momento: saber que tareas se están desarrollando, quien lo está desarrollando, ver las tareas pendientes y las tareas que ya se han completado.

En este proyecto se utiliza un tablero digital mediante la herramienta *Trello*¹¹. En la Figura 2.1 se puede ver el estado del tablero de tareas de Big Data Analytics Lab durante uno de los *sprints* del desarrollo.

El panel está formado por cinco columnas:

- **Backlog:** Tareas pendientes de ser realizadas en futuros *sprints* para la finalización del desarrollo del producto. Representa el *product backlog*.
- **To do next sprint:** Tareas listas para ser realizadas en el *sprint* actual. Representa el *sprint backlog*.
- **Doing:** Tareas que cada miembro del equipo de desarrollo está llevando a cabo.
- **A validar:** Tareas realizadas en el *sprint* actual, pendientes de ser validadas en la reunión de *sprint review*.
- **Done:** Tareas realizadas en un determinado *sprint*, validadas por todo el equipo en la reunión de *sprint review* de ese *sprint*.

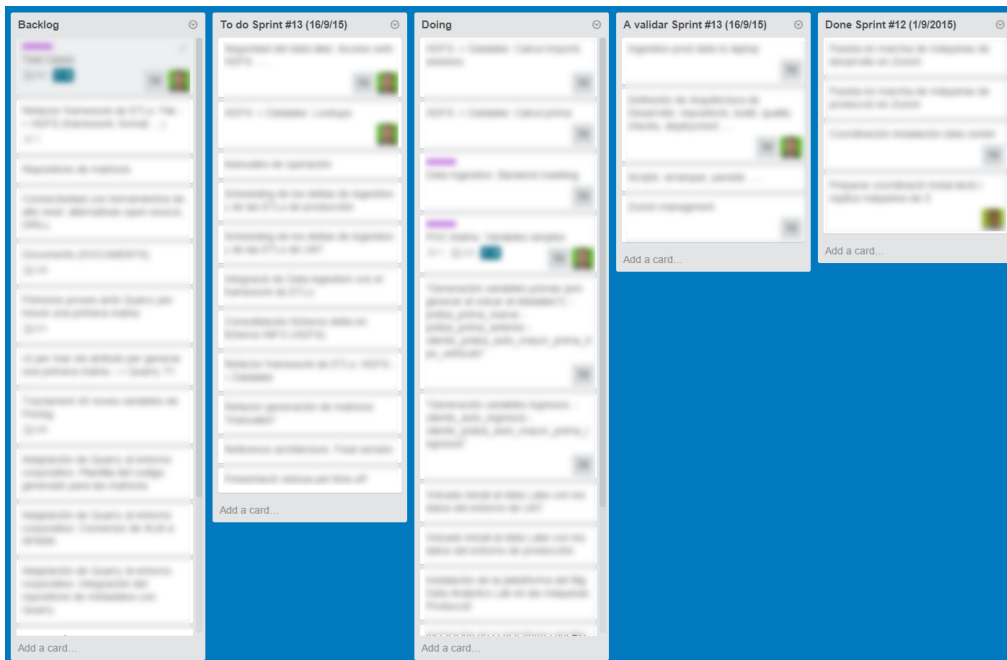


Figura 2.1: *Task Board* del equipo de Big Data Analytics Lab.

¹¹<https://trello.com/>

2.3. Planificación

El proyecto de Big Data Analytics Lab empezó el mes de Febrero de 2015 i finalizará en el mes de Diciembre de 2015. En cambio la duración de la parte relacionada con el repositorio de metadatos y la automatización de la ingesta de metadatos, competencia de este trabajo final de grado, empezó en Junio de 2015.

En la planificación temporal se han establecido todos los *sprints*, de dos semanas de duración cada uno, tal como se mencionó en el apartado de Metodología de esta memoria. En cambio, al empezar en el mes de Julio, la planificación de esta parte del proyecto contará con menos *sprints* que el proyecto Big Data Analytics Lab. Esta parte del proyecto se llevará a cabo en un total de 12 *sprints*.

Cada *sprint* dura dos semanas, que son un total de 10 días laborables, en los que el autor de este trabajo de final de grado dedica 5 horas por día laborable.

Además, también se tiene en cuenta en la planificación del proyecto la última fase de este, la redacción de la memoria. Esta fase se llevará a cabo tras el último *sprint* y tendrá una duración de cuatro semanas, en las que se dedicará el mismo número de horas diarias que a los *sprint* durante 20 días.

La dedicación en horas a este trabajo de final de grado será:

$$\begin{aligned} 12 \text{ sprints} \cdot 10 \text{ días/sprint} \cdot 5 \text{ horas/día} &= 600 \text{ horas} \\ 20 \text{ días de redacción de memoria} \cdot 5 \text{ horas/día} &= 100 \text{ horas} \\ 3 \text{ créditos de GEP} \cdot 30 \text{ horas/crédito} &= 90 \text{ horas} \\ 600 \text{ horas} + 100 \text{ horas} + 90 \text{ horas} &= 790 \text{ horas} \end{aligned}$$

Por lo tanto, se dedicará a este proyecto el tiempo que requiere un trabajo de final de grado, ya que la duración de este se estima que debe de dedicarse a un trabajo final de grado es de 540 horas, según la normativa^{III}.

^{III}[http://www.fib.upc.edu/fib/estudiar-enginyeria-informatica/treball-final-grau/mainColumnParagraphs/0/document/NormativaTFG-GEI%20\(document%20final\).pdf](http://www.fib.upc.edu/fib/estudiar-enginyeria-informatica/treball-final-grau/mainColumnParagraphs/0/document/NormativaTFG-GEI%20(document%20final).pdf)

2.3.1. Descripción de las tareas

Para desarrollar este proyecto se está siguiendo la metodología ágil *Scrum*. Por este motivo no se pueden definir unas fases de desarrollo del proyecto con fecha de inicio y final. Por lo tanto, solo podemos dividir el proyecto en 3 fases, que se irán repartiendo entre los diferentes *sprints* en función de los resultados de cada *sprint planning* y *sprint review*, y una fase adicional en la que se llevará a cabo la redacción de la memoria de este Trabajo Final de Grado.

Modelado de la ontología

En esta fase del proyecto, explicada en detalle en la Sección 4.1 del capítulo Desarrollo del proyecto de esta memoria, deberán estudiarse los conceptos de negocio junto a la empresa cliente para entender su modelo de negocio y poder modelar la ontología de los conceptos que le interesen. Esta ontología será modelada en varias iteraciones, añadiendo cada vez más información al modelo conforme la empresa vaya aportando nueva información a ser incluida en la ontología.

Requisitos: Ninguno

Dificultad estimada: Media

Tareas:

- Obtener conocimientos de Semantic Data (ontologías, RDF, ...) [5] [6]
- Modelar una primera ontología muy básica
- Modificar el modelo de la ontología para incluir nueva información aportada por la empresa tantas veces como se vea oportuno hasta obtener un modelo completo

Repositorio de metadatos

En esta fase se deberá instalar el repositorio de metadatos (Virtuoso) junto a todas sus dependencias en las máquinas de desarrollo de inLab FIB. Una vez esto esté funcionando usaremos una API para facilitarnos la interacción con el repositorio. Esta API ha de ser modificada, ya que nuestro proyecto tiene algunos requisitos que esta API no satisface, como la integración con Quarry. Para ello se estudiará el funcionamiento de Quarry así como el formato de ontología que puede visualizar, para implementar nuevos métodos en la API que permitan la integración de ambas herramientas.

Requisitos: El desarrollo de determinadas tareas de esta fase dependerá del modelo de la ontología construido en la fase anterior.

Dificultad estimada: Alta

Tareas:

- Instalar dependencias
- Instalar Virtuoso
- Estudiar funcionamiento de la API y Quarry
- Hacer los cambios necesarios en la API
- Poner en marcha la API
- Validación del funcionamiento de los cambios implementados

Desarrollo de los procesos de obtención de metadatos

Durante la fase final del proyecto deberán desarrollarse los procesos encargados de obtener los metadatos necesarios durante los procesos ETL, que se llevarán a cabo para migrar los datos de la base de datos operacional al Data Lake. Estos metadatos serán guardados automáticamente en el repositorio de metadatos usando la API durante estos procesos.

Requisitos: Todas las fases anteriores del proyecto finalizadas

Dificultad estimada: Alta

Tareas:

- Desarrollar procesos
- Hacer pruebas con pequeños ETL para comprobar que todos los componentes que intervienen en este proceso funcionan correctamente
- Lanzar los ETL junto a los procesos de ingesta de metadatos preparados que guardarán todos los metadatos en el repositorio de metadatos
- Comprobar que todo se visualiza correctamente en Quarry

Redacción de la memoria del proyecto

Una vez finalizado todo el proyecto pasara a redactarse la memoria. Durante el transcurso del proyecto se han podido redactar varias partes de la memoria a modo de documentación de las tareas que se iban realizando, pero la mayoría de esta será redactada una vez se haya finalizado el desarrollo del proyecto. De esta manera se podrá redactar la memoria con una visión global del proyecto. Además, se revisarán las partes de la memoria que hayan podido ser redactadas previamente, modificándolas o reescribiéndolas si fuera necesario. El tiempo estimado para redactar la memoria es de cuatro semanas, comenzando una vez finalizado el último sprint del proyecto y acabando una semana antes de la defensa del Trabajo Final de Grado.

Requisitos: Todas las fases previas del proyecto finalizadas

Dificultad estimada: Media

Tareas:

- Redactar la memoria

2.3.2. Recursos temporales

Para desarrollar este proyecto no será necesario el uso de recursos adicionales en ningún momento del desarrollo. Tan solo puntualmente se consulta a los autores de algunas herramientas que usamos para resolver dudas; como a la autora de la API, Varunya Thavornun, y a uno de los autores del Quarry, Sergi Nadal, estudiante de *IT4BI Erasmus Mundus Master*. Este recurso será utilizado en caso de necesidad y servirá para resolver contratiempos de forma rápida.

No se consideran más recursos materiales ya que todas las herramientas que se utilizarán son Open Source.

2.3.3. Valoración de alternativas i plan de acción

Debido a la metodología que se sigue para desarrollar el proyecto, el seguimiento de la planificación es muy sencillo de verificar que se cumple, ya que al principio del proyecto se definieron las fechas para todas las reuniones de *sprint planning* y *sprint review*, que se realizan cada dos semanas.

Una desviación eventual que se podría dar, por motivos de agenda, es que alguna de las reuniones no se pueda realizar exactamente el día en el que se planificó. Pero no afectaría al tiempo de desarrollo del proyecto, ya que el equipo que participa en estas reuniones está en comunicación constantemente y no habría ningún problema en fijar otra fecha lo más cercana posible al día en que se acordó en un principio.

Otra posible desviación que se prevé que pueda pasar es que necesitemos mas horas de las planificadas para llevar a cabo todas las tareas de un *sprint* determinado. Esto se detectaría en la reunión de *sprint planning*, que es el momento en el que se asignan las tareas a hacer en el *sprint*. Con lo cual, el equipo presente en la reunión, junto a los clientes, decidiría en ese momento si es viable dedicar más horas a ese *sprint*, con el incremento en el presupuesto que conlleva esta decisión, o si se descarta alguna de las tareas que iban a ser incluidas en el *sprint* o se cambia esta tarea por otra que requiera menos tiempo para ser llevada cabo.

El estudio del coste de las posibles desviaciones que pueda sufrir el proyecto durante su desarrollo se hace en el apartado Control de gestión de la sección Identificación de los costes de esta memoria.

2.3.4. Diagrama de Gantt

En la planificación se han definido las fechas entre las que se llevarán a cabo todos los *sprints*; entre estos *sprint* se realizarán las reuniones de seguimiento, donde se llevará a cabo tanto el *sprint planning* del siguiente *sprint* como el *sprint review* del anterior. En la Figura 2.2 se puede ver el diagrama de Gantt de la planificación, donde vemos la distribución de todos los *sprints* de desarrollo del proyecto.

No se puede detallar en esta planificación inicial el contenido de cada uno de los *sprint* ya que, debido a la metodología que usamos para el desarrollo de este proyecto, los objetivos de cada *sprint* se definen en la reunión de *sprint planning* previa al *sprint* en función del resultado obtenido en el *sprint* anterior.

En el diagrama de Gantt, cada una de las casillas que representan las semanas de cada mes está representada por el miércoles de esa semana. De esta manera podemos diferenciar los días en los que acaba un *sprint* y empieza el siguiente, llevándose a cabo la reunión de seguimiento este mismo día.

Además, en el diagrama de Gantt figura la última fase de este proyecto, la redacción de la memoria de Trabajo Final de Grado. Esta fase tiene una duración de cuatro semanas, comienza tras el último *sprint* del proyecto y finaliza el día 20 de enero, día en el que se entregará la memoria. De esta manera, se presenta la memoria al menos una semana antes de la defensa del proyecto, día 28 de enero.

2.3.5. Dimensión del proyecto

En la Figura 2.3 se muestra la dimensión del proyecto con los tres *workflows* de desarrollo, definidos en la sección Descripción de las tareas de esta memoria, y las tres fases escogidas (definición, establecimiento y ejecución).

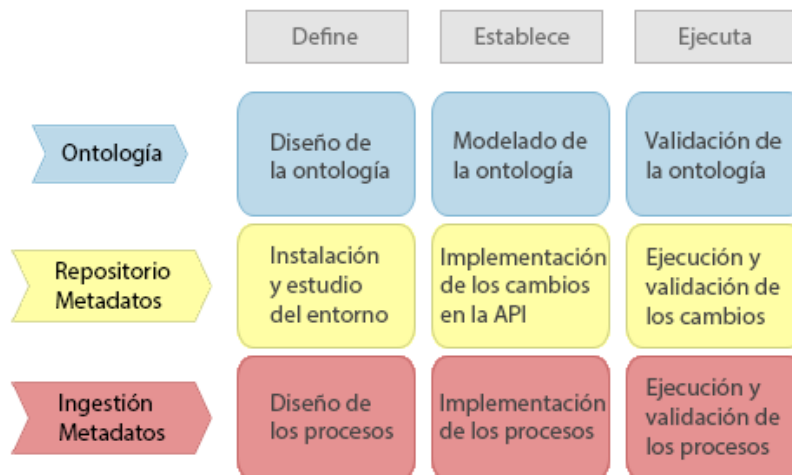


Figura 2.3: Dimensión del proyecto: representados los diferentes workflows y fases

2.3.6. Desviación de la planificación

Durante el desarrollo del proyecto han surgido algunos imprevistos, ajenos a este TFG, que han limitado el alcance de este trabajo de final de grado. La parte que no se ha llegado a realizar es la generación de las estadísticas sobre los datos y, por lo tanto, la inclusión de estas estadísticas en la ontología y la posterior ingestión en el repositorio de metadatos.

Los constantes cambios en la infraestructura del Data Lake, con el fin de mejorar el almacenamiento y procesamiento de los datos, han consumido la totalidad del tiempo planificado para la generación de las estadísticas y la inserción de estas en el repositorio de metadatos.

De manera que, se ha dado mayor prioridad a la realización de otras tareas del proyecto por frente a esta, con el objetivo de dotar al sistema de una mayor consistencia, entregando así a la empresa cliente un sistema mucho mas robusto.

En cuanto al resto de la planificación, no se han producido desviaciones, se ha seguido la planificación de los *sprint* y *sprint meeting* correctamente.

2.4. Identificación de los costes

Identificaremos los costes tomando como referencia las horas totales dedicadas al proyecto. Se calculan los costes de los recursos humanos, recursos de hardware y software, amortizaciones y gastos indirectos.

2.4.1. Costes directos

Recursos Humanos

El gasto principal de este proyecto es el de los recursos humanos. Se pueden diferenciar cuatro roles que tomarán parte en este proyecto: el rol de jefe de proyecto, analista, desarrollador y *tester*. Para poder calcular los gastos asociados a los recursos humanos necesitaremos conocer tanto el precio por hora de cada uno de estos roles, como las horas que dedicará cada uno al proyecto.

Rol	Salario
Jefe de proyecto	23 €/h
Analista	18 €/h
Desarrollador	14 €/h
<i>Tester</i>	17 €/h

Tabla 2.1: Salario estimado asociado a cada rol involucrado en el proyecto

El presupuesto se basará en los precios por rol de la Tabla 2.1, obtenidos de un estudio de remuneraciones en el sector tecnológico [7] llevado a cabo por Page Personnel^{IV} este mismo año. Este estudio nos aporta el salario en bruto anual, para saber el salario por hora de cada uno de los roles se ha usado calculado aplicando la siguiente fórmula:

$$\text{Salario por hora} = \frac{\text{€}}{\text{año}} \cdot \frac{\text{año}}{12 \text{ meses}} \cdot \frac{\text{mes}}{20 \text{ días laborables}} \cdot \frac{\text{día laborable}}{8 \text{ horas}}$$

Para calcular el coste en recursos humanos se seguirá la planificación que se ha realizado previamente, en concreto se seguirá el diagrama de Gantt. En esta planificación se han previsto hacer 12 *sprints*. Si se considera que por cada *sprint*, al que se le dedica un total de 50 horas (25 horas por semana), se distribuyen las horas entre los diferentes roles como indica la Tabla 2.2.

Rol	Horas
Jefe de proyecto	3 h
Analista	15 h
Desarrollador	24 h
<i>Tester</i>	8 h

Tabla 2.2: Distribución de las horas de cada *sprint* entre los roles

^{IV}Consultora líder en selección y trabajo temporal especializado, forma parte de Page Group

Se puede estimar que el coste total de recursos humanos será de 9.730 €. En la Tabla 2.3 se puede ver el detalle del coste por cada rol, que se ha calculado de la siguiente manera:

$$\text{Coste por rol} = \frac{\text{€}}{\text{hora}} \cdot \frac{\text{horas}}{\text{sprint}} \cdot 12 \text{ sprints}$$

Rol	Coste
Jefe de proyecto	828 €
Analista	3.240 €
Desarrollador	4.032 €
<i>Tester</i>	1.632 €
Total	9.732 €

Tabla 2.3: Coste estimado de los recursos humanos por cada rol

Recursos Hardware

Durante todo el proyecto se ha trabajado en dos máquinas. El equipo donde se ha llevado a cabo todo el desarrollo, donde trabaja el autor de este trabajo de final de grado, y la máquina donde se ha desplegado el repositorio de metadatos, la API y el Quarry. Esta máquina forma parte de un *cluster* de tres, en el que se está desarrollando el proyecto Big Data Analytics Lab. Las tres máquinas son máquinas virtuales alojadas en un servidor. Se estima que se trabajará con todo el *cluster* durante los últimos 4 *sprints* de la planificación.

Por lo tanto el hardware utilizado durante todo el proyecto consta de una máquina de desarrollo, un servidor que aloja las tres máquinas del *cluster*, y los periféricos (monitor, ratón y teclado) de la máquina donde desarrolla este proyecto. Para un uso más eficiente de los servidores compartimos los recursos del servidor donde alojamos las tres máquinas virtuales; se estima que el consumo de recursos del servidor que hacen estas tres máquinas es de un 40%. Para los cálculos se estima que el consumo de las máquinas virtuales es el mismo, por lo tanto el consumo de una de ellas es una tercera parte del total.

Los costes de los recursos que veremos a continuación han sido facilitados por la responsable del Área de Sistemas, Aulas y Comunicaciones de inLab FIB, Antonia Gómez.

Hardware	Coste
Servidor	705,44 (1/3 del 40% de 5.290,82) €
Máquina desarrollo	605 €
Periféricos	200 €
Total	1.510,44 €

Tabla 2.4: Coste del hardware utilizado durante los 12 *sprints*

Hardware	Coste
Servidor	1.410,89 (2/3 del 40 % de 5.290,82) €
Total	1.410,89 €

Tabla 2.5: Coste del hardware adicional utilizado durante 4 *sprints*

Para conocer el coste real de este hardware asociado al desarrollo del proyecto calcularemos la amortización de estos costes, teniendo en cuenta que dos máquinas tan solo se utilizarán durante 4 *sprints*. El coeficiente de amortización para programas y equipos informáticos es de 26 % por año [8]; la duración total del proyecto equivale a 75 días a jornada completa y la duración correspondiente al uso de las dos máquinas adicionales es de 25 días.

$$\text{Duración proyecto} = 12 \text{ sprints} \cdot \frac{50h}{\text{sprint}} \cdot \frac{\text{día}}{8h} = 75 \text{ días}$$

$$\text{Uso cluster} = 4 \text{ sprints} \cdot \frac{50h}{\text{sprint}} \cdot \frac{\text{día}}{8h} = 25 \text{ días}$$

La amortización de estos gastos y, por lo tanto, el coste de los recursos Hardware correspondiente a este proyecto es:

$$\text{Amortización} = \left(1.510,44 \text{ €} \cdot \frac{75}{365} + 1.410,89 \text{ €} \cdot \frac{25}{365} \right) \cdot 0,26 = 105,82 \text{ €}$$

Recursos Software

Durante el desarrollo de este proyecto se ha optado siempre por el software Open Source, por lo tanto, gran parte del software utilizado para llevar a cabo el desarrollo no tiene coste, como se puede apreciar en la Tabla 2.6.

Software	Coste
Windows 8.1	45,4 €
Ubuntu 12.04.5 LTS	0 €
Eclipse	0 €
Bitvise SSH Client	0 €
Virtuoso OpenSource	0 €
Apache Tomcat	0 €
Mongo DB	0 €
Node.js	0 €
Maven	0 €
Quarry	0 €
Varunya's API	0 €
LaTeX	0 €
Total	45,4 €

Tabla 2.6: Coste del software utilizado durante el desarrollo del proyecto

Los costes de los recursos que vemos en este apartado, al igual que con los Recursos Hardware, han sido facilitados por la responsable del Área de Sistemas, Aulas y Comunicaciones de inLab FIB, Antonia Gómez.

Para conocer el coste real de este software asociado al desarrollo del proyecto calcularemos la amortización de estos costes, como hicimos con los costes en Recursos Hardware. El coeficiente de amortización para programas y equipos informáticos es el mismo, 26 % por año [8], y se utilizará este software durante toda la duración del proyecto, que equivale a 75 días a jornada completa (cálculo hecho en el cálculo de la amortización en Recursos Hardware).

La amortización de estos gastos y, por lo tanto, el coste de los recursos Software correspondiente a este proyecto es:

$$\text{Amortización} = 45,4 \text{ €} \cdot \frac{75}{365} \cdot 0,26 = 2,43 \text{ €}$$

2.4.2. Costes indirectos

También tenemos que contemplar los gastos indirectos necesarios para desarrollar el proyecto. En inLab FIB se valoran los costes indirectos de los proyectos a partir de un porcentaje de los gastos directos; actualmente este porcentaje es del 7%.

Estos gastos contemplan el espacio y material de oficina utilizado, los gastos de gestión, los gastos de agua y luz, entre otros.

2.4.3. Coste total

El coste total estimado del proyecto se obtiene a partir de la suma de los costes directos y los costes indirectos. Los costes directos son la suma de los Recursos Humanos, los Recursos Hardware y los Recursos Software.

Recursos	Coste
Humanos	9.732 €
Hardware	105,82 €
Software	2,43 €
Total	9.840,25 €

Tabla 2.7: Costes directos del proyecto

$$\text{Costes indirectos estimados} = 9.840,25 \text{ €} \cdot \frac{7}{100} = 688,82 \text{ €}$$

El coste total estimado del proyecto, teniendo en cuenta tanto los costes directos como los costes indirectos calculados en este apartado, es de:

Costes	Coste
Directos	9.840,25 €
Indirectos	688,82 €
Total	10.529,07 €

Tabla 2.8: Coste total estimado del proyecto

2.4.4. Coste a nivel de actividades de Gantt

Como se ha visto anteriormente, debido a la metodología que usamos durante el desarrollo del proyecto, la planificación ha sido dividida en *sprints*. En la distribución de estos durante el desarrollo del proyecto, que se puede observar en el Diagrama de Gantt, se ha visto que hay un total de 11 *sprints* de dos semanas de duración y 1 de tres semanas de duración.

Por lo tanto, se calculará el coste de cada uno de estos *sprints* dependiendo de su duración.

$$\text{Costes por semana de desarrollo} = \frac{10.534,18 \text{ €}}{25 \text{ semanas}} = 421,37 \text{ €}$$

$$\text{Costes } \textit{Sprint} 5 = 421,37 \text{ €} \cdot 3 \text{ semanas} = 1.264,1 \text{ €}$$

$$\text{Costes de cada uno del resto de } \textit{sprints} = 421,37 \text{ €} \cdot 2 \text{ semanas} = 842,73 \text{ €}$$

2.4.5. Control de gestión

Durante el desarrollo del proyecto pueden surgir desviaciones, las cuales afectarán al coste total del proyecto. Calcularemos el coste de estas desviaciones de la siguiente manera:

- Desviaciones en las horas de un *sprint*: (horas estimadas - horas reales) · coste estimado
- Desviaciones en el coste de un recurso: (coste estimado - coste real) · consumo real

Debido a que el presupuesto de este proyecto esta relacionado directamente con las horas dedicadas a su desarrollo. Una desviación de este tiempo implicaría un notable incremento del coste final del proyecto.

Para evitar un aumento importante del presupuesto el equipo se limitará a cumplir con la planificación acordada. Si el cliente solicita el desarrollo de nuevas funcionalidades que provocan desviaciones importantes en la planificación y, por lo tanto, en el presupuesto, se tendrá que prescindir de alguna otra funcionalidad con tal de no alterar demasiado la planificación. Será el cliente el encargado de priorizar las funcionalidades que aporten más valor al producto y, por lo tanto, formen parte del alcance del proyecto.

Si se da el caso que el cliente decide que hay una funcionalidad imprescindible, pero no se previó al inicio y es más importante desarrollarla que otra funcionalidad prevista, entonces se desarrollará la nueva funcionalidad y no se desarrollará la inicial. Esta decisión será tomada por el cliente y consensuada con el equipo.

2.5. Informe de sostenibilidad

Valoraremos la sostenibilidad y compromiso social del proyecto analizando la sostenibilidad económica, social y ambiental del mismo. Finalmente, otorgaremos una puntuación de sostenibilidad a cada uno de estos aspectos desde diferentes puntos de vista: planificación, resultados y riesgos. Aunque la valoración actual se llevará a cabo teniendo en cuenta únicamente la planificación.

2.5.1. Sostenibilidad económica

Como se ha visto durante el estudio de los costes de este proyecto en la sección 2.4, el grueso del presupuesto de este proyecto reside en los costes de Recursos Humanos. Esto se debe a que se han minimizado los gastos para desarrollar el proyecto en los demás ámbitos. Como en el referente al software, en el que utilizamos herramientas Open Source y proyectos académicos, como Quarry o la API.

De esta manera, reutilizando tecnologías y colaborando con algunos proyectos existentes, reducimos mucho el tiempo de desarrollo del proyecto y, por lo tanto, su coste final. Este ajuste del presupuesto ha hecho que la empresa cliente haya visto con buenos ojos la viabilidad de este proyecto.

Finalmente, una de las tareas planificadas al principio del proyecto no se llegó a realizar, puesto que el cliente priorizó otras tareas por frente de esta. Pese a esto, se cumplió con el presupuesto de la planificación, ya que el tiempo dedicación del equipo al proyecto fue el previsto. Se realizaron otras tareas en el tiempo en el que debía realizarse esta tarea que quedó fuera de la planificación final.

2.5.2. Sostenibilidad social

Como se ha dicho anteriormente, este proyecto nace de la necesidad de una empresa por controlar de forma eficiente los datos con los que trata día a día. Por lo tanto, este proyecto no se hace con un único fin académico, sino también para satisfacer una necesidad real que existe en la sociedad ahora mismo y que así nos ha manifestado el cliente.

Con el desarrollo del proyecto mejoramos el proceso de obtención de los datos por el que hasta ahora tenían que pasar los analistas de datos de la empresa. De manera que los usuarios ya no tienen que pasar por un proceso tan tedioso, mejorando así su calidad de vida y su productividad, ya que podrán dedicar el tiempo que dedicaban a este proceso a otras tareas al tratarse de un proceso automático la obtención de estos datos.

Tras finalizar el desarrollo del proyecto, el cliente quedó satisfecho por el trabajo realizado en este proyecto, y este ha tenido una buena acogida entre los empleados de la empresa, posibles usuarios del sistema.

2.5.3. Sostenibilidad ambiental

El uso de software Open Source y la colaboración con proyectos académicos es la principal razón que hace que tengamos que dedicar mucho menos tiempo y recursos al desarrollo de este proyecto. Por lo tanto, se minimiza también el impacto ambiental del proyecto.

Además, las máquinas donde se ha instalado la plataforma Big Data son máquinas virtuales alojadas en un servidor donde también se llevan a cabo otros proyectos. De manera que compartimos los recursos del servidor donde trabajamos para disminuir la huella ecológica causada por el proyecto.

Teniendo en cuenta el uso de las máquinas virtuales durante el transcurso del proyecto, el consumo de recursos que hacen estas del servidor, y que las usaremos un máximo de 5 horas al día (en el peor caso, no trabajamos sobre las máquinas virtuales el 100% del tiempo). Partiendo de un consumo diario de 4,42 kWh del servidor, el consumo energético diario que hace este proyecto es de:

$$\text{Consumo diario} = 4,42 \text{ kWh} \cdot \frac{40}{100} \cdot \frac{5}{24} \cdot \left(\frac{1}{3} + \frac{2}{3} \cdot \frac{4}{12} \right) = 0,205 \text{ kWh}$$

2.5.4. Matriz de sostenibilidad

Sostenibilidad	Económica	Social	Ambiental
Planificación	9	7	9
Resultados	9	7	8
Riesgos	-5	0	0
valoración total	44		

Tabla 2.9: Matriz de sostenibilidad del trabajo de final de grado

2.6. Identificación de leyes y regulaciones

Como se ha mencionado anteriormente, el software en el que se basa este TFG es OpenSource. El código fuente de todo el software resultado de proyectos académicos, como Quarry y la API, que se ha utilizado en este proyecto está publicado en UPCommons y se ha obtenido autorización expresa del autor para poder utilizarlo y modificarlo, para adaptarlo a los requisitos de nuestro proyecto.

Otro software como Virtuoso OpenSource y Apache Tomcat están publicados bajo la licencia *GNU General Public License version 2* (GPLv2^V) y Apache License, respectivamente. Por lo tanto, tanto el uso como la modificación del código para su posterior uso, que es lo que se ha hecho en este proyecto, está permitido en el software liberado bajo estas licencia.

También se han tenido en cuenta leyes y regulaciones respecto a la privacidad de los datos utilizados en este proyecto, ya que todos los datos son de carácter personal. Por lo tanto, estos datos han sido anonimizados antes de ser utilizados, cumpliendo así con la LOPD^{VI}.

^V<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

^{VI}Ley Orgánica de Protección de Datos

Capítulo 3

Arquitectura del sistema

En esta sección se explica como se compone el sistema Big Data de Big Data Analytics Lab. Para facilitar la comprensión, se explicarán las diferentes partes del sistema y su funcionamiento de forma aislada, para finalmente ver como encajan cada una de estas partes y tener una visión global del proyecto.

No se hace un análisis exhaustivo de cada una de las herramientas e integraciones llevadas a cabo en el sistema, sino que se pretende dar un conocimiento general sobre la composición de cada una de las partes de este sistema. Profundizando más adelante, en capítulos posteriores de esta memoria, en la parte del sistema en la cual se centra este Trabajo Final de Grado.

3.1. Data Lake

Se ha hecho referencia varias veces al concepto Data Lake durante esta memoria, pero solo ha sido introducido brevemente hasta ahora. Llevamos décadas acostumbrados a almacenar la información en bases de datos y sistemas de ficheros convencionales.

Un Data Lake dista mucho de estos sistemas tradicionales, la información ya no se guarda tan solo en una única máquina, sino en varias. Nos encontramos con sistemas de ficheros y bases de datos distribuidas, y no tan solo se distribuye la información entre las máquinas, sino que también se replica, de manera que tenemos varias copias de los datos distribuidas entre las máquinas mejorando así tanto el acceso como la seguridad de estos.

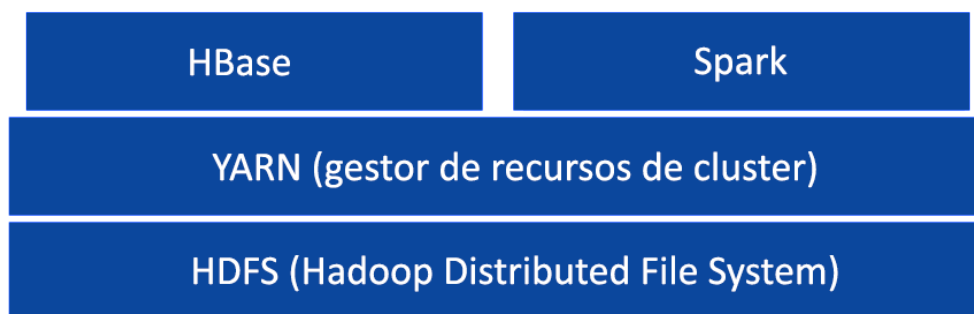


Figura 3.1: Esquema de los diferentes componentes del Data Lake.

En Big Data Analytics Lab el Data Lake está compuesto de herramientas Open Source del ecosistema Apache Hadoop. Como podemos ver en la Figura 3.1, esta formado por el sistema de ficheros distribuido *Hadoop Distributed File System* (HDFS), el gestor de recursos de *cluster* YARN, la base de datos distribuida no-relacional HBase, y la plataforma de análisis y procesamiento de datos Spark.

HDFS distribuye la información a lo largo del *cluster* de máquinas que componen el Data Lake, replicando la información en diferentes máquinas. YARN gestiona los recursos de todas las máquinas del *cluster*, distribuye los procesos entre las diferentes máquinas dependiendo del consumo de recursos actual de estas e intentando sacar partido de la localidad de los datos. HBase es una base de datos distribuida basada en columnas que nos permite guardar la información en HDFS y acceder posteriormente a ella de una forma más ordenada, como si extrajéramos datos de una base de datos en lugar de ir directamente al sistema de ficheros a buscarla. Por último, Spark nos permite procesar datos, de diferentes fuentes de datos, en memoria, con lo cual la eficiencia a la hora de ejecutar análisis de datos que utilizan algoritmos que realizan múltiples iteraciones sobre un mismo conjunto de datos se dispara.

3.2. Repositorio de metadatos

El repositorio de metadatos es la parte del sistema Big Data en la cual se centra este Trabajo Final de Grado. Este repositorio alberga metadatos semánticos, en forma de ontología, que nos proporcionan información sobre el contenido, el significado y la relación de los datos del Data Lake. Se compone de dos herramientas fundamentales, como podemos ver en la Figura 3.2.

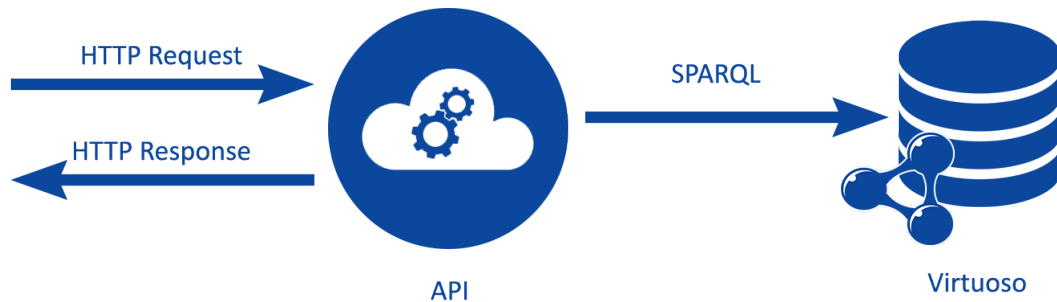


Figura 3.2: Esquema de los diferentes componentes del repositorio de metadatos.

Por una parte, Virtuoso OpenSource, un sistema de gestión de bases de datos relacionales (RDBMS^I) que puede representar los datos como tablas relacionales y/o grafos de propiedades. Entre otras cosas, también nos permite guardar todo el contenido de nuestro repositorio en formato RDF, actuando a modo de Triplestore. De esta manera podemos guardar la ontología en forma de grafo en este RDBMS, con el cual interactuaremos mediante SPARQL (*SPARQL Protocol and RDF Query Language*)^{II} para almacenar y lanzar consultas sobre la base de datos.

^IRelational Database Management System

^{II}Protocolo SPARQL y lenguaje de consulta RDF

Por otra parte, una API, fruto de la tesis de máster de una estudiante de la UPC, Varunya Thavornun [9]. Esta API permite interactuar con Virtuoso mediante peticiones HTTP (*Hypertext Transfer Protocol*). Por lo tanto, permite gestionar las tripletas almacenadas en Virtuoso, de manera que podemos consultar, insertar y eliminar contenido sin tener conocimiento alguno de SPARQL, actuando como una capa de abstracción para el usuario.

De esta manera, la interacción con Virtuoso se puede realizar únicamente a través de la API mediante los métodos GET, POST y DELETE de HTTP, que nos permiten pedir, insertar y eliminar contenido, respectivamente, representando los datos en ficheros JSON (*JavaScript Object Notation*).

Por ejemplo, para obtener datos del repositorio, tan solo habría que hacer una petición HTTP concreta mediante el método GET a la API y esta nos devolvería en un fichero JSON el resultado de nuestra consulta, y en caso de insertar contenido en el repositorio se envían los datos en un fichero JSON en el cuerpo de la petición a la API mediante el método POST.

En la Sección 4.2 del capítulo Desarrollo del proyecto de esta memoria se explica en detalle el trabajo realizado, las adaptaciones y funcionalidades nuevas implementadas sobre estas herramientas para cumplir con las necesidades requeridas por el proyecto.

3.3. Quarry

Quarry[1] es la herramienta utilizada en Big Data Analytics Lab para dotar al usuario de un método de interacción con el sistema totalmente transparente a todas las tecnologías de las que está compuesto.

Esta herramienta es capaz de mostrar una ontología al usuario y que este pueda extraer los datos que quiere de la base de datos, seleccionando los conceptos en los que esté interesado y aplicando una serie de reglas sobre estos. De esta manera, Quarry, a partir de los archivos que contienen la ontología y toda la información necesaria sobre todos los conceptos de esta (por ejemplo, los *mappings* de los conceptos hacia la base de datos), es capaz de conocer con precisión el lugar en el que se encuentran los datos que el usuario quiere extraer.

En la Figura 3.3 podemos ver gráficamente el funcionamiento de Quarry junto a todas las partes que entran en juego.

En este Trabajo Final de Grado no se ha trabajado sobre esta herramienta salvo para estudiar ligeramente su funcionamiento para hacer algunas modificaciones sobre la API. Se habla sobre esto más adelante, en la Sección 4.2.3 del capítulo Desarrollo del proyecto de esta memoria.

3.4. Visión global

En poner todos estos elementos, vistos en los apartados anteriores de esta sección, en conjunto se obtiene una imagen global de la solución implementada por el equipo de Big Data Analytics Lab, que es la que se puede ver en la Figura 3.4.

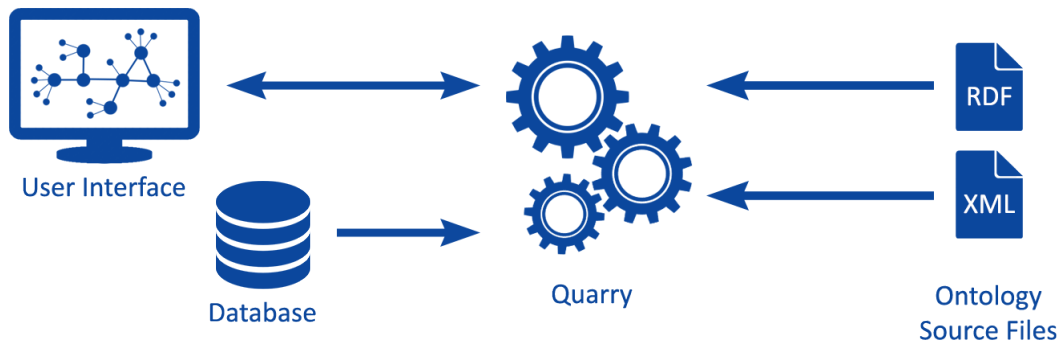


Figura 3.3: Esquema de funcionamiento del Quarry.

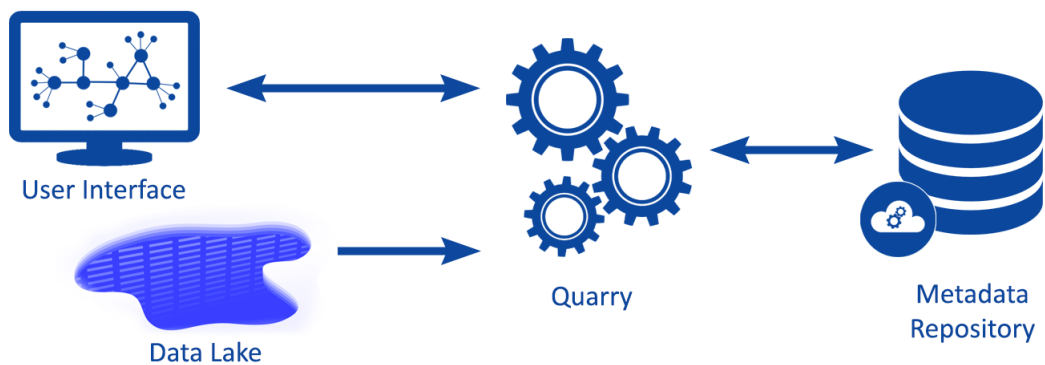


Figura 3.4: Esquema de las diferentes partes del sistema Big Data en conjunto.

Si lo comparamos con el esquema de funcionamiento de Quarry (ver en Figura 3.3) se puede ver que ambos tienen una estructura similar. Esto se debe a que Quarry es el elemento central del sistema, encargado de conectar al usuario, al Data Lake y al repositorio de metadatos.

Partiendo de lo explicado en el apartado anterior sobre Quarry, en la solución final nos encontramos con el mismo esquema, salvo por algunos cambios. Uno de estos cambios se refleja en la parte encargada de almacenar los datos, en el caso de la solución implementada por el equipo de Big Data Analytics Lab estamos hablando de una Data Lake compuesto por diferentes herramientas (ver Sección 3.1) en lugar de una base de datos tradicional como utiliza Quarry. Además, en nuestra solución Quarry en lugar de ir a buscar la ontología a unos ficheros de un sistema de ficheros local, pide esta información a la API (vista en la Sección 3.2).

De esta manera, cuando el usuario selecciona una ontología, Quarry carga del repositorio de metadatos toda la información necesaria. Entonces, en el momento en que deba acceder al Data Lake para ir a buscar los datos, sabrá exactamente donde ir a buscarlos, ya que habrá cargado los datos de los *mappings* de los conceptos de esa ontología previamente.

Capítulo 4

Desarrollo del proyecto

Durante el desarrollo del proyecto se ha trabajado paralelamente en las diferentes fases del proyecto, explicadas en la Sección 2.3.1 de esta memoria.

4.1. Ontología

Entendiendo el modelo de negocio del cliente y sus necesidades, debía modelarse una ontología lo más completa posible para describir los datos del cliente que se iban a trasladar a la nueva plataforma Big Data. A partir de la información proporcionada sobre los datos que debían ser representados en esta ontología se empezaron a hacer los primeros esbozos del modelo.

En los siguientes apartados de esta sección veremos como se fue modelando la ontología durante el transcurso del proyecto, añadiendo en determinados momentos tanto información adicional sobre los datos que ya estaban representados en la ontología, como información sobre nuevos datos que debían ser representados. De manera que veremos los cambios que se fueron aplicando sobre el modelo de la ontología cronológicamente.

Además, previamente a la explicación de la evolución del modelo de la ontología, se explicará la estructura de la ontología, viendo las diferentes partes de las que se compone.

4.1.1. Composición de la ontología

La ontología se ha dividido en diferentes partes, separando el contenido de esta en grafos independientes, a los que a partir de ahora llamaremos *namespaces*.

El contenido de cada *namespace* tiene un propósito y estructura diferente, de ahí que se haya decidido separarlos. Aún así, como la información que contienen hace referencia a los mismos datos, encontramos conexiones entre los diferentes *namespaces*, de manera que toda la información sobre unos determinados datos en la ontología quedará conectada y fácilmente accesible.

Como se puede ver en la Figura 4.1, se han definido tres *namespaces* diferentes, debido al contenido que tenemos que diferenciar en la ontología. El *namespace* de vocabulario es el principal, en él se hayan todos los conceptos de negocio con los que los empleados de la compañía trabajan a diario. Además, este *namespace* es el que conecta con los otros dos, el de origen y *mappings*. Las flechas rojas de la Figura 4.1 son las que representan

estas conexiones.

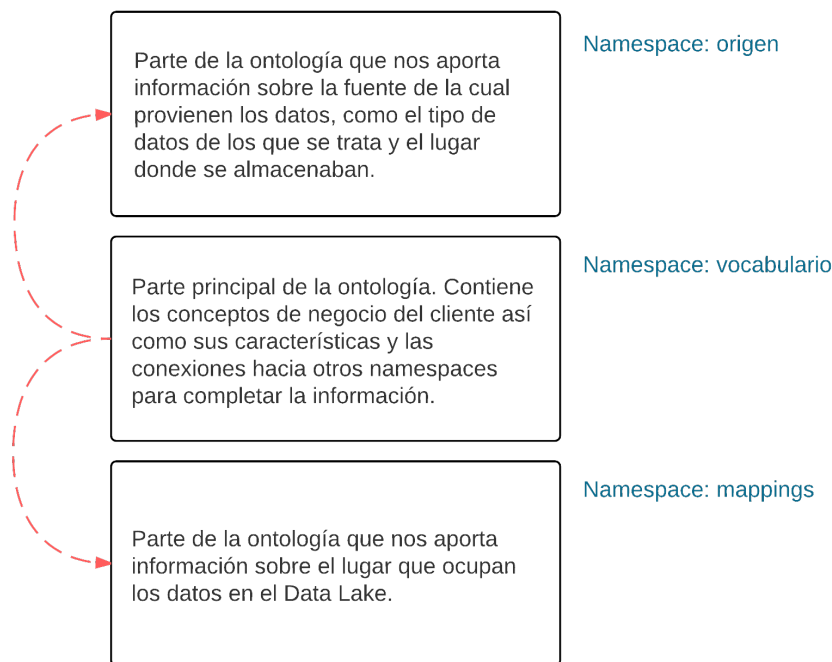


Figura 4.1: Esquema de los diferentes namespaces de la ontología y su contenido.

El *namespace* de origen contiene información sobre el lugar del cual provienen los datos, como la tabla y el atributo físico en el que se representaban estos datos en la base de datos operacional, y características sobre estos datos, como el tipo de datos del que se trata (*boolean*, *number*, *char*, ...).

Finalmente, el *namespace* de mappings nos indica en qué lugar se encuentran exactamente los datos que representan cada concepto del *namespace* de vocabulario en el Data Lake.

4.1.2. Modelo básico

Entre la información de la que se disponía en un principio para realizar un primer modelo de la ontología se pudieron extraer una serie de conceptos de negocio de alto nivel, los conceptos principales de la ontología, que están relacionados entre sí mediante diferentes predicados, como se puede ver en el esquema de ejemplo de la Figura 4.2.

Además, se extrajeron de esta misma información otros conceptos de más bajo nivel, a los que llamaremos a partir de ahora Características, que están relacionados con los conceptos principales mediante el mismo predicado *#hasFeature* (Figura 4.3).

La razón por la cual se asigna el mismo predicado a todas las relaciones entre Conceptos de negocio y Características es para reducir la complejidad de algunas consultas SPARQL, facilitar el trabajo a Virtuoso y obtener las tripletas de forma eficiente.

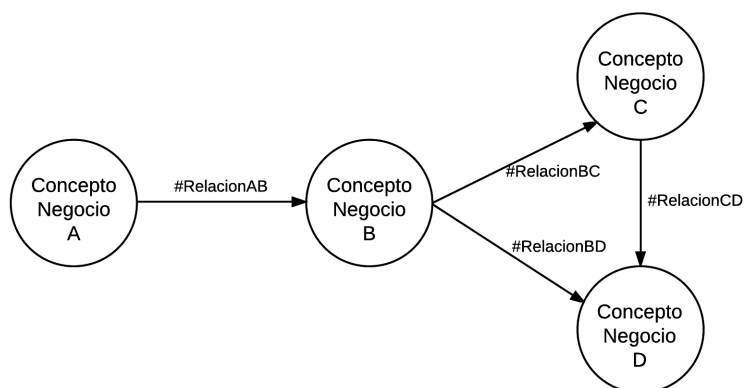


Figura 4.2: Esquema básico de ejemplo de conceptos de la ontología.

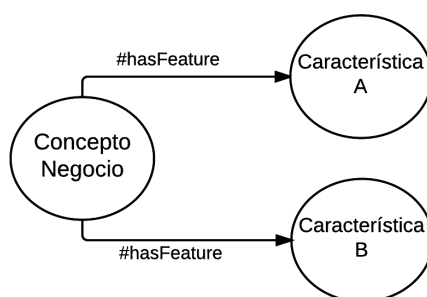


Figura 4.3: Relación entre conceptos principales y características de la ontología.

Por ejemplo, al modelar esta parte de la ontología de esta forma, si quisiéramos obtener todas las Características asociadas a un Concepto de negocio X la consulta SPARQL que se lanzaría sería muy sencilla, como se puede ver en el Fragmento de código 4.1.

Fragmento de código 4.1: Ejemplo de simplicidad de las consultas SPARQL

```

SELECT ?objeto
FROM <repositorio>
WHERE {
  ?sujeto ?predicado ?objeto .
  FILTER(REGEX(STR(?sujeto), '#ConceptoX') &&
    REGEX(STR(?predicado), '#hasFeature'))
}

```

Todo lo explicado hasta ahora sobre este primer modelo de la ontología corresponde al *namespace* de vocabulario, en el cual trabajamos con la semántica de los datos (Conceptos de negocio, Características y relaciones), que es la parte de la ontología más cercana al lenguaje natural.

Para facilitar la comprensión, veremos un ejemplo de un posible caso de uso real para esta parte de la ontología.

Ejemplo

Supongamos que la empresa cliente, para la cual se está desarrollando el proyecto, es una aerolínea llamada *Landing*. Esta compañía nos ha proporcionado la información necesaria para empezar a modelar la ontología, entre la cual disponemos de información

como Conceptos de negocio y Características asociadas a estos.

Partiendo del modelo básico de la ontología que hemos definido hasta ahora, volcando alguna de la información de la que disponemos de esta aerolínea en este modelo se obtiene la ontología de la Figura 4.4. En esta figura podemos ver como los Conceptos de negocio de *Landing* (círculo negro) representan un concepto a más alto nivel que las Características (círculo azul), que tratan de cosas más concretas, propias de cada Concepto de negocio.

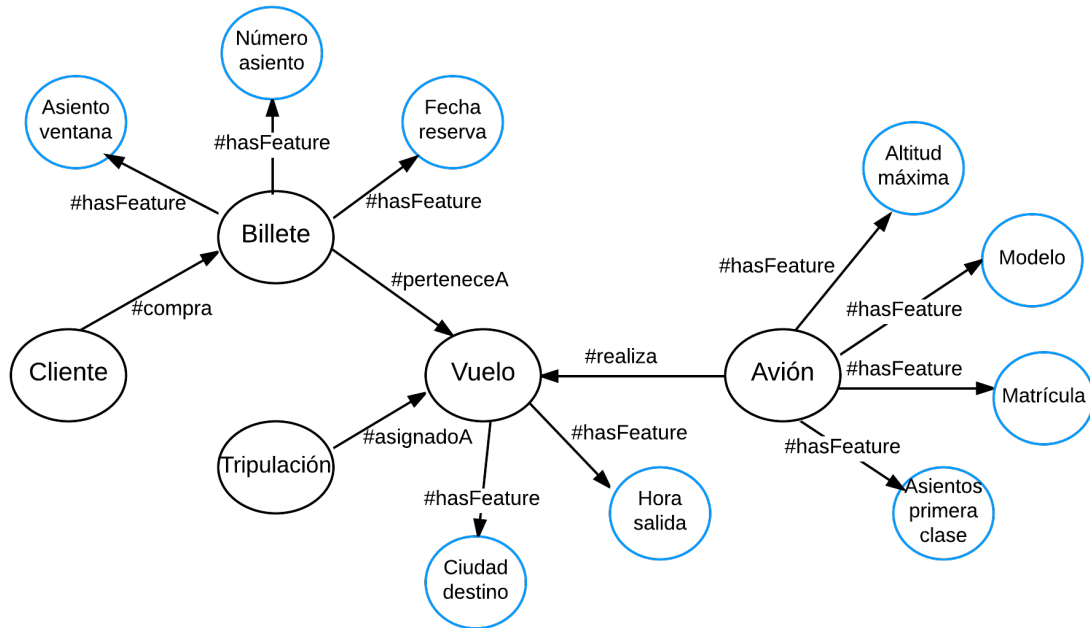


Figura 4.4: Ejemplo del modelo básico de la ontología.

Finalmente, podría parecer tras ver este ejemplo que una ontología, o al menos esta parte, es muy similar a un diagrama UML

podemos destacar que un Concepto de negocio de por sí solo no nos aporta información, se podría entender como un conjunto de Características que representan o nos aportan información sobre una misma entidad.

Origen de los datos

En este modelo más básico de la ontología también se modeló el *namespace* de origen, ya que entre los metadatos de los que disponemos también encontramos la información sobre de dónde vienen los datos. Es decir, se sabe en que lugar de la base de datos operacional se almacenaba cada una de las Características del *namespace* de vocabulario, ya que disponemos de la tabla y atributos físicos, además de algunos metadatos adicionales que nos proporcionan información que describen el tipo de dato final que representa cada Característica. Esto último tiene una gran utilidad una vez se hayan migrado los datos al Data Lake, ya que una vez allí no sabremos de qué tipo de datos se tratan y esto nos ayudará a identificarlos y tratarlos adecuadamente.

En la Figura 4.5 podemos ver como encajan estos metadatos en el modelo de la ontología que habíamos visto hasta ahora, así como la manera en la que se relacionan con los

metadatos del *namespace* de vocabulario. La relación entre diferentes *namespaces* se establece desde el de vocabulario, los Conceptos de negocio y Características de este *namespace* representan respectivamente tablas y atributos en el *namespace* de origen. Entonces, todo Concepto de negocio del *namespace* de vocabulario formará parte de, al menos, una tripleta con el predicado *#comesFrom*, mientras que toda Característica formara parte de exactamente una tripleta con este predicado. Esta tripleta nos indicará la equivalencia entre las entidades del *namespace* de vocabulario y origen.

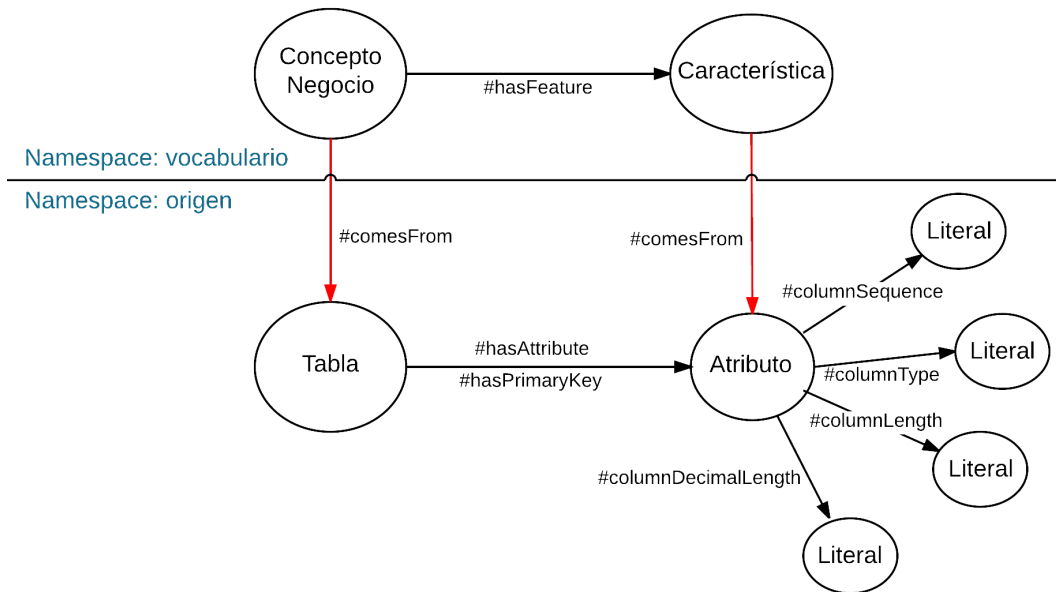


Figura 4.5: Modelo básico de la ontología.

Se ha dicho que los Conceptos de negocio pueden formar parte de varias tripletas con el predicado *#comesFrom*, esto es debido a que un Concepto de negocio puede estar compuesto por más de una tabla física en la base de datos operacional, como se verá más adelante.

Además, en la figura anterior (Figura 4.5) vemos como en el *namespace* de origen las tablas y atributos se relacionan mediante tripletas con el predicado *#hasAttribute*, que indica que el Atributo pertenece a esa Tabla, y *#hasPrimaryKey*, que nos indica que este Atributo es clave primaria de esta Tabla, en caso de existir esta relación entre la Tabla y el Atributo.

Finalmente, se puede apreciar que los Atributos tienen ciertas propiedades que lo definen, tanto al propio atributo como al lugar que ocupaban en el esquema relacional de la base de datos operacional. Estas propiedades están representadas en la ontología como tripletas que en uno de sus extremos, en el del objeto, tienen un valor literal en lugar de una URI. Representamos estos valores mediante este tipo de tripletas debido a que se tratan de valores fijos que no tienen entidad suficiente como para tratarse como a un objeto, ya que únicamente nos aportan información sobre un sujeto pero su valor no va más allá del propio sujeto y, por lo tanto, no nos interesa relacionarlo con ninguna otra entidad.

4.1.3. Modelo avanzado

Los siguientes metadatos de los que se dispusieron fueron sobre nuevas Características de los mismos Conceptos de negocio que tratamos anteriormente. Esta vez se trataba de datos almacenados de una forma muy peculiar, en unas tablas de la base de datos operacional a las que en esta compañía llaman tablas *cremallera*. Pero, antes de ver como se modeló esta parte de la ontología, veamos a que se refieren con tablas cremallera.

¿Qué son y cómo funcionan las tablas cremallera?

Anteriormente se ha mencionado que los Conceptos de negocio están asociados a una o más tablas. Esto es porque, en un principio, todo Concepto de negocio está representado por una tabla en la base de datos operacional, pero con el tiempo la compañía necesitó muchos más atributos para representar algunos Conceptos de negocio. Entonces, decidieron utilizar tablas auxiliares para representar todos aquellos datos adicionales sobre algunos Conceptos de negocio.

Para algunos Conceptos de negocio, empezaron a discernir entre diferentes tipos con Características singulares que los diferenciaban. Estas nuevas Características, propias de cada Tipo de concepto de negocio, son las que debían ser representadas de alguna manera en la base de datos operacional.

Pero, ¿por qué no ampliar la tabla principal para dar cabida a estos atributos? El caso es que no se trataba de unos pocos atributos, sino de cientos de ellos por cada Concepto de negocio, y esta información no era necesaria en todo registro de la tabla. Por lo tanto, no iban a ampliar el tamaño de las tablas que más se consultaban en la base de datos operacional si no era estrictamente necesario, haciendo que estas consultas se volvieran mucho más ineficientes.

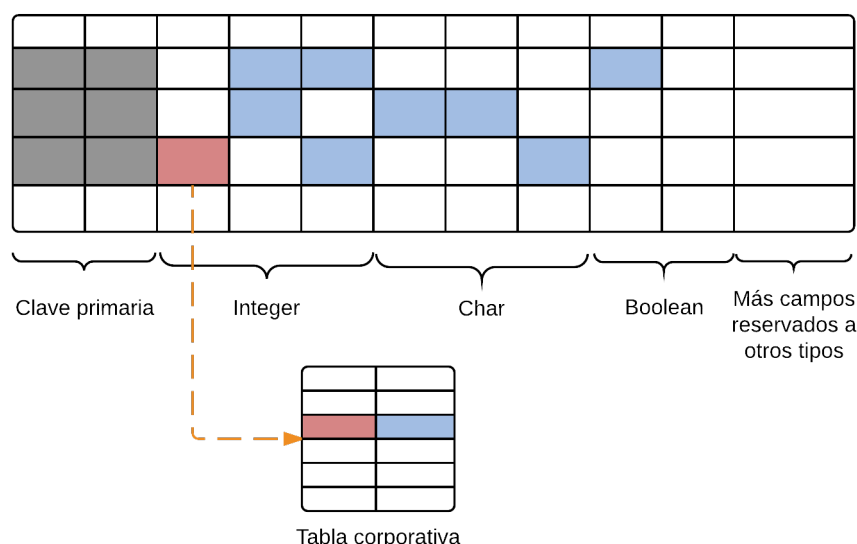


Figura 4.6: Esquema de las tablas cremallera.

Estos cientos de atributos de los que están compuestas estas tablas contienen información de todo tipo. En la Figura 4.6 podemos ver una representación del funcionamiento de estas

tablas. En esta figura vemos como todo registro está identificado por la clave primaria de la tabla principal de ese Concepto de negocio, seguido del resto de atributos de la tabla, que son un conjunto de campos de todos los tipos que podían necesitar. Por ejemplo, estas tablas podrían contener 15 atributos de tipo *Integer*, 25 de tipo *Varchar*, 10 de tipo *Boolean*, 35 de tipo *Decimal*, 10 de tipo *Bigint*, 10 de tipo *Date*, etc.

Dada una misma tabla cremallera, cada campo significa algo totalmente diferente dependiendo del Tipo de concepto de negocio, ya que cada uno utiliza los campos que le son necesarios para almacenar una información específica para ese tipo. Esto se puede apreciar en la Figura 4.6, donde cada una de las filas podría representar un Tipo de concepto de negocio diferente, y vemos como los campos utilizados por cada uno (resaltados en azul o rojo) son diferentes. Algunos Tipos de conceptos de negocio incluso utilizan más de una fila de esta tabla para poder representar toda la información.

Además, en la figura anterior apreciamos que algunos de los campos, los resaltados en rojo, hacen referencia a otra tabla a la que llaman *Tabla Corporativa*. En estos campos de la tabla cremallera no nos encontramos con el valor final del atributo, como pasa con los campos resaltados en azul, sino que damos con una referencia a una Tabla Corporativa donde se encuentran una serie de valores predefinidos, entre los cuales se haya el valor final de ese atributo. Por lo tanto, una tabla cremallera representa muchos más atributos lógicos que el número de campos físicos de los que dispone, ya que cada Tipo de concepto de negocio almacena diferente información en los mismos campos físicos de esta tabla.

Estas tablas no solo sirven como extensión de las tablas principales de cada Concepto de negocio, sino que algunos Conceptos de negocio están representados únicamente por una única tabla de este tipo aprovechando el polimorfismo de estas tablas.

Extensión del modelo de la ontología

Como se puede ver en la figura 4.7, se ha conservado el modelo anterior de la ontología y se ha extendido para poder integrar la nueva información en el modelo. En el *namespace* de vocabulario se ha añadido un nivel más entre los Conceptos de negocio y las nuevas Características que debían ser incluidas en la ontología. De esta forma, se agrupan las Características propias de cada Tipo de Concepto de negocio en este nuevo nivel que representa el Tipo de concepto de negocio per se.

En el *namespace* de origen sí que nos encontramos más cambios, en un primer vistazo a las novedades del modelo de la ontología nos encontramos con que el Atributo de la Tabla ya no representa únicamente un campo físico de esta tabla, sino también puede representar uno lógico. Estos campos lógicos representarían un campo físico y una fila de una tabla cremallera, además de una Tabla Corporativa en caso de que la información de esa Característica no se encuentre en el campo físico indicado de la tabla cremallera.

De esta manera, nos encontramos también con un nuevo nivel en el *namespace* de origen, entre el Atributo y los metadatos que determinan que tipo de dato vamos a encontrarnos. Este nuevo nivel nos indica donde se encuentra la información que estamos buscando. Así como en el modelo básico de la ontología únicamente se necesitaba el nombre del campo físico de la tabla para saber donde se encontraba el valor correspondiente a una Característica, para representar las tablas cremallera hace falta el nombre del campo físico de la tabla, la fila e incluso se podría necesitar el nombre de otra tabla adicional, la Tabla

Corporativa. Estas novedades las encontramos representadas en el nuevo modelo mediante tripletas con los predicados *#hasField*, *#hasRow* y *#hasCorporateTable*, respectivamente, y se relacionan directamente con el Atributo, que en este caso representa un campo lógico, mientras que el campo físico es el representado por los valores que se encuentran al otro extremo de las tripletas mencionadas anteriormente.

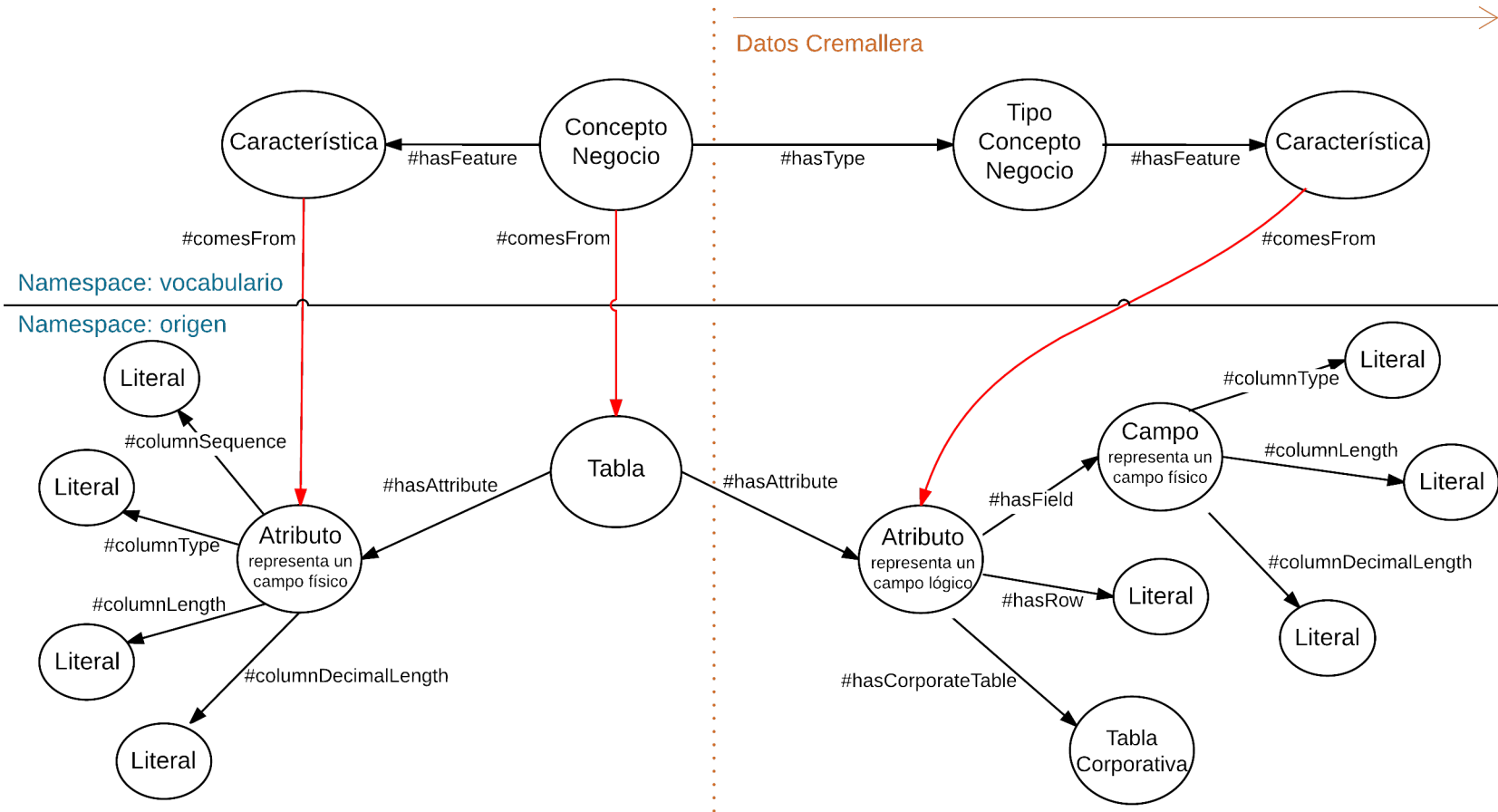


Figura 4.7: Modelo avanzado de la ontología.

4.1.4. Modelo final

La información que quedaba por modelar en la ontología era la referente a los *mappings* de los datos en el Data Lake. Es decir, la información gracias a la cual se puede saber exactamente donde se encuentran los datos correspondientes a una Característica concreta en el nuevo sistema Big Data, que guardaremos en el *namespace* de *mappings* (ver Figura 4.8).

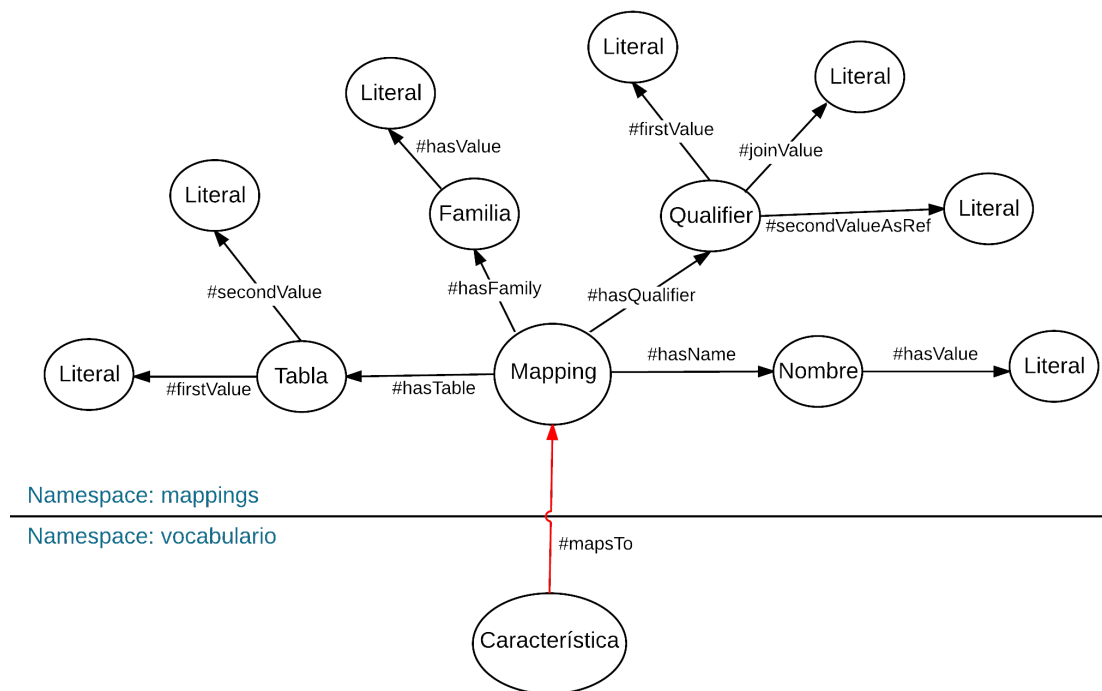


Figura 4.8: Esquema de las novedades del modelo final de la ontología.

Estos *mappings* son hacia HBase, uno de los componentes que forman el Data Lake, que vimos en la Sección 3.1. Un *mapping* hacia HBase[10] está formado por una Tabla, una Familia y un Qualifier. Esto es debido a que en HBase los datos se guardan en unas tablas que están divididas en familias, que son agrupaciones de columnas que se guardan en un mismo fichero HDFS. Por lo tanto para hacer referencia a una columna en concreto, que es representada dentro de una familia mediante un *Column Qualifier* o Qualifier, necesitamos estos tres elementos, Tabla, Familia y Qualifier.

Además, junto a cada *mapping* guardamos un Nombre que hace referencia a la Característica, ya que la información que se guarda en HBase son ficheros JSON¹ que contienen el valor de varias Características. Por lo tanto, para acabar de definir el *mapping* hacia una Característica en concreto, también se debe indicar en que posición del JSON se encuentra.

En el nuevo modelo de la ontología conectamos cada Característica del *namespace* de vocabulario con el *mapping* correspondiente del nuevo *namespace* de la ontología. El *mapping* de esta Característica está conectado con la propiedades que lo definen, que se han nombrado anteriormente, mediante los predicados *#hasTable*, *#hasFamily*, *#hasQualifier* y *#hasName*. Cada una de estas propiedades que definen el *mapping* pueden estar forma-

¹HBase guarda tiras, o *arrays*, de *bytes*, pero en este caso estas tiras de *bytes* representan ficheros JSON

das por un único valor, el cual se indica mediante el predicado *#hasValue*, o puede estar compuesto por varios valores o referencias a valores. En el segundo caso, en la ontología se guarda la información de cómo se compone el valor final de estas propiedades a partir de los otros valores o referencias.

En caso que se trate de una referencia no contamos con el valor directamente, sino con el nombre de una Característica a la que hace referencia. Entonces el valor de esta Característica será aportado por el usuario, ya que los datos han sido clasificados según el valor de esta Característica para acotar el rango de datos al que se accede y, de esta manera, optimizar el acceso a los datos. En caso de no aportarse este valor, se accederá a todos los datos, independientemente del valor de esta Característica, lo cual hará que accedamos a los datos de una forma menos eficiente. Por ejemplo, en lugar de acceder a una familia concreta de una tabla de HBase, accederíamos a todas las familias de esta tabla, lo cual haría que el acceso a estos datos no sea tan eficiente en caso que no estemos interesados en acceder a toda la tabla.

Esta información se ha modelado dividiéndola en cada una de las partes de las que se compone. Por una parte los valores o referencias, de los cuales debe guardarse el orden también, están definidos mediante el predicado *#firstValue*, en caso de que se trate del valor, o *#firstValueAsRef*, en caso de que se trate de una referencia. La primera parte de este predicado varía dependiendo del lugar que ocupa este valor o referencia al construir el valor final, pudiendo ser: *first*, *second*, *third*, etc. Por otra parte, nos encontramos con que algunos de estos valores compuestos tienen un carácter de separación entre ellos, en la ontología se ha representado mediante el predicado *#joinValue*, se entiende que este debe situarse entre todas las partes que compongan el valor final. En caso que no haya *joinValue* y se trate de un valor compuesto, se concatenarán todas las partes sin más, formando así el valor final.

En la Figura 4.9 se puede ver una imagen completa del modelo final de la ontología, tras haber aplicado todos los cambios vistos en esta sección durante el desarrollo del proyecto.

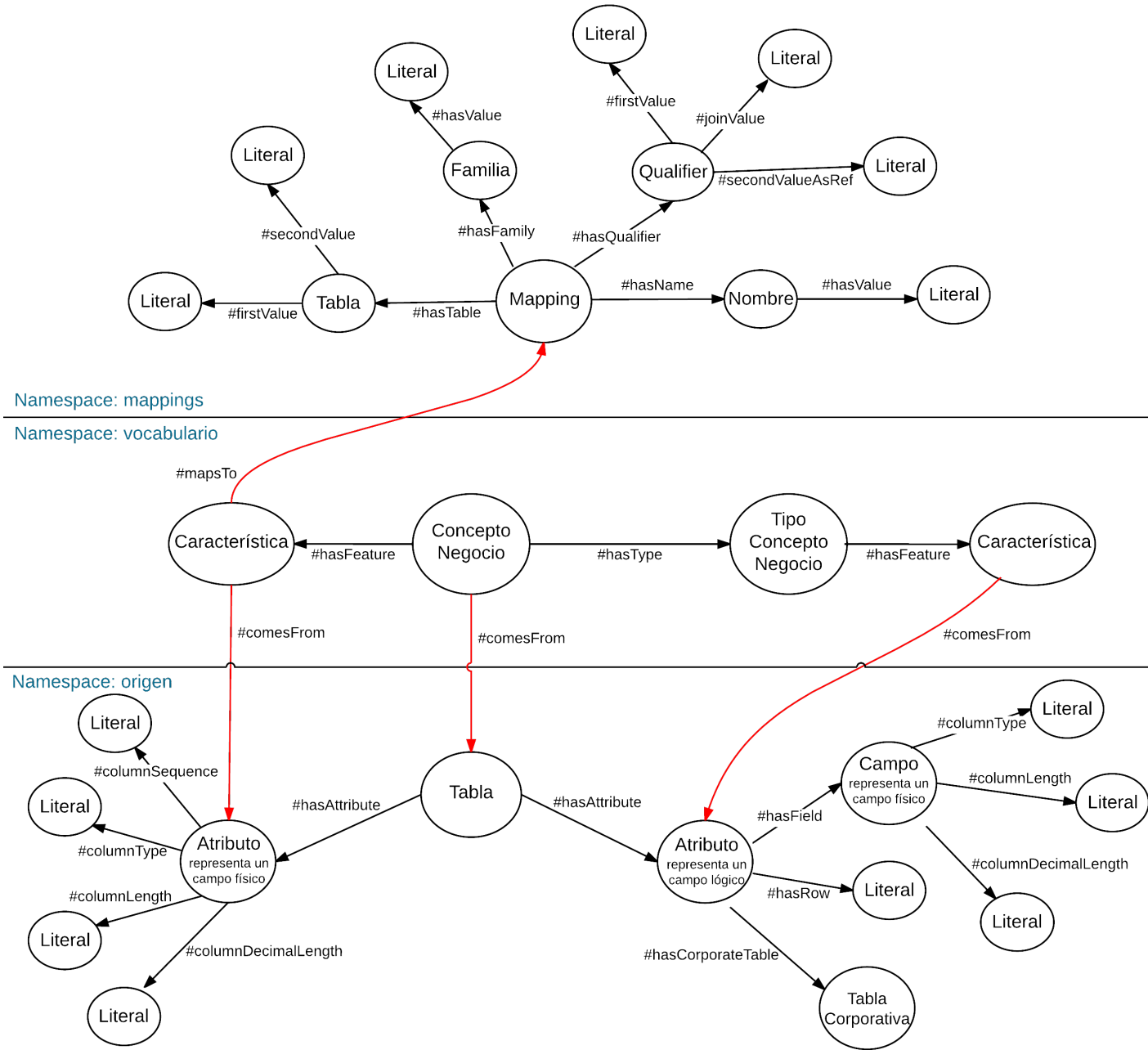


Figura 4.9: Modelo final de la ontología.

4.2. Repositorio de metadatos

4.2.1. Instalación y configuración del entorno

En los apéndices A y B se describe el proceso de instalación y configuración de varias herramientas, en un entorno basado en GNU/Linux, que deben ponerse en marcha para el correcto funcionamiento del repositorio de metadatos. En estos apéndices nos encontramos con los siguientes manuales:

- Manual de instalación y configuración de Virtuoso Opensource
- Manual de instalación y configuración de la API

Tras tenerlo todo funcionando, lejos de haber terminado con el trabajo a realizar con el repositorio de metadatos, todavía queda la parte más importante, la adaptación de la API a los requisitos del proyecto. En los próximos apartados de esta misma sección veremos en detalle los cambios realizados en los métodos existentes de la API, así como los nuevos métodos implementados para satisfacer todas las necesidades del proyecto.

4.2.2. Adaptación de la API para el uso de namespaces

En un principio la API solamente fue implementada para utilizarse junto a Virtuoso como Triplestore pero, debido a las necesidades que tenía el proyecto, debió de adaptarse la API para que permitiera la división del grafo completo, que representa toda la ontología, en grafos lógicos para representar los *namespaces*.

Por lo tanto, debieron de implementarse los cambios necesarios en el código correspondiente a la última versión de la API para el uso de *namespaces*.

La manera en la que se representa el *namespace* en la Triplestore es guardando el *namespace* junto a la URI de cada componente de la tripleta. Por lo tanto, la tripleta se guarda en el repositorio de la siguiente forma:

sujeto: < namespace: + *namespace* de la tripleta + / + URI del sujeto >
predicado: < namespace: + *namespace* de la tripleta + / + URI del predicado >
objeto: < namespace: + *namespace* de la tripleta + / + URI del objeto > o literal

Este cambio será transparente para el usuario. En ningún momento se pedirá al usuario que inserte las tripletas en este formato, ni las URIs serán mostradas al usuario de esta forma. Únicamente se le pedirá al usuario la inclusión del parámetro *namespace* en las llamadas que haga a la API cuando quiera listar, insertar o eliminar tripletas en un determinado grafo y la API se encargará de hacer el resto.

Por ejemplo, una tripleta del *namespace* origen, del que se ha hablado en la sección anterior, se guardaría con este formato en Virtuoso:

sujeto: < namespace:origen/http://urideejemplo.tfg/ontology#ConceptoNegocio >
predicado: < namespace:origen/http://urideejemplo.tfg/ontology#hasFeature >
objeto: < namespace:origen/http://urideejemplo.tfg/ontology#Caracteristica >

Para implementar este funcionamiento, se hicieron varios cambios en los métodos principales de la API.

Añadir tripletas al repositorio

Se han hecho cambios en las siguientes funciones para implementar el uso de *namespaces* al añadir tripletas al repositorio:

- *addTriples* en *APIController.java*: esta función se encarga de llevar a cabo la inserción de nuevas tripletas mediante el método POST.
 - Añadido el parámetro *namespace* a la función: este parámetro no es obligatorio de manera que se mantiene la retrocompatibilidad con la versión anterior de la API.
 - En caso que no se defina el parámetro *namespace* en el POST se le asigna un valor nulo, de esta manera la API tiene el mismo comportamiento que en su versión anterior.
 - Paso del parámetro *namespace* a la función *getFullUrl* de *APIController.java*.
- *getFullUrl* en *APIController.java*: esta función devuelve la URI absoluta en el caso que se hayan usado prefijos.
 - Añadido el parámetro *namespace* a la llamada de la función.
 - Si el parámetro *namespace* es diferente de nulo añade este al principio de la URI.

Eliminar tripletas del repositorio

Se han hecho cambios en las siguientes funciones para implementar el uso de *namespaces* al eliminar tripletas del repositorio:

- *deleteTriples* en *APIController.java*: esta función se encarga de eliminar tripletas del repositorio mediante el método POST.
 - Añadido el parámetro *namespace* a la función: este parámetro no es obligatorio de manera que se mantiene la retrocompatibilidad con la versión anterior de la API.
 - En caso que no se defina el parámetro *namespace* en el POST se le asigna un valor nulo, de esta manera la API tiene el mismo comportamiento que en su versión anterior.
 - Paso del parámetro *namespace* a la función *getFullUrl* de *APIController.java*.
- Esta funcionalidad de la API también hace uso de los cambios en la función *getFullUrl* en *APIController.java* detallados anteriormente.

Listar tripletas del repositorio

Se han hecho cambios en las siguientes funciones para implementar el uso de *namespaces* al listar tripletas del repositorio:

- *queryTuples* en *TripleStoreConnector.java*: esta función devuelve las tripletas del repositorio dada una query generada en el controlador de la API.
 - Añadido el parámetro *namespace* a la llamada de la función.
 - Paso del parámetro *namespace* a la función *queryTuples* de *DBConnector.java*.

- *queryTriples* en *DBConnector.java*: esta función devuelve las tripletas del repositorio dada una query generada en el controlador de la API.
 - Añadido el parámetro *namespace* a la llamada de la función.
 - Modificación del código para la búsqueda en el repositorio de las tripletas de el grafo correspondiente a ese namespace, devolviendo las tripletas con las URIs originales para que la implementación del uso de namespaces sea transparente para el usuario.

4.2.3. Implementación de la conexión con Quarry en la API

Con tal de conectar estas dos herramientas, hay que hacer que la API proporcione a Quarry los dos archivos que este necesita para obtener la información de la ontología y los *mappings*, como vimos en la Sección 3.4 del capítulo Arquitectura del sistema de esta memoria.

Estudio de Quarry

Para llevar a cabo las modificaciones necesarias en la API se debe estudiar el formato de los archivos que lee Quarry, con tal de comprender su estructura y hacer que la API transforme la ontología y los *mappings* que Virtuoso almacena en RDF al formato que necesita Quarry.

Tras entender la estructura de estos archivos, se ha generado una plantilla de cada uno que sirva como referencia para implementar las nuevas llamadas de la API, como podemos ver en los Fragmentos de código de este apartado.

Fragmento de código 4.2: Plantilla del fichero de ontología que lee Quarry.

```

<owl:Class rdf:about="URI_de_Concepto_de_negocio">
  <rdfs:label>Concepto de negocio</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="URI_predicado_hacia_otro_Concepto_de_negocio"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="URI_predicado_hacia_otro_Concepto_de_negocio"/>
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:about="URI_predicado_entre_Conceptos_de_negocio">
  <rdfs:domain rdf:resource="URI_Concepto_de_negocio_1"/>
  <rdfs:range rdf:resource="URI_Concepto_de_negocio_2"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="URI_Caracteristica">
  <rdfs:label>Caracteristica</rdfs:label>
  <rdfs:domain rdf:resource="URI_Concepto_de_negocio"/>
  <rdfs:range rdf:resource="Tipo_de_dato"/>
</owl:DatatypeProperty>

```

En el Fragmento de código 4.2, referente a la plantilla de la ontología, vemos como aparecen tanto Conceptos de negocio como Características. En este fichero que generamos para Quarry, en formato OWL (Ontology Web Language), los Conceptos de negocio deben ser representados como clases, para las que tendremos que definir la cardinalidad mínima y máxima de cada conexión que tiene con otro Concepto de negocio o clase. OWL es un lenguaje web semántico, diseñado para representar conocimiento sobre términos y relaciones entre estos.

Además, vemos como las conexiones entre Conceptos de negocio deben ser representadas como *ObjectProperties* y las Características como *DatatypeProperties* de los Conceptos de negocio, donde deberemos indicar además del tipo de dato que se trata.

ObjectProperties equivalen a las triplas vistas en el modelo de la ontología que están formadas por sujeto, predicado y objeto. Mientras que *DatatypeProperties* equivalen a las triplas que están formadas por sujeto, predicado y un valor literal.

En el Fragmento de código 4.3 vemos la plantilla de los *mappings* de la ontología, donde nos encontramos con una primera parte donde se define una conexión. Esta primera parte la obviaremos, únicamente ha sido incluida en la plantilla para ser conscientes de que en caso de ser necesario que Quarry acceda a una base de datos, la conexión a esta sería definida siguiendo esta parte de la plantilla. En este caso, no debemos definir conexión alguna a base de datos ya que solo trabajaremos con HBase y, por lo tanto, esta parte no será incluida en el archivo que finalmente será enviado a Quarry. Aunque podría ser incluido en caso que extrajáramos datos de diferentes fuentes, definiendo esta conexión y todos sus parámetros en el *namespace* de *mappings* de la ontología.

Fragmento de código 4.3: Plantilla del fichero de *mappings* que lee Quarry.

```

<connection>
  <name>ConnectionName</name>
  <server>Server</server>
  <type>POSTGRESQL/MYSQL/...</type>
  <access>Native</access>
  <database>DB</database>
  <port>Port</port>
  <username>DBuser</username>
  <password>password</password>
</connection>

<OntologyMapping sourceKind="relational/text/xml/excel">
  <Ontology type="concept/property">URI</Ontology>
  <Mapping sourceKind="relational/text/xml/excel" connectionName="
    ConnectionName/file" >
    <Tablename>table</Tablename>
    <Projections>
      <Attribute>attribute1</Attribute>
      <Attribute>SUBSTR(x,y,z)</Attribute>
      ...
    </Projections>
    <Selections>
    <Selection>
      <Column>SUBSTR(x,a,b)</Column>
      <Operator>operator</Operator>
      <Constant>constant</Constant>
    </Selection>
    </Selections>
  </Mapping>
</OntologyMapping>

```

El resto del contenido de la plantilla sí que es relevante, esta parte ya contiene la información referente a los *mappings*. Como está orientado al uso con bases de datos relacionales, nos encontramos con mucha información que no necesitamos. En nuestro caso particular, omitiremos algunos de los campos que podemos ver en esta parte de la plantilla, simplificando el trabajo de Quarry. Podemos ver un ejemplo de la información que contiene el archivo de *mappings* que genera la API en el Fragmento de código 4.4.

Fragmento de código 4.4: Ejemplo del fichero de *mappings* generado por la API.

```
<OntologyMapping sourceKind="hbase">
  <Ontology type="property">URI.Caracteristica</Ontology>
  <Mapping sourceKind="hbase">
    <Projections>
      <Attribute>name:v:value1;</Attribute>
      <Attribute>family:v:value2;</Attribute>
      <Attribute>qualifier:r:reference1:v:_:r:reference2:v:_:r:reference3;</Attribute>
      <Attribute>table:v:value3:r:reference4;</Attribute>
    </Projections>
  </Mapping>
</OntologyMapping>
```

En este ejemplo, podemos ver como se define el tipo de fuente de los datos como *hbase*, de esta manera Quarry sabe como leer los *mappings* que tiene a continuación y sabe que son hacia HBase. Se trata del *mapping* de una Característica, por lo tanto, nos encontramos con todos los elementos que definen este *mapping*, vistos en el apartado 4.1.4 de la sección Ontología de este mismo capítulo de la memoria.

Estos cuatro elementos que conforman el *mapping* se definen de la misma forma en este archivo. Nos encontramos con el nombre del elemento primero, seguido del valor final de este. Recordemos que el valor final de los elementos de un *mapping* puede estar formado por varios valores y/o referencias (visto en el apartado al que se hace referencia en el párrafo anterior). Esto se indica en cada uno de estos valores o referencias, si se trata de un valor, este va precedido por el carácter *v*. Si, en cambio, se trata de una referencia, esta irá precedida por el carácter *r*. De esta manera Quarry puede discernir entre ambos.

Además, en caso que haya un valor que sirva de separación entre estos valores y/o referencias, definido en la sección anterior de este capítulo como *joinValue*, este valor irá precedido también por el carácter *v*, como pasa con el carácter '_' en el Fragmento de código anterior. Quarry no distingue entre valores y *joinValues*, los concatena sin más para formar el valor final del elemento y, en el caso de las referencias, irá a buscar el valor de estas como paso previo a concatenarlas.

Extracción de la ontología en formato Quarry

Para extraer la ontología del repositorio y darle el formato que Quarry necesita, visto en el apartado anterior, se hizo la extracción por partes. Quarry entiende por ontología aquella información que mostrará al usuario, por lo tanto, solo necesita la información del *namespace* de vocabulario.

Primero se extrajeron los Conceptos de negocio y las conexiones establecidas entre ellos para así generar las clases y las *ObjectProperties* que necesita Quarry, como se puede ver en el Fragmento de código 4.5.

Fragmento de código 4.5: Consulta de los Conceptos de negocio y conexiones en SPARQL

```

SELECT ?sujeto ?predicado ?objeto
FROM <repositorio>
WHERE {
  ?sujeto ?predicado ?objeto .
  FILTER(
    REGEX(STR(?sujeto), '^namespace:vocabulario/') &&
    REGEX(STR(?sujeto), 'ontology#') &&
    REGEX(STR(?objeto), 'ontology#') &&
    !REGEX(STR(?predicado), '#ComesFrom')
  )
}

```

Para obtener estos datos, lo que buscamos en el repositorio son tripletas básicas en el *namespace* de vocabulario. Es decir, tripletas que tanto sujeto como objeto contengan URIs de Conceptos de negocio.

Esto lo vemos cuando la URI contiene *ontology#*, ya que así es como acaba la URI raíz que utilizamos, y el único contenido de la ontología que se concatena inmediatamente después de la URI raíz son los Conceptos de negocio y los predicados. Si se tratara de una Característica, por ejemplo, la URI contendría algo como *ontology/ConceptoNegocioX#CaracteristicaY*. Lo mismo sucedería con los Tipos de Conceptos de negocio y sus Características, cuyas URIs contendrían algo como *ontology/ConceptoNegocioX#TipoConceptoNegocioY* y *ontology/ConceptoNegocioX/TipoConceptoNegocioY#CaracteristicaZ* respectivamente.

Tras extraer estos datos, se parsean para generar las clases y las *ObjectProperties*. Se recorre la lista de tripletas resultante de la consulta anterior, de manera que por cada sujeto u objeto diferente (que corresponden a los Conceptos de negocio) se crea una nueva clase. Además, por cada tripleta de la lista se añade la cardinalidad a las clases correspondientes (las que representan el Concepto de negocio del sujeto y del objeto de la tripleta), y se genera la *ObjectProperty* que representa la relación entre ambas clases.

Finalmente, se extrae del repositorio la información correspondiente a las Características para generar las *DatatypeProperties*, como muestra el Fragmento de código 4.6, y completar así esta parte de la ontología que necesita Quarry.

Fragmento de código 4.6: Consulta de todas las Características en SPARQL

```

SELECT ?sujeto ?predicado ?objeto
FROM <repositorio>
WHERE {
  {
    ?sujeto ?predicado ?objeto .
    FILTER(
      REGEX(STR(?sujeto), '^namespace:vocabulario/') &&
      REGEX(STR(?sujeto), 'ontology#') &&
      REGEX(STR(?predicado), '#hasFeature')
    )
  }
  UNION
  {
    ?sujeto ?as ?type .
    ?suj2 ?predicado ?objeto .
    FILTER(
      REGEX(STR(?sujeto), '^namespace:vocabulario/') &&
      REGEX(STR(?suj2), STR(?type)) &&
      REGEX(STR(?as), '#hasType') &&
      REGEX(STR(?predicado), '#hasFeature')
    )
  }
}

```

```
ORDER BY ?objeto
```

Pueden identificarse dos partes en esta consulta. La primera se encarga de obtener las Características principales de cada Concepto de negocio, buscando en el *namespace* de vocabulario las tripletas cuyo sujeto sea un Concepto de negocio, por el método explicado anteriormente, y que la URI del predicado contenga *#hasFeature*, lo cual nos indica que se trata de una Característica.

La segunda parte extrae las Características específicas de los Conceptos de negocio, es decir, aquellas propias de un Tipo de concepto de negocio en concreto, pero conservando el Concepto de negocio como sujeto de la triplete en lugar del Tipo de concepto de negocio, como podemos ver en la Figura 4.10, donde la triplete que se extrae del repositorio es la formada por *?sujeto*, *?predicado* y *?objeto*. Una vez se han extraído todas las Ca-

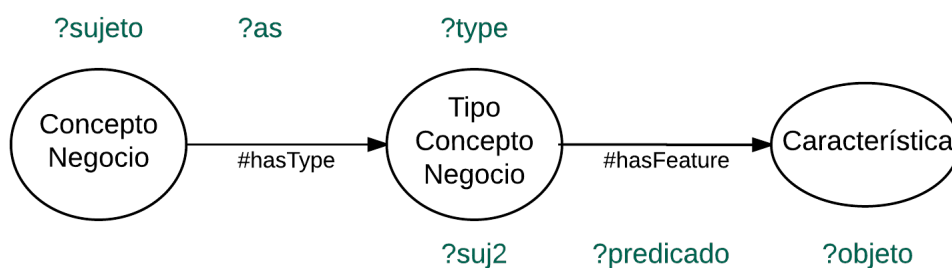


Figura 4.10: Extracción de las Características específicas del modelo.

racterísticas, junto a los Conceptos de negocio correspondientes, se parsea cada triplete obtenida para generar la *DatatypeProperty* que relaciona la Característica con el Concepto de negocio correspondiente de la forma que necesita Quarry.

Extracción de los *mappings* en formato Quarry

La extracción que se hace de los *mappings* es muy diferente a la explicada en el apartado anterior para la ontología, algo que se puede percibir tan solo viendo lo diferente que es el modelo de ambos *namespaces* (vistos en el apartado Modelo final de la sección anterior).

En este caso primero extraemos del *namespace* de vocabulario las URIs de las Características junto a su Mapping, como muestra el Fragmento de código 4.7. De manera que tenemos los nodos principales de los *mappings* que queremos obtener junto a su equivalente en el *namespace* de vocabulario.

Fragmento de código 4.7: Consulta de los Mappings en SPARQL

```
SELECT ?sujeto ?objeto
FROM <repositorio>
WHERE {
  ?sujeto ?predicado ?objeto .
  FILTER(
    REGEX(STR(?sujeto), '^namespace:vocabulario/') &&
    REGEX(STR(?predicado), '#mapsTo')
  )
}
```

Una vez tenemos la URI del Mapping equivalente a cada Característica, se extrae la información de cada elemento de estos *mappings*, esta vez del *namespace* de *mappings*, tal como se muestra en el Fragmento de código 4.8 para la extracción de la información del elemento Familia del Mapping.

Fragmento de código 4.8: Consulta específica de los elementos de un *mapping* en SPARQL

```

SELECT *
FROM <repositorio>
WHERE {
  ?sujeto ?predicado ?objeto .
  ?suj2 ?descripcion ?valor .
  FILTER(
    REGEX(STR(?sujeto), '^namespace:mappings/') &&
    REGEX(STR(?sujeto), 'URI:Mapping') &&
    REGEX(STR(?predicado), '#hasFamily', 'i') &&
    REGEX(STR(?suj2), STR(?objeto))
  )
}

```

Tras realizar estas extracciones del repositorio, obtenemos las tripletas que nos indican que valores forman cada elemento. Entonces, se parsea la información de cada elemento del Mapping recorriendo todas las tripletas que nos indican su valor, aquellas con los predicados *#hasValue*, *#firstValue*, *#joinValue*, *#secondValueAsRef*, etc. que se encuentran en *?descripcion* como podemos ver en la Figura 4.11. De esta manera le damos a cada

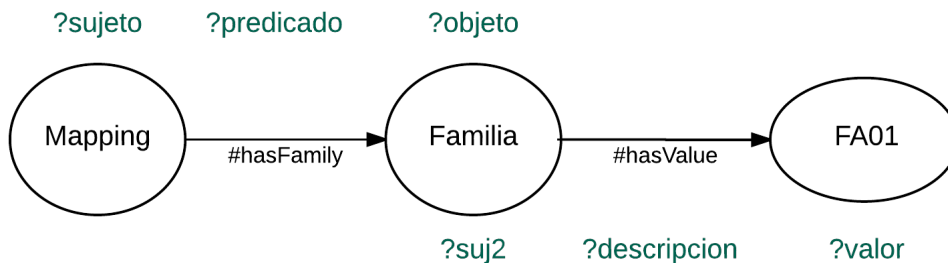


Figura 4.11: Extracción de los valores de un elemento del Mapping.

elemento el formato visto en el Fragmento de código 4.4 al principio de este apartado, en Estudio de Quarry, para generar el fichero que necesita Quarry y formatear los valores de cada elemento del *mapping* de forma que pueda ser interpretado fácilmente.

4.2.4. Extensión de la búsqueda de tripletas

Con tal de extraer metadatos concretos de la ontología de la forma más eficiente posible se tuvo que trabajar en el método de búsqueda de tripletas. Concretamente en la llamada a la API */triples/search*. Esta llamada busca una cadena de caracteres en cualquier URI que forme parte de la ontología, ya pertenezca a sujeto, predicado u objeto de cualquier tripla, y devuelve las tripletas que cumplen esta condición.

A la hora de acceder al repositorio para extraer una información concreta que se necesita durante los procesos ETL, nos dimos cuenta de que este método para buscar tripletas en el repositorio no filtraba lo suficiente y obteníamos mucha más información de la deseada. Por lo tanto, se tuvieron que hacer algunos cambios en esta llamada a la API.

Además de los cambios llevados a cabo para implementar el uso de *namespaces*, se añadieron tres parámetros más a la llamada: sujeto, predicado y objeto. De modo que, ahora podían llevarse a cabo dos tipos de búsquedas, una de ellas se ha explicado en el primer párrafo de este apartado, y otra concretando diferentes cadenas de caracteres a buscar en sujeto, predicado y objeto. Esto es posible ya que, como se hizo al implementar el uso de *namespaces*, se han hecho los cambios de forma que se conserva el comportamiento

original de la llamada a la API. Por lo tanto, toda llamada a la API que se hiciera antes de aplicar estas modificaciones, si se volviera a hacer ahora, se comportaría del mismo modo y tendría el mismo resultado si no ha cambiado el contenido del repositorio.

4.3. Ingestión de metadatos

Vista la evolución del modelo de la ontología y los cambios hechos en la API para adaptarla a las necesidades del proyecto, en esta sección se explica el proceso por el cual se ha nutrido el repositorio de metadatos con la información proporcionada por el cliente. Aunque no se podrá mostrar ni la información que tratamos ni el código de los *parsers* y ETLs por confidencialidad.

Este proceso se ha ido realizando durante el desarrollo del proyecto, volcando nueva información sobre el repositorio a medida que se iba añadiendo al modelo de la ontología cuando el cliente nos iba aportando esta información.

4.3.1. Inserción de las tripletas básicas

En el primer volcado de información que se hizo sobre el repositorio de metadatos, se incluyeron los metadatos correspondientes a la parte más básica del primer modelo de la ontología, explicado en el apartado 4.1.2 de este mismo capítulo. Esta información corresponde a los Conceptos de negocio y las relaciones entre ellos, información representada en la Figura ??.

Estos metadatos se introdujeron manualmente, ya que difícilmente se puede automatizar todo el proceso de ingestión de datos en una ontología, sobretodo la parte de *domain knowledge* (conocimiento del dominio), que nosotros representamos en el *namespace* de vocabulario. Pese a ello, al tratarse únicamente de las conexiones entre los Conceptos de negocio del *namespace* de vocabulario, la cantidad de tripletas a insertar no sobrepasaba la decena.

Para insertar estas tripletas se utilizó Postman, una API que permite realizar peticiones HTTP mediante una interfaz muy simple e intuitiva. Se realizó una llamada al método de inserción de tripletas de la API del repositorio de metadatos, mediante el método POST de HTTP. En este mensaje, le indicamos las tripletas a insertar en un fichero JSON, que forma parte del cuerpo del mensaje HTTP, junto al *namespace* en el que deben ser insertadas.

Las tripletas que se insertaron, mediante este fichero JSON, debían tener un determinado formato como el que se puede ver en el Fragmento de código 4.9.

Fragmento de código 4.9: Formato de las tripletas insertadas mediante la API.

```
[{
  "subject": "<URI-sujeto>",
  "predicate": "<URI-predicado>",
  "object": "<URI-objeto>"
}]
```

Todas las tripletas que se inserten en el repositorio tendrán este mismo formato, con la diferencia que en las demás fases de ingestión de metadatos la generación e inserción de las tripletas en el repositorio será automatizada.

4.3.2. Generación de las tripletas del modelo básico

Una vez volcada la parte más básica de la ontología en el repositorio, se pasó a insertar el resto de tripletas del modelo básico una vez se acabó de modelar este, en base a la información aportada por la empresa.

Esta información contenía las Características de los Conceptos de negocio, así como su origen. Por lo tanto, se insertaron tripletas tanto en el *namespace* de vocabulario como en el de origen. Los metadatos se extrajeron de ficheros XML donde se encontraba toda la información sobre las Características, tanto a nivel descriptivo como detalles sobre los datos a los que hacen referencia.

Entonces, mediante unos *parsers*, se recorrieron todos estos metadatos haciendo una selección de los que nos iban a ser de utilidad, que son los que se modelaron en la ontología. Estos parsers extraen la información y la transforman en tripletas cumpliendo con el modelo básico de la ontología. Tripletas que eran añadidas a dos ficheros JSON, uno por *namespace*, para después ser insertadas en el repositorio a través de la API, haciendo una llamada por cada fichero de tripletas a insertar.

4.3.3. Generación de las tripletas del modelo avanzado

Tras modelar la parte de la ontología que representa las tablas cremallera, como se explica en el apartado Modelo avanzado de esta memoria, y obtener así una nueva versión del modelo de la ontología. Se insertaron todos los metadatos correspondientes a este modelo.

La información aportada por la empresa volvía a hacer referencia a los *namespaces* de vocabulario y origen, pero esta vez era algo más compleja de representar, como se ha podido ver en el modelo. Estos metadatos eran sobre los Conceptos de negocio y sus características de nuevo, pero añadiendo un nivel intermedio en el *namespace* de vocabulario y cambiando ligeramente la parte del modelo del *namespace* de origen para dar cabida a la información sobre las tablas cremallera.

De nuevo, se ha trabajado con información en ficheros XML, que han sido *parseados* para obtener las tripletas correspondientes a este modelo avanzado de la ontología. Aunque esta vez con una mayor complejidad, no solo para entender conceptualmente el funcionamiento de estas tablas cremallera, sino para extraer los datos que realmente fueran relevantes dada esta nueva estructura en el origen de los datos.

4.3.4. Inserción de las tripletas del modelo final durante los procesos ETL

La ingesta de metadatos correspondiente al modelo explicado en el apartado Modelo final fue muy diferente a las anteriores. Esta vez la empresa no nos aportó la información en la que nos debíamos basar para modelar la ontología e insertar los metadatos, *parseandolos* desde unos ficheros XML como veníamos haciendo hasta ahora. Esta información debía ser extraída durante los procesos ETL mediante los cuales se iban a migrar los datos de su base de datos operacional al Data Lake.

Se modificaron unos procesos ETL, que procesa Apache Spark para leer la información del sistema actual de la empresa y migrarla al nuevo sistema Big Data, para extraer los datos correspondientes al *namespace* de *mappings*. Estas modificaciones consistieron en adaptar estos procesos para extraer la información necesaria sobre los datos que estaban siendo migrados, identificarlos mediante la información que teníamos guardada en el repositorio y, finalmente, antes de guardar los datos en el nuevo sistema, extraer esta información y

guardarla en el repositorio.

Todas las funciones y métodos creados para la automatización de la extracción e ingestión de estos metadatos, utilizados durante los procesos ETL, han sido incluidos en una librería. De esta manera, será mucho más fácil utilizar estas funciones y métodos en futuros procesos ETL si se migran más datos al Data Lake.

Al extraer la información durante la migración de los datos, estos datos hacían referencia a tablas y atributos de la base de datos operacional, información que nosotros guardamos en el *namespace* de origen. Por lo tanto, lo más fácil sería asociar la información de cada *mapping* a sus equivalentes en esta parte de la ontología, pero esto no nos interesa. Si revisamos de nuevo el Modelo final, vemos como los *mappings* están conectados con las Características del *namespace* de vocabulario, y no con Tablas o Atributos del *namespace* de origen.

Esto es debido a que el usuario trabajará desde el *namespace* de vocabulario. Entonces, para saber donde se guardan los datos correspondientes a la información que se muestra en este *namespace* será más eficiente si conectamos la información sobre los *mappings* directamente a las Características, en lugar de tener que acceder al *namespace* de origen para poder acceder finalmente a los *mappings*.

Para evitar tener que consultar *namespaces* inútilmente, preferimos complicar ligeramente el proceso de ingestión de metadatos en beneficio de un mejor futuro rendimiento del repositorio. De manera que el proceso de extracción de metadatos durante los procesos ETL es el siguiente:

- Extraer la información sobre los datos que están siendo migrados mediante Apache Spark, como resultado se obtienen un conjunto con los diferentes Atributos de cada Concepto de negocio que están siendo guardados en el Data Lake.
- Conservar la información correspondiente a los *mappings* cuando los datos van a ser almacenados en el nuevo sistema Big Data, conservando el valor (o como se forma este valor) de la Tabla, Familia y Qualifier, además del Nombre correspondiente a los Atributos en el fichero JSON.
- Identificar el equivalente en el *namespace* de vocabulario a los Atributos mediante la llamada a la API */triples/search*, cuyo funcionamiento se extendió para satisfacer esta necesidad. Con lo cual obtenemos las Características.
- Parsear la información extraída sobre los *mappings* de los datos para generar las nuevas tripletas. Además de generar todas las URIs de los componentes de las nuevas tripletas que van a ser insertadas, se ha de vigilar a la hora de construir estas tripletas de hacerlo adecuadamente. Respetando el orden de los valores que compongan un mismo elemento, así como tener en cuenta cuales son valores y cuales referencias. Finalmente todas estas tripletas son agrupadas en un fichero JSON para ser insertadas en el repositorio.
- Insertar las tripletas en el repositorio a través de la API.

Una vez finalizados todos los procesos ETL modificados para nutrir el repositorio de metadatos a la vez que se llevaba a cabo la migración de los datos, ha finalizado por completo el proceso de ingestión de metadatos en el repositorio.

Capítulo 5

Conclusiones

Ha sido sumamente interesante llevar a cabo este Trabajo Final de Grado en un laboratorio de innovación e investigación como inLab FIB. Ya que normalmente de un TFG se extraen muchos conocimientos que no se pueden adquirir durante el grado, además de un crecimiento a nivel tanto profesional como personal indudable. Pero el haber podido desarrollar este TFG como parte de un proyecto real mayor, junto a un equipo de grandes profesionales y docentes, y estando en contacto constantemente con la empresa cliente respondiendo a sus necesidades ad hoc, ha sido un auténtico privilegio y ha hecho que este proyecto sea mucho más enriquecedor.

5.1. Objetivos alcanzados

Tras todo el trabajo realizado durante el desarrollo de este proyecto, se puede decir que se ha cumplido con los objetivos satisfactoriamente.

Pese a haber sido descartado por el cliente un último objetivo de este proyecto, con el fin de reforzar otras partes del sistema Big Data del proyecto Big Data Analytics Lab, los demás han sido cumplidos con éxito.

Finalmente, se ha obtenido un repositorio de metadatos semántico compuesto por herramientas Open Source, a cuya información se puede acceder fácilmente a través de una API adaptada a los requisitos del proyecto. Este repositorio contiene toda la información necesaria para describir los datos almacenados en el Data Lake y su localización. Esta información es representada mediante una ontología, modelada expresamente para este caso de uso, que ha sido generada y almacenada automáticamente mediante procesos automatizados.

5.2. Trabajo futuro

La aportación al sistema Big Data que realiza el repositorio de metadatos obtenido en este Trabajo Final de Grado podría mejorarse si se estudiaran los datos almacenados en el Data Lake y se generaran metadatos de interés adicionales, como estadísticas sobre los datos.

Estas estadísticas deberían ser las que mejor describan los datos y las que más optimicen el acceso a estos. Por lo tanto, también debería realizarse un estudio sobre el tipo de consultas que se realizan sobre los datos, determinando así que estadísticas pueden

aportarnos información de mayor utilidad a la hora de optimizar estas consultas.

Finalmente, tras conocer cuales son estas estadísticas, debería modelarse la parte de la ontología correspondiente a esta nueva información, de forma que pudiera integrarse fácilmente con el modelo actual, y automatizar la ingestión de estos nuevos metadatos en el repositorio.

Apéndice A

Manual de instalación y configuración de Virtuoso OpenSource

Para instalar Virtuoso OpenSource, primero debemos comprobar el listado de paquetes disponibles, para asegurarnos de que figura entre los paquetes que podemos instalar desde los repositorios de nuestra distribución GNU/Linux:

```
$ apt-cache search virtuoso-opensource
```

En nuestro caso, utilizando Ubuntu 12.04.5 LTS, nos salió entre una lista de paquetes. Por lo tanto, pasamos a instalar este paquete en la máquina seleccionando la respuesta por defecto a las preguntas que nos hagan durante la instalación (*Yes* a instalar paquetes recomendados y *No* a instalar paquetes sugeridos). De esta manera nos instala todas las dependencias que necesitamos.

```
$ sudo apt-get install virtuoso-opensource
```

Tras completarse esta instalación y la de todas sus dependencias deberemos configurar Virtuoso adecuadamente. Para ello debemos parar primero el servicio, ya que se arranca automáticamente tras la instalación.

```
$ sudo /etc/init.d/virtuoso-opensource-6.1 stop
```

Una vez parado el servicio, se ha de revisar el fichero de configuración por si alguno de los puertos que utiliza por defecto Virtuoso esta siendo ocupado por otro servicio, si es así deberemos cambiar ese puerto por uno de nuestra elección. Un puerto a destacar es el que utiliza Virtuoso para su interfaz web, que podemos encontrar en los siguientes puntos del fichero de configuración `/etc/virtuoso-opensource-6.1/virtuoso.ini`:

- En la variable `ServerPort` bajo la cabecera `HTTPServer`
- En la variable `DefaultHost`

Después de hacer estos cambios de puerto, si han sido necesarios, podemos volver a arrancar el servicio, ya que no hace falta hacer ninguna configuración más.

```
$ sudo /etc/init.d/virtuoso-opensource-6.1 start
```

Apéndice B

Manual de instalación y configuración de la API

Para poner en funcionamiento la API primero deberemos instalar Apache Tomcat, un contenedor de *Servlets* Open Source que nos permitirá poner en marcha la API.

Como ya hicimos con Virtuoso, lo primero que debemos hacer es mirar el listado de paquetes disponibles para asegurarnos de que figura entre los paquetes que podemos instalar desde los repositorios de nuestra distribución y además ver cual es la última versión que tenemos a nuestra disposición:

```
$ apt-cache search tomcat
```

En Ubuntu 12.04.5 LTS nos salió la versión 7 como la última estable disponible en el repositorio de paquetes. Por lo tanto, pasamos a instalar este paquete en la máquina seleccionando la respuesta por defecto a las preguntas que nos hagan durante la instalación (*Yes* a instalar paquetes recomendados y *No* a instalar paquetes sugeridos). De esta manera nos instala todas las dependencias que necesitamos.

```
$ sudo apt-get install tomcat7
```

Deberemos revisar primero los puertos usados por Tomcat, como ya hicimos con Virtuoso, para intentar no encontrarnos con algún problema más adelante. En este caso, revisaremos el puerto a través del cual se accede a las aplicaciones web para cambiarlo si fuera necesario. Este puerto se encuentra en el fichero de configuración de Tomcat `/var/lib/tomcat7/conf/server.xml`, en *Connector port*. Tras cambiar esto deberemos reiniciar Tomcat para que se apliquen los cambios si lo teníamos iniciado:

```
$ sudo service tomcat7 restart
```

Por último deberemos descargar el código fuente de la API del repositorio de código público donde está publicado, *GitHub*, y compilarlo. Para ello primero deberemos instalar dos paquetes más, *Git*, un sistema de control de versiones, y *Apache Maven*, un software de gestión de proyectos.

```
$ sudo apt-get install git
```

```
$ sudo apt-get install maven
```

Para descargarnos el código y compilarlo deberemos ejecutar las siguientes instrucciones:

```
$ git clone https://github.com/vamhan/master_thesis_api
```

```
$ cd master_thesis_api
```

```
$ mvn clean package
```

Entonces ya tendremos lista la API para su correcto funcionamiento, únicamente tendremos que copiar el archivo WAR (*Web application ARchive*) de la aplicación web a la carpeta de aplicaciones web de Tomcat para que este se encargue de ponerla en funcionamiento.

```
$ cp /target/gs-rest-hateoas-0.1.0.war /var/lib/tomcat7/webapps/
```

Bibliografía

- [1] P. Jovanovic, O. Romero, A. Simitsis, A. Abelló, H Candón, and S. Nadal. Quarry: digging up the gems of your data treasury. 2015.
- [2] A. L. Porter, J. Schuehle, J. Youtie, and Y. Huang. Metadata: Bigdata research evolving across disciplines, players, and topics. *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 262–267, 2015.
- [3] D.E. O’Leary. Embedding ai and crowdsourcing in the big data lake. *Intelligent Systems, IEEE*, 29:70–73, 2014.
- [4] J. Liu and Y. Chen. Improving data analysis performance for high-performance computing with integrating statistical metadata in scientific datasets. *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion.*, pages 1292–1295, 2012.
- [5] Shelley Powers. *Practical RDF*. O’Reilly Media, 2003.
- [6] Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the Semantic Web*. O’Reilly Media, 2009.
- [7] Page Personnel. Estudios de remuneración 2015: Tecnología. 2015. http://www.pagepersonnel.es/productsApp_pp_es/Estudios%20Remuneracion/er_tecnologia.pdf.
- [8] Agencia Tributaria. Regímenes para determinar el rendimiento de las actividades económicas. 2015. http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresarios_individuales_y_profesionales/Rendimientos_de_actividades_economicas_en_el_IRPF/Regimenes_para_determinar_el_rendimiento_de_las_actividades_economicas/Estimacion_Directa_Simplificada.shtml.
- [9] V. Thavornun. Metadata management for knowledge discovery. 2015. <http://upcommons.upc.edu/handle/2117/77842>.
- [10] A. Khurana. Introduction to hbase schema design. *login.*, 37(5), 2012. <https://www.usenix.org/publications/login/october-2012-volume-37-number-5/introduction-hbase-schema-design>.