UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

MÁSTER UNIVERSITARIO
EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

SparkDQ: A Spark Stack to Support Data Quality Context-Aware Assessment of Semantic Triples

Raúl Reguillo Carmona

Octubre, 2017

SparkDQ: A Spark Stack to Support Data Quality Context-Aware Assessment of Semantic Triples

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

TRABAJO FIN DE MÁSTER

SparkDQ: A Spark Stack to Support Data Quality Context-Aware Assessment of Semantic Triples

Autor: Raúl Reguillo Carmona

Tutor: Ismael Caballero Muñoz-Reja

Cotutor: Bibiano Rivas García

Octubre, 2017

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

TRABAJO FIN DE MÁSTER

SparkDQ: A Spark Stack to Support Data Quality Context-Aware Assessment of Semantic Triples

Fdo.: Raúl Reguillo Carmona Fdo.: Ismael Caballero Muñoz-Reja

Octubre, 2017

Raúl Reguillo Carmona

Ciudad Real – Spain

E-mail: raul.reguillo@alu.uclm.es
Teléfono: 638 175 608

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

La Web Semántica es un tipo de web cuyo contenido puede ser procesado tanto por máquinas de forma automática como por personas. Esto es posible debido a una serie de avances significativos en la representación del conocimiento durante los últimos años. Esta representación del conocimiento se basa en el concepto de *triplas* (sujeto, predicado, objeto) que establecen relaciones entre los distintos datos. Estas relaciones a su vez se representan utilizando lenguajes y especificaciones desarrollados para tal fin, como son Framework de Descripción de Recursos (RDF) y Lenguaje de Ontología Web (OWL).

Paralelamente surge el paradigma de Linked Data, que se puede definir como el uso del modelo web para publicar datos estructurados de manera que puedan ser fácilmente consumidos y enlazados con otros. Aprovechando la capacidad semántica obtenida mediante la representación en triplas y la capacidad de poder interconectar los datos de esta manera, se pueden tener volúmenes significativos de datos interrelacionados sobre los cuales aplicar razonamiento automático, consultas o cualquier otra operación como parte del procesamiento automático anteriormente citado.

Por otra parte, tener en cuenta la calidad de los datos resulta algo imprescindible en el desarrollo con éxito de cualquier tarea. Se puede encontrar una gran cantidad de *frameworks* (Jena, Sesame, ...) con los que elaborar aplicaciones semánticas, pero actualmente ninguno de ellos incorpora primitivas para dar soporte a las operaciones de gestión de calidad de datos que están siendo utilizados. Dada la importancia de la calidad de datos y la carencia de dichas utilidades en los frameworks anteriormente citados, es patente la necesidad de elaborar mecanismos que permitan llevar a cabo evaluaciones de calidad de los datos usados en aplicaciones de tecnología semántica en general y Web Semántica en particular.

Así pues el objetivo del Proyecto Fin de Carrera consistió en la elaboración de una extensión de un framework de desarrollo de aplicaciones de Web Semántica y Linked Data: Apache Jena. La finalidad de dicha extensión fue, dado un conjunto de Linked Data, dar soporte a primitivas de medición de calidad de datos teniendo en cuenta el contexto en el que dichos datos deben ser considerados.

El objetivo principal de una posible iteración sobre este proyecto fue extender estas funcionalidades en el contexto Big Data, debido a la falta de eficiencia encontrada cuando las métricas calculadas ganaban complejidad e involucraban un número considerable de triplas.

De esta manera, el objetivo principal de este Trabajo Fin de Máster es el desarrollo de un stack para un framework de procesamiento de datos en entornos distribuidos, Apache Spark, de manera que se pueda suplir la necesidad de evaluación de calidad de datos enlazados en entornos Big Data.

Abstract

The Semantic Web is a type of Web which content can be processed automatically. This is possible due to a number of significant advances in knowledge representation in recent years. Knowledge in the Semantic Web is represented by triples (subject, predicate, object) that establish relations between different data using languages and specifications developed for this purpose, such as Resource Definition Language (RDF) and Ontology Web Language (OWL).

Meanwhile, Linked Data paradigm appears, which can be defined as the use of the Web model to publish structured data so it can be easiliy consumed and linked to other data. Considering the semantic capacity obtained by triples representation and the ability to interconnect data thus, users could have significant volumes of interrelated data where apply automated reasoning, inquiries or any other operation.

Moreover, it should be considered data quality as a need in the successful perform of any task. People can find a lot of frameworks (Jena, Sesame, . . .) to develop semantic application, but currently none of them incorporates basic operations for assessing quality of data in use. Many authors consider that this part has not yet been taken int account or has been received little attention. Therefore, there is a need to develop mechanisms to carry out quality assessments of the data used in semantic applications.

Thus, the objective of the original project involved the development of an extension for a Semantic Web and Linked Data framework: Apache Jena. The purpose of this extension was, having a set of Liked Data, to support a set of data quality measures taking into account the context in which these data should be considered.

The main target of a possible next iteration over the project was to extend this functionalities on a Big Data context, due to lack of efficiency found when the metrics calculation get complex and involves an increasing number of triples.

Thus, the goal of this project is the development of a stack for a distributed processing framework, Apache Spark, in order to cover the need of Data Quality Assessment of Linked Data in a Big Data environment.

Listado de acrónimos

AAA	*Anyone can say Anything about Any topic*
API	Application Programming Interface
CASE	Computer Aided Software Engineering
CdU	Caso de Uso
CSV	Comma Separated Values
DQ	Calidad de Datos
DQD	Dimensión de Calidad de Datos
DOAP	Description Of A Project
ER	Entidad-Interrelación
FOAF	Friend Of A Friend
HDFS	Hadoop Distributed File System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
ISO	International Organization for Standarization
JSON	Java Script Object Notation
LD	Datos Enlazados
LOD	Datos Enlazados Abiertos
NT	N-Triples
OD	Datos Abiertos
OWL	Lenguaje de Ontología Web
PFC	Proyecto Fin de Carrera
PoC	Proof of Concept
PUD	Proceso Unificado de Desarrollo
QA	Quality Assurance
RDD	Resilient Distributed Dataset

0.

RDF	Framework de Descripción de Recursos
REST	Representation State Transfer
RF	Requisitos Funcionales
RQL	Relationship Query Language
SGML	Standard Generalized Markup Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TDB	Triple Data Base
TFM	Trabajo Fin de Máster
UML	Unified Modeling Language
URI	Identificador Uniforme de Recurso
URL	Localizador Uniforme de Recurso
W3C	Consorcio para la World Wide Web
XML	Lenguaje de Marcas Extensible

Agradecimientos

Escribe aquí algunos chascarrillos simpáticos. Haz buen uso de todos tus recursos literarios porque probablemente será la única página que lean tus amigos y familiares. Debería caber en esta página (esta cara de la hoja).

Juan[1]

[1] Sí, los agradecimientos se firman

A alguien querido y/o respetado

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Listado de acrónimos

AAA	*Anyone can say Anything about Any topic*
API	Application Programming Interface
CASE	Computer Aided Software Engineering
CdU	Caso de Uso
CSV	Comma Separated Values
DQ	Calidad de Datos
DQD	Dimensión de Calidad de Datos
DOAP	Description Of A Project
ER	Entidad-Interrelación
FOAF	Friend Of A Friend
HDFS	Hadoop Distributed File System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
ISO	International Organization for Standarization
JSON	Java Script Object Notation
LD	Datos Enlazados
LOD	Datos Enlazados Abiertos
NT	N-Triples
OD	Datos Abiertos
OWL	Lenguaje de Ontología Web
PFC	Proyecto Fin de Carrera
PoC	Proof of Concept
PUD	Proceso Unificado de Desarrollo
QA	Quality Assurance
RDD	Resilient Distributed Dataset

0.

RDF	Framework de Descripción de Recursos
REST	Representation State Transfer
RF	Requisitos Funcionales
RQL	Relationship Query Language
SGML	Standard Generalized Markup Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TDB	Triple Data Base
TFM	Trabajo Fin de Máster
UML	Unified Modeling Language
URI	Identificador Uniforme de Recurso
URL	Localizador Uniforme de Recurso
W3C	Consorcio para la World Wide Web
XML	Lenguaje de Marcas Extensible

Capítulo 1

Introducción

1.1 Contexto del proyecto

En [BLHL01] se define la Web Semántica como un tipo de Web cuyo contenido resulta procesable de manera automática. Para conseguir un contenido procesable se debe antes llevar a cabo una representación o formalización del conocimiento que se pretende exponer. Esto ha sido posible gracias al concepto de *tripla* que no es más que la relación, expresada en un lenguaje formal, existente entre un sujeto y un objeto a través de un predicado que los relaciona. De esta manera, si se desea expresar formalmente una sentencia respecto de cualquier concepto bastará con establecer un sujeto o recurso (el propio concepto) una relación o propiedad que se desee modelar, y un objeto que a su vez puede ser un nuevo recurso de manera que agrupando conjuntos de triplas se logra definir conceptos completos de manera formal.

Durante los últimos años se han desarrollado lenguajes como RDF y OWL, y especificaciones, con el fin de dar soporte a la interoperabilidad semántica [SBLH06]. En 1997 el Consorcio para la World Wide Web (W3C) [W3Ca] define la primera especificación de RDF lo que sentará los cimientos de la Web Semántica: el objetivo de RDF será aportar una descripción semántica del conocimiento en la Web [SBLH06] como lenguaje de definición de recursos, haciendo posible trabajar con la totalidad de la Web como un conjunto de recursos y sus interrelaciones.

Posteriormente surge el paradigma Linked Data [BL09], que se puede definir como el uso del modelo Web para publicar datos estructurados de manera que puedan ser fácilmente consumidos y combinados con otros [BHBL09]. Valiéndose del concepto de Identificador Uniforme de Recurso (URI), se consiguen identifcar recursos en la red de forma unívoca y así poder enlazarlos sin ambigüedad. Linked Data aprovecha la potencia descriptiva de RDF para facilitar el procesado y razonamiento automático sobre grandes conjuntos de datos. En [BL09] se enumera una serie de principios que deben cumplir los datos para ser considerados Linked Data:

1. Usar URI como identificadores de los recursos publicados en la Web.
2. Usar las URL de estas URI para que la gente pueda localizar y consultar estos recursos.

3. Proporcionar información útil cuando la URI sea desreferenciada.
4. Incluir vínculos a otras URI relacionadas con los datos en el recurso.

Por otra parte, el concepto de *Calidad* puede aplicarse a cualquier tarea u objeto. Es importante matizar para el contexto del proyecto qué se entiende por calidad y más concretamente por calidad de los datos (DQ).

Existen varias definiciones de calidad que conviene considerar a la hora de abordar el proyecto. En primer lugar, se va a exponer la definición que la Real Academia Española de la lengua otorga a este término:

Propiedad o conjunto de propiedades inherentes a algo que permiten juzgar su valor.

Sin embargo este concepto ha ido variando con el tiempo, adaptándose a las necesidades y a los procesos. Relacionando este concepto con la *fabricación* de un determinado bien, en [ISH85] se define la calidad como:

Un producto tiene calidad cuando es desarrollado, diseñado y mantenido de la forma más económica, útil y satisfactoria para el consumidor.

Autores como [CMC08] o [FH10] dejan constancia de cómo la calidad de los datos resulta imprescindible en el desarrollo con éxito de cualquier tarea o toma de decisión. Un nivel inadecuado de calidad en los datos puede tener impactos sustanciales en ámbitos sociales y económicos. Las empresas están mejorando la calidad de los datos con enfoques y herramientas prácticas [WS96].

En el ámbito de las tecnologías semánticas, la comunidad ha prestado poca atención a la calidad de los datos que han sido representados mediante la utilización de tecnologías semánticas [FH10]. Existen en la actualidad un número considerable de frameworks disponibles para el desarrollo de aplicaciones basadas en el uso de tecnologías semánticas, no obstante, ninguno de ellos incluye funcionalidades específicas que permitan procesar una evaluación del nivel de calidad de los datos que se están usando.

Puesto que ningún framework de desarrollo de Web Semántica facilita este tipo de operaciones de medición, existe una necesidad de elaborar mecanismos que permitan llevar a cabo evaluaciones de calidad de los datos usados en aplicaciones de tecnologías semánticas.

Con la irrupción de las tecnologías Big Data y dadas las necesidades que existen de procesamiento distribuido sobre grandes volúmenes de información, encontramos que el problema de la evaluación de la calidad de los datos semánticos se extiende a este ámbito, no pudiendo encontrar ningún framework que permita trabajar la calidad de los datos semánticos de manera distribuida.

Considerando la no disponibilidad de operaciones de evaluación de calidad de datos en los frameworks de desarrollo de tecnologías semánticas, el objetivo del Trabajo Fin de Máster (TFM) consiste en el desarrollo de una capa tecnológica (stack) para uno de estos frame-

works. La finalidad de dicho stack será la de diseñar, implementar y poner a disposición de la comunidad un conjunto de primitivas de evaluación de calidad de datos para uno de los frameworks de desarrollo de aplicaciones de tecnología semántica más importantes, Apache Jena, considerando el contexto en el que los datos deben son usados mediante el uso de reglas que lo modelen y sean procesables.

1.2 Evolución respecto del PFC

Tras llevar a cabo un estudio acerca del estado actual de los frameworks de desarrollo de Web Semántica y Linked Data, durante el desarrollo del Proyecto Fin de Carrera (PFC) se escogió uno de ellos basado en Java, **Apache Jena**, y se extendió con la finalidad de ofrecer estos mecanismos para la evaluación de la calidad de datos semánticos.

Paralelamente, durante los últimos años han tenido lugar una serie de cambios muy significativos en la industria de las tecnologías de la información, tales como el nacimiento del Internet de las Cosas, Computación en la Nube y la aparición de las Redes Sociales.

El desarrollo y explotación de estas tecnologías ha propiciado que la cantidad de los datos que se generan por unidad de tiempo se haya incrementado a una velocidad sin precedentes, trayendo consigo la necesidad de cómputo de tales volúmenes de información heterogénea, en tiempos razonables y de manera escalable [DG04].

Esta necesidad ha propiciado la aparición de paradigmas, tecnologías y numerosos frameworks con los que abordar esta problemática desde diferentes puntos de vista (procesamiento por lotes, en tiempo real, extracción, transformación y carga de datos, seguridad,...), agrupándose en ecosistemas tal y como pueda ser el caso del **ecosistema Hadoop** [HAD].

Apache Spark [SPAa] es un framework de procesamiento distribuido, perteneciente al ecosistema Hadoop, que en los últimos años se ha convertido en el estándar *de facto* a la hora de abordar problemas de procesamiento de grandes volúmenes de información en tiempos razonables, obteniendo unos resultados de rendimiento varios órdenes de magnitud superior frente a modelos y tecnologías anteriores, debido al procesamiento en memoria.

Mientras que el proyecto de Jena ha ido evolucionando para permitir el procesamiento de datos semánticos de manera cetralizada y habiendo estudiado el estado actual de los frameworks de procesamiento distribuido, se concluye que ninguno de ellos ofrece primitivas para el procesamiento de datos semánticos en distribuido y por lo tanto, tampoco mecanismos para llevar a cabo cálculos de métricas de calidad sobre datos semánticos en entornos distribuidos.

Debido a la necesidad de incluir primitivas de evaluación de calidad de los datos a la hora de desarrollar cualquier tarea o toma de decisión y puesto que el ritmo de crecimiento de los datos dificulta su análisis en tiempos razonables, se establece como objetivo principal de este Trabajo Fin de Máster ofrecer un stack tecnológico basado en un framework de proce-

samiento distribuido para que permita llevar a cabo análisis de calidad de datos semánticos en entornos Big Data.

Para el presente Trabajo fin de Máster, será **Apache Spark** el framework sobre el cual se construirá dicho stack para dar soporte a primitivas de evaluación de calidad de los datos de una manera distribuida, escalable y tolerante a fallos.

Como incremento respecto al PFC, cabe destacar los siguientes aspectos:

1. Elaboración de un stack tecnológico basado en un framework de desarrollo distribuido, **Apache Spark**, para permitir el procesado de datos RDF: **SparkRDF**. Este stack estará basado en el framework **Apache Jena** para el procesamiento de datos en formatos semánticos y en **Apache Spark** para ofrecer primitivas de procesamiento en distribuido.
2. Extensión de dicho stack para permitir evaluación de calidad de datos semánticos de manera distribuida, **SparkDQ**.
3. Elección o desarrollo de una ontología que describa evaluaciones de calidad de los datos.
4. Desarrollo de primitivas de razonamiento sobre evaluaciones de calidad.
5. Incremento del rendimiento a la hora de calcular métricas sobre grafos de datos enlazados.
6. Estudio y comparativa entre resultados de **JenaDQ** y **SparkDQ** con el fin de establecer unos criterios de mejora en términos de tiempo y eficiencia a la hora de llevar a cabo estos cálculos de métricas de calidad.
7. Elaboración de una prueba de concepto de extracción de grandes volúmenes de datos, evaluación y presentación de las métricas.

1.3 Logro de competencias planteadas

`TODO`

Pues ha ido flaman.

1.4 Estructura del documento

La forma en la que se ha organizado el presente documento se expone a continuación:

Capítulo 1: Introducción

En este apartado se introduce el problema y se plantea el trabajo en base a los objetivos fijados.

Capítulo 2: Objetivos

En este capítulo se describe el principal objetivo del TFM así como un conjunto de

objetivos parciales que es necesario alcanzar para cubrir el objetivo final.

Capítulo 3: Estado del Arte

Esta sección introducirá los conceptos más importantes tratados en el documento y el proyecto, acompañado de conceptos y definiciones. Incluirá referencias al ámbito de la calidad de los datos, la Web Semántica y las tecnologías asociadas a ésta.

Capítulo 4: Método de Trabajo

Se expondrá detalladamente cómo se ha utilizado la metodología de desarrollo elegida para la realización del TFM.

Capítulo 5: Resultados

En este capítulo se expondrán los resultados obtenidos durante las diferentes iteraciones que se han llevado a cabo durante el desarrollo del TFM.

Capítulo 6: Conclusiones

En este capítulo se expondrán las conclusiones que se han obtenido tras la realización de este TFM además de una serie de propuestas de trabajo futuro.

Capítulo 2

Objetivos

2.1 Objetivos principales

El objetivo de este Trabajo Fin de Máster (TFM) es:

- Desarrollar un stack para el framework **Apache Spark** con el fin de dar soporte a primitivas de evaluación de calidad de datos semánticos en un entorno Big Data. Dicha stack se apoyará sobre el framework **Apache Jena** para procesar datos semánticos y finalmente se ofrecerá una API cuya finalidad será encapsular todas las primitivas desarrolladas.
    - Se establecerá un modelo de desarrollo distribuido para Spark con el fin de dar soporte a primitivas de evaluación de calidad de datos semánticos en entornos Big Data.

2.2 Objetivos parciales

El objetivo principal implica el abordar una serie de objetivos parciales relacionados con la arquitectura y funcionalidades del sistema que se pretenden desarrollar. Dichos objetivos parciales se detallan a continuación.

O1. Representación del contexto de evaluación de calidad de datos
Implementar las reglas de negocio que representen el contexto necesario para la evaluación de calidad de datos usando la tecnología más adecuadas.

O2. Investigación o desarrollo de un vocabulario para los resultados de evaluaciones de calidad de datos
El fin de dicho vocabulario será el de permitir publicar los resultados de las evaluaciones que se lleven a cabo de manera formal y basándose en los paradigmas de la Web Semántica. Los conceptos de ontología y vocabulario serán detallados en la Sección 3.1.5.

O3. Diseño e implementación de las primitivas de evaluación de calidad de datos
Diseñar e implementar primitivas para la evaluación de calidad de los datos como una parte del stack desarrollado, considerando el contexto de uso mediante el uso de reglas semánticas.

Estas primitivas se desarrollarán tomando diferentes perspectivas de la calidad de los datos, que se pueden consultar en la Sección 3.3.

O4. Elección o desarrollo de un razonador de reglas para datos semánticos

Se realizará un estudio sobre los razonadores de reglas de inferencia para datos semánticos. Las reglas definidas deberán ser ejecutadas por un razonador que generará los resultados finales del proceso de medición.

O5. Desarrollo de una aplicación de prueba de concepto

Elaborar un pipeline que haga uso del stack previamente desarrollado. Para ello se deberá implementar la arquitectura de la aplicación teniendo en cuenta las restricciones de la tecnología a utilizar.

Capítulo 3

Estado del Arte

En el presente capítulo se describen los conceptos teóricos que son necesarios para establecer las bases de la elaboración del TFM. La descripción incluye conceptos como Calidad de Datos (DQ), Datos Abiertos (OD), Datos Enlazados Abiertos (LOD), Web Semántica y Big Data. También se describe un conjunto de tecnologías que se han utilizado para el desarrollo del proyecto.

3.1 Web Semántica

En [BLHL01] se define la Web Semántica como una evolución o extensión de la Web tradicional en la que la información es dada mediante significados bien definidos, lo que facilita el procesamiento automático del contenido y permita a las personas y a los ordenadores trabajar en cooperación.

Este concepto ha ido evolucionando y refinándose con el paso del tiempo. En los siguientes apartados se introduce el concepto de Web Semántica y se explicarán sus principales características.

3.1.1 Concepto

El Consorcio para la World Wide Web (W3C) en [W3Cb] define la Web Semántica como:

Una Web extendida, dotada de mayor significado en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Al dotar a la Web de más significado y, por lo tanto, de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante.

[SBLH06] define a su vez la Web Semántica como sigue:

La Web Semántica es una Web de información procesable - información derivada de los

datos mediante una teoría semántica para la interpretación de símbolos. La teoría semántica proporciona una noción de "significado" en la que la conexión lógica de términos establece la interoperabilidad entre sistemas.

Por otro lado, [HBLM02] definen nuevamente la Web Semántica de la siguiente manera:

La Web Semántica es una extensión de la Web actual, en la que a la información disponible se le otorga un significado bien definido que permita a los ordenadores y a las personas trabajar en cooperación. Está basada en la idea de proporcionar en la Web datos bien definidos y enlazados, permitiendo que aplicaciones heterogéneas localicen, integren, razonen y reutilicen toda la información presente en la Web.

Por lo tanto se puede establecer que la Web Semántica es una extensión de la Web en la que se dota de capacidad de anotar información semántica a los datos de manera que proporcionen un significado. En la última definición aparece el concepto de "datos enlazados" como precursor de lo que posteriormente se considerará Linked Data (LD), con la idea de vincular datos mediante estándares de Web Semántica para alcanzar los siguientes objetivos:

1. Reutilizar entidades ya definidas en el modelo de Web Semántica.
2. Hacer de estas entidades conceptos únicos, evitando redundancia y ambigüedad en los datos.
3. Permitir la interoperabilidad semántica entre aplicaciones heterogéneas.

La reutilización es posible gracias al concepto de Identificador Uniforme de Recurso (URI) utilizado para identificar de forma unívoca, universal y expansible un espacio de nombres de recursos de información.

Respecto de la interoperabilidad semántica, [HT06] explica que la *Web Semántica es la solución al problema de la integración de datos* sin necesidad de llevar a cabo procesos de conversión, gracias a la utilización de un modelo de representación como Framework de Descripción de Recursos (RDF) y utilizando Lenguaje de Marcas Extensible (XML) como fuente sintáctica para su intercambio.

3.1.2 Terminología de Web Semántica

- **Identificador Uniforme de Recurso (URI)**: cadena de caracteres que identifica los recursos de una red de forma unívoca. La diferencia con una Localizador Uniforme de Recurso (URL), es que éstos pueden variar en el tiempo. En el ámbito de la Web Semántica, una URI identificará a un recurso de manera unívoca dentro del conjunto de datos.
- **Recurso**: Se dice que un recurso es cualquier concepto que se pueda identificar.
- **Tripla**: Usando el estándar Framework de Descripción de Recursos (RDF), se define

tripla como una asociación de dos recursos a través de una relación o propiedad. Esta asociación se representa mediante dos nodos conectados por un arco (véase figura 3.1) (sujeto, predicado y objeto), también llamado *sentencia* (statement):

- **Sujeto**: es el recurso desde el que parte el arco (la propiedad).
- **Predicado**: es la propiedad que etiqueta el arco.
- **Objeto**: es el recurso o literal apuntado por el arco.

- **Endpoint**: Interfaz que se ofrece como extremo de una comunicación o como terminal que permite dar un servicio determinado. En el ámbito de la Web Semántica, un Endpoint permitirá tener acceso a las URI de un dataset, así como a las triplas mediante la utilización de protocolo HTTP.
- **Graph/Named Graph**: El término *Graph* se refiere a un conjunto de triplas relacionadas entre sí, de tal forma que tanto predicados como objetos dentro de una tripla hacen referencia a sujetos de otras triplas, enlazándose de esta manera. Un *Named Graph* es un conjunto de triplas que tiene sentido en sí misma (por ejemplo, las triplas que definen a un grupo de música en concreto).
- **Almacenamiento de triplas**: Es una base de datos especializada en almacenar archivos semánticos, es decir, conjuntos de triplas o *graphs*. Su funcionamiento interno dista del modo en que lo hacen las bases de datos convencionales, pues debe permitir inferencia gracias a las propiedades descritas.
- **Servidor de triplas**: Un servidor de triplas es una aplicación que, dado un almacenamiento de triplas, permite que ese contenido sea accesible por otras aplicaciones y mediante protocolos tales como HTTP. Además debe permitir operaciones de consulta, actualización, inserción y borrado sobre los datos almacenados.

3.1.3 Estándares

A continuación, se expondrán los estándares que definen la Web Semántica en el marco de la W3C.

Lenguaje de Marcas Extensible (XML)

En [XML] se define XML como un formato de texto simple, muy flexible, derivado de Standard Generalized Markup Language (SGML) (ISO 8879). Originalmente diseñado para cumplir con los desafíos de la publicación electrónica a gran escala, XML también está desempeñando un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y otros sistemas.

XML establece las bases para la elaboración de lenguajes orientados a la representación de información estructurada mediante la descripción de gramáticas, permitiendo diferentes niveles de abstracción.

La principal ventaja de XML es que permite la intercomunicación de aplicaciones y la integración de información, también en el ámbito de las bases de datos.

XML constituye la base de otros lenguajes y en particular, fue precursor de RDF.

Framework de Descripción de Recursos (RDF)

Se define RDF en [RDF] como un modelo estándar para el intercambio de datos en la Web. RDF tiene características que facilitan la fusión de datos incluso si los esquemas subyacentes difieren y soporta específicamente la evolución de esquemas con el tiempo sin necesidad de realizar cambios en los consumidores de los datos.

RDF permite extender la estructura de los enlaces de la Web para hacer uso de las URI como nombre de las relaciones entre conceptos. De esta manera, se establecen lo que se conoce como *triplas* como una relación *(subjeto, predicado, objeto)* (véase figura 3.1 y Sección 3.1.2) en la que todos los componentes son URI.

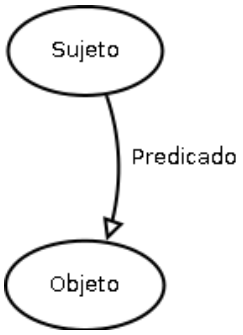

Figura 3.1: Esquema de tripla

Usando este modelo se permite la mezcla de datos estructurados y semi-estructurados a través de diferentes aplicaciones.

Esta estructura de enlaces forma grafos etiquetados y dirigidos donde los arcos representan el tipo de relación entre dos recursos, representados por nodos del grafo (véase figura 3.2).

Lenguaje de Ontología Web (OWL)

Según [OWL], se define OWL como un lenguaje de Web Semántica diseñado para representar conocimiento enriquecido y complejo acerca de conceptos, grupos de conceptos y relaciones entre conceptos. OWL es un lenguaje computacional basado en la lógica de manera que el conocimiento que expresa puede ser explotado por programas de computador.

Los documentos generados según el lenguaje OWL se conocen como **ontologías**. Las ontologías pueden ser publicadas en la W3C o pueden referise o derivarse de otras ontologías.

En el contexto de la computación y ciencias de la información, una ontología define un

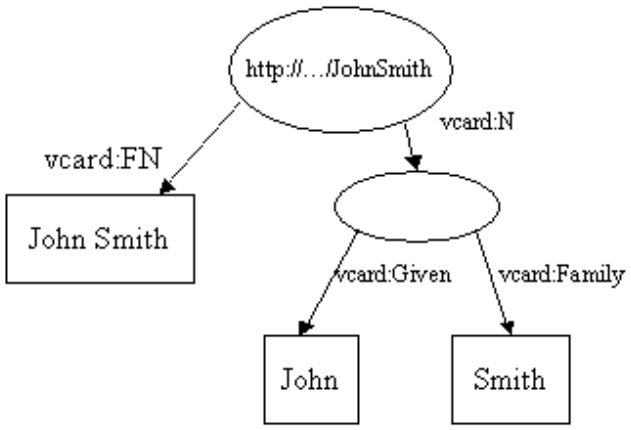

Figura 3.2: Grafo basado en triplas. Extraído de [JEN14]

conjunto de primitivas de representación con la que modelar un dominio de conocimiento [Gru]. La finalidad de las ontologías es ofrecer un modelo formal acerca de un conjunto de conceptos sobre el cual poder aplicar razonamiento automático .

De OWL derivan tres sub-lenguajes basados en la capacidad expresiva, tal y como cita [OWL]:

1. **OWL Lite**: que da soporte a las necesidades básicas del usuario cuando lo que se presente es una representación no tan exhaustiva, como por ejemplo, recursos, clasificaciones jerárquicas y restricciones simples.
2. **OWL DL (Description Logics)**: soportando más expresividad semántica y garantiza que todas las inferencias puedan ser calculadas en un tiempo finito.
3. **OWL Full**: el grado de expresividad es total, pero no asegura que las inferencias puedan ser calculadas en un tiempo finito.

SPARQL Protocol and RDF Query Language (SPARQL)

SPARQL es un lenguaje estándar para la consulta de grafos RDF. En [SPAb] se puede encontrar toda la información y especificación de la tecnología.

Muy similar a Structured Query Language (SQL), es un lenguaje declarativo que permite estructurar las consultas como patrones de *triplas* sobre los cuales extraer instancias concretas.

3.1.4 Arquitectura

Tal y como se puede ver en las figuras 3.3 y 3.4, la arquitectura de la Web Semántica se fundamenta sobre los principios de la Web tradicional. Estos principios se pueden resumir:

1. Utilización de Localizador Uniforme de Recurso (URL) para la localización de recursos
2. Uso de HyperText Markup Language (HTML) para la elaboración de documentos, de modo que sea entendible por las personas y procesable por los computadores.
3. Uso de HyperText Transfer Protocol (HTTP) de comunicación cliente-servidor.

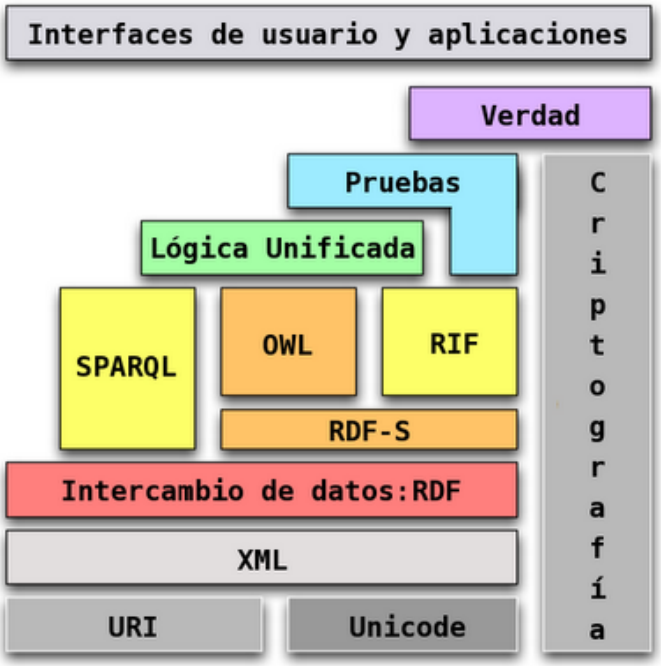

Figura 3.3: Tecnologías de Web Semántica

Las diferentes capas que se muestran en la figura 3.4 [San11] se pueden definir como :

- **Capa de localización y codificación**: el estándar para la codificación de caracteres es UNICODE y para la identificación y localización de recursos se utilizarán las URI o URL.
- **Capa de sintaxis**: estándares necesarios para la representación de información. Cobra especial importancia XML puesto que ofrece un formato de fácil procesamiento con una sintaxis jerárquica. Como derivación de XML surge XML *namespaces* que habilitará la aparición de diferentes vocabularios XML en un mismo documento. Esto permite la reutilización de recursos que previamente ya se hayan definido.
- **Capa de descripción y estructura**: el uso de RDF como estándar de representación de recursos, propiedades y relaciones, es el siguiente gran salto en la Web Semántica. Al tener una representación de la semántica formal, se consigue la interoperabilidad semántica entre aplicaciones heterogéneas.

- **Capa de integración lógica de ontologías y reglas de inferencia**: estrechamente vinculada con la capa de descripción y estructura, en esta capa se amplía la definición de clases, relaciones y propiedades entre recursos, utilizando para ello el lenguaje específico OWL.
- **Capa de consulta**: los recursos ya representados junto con sus relaciones ya conforman un hito. A continuación se requiere establecer unos mecanismos para búsqueda de triplas. SPARQL cumplirá la función de motor de búsqueda declarativo sobre archivos RDF.

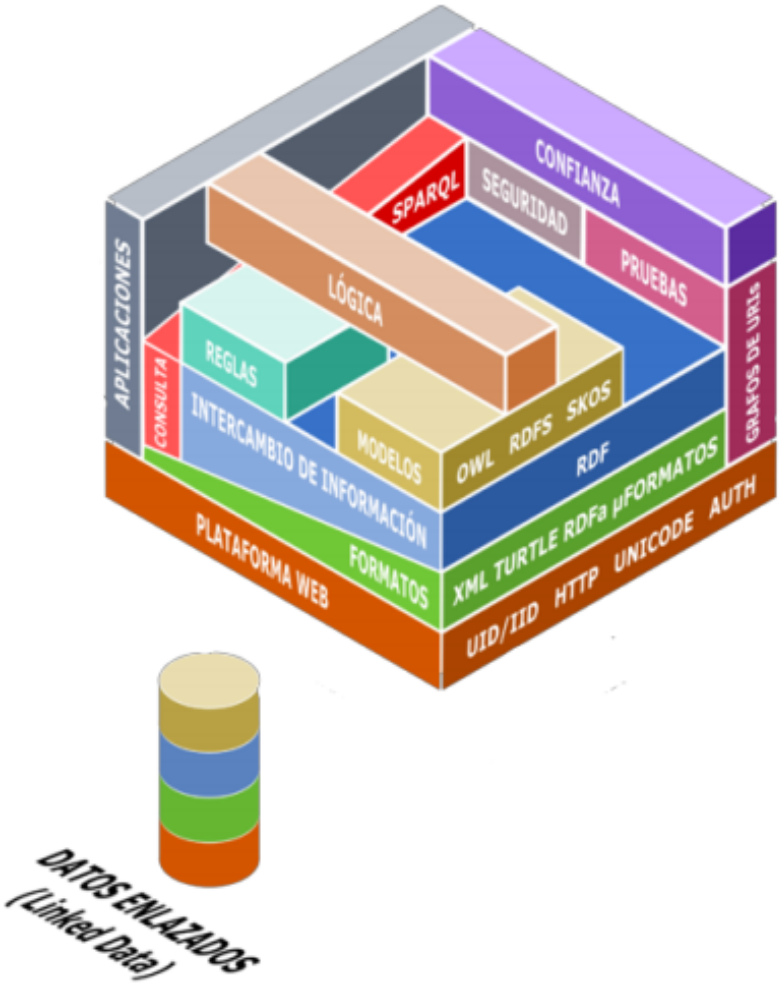

Figura 3.4: Arquitectura de la Web Semántica. Extraído de [San11]

3.1.5 Ontologías

Las ontologías pueden jugar un papel crucial para permitir el procesamiento del conocimiento, el intercambio y la reutilización basada en la Web entre aplicaciones. Las ontologías ofrecen una comprensión común de conceptos, con el fin de que la comunicación entre personas y aplicaciones sea homogénea [DMVH$^+$00].

Definiciones de ontologías

[Gru] define las ontologías como *una especificación explícita de una conceptualización*, refiriéndose este término al modelo abstracto de una realidad concreta.

En [SBF98] se profundiza en los términos que engloba la definición de ontología:

Una ontología es una especificación formal y explícita de una conceptualización compartida, donde:

- el término "conceptualización" se refiere a un modelo abstracto de algún fenómeno en el mundo, identificando los conceptos relevantes del mismo,
- "explícita" porque los tipos de conceptos utilizados así como las constantes en su uso están explícitamente definidas,
- "formal" se refiere al hecho de que la ontología debe ser legible para las computadoras, lo que excluye al lenguaje natural, y
- "compartida" refleja la noción de que una ontología captura el conocimiento consensuado, es decir, no es privativo para ningún individuo, sino que es aceptado por un grupo.

Luego una ontología es una jerarquía que define una serie de clases, atributos y relaciones para describir un dominio sobre un concepto en concreto cuya finalidad es servir de herramienta para la representación del conocimiento.

Componentes de una ontología

En [San11] se enumeran los distintos componentes de las ontologías:

- **Clases**: conceptos generales acerca de un determinado dominio. La idea básica que se pretende formalizar.
- **Relaciones**: enlace entre conceptos (clases) de un mismo dominio.
- **Atributos**: representan la estructura del concepto.
  - **Funciones**: identifican un elemento mediante el cálculo de una función.
- **Instancias**: representa un individuo concreto perteneciente a una clase.
- **Axiomas**: expresiones siempre ciertas sobre relaciones.

Clasificación de las ontologías

A su vez, dependiendo del objetivo de la ontología, se pueden clasificar en distintos ámbitos tal y como propone [Gua98] (véase figura 3.5):

- Ontologías de alto nivel (*Top-level ontologies*): describen conceptos muy generales que son independientes de un problema particular, tales como espacio, tiempo, objeto, evento o acción.

- Ontologías de dominio y tarea (*Domain ontologies and task ontologies*): describen respectivamente, el vocabulario relacionado con un dominio genérico o con una tarea genérica o actividad, especializando los términos de la ontología de alto nivel.
- Ontologías de aplicación (*Application ontologies*): describen conceptos dependiendo tanto del dominio particular como de la tarea, es decir, es una especialización que generalmente particulariza *ambos* tipos de ontología. Generalmente corresponde con *roles* desempeñados por entidades de dominio mientras desarrollan alguna tarea.

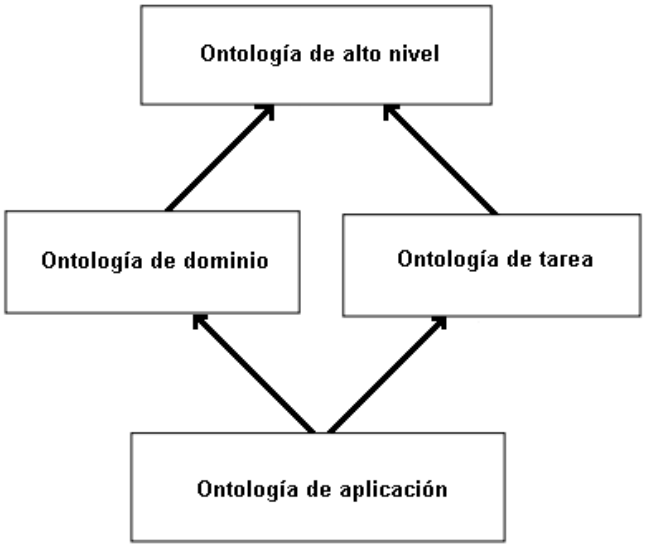

Figura 3.5: Clasificación de Ontologías. Extraído de [Gua98]

Metodología para desarrollo de una ontología en entornos de Web Semántica

Finalmente, [NMo01] propone una secuencia de tareas que todo desarrollo de cualquier ontología debería incluir:

1. Definir las clases en la ontología
2. Organizar dichas clases según una taxonomía
3. Definir propiedades de las clases y los valores que se asocian a esas propiedades
4. Completar los valores de las propiedades para cada una de las instancias.

3.1.6 Vocabularios

Como complemento al concepto de ontología, se describe a continuación el término vocabulario, frecuentemente usado en gran parte de la bibliografía.

Definición

En la Web Semántica, tal y como se explica en [W3Ca], los vocabularios definen los conceptos y relaciones usados para describir y representar un área de conocimiento y así clasificar los términos en una aplicación particular caracterizando relaciones y restricciones. Los vocabularios en la práctica pueden ser enormemente complejos o muy simples (llegando a describir únicamente uno o dos conceptos).

Según esta definición no queda claro en qué se diferencia un vocabulario de una ontología, pues realmente no existe una división clara entre estos dos conceptos. La tendencia es utilizar el término "ontología" para una colección de términos más compleja y formal, mientras que "vocabulario" se usa cuando la flexibilidad en el formalismo no implica necesariamente una pérdida de significado [W3Ca].

Por lo tanto, la función de un vocabulario es similar a la de una ontología en términos de objetivo final: establecer una representación formal de un determinado concepto.

Ejemplos de vocabulario

Existen numerosos vocabularios en uso actualmente. A continuación se citan algunos ejemplos:

Description Of A Project (DOAP)

El objetivo de este vocabulario es la descripción de proyectos software. Para ello, incluye toda la termonología referente al proyecto (licencia, versión de producto, dirección de repositorio, ...).

Simple Knowledge Organization System (SKOS)

Este vocabulario tiene por objetivo la representación y estructuración de esquemas conceptuales tales como taxonomías, esquemas de clasificación o tesauros.

Friend Of A Friend (FOAF)

Quizá uno de los más extendidos sea Friend Of A Friend (FOAF) [FOA]. Su finalidad es describir personas, relaciones entre ellas así como aspectos de su actividad. Esta tecnología está en alza debido a su extensión en las redes sociales. FOAF a día de hoy es la base de un número considerable de esfuerzo en lo que se conoce como movimiento "open social", que tratan de facilitar al usuario integrar su propia información a través de aplicaciones sociales a través de la Web [AH08].

FOAF trabaja según el principio de *Anyone can say Anything about Any topic* (AAA). En el caso de FOAF los temas usualmente son otras personas [AH08].

En la figura 3.6 y en el listado 3.1 pueden verse ejemplos de uso de este vocabulario. En la figura se aprecian dos recursos (`foaf:ian` y `foaf:mary`), dos propiedades (`foaf:knows` y `foaf:firstName`) y un literal ("`Mary`"). En el listado 3.1 se expone un documento describiendo algunas entidades y relaciones.

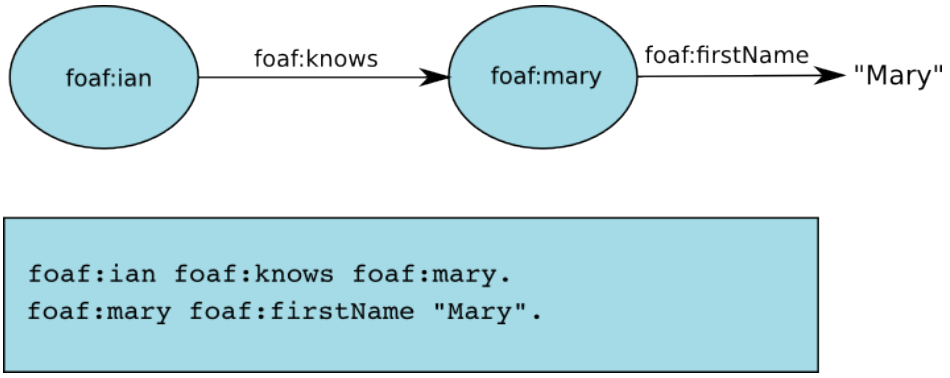

Figura 3.6: Ejemplo de FOAF. Extraído de [JEN14]

```
<rdf:RDF
  xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
  xmlns:rdfs=''http://www.w3.org/200/01/rdf-schema#''
  xmlns:foaf=''http://xmlns.com/foaf/0.1/''>

  <foaf:Person>
    <foaf:name>Raul Reguillo Carmona</foaf:name>
    <foaf:title>Mr</foaf:title>
    <foaf:nick>Radulfr</foaf:nick>
    <foaf:weblog rdf:resource=''http://geeklandtryp.blogspot.com.es />
    <foaf:knows>
      <foaf:Person>
        <foaf:name>Ismael Caballero</foaf:name>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
</rdf:RDF>
```

Listado 3.1: Ejemplo de FOAF

3.1.7 Frameworks para el desarrollo de aplicaciónes de Web Semántica

En [W3Cc] se puede consultar una lista con todos los frameworks de Web Semántica disponibles hasta la fecha. En este apartado, se van a nombrar solamente algunos de ellos por ser especialmente significativos.

Apache Jena

Jena es un framework para la construcción de aplicaciones que utilizan tecnologías semánticas y Linked Data [JEN14].

En la arquitectura de Jena (véase figura 3.7) se pueden encontrar tres bloques diferenciados en función de las herramientas que engloban:

- **RDF**
  - Jena ofrece una API para el tratamiento con grafos RDF, serializando las triplas y tratando con ellas de manera ágil y eficiente.

- Paralelamente ofrece un motor SPARQL para las consultas sobre los archivos semánticos.

- **Triple Store**
  - Para hacer persistentes los datos, Jena ofrece Triple Data Base (TDB): una base de datos de triplas nativa de alto rendimiento y alineada con el resto de tecnologías de Jena.
  - Por otra parte, Jena ofrece **Fuseki** como *endpoint*, es decir, como servidor de triplas accesible a través de HTTP, integrándose a la perfección con TDB.

- **OWL**
  - Para trabajar con modelos y OWL, Jena propone una API específica orientada a ontologías.
  - Jena igualmente propone una API para facilitar el razonamiento y comprobar el contenido de los archivos semánticos. Permite especificar distintos razonadores.

A continuación se van a definir algunos de los conceptos que maneja Jena así como se va a introducir brevemente el ámbito de la inferencia en este framework.

Modelo

A la hora de trabajar con triplas, jena utiliza los llamados **modelos**. Un modelo de Jena es un conjunto de triplas RDF con el que se puede trabajar de diversas formas: bien para hacer consultas sobre él o para aplicar inferencia. Estos modelos son la piedra angular de la tecnología puesto que son la fuente del resto de operaciones que se desarrollan en este framework.

Inferencia

La inferencia es un proceso abstracto de derivación de información adicional partiendo de los datos [JEN14]. De esta manera se utilizará la inferencia para hacer explícita información que aparece de forma implícita en los datos.

La inferencia en Jena consta de dos partes fundamentales:

- **Razonadores**: encargados de llevar a cabo la inferencia, pudiendo encontrar en Jena los siguientes [JEN14]:
  1. Razonador transitivo: ofrece soporte para el razonamiento a través de los conceptos de clase y propiedad. Únicamente afecta a propiedades transitivas y reflexivas de `rdfs:subPropertyOf` y `rdfs:subClassOf`.
  2. Razonador de reglas RDFS: Implementando un subconjunto configurable de vínculos RDFS.
  3. Razonadores para OWL, OWL Mini, OWL Micro: Un conjunto no completo para la implementación de inferencia basada en OWL/Lite y OWL/Full.

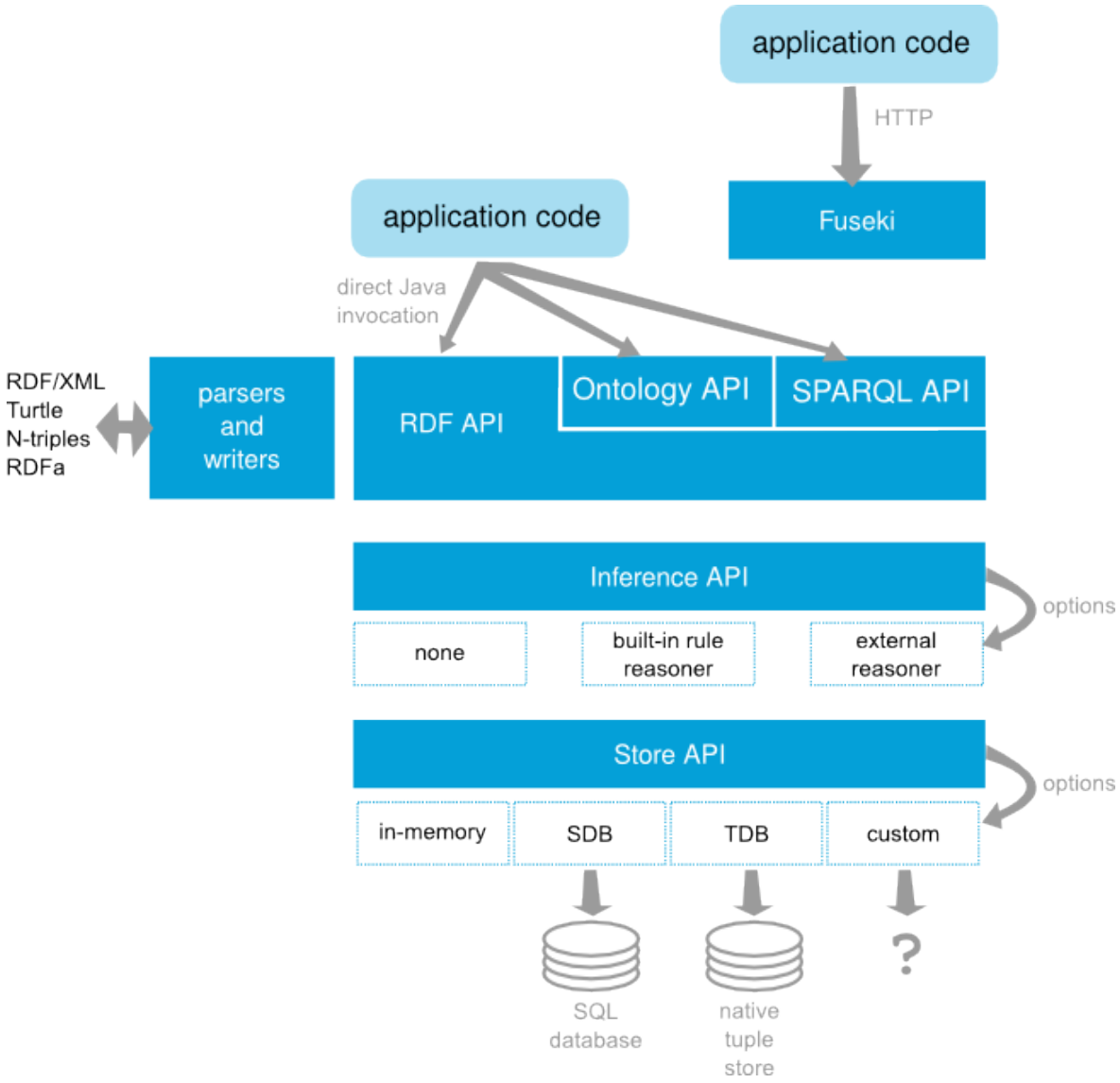

Figura 3.7: Diagrama de la arquitectura de Jena. Extraída de [JEN14]

4. Razonador de reglas genéricas: Se basa en reglas definidas, mediante el encadenamiento hacia adelante, hacia atrás e híbrido.

- **Reglas**: las reglas son sentencias usadas para aplicar inferencia en un determinado modelo. Jena implementa un lenguaje propio para la edición de estas reglas.

Se puede consultar un esquema del razonamiento en Jena en la figura 3.8.

Sesame

OpenRDF Sesame es un framework estándar *de-facto* para el procesado de datos RDF [OPE]. Sesame incluye operaciones para el procesamiento, consulta, almacenamiento e inferencia sobre RDF. Al igual que Jena, Sesame contiene un número considerable de herramientas para la construcción de aplicaciones de tecnologías semánticas.

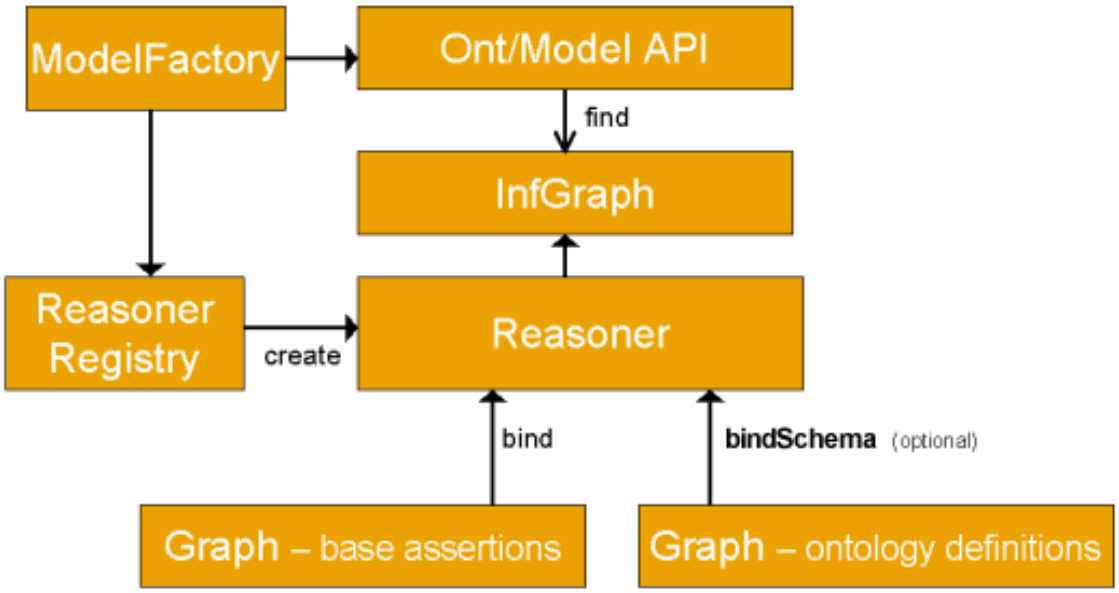

Figura 3.8: Razonamiento en Jena. Extraída de [JEN14]

CubicWeb

CubicWeb [CUB] es otro framework, *open source*, para la realización de aplicaciones semánticas. Está escrito en Python. Sus características engloban:

- Soporta OWL y RDF
- Relationship Query Language (RQL)
- Herramientas de migración para desarrollo ágil
- Una librería de *cubes* como pequeños módulos al estilo de Ruby.

Comparativa

En el cuadro 3.1 se pueden contemplar algunas características de los frameworks anotados anteriormente, pudiendo encontrarse una relación más completa en [W3Cc]:

Framework	Lenguaje	Licencia	Versión
Apache Jena	Java	Apache License 2.0	2.11.0 / September 18, 2013
Sesame	Java	BSD-style license	2.7.11 / March 27, 2014
CubicWeb	Python	Lesser General Public License	3.18.4 / April 7, 2014

Tabla 3.1: Comparativa de frameworks de Web Semántica. Extraída de [W3Cc]

3.2 Datos Enlazados (LD)

En esta sección se incluyen los conceptos clave respecto de los principales movimientos de datos abiertos y datos enlazados.

3.2.1 Definición de LD

Berners-Lee en [BL09] enfatiza la publicación de los datos en la Web no únicamente como exposición de los mismos, sino mediante el establecimiento de enlaces de forma que personas o máquinas puedan explorar una Web de datos. De esta manera se pueden encontrar datos relacionados.

La mayor parte del contenido en la Web está construido con documentos. Estos documentos a su vez tienen enlaces hacia otros documentos cuyo contenido puede estar o no formalizado. El movimiento LD pretende dar un paso más allá, estableciendo relaciones entre los datos de manera global (véase figura 3.9). Para ello, RDF y las URI toman un papel crucial. [BL09] expone sus cuatro reglas para LD:

1. Usar URI como nombres para los conceptos.
2. Usar HTTP para que las personas puedan acceder a esos nombres.
3. Proporcionar información útil cuando la URI sea desreferenciada, usando estándares como RDF o SPARQL.
4. Incluir enlaces a otras URI, de manera que los usuarios puedan descubrir nuevos conceptos.

[BHBL09] establece a su vez una serie de pasos básicos para la publicación de LD:

1. Asignar URI a las entidades descritas por el conjunto de datos y proveer el desreferenciado de las URI a través del protocolo HTTP en representaciones de RDF.
2. Establecer enlaces RDF a otros recursos de datos en la Web, de tal manera que los usuarios puedan navegar a través de la Web de los datos siguiendo enlaces RDF.
3. Facilitar metadatos sobre los datos publicados, de manera que los usuarios puedan evaluar la calidad de los datos publicados y escoger entre diferentes medios de acceso.

Aprovechando LD la información en la Web Semántica puede verse desde diferentes niveles de granularidad, desde el grafo universal formado por todos los documentos RDF en la Web, pasando por documentos individuales hasta sus triplas [DFP+05] (véase figura 3.10).

3.2.2 Datos Abiertos (OD) y Datos Enlazados Abiertos (LOD)

Por otra parte los Datos Abiertos (OD), se presentan en ausencia de formatos privativos, información pública y reutilizables procedentes de organizaciones tales como las gubernamentales. Así, Datos Enlazados Abiertos (LOD) serán todos aquellos datos enlazados que

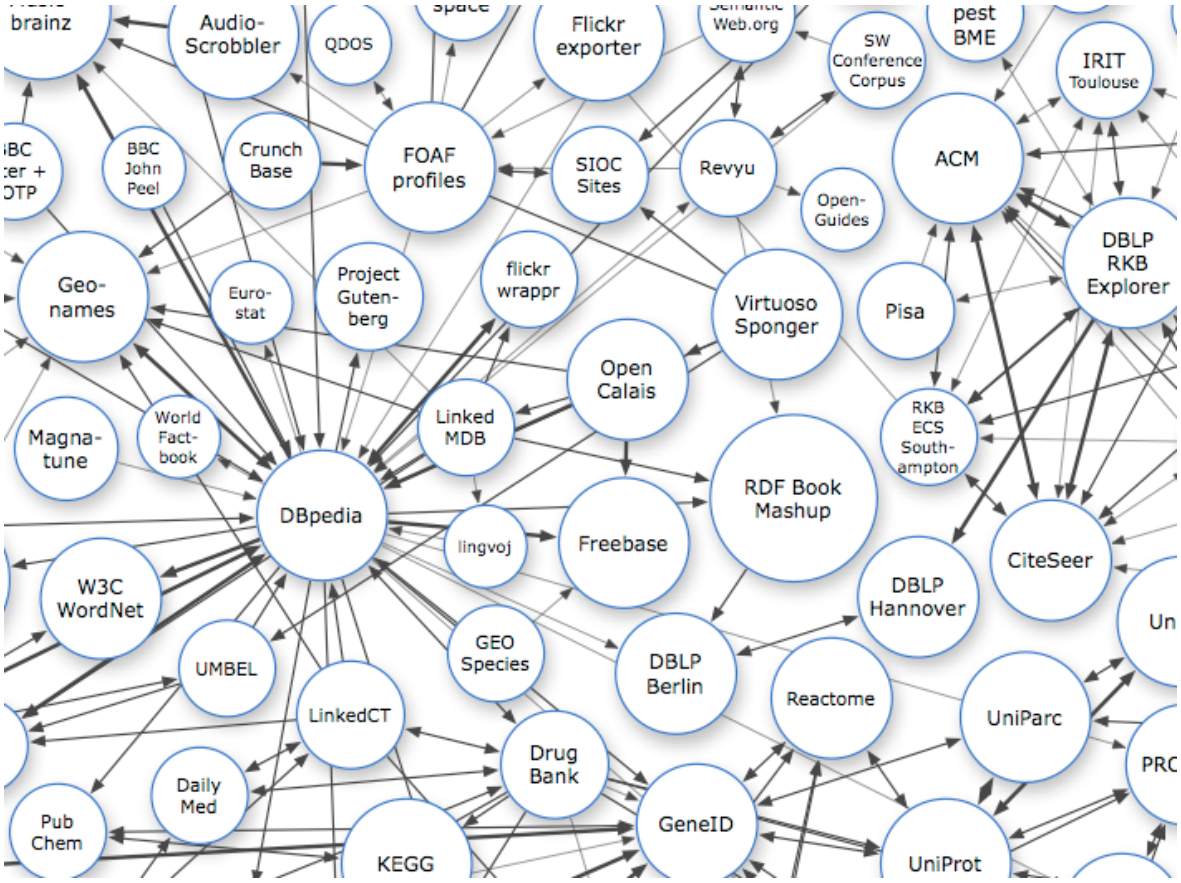

Figura 3.9: Esquema LD DBpedia

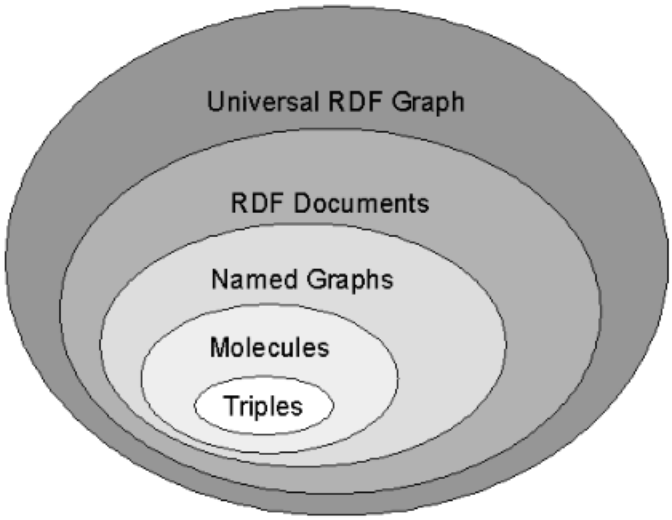

Figura 3.10: Niveles de granularidad en la Web Semántica. Extraído de [DFP+05]

sean abiertos. [BL09] define LOD como LD que es liberado bajo licencia abierta que no impida su reuso de manera gratuita.

Asimismo, define las cinco estrellas de calidad de LOD:

- ⋆ Los datos están disponibles en la Web, independientemente del formato, pero con licencia abierta, para que sean OD.
- ⋆⋆ Disponible para la lectura por parte de máquinas, es decir, datos estructurados. Por ejemplo, utilizar una tabla Excel en lugar de una imagen escaneada de dicha tabla.
- ⋆ ⋆ ⋆ Como el anterior, pero en formato no propietario. En este caso formato Comma Separated Values (CSV) en lugar de Excel.
- ⋆⋆⋆⋆ Todo lo anterior y además, usar estándares abiertos de W3C (RDF y SPARQL) para identificar conceptos, de manera que otros usuarios puedan apuntar a este contenido.
- ⋆ ⋆ ⋆ ⋆ ⋆ Lo anterior y además, enlazar estos datos a los de otros usuarios para proporcionar un contexto.

3.2.3 LOD en la actualidad

Empresas del sector privado y público se han dado cuenta de las ventajas que OD y LOD pueden ofrecerles. Es por ello que están surgiendo muchas aplicaciones tanto a nivel nacional, donde estos movimientos aún no están demasiado extendidos, como a nivel internacional. En [APP] se encuentra una lista con una serie de aplicaciones realizadas utilizando tecnología OD y LOD. A continuación se exponen algunos ejemplos.

Bus Guru

Bus Guru es una aplicación para iOS que monitoriza en tiempo real la situación de los autobuses urbanos así como su hora de llegada estimada para una parada (véase figura 3.12).

Bus Gijón

Igualmente en el ámbito nacional, existen aplicaciones que aprovechan los datos abiertos generados en tiempo real por dispositivos empotrados en autobuses y marquesinas. Un caso homólogo a **Bus Gurú** es **Bus Gijón** (figura 3.11), que recaba información del portal de OD `https://datos.gijon.es` para obtener información de los autobuses.

3.3 Calidad de Datos

En [SLW97] se identifican tres roles a la hora de tratar los datos:

1. *Data producers*: aquellas entidades encargadas de **generar** los datos.
2. *Data custodians*: personas responsables del **almacenamiento y procesado** de los datos.
3. *Data consumers*: personas o grupos que **usan** los datos.

Debido a que los diferentes roles tendrán concepciones distintas de lo que los datos significan para ellos según su papel en un sistema, pueden encontrarse distintas perspectivas de lo

Figura 3.11: Bus Gijón App

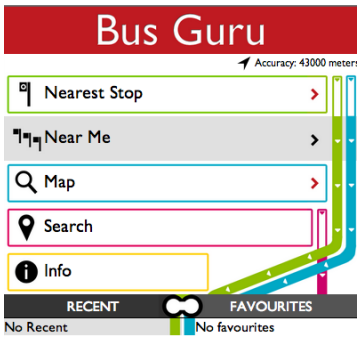

Figura 3.12: Bus Gurú App

que significa calidad de datos. Dos de las principales perspectivas son *Meeting Requirements* y *Fitness for Use*.

- *Meeting requirements* [BH12]: Los datos tienen calidad si satisfacen los requisitos que fueron establecidos. Los **requisitos de datos** que son especificados, por ejemplo, con un modelo Entidad-Interrelación (ER) en el que se subrayan cómo se van a relacionar los datos como conjunto, es decir, un marco arquitectónico para los datos. En [FH10] y [FH11] se pueden consultar aproximaciones a la calidad de datos desde este punto de vista.

- *Fitness for use* [SLW97]: Se dice que los datos tienen calidad si son válidos para el propósito por el que son requeridos, es decir, considerando la calidad de los datos en el **contexto** de uso. En [CVCP08] se adopta esta aproximación.

3.3.1 Dimensiones de Calidad de Datos

La perspectiva *Fitness for Use* se centra en los *Data consumers*, considerando datos de alta calidad aquéllos que son apropiados al usuario final en el contexto de uso. Debido a ello, se deben considerar aspectos tales como *utilidad* o *usabilidad* y en definitiva, todo aquel aspecto que pueda repercutir en la experiencia del usuario.

Puesto que la calidad de los datos dependerá del propósito, se requiere considerar una serie de **categorías**. La calidad de datos es un concepto que es necesario evaluar desde distintos criterios o **dimensiones** (Dimensión de Calidad de Datos (DQD)) [SLW97]. Al conjunto de dimensiones de calidad de datos utilizados para evaluar un conjunto de datos se le conoce como **Modelo de Calidad de Datos**. En el cuadro 3.2 se detallan conjuntos de dimensiones de calidad agrupadas por categorías.

Categoría de Calidad de Datos	Dimensiones de Calidad de Datos
Intrínseca	Precisión, Objetividad, Credibilidad, Reputación
Accesibilidad	Accesibilidad, Acceso seguro
Contextual	Relevancia, Valor añadido, Temporalidad, Completitud, Cantidad de datos adecuada
Representacional	Interpretabilidad, Facilidad de entendimiento, Representación concisa, Representación consistente

Tabla 3.2: Categorías y dimensiones de DQ. Extraído de [SLW97]

Pese a que existan modelos de calidad de datos consistentes, la calidad de datos no deja de ser un concepto subjetivo, puesto que para un mismo conjunto de datos se pueden obtener niveles de calidad radicalmente distintos dependiendo del usuario o el rol que haga uso de ellos, incluso sobre la misma dimensión de calidad [Wan98].

A continuación se detallarán algunas dimensiones de calidad de datos.

3.3.2 Completeness

Dentro de todas las posibles dimensiones de calidad, existen subconjuntos que cobran especial relevancia. *Completeness* es, por norma general para todos los autores, una dimensión de obligada presencia en todos los trabajos.

Dentro de esta dimensión, en la bibliografía se proponen distintas métricas que contemplan el concepto de *completitud* desde diversas perspectivas. Como se comprobará en adelante (véase Sección 3.4), en el presente trabajo se ha optado por una única visión, más compacta, sencilla y acorde a lo que un usuario puede esperar acerca de la presencia de valores no nulos en Linked Data.

Definición

[ZRM+13] define **Completeness** como el grado en el que toda la información requerida está presente en un conjunto de datos particular. En general, esta definición se puede extender sobre otros factores tales como profundidad de los datos, anchura y alcance para la tarea que se quiera realizar.

[PLW02] amplía esta definición. Define Completeness como el grado en que los datos no han desaparecido y poseen la suficiente amplitud y profundidad para la tarea en cuestión.

Por otro lado, la norma ISO 25012 (véase [ISO]) define Completeness como el grado en el que los datos asociados con una entidad tienen valores para todos los atributos esperados e instancias relacionadas en un contexto de uso específico.

3.3.3 Accesibility

En entornos de Web Semántica y Linked Data tiene especial importancia el hecho de que dichos datos sean de fácil acceso. La palabra *accesiblidad* cuando se habla de datos enlazados cobra un valor mayor que en otras dimensiones. Es preciso que los datos enlazados estén, en efecto, adecuadamente enlazados.

Definición

ISO 25012 (véase [ISO]) define *Accessibility* como el grado en el que los datos puedan ser accedidos en un contexto de uso específico, particularmente por gente que necesite tecnología de apoyo o una configuración especial debido a alguna discapacidad.

Por otra parte [PLW02] define Accessibility como el grado en que los datos están disponibles y son accesibles fácil y rápidamente.

3.4 Calidad de Datos en LD

Actualmente, existen autores tales como [ZRM+13] o [FH11] que comienzan a aplicar conceptos de LD para algunas dimensiones de calidad. Para ilustrarlo, se muestran algunas métricas propuestas en estos trabajos para las DQD Completeness y Accessibility.

Métricas para Completeness

[ZRM+13] propone una serie de métricas para la evaluación de esta dimensión.

1. *Schema Completeness*: grado en el que las clases y propiedades de una ontología están representadas. También conocido como *ontology completeness*.
2. *Property Completeness*: evaluación sobre los valores perdidos de una propiedad específica.
3. *Population Completeness*: siendo el porcentaje de todos los objetos del mundo real de un determinado tipo que están representados en los conjuntos de datos.

4. *Interlinking completeness*: refiriéndose al grado en el que las instancias en el conjunto de datos están interconectadas. Esta métrica consta de especial importancia en Linked Data. No obstante esta métrica concreta puede entenderse, como así se abordará en este trabajo, como una particularización de *Accesibility*.

[FH10] ofrece un conjunto de consultas SPARQL que permiten identificar problemas de integración de calidad de datos. Se pueden comprobar dos ejemplos en los listados 3.2 y 3.3 extraídos de ese mismo trabajo.

```
SELECT ?s
WHERE { {
  ?s a <class1> .
  ?s <prop1> "". }
UNION{
  ?s a <class1> .
  NOT EXISTS {
  ?s prop1> ?value}}}
```

Listado 3.2: Consulta SPARQL para identificación de literales perdidos (I)

```
SELECT ?s
WHERE { {
      ?s a <class1> .
      ?s <prop1> <value1>.
   NOT EXISTS{
      ?s <prop2> ?value2 .
   }
   }UNION{
      ?s <prop1> <value1> .
      ?s <prop2> "".
   }}
```

Listado 3.3: Consulta SPARQL para identificación de literales perdidos (y II)

Métricas para Accessibility

[ZRM+13] considera Accessibility como una categoría que a su vez contiene cuatro dimensiones, cada una con sus métricas:

- **Disponibilidad**: grado en el la información está presente y preparada para su uso. Sus métricas propuestas son:
  - *Accessibility of the server*: comprobación sobre el servidor SPARQL ante una consulta.
  - *Accessibility of the* SPARQL *endpoint*: comprobación sobre el servidor SPARQL ante una consulta.

- *Accessibility of the* RDF *dumps*: comprobación sobre la recuperación de datos en un contenedor de datos RDF.
- *Dereferencability issues*: comprobación en el momento que una URI devuelva un error del código de respuesta o enlace roto.
- *No structured data available*: detección de enlace caído o cuando una URI sin metadatos RDF o sin redirección, devuelva el código de error pertinente.
- *Misreported content types*: detección de si el contenido es susceptible a ser consumido y si el contenido puede ser accedido.
- *No dereferenced back-links*: detección de todos los enlaces propios al conjunto de datos: localmente disponibles.

- **Rendimiento**: Eficiencia del sistema vinculado al conjunto de datos de manera que cuanto más eficiente sea la fuente de datos, un sistema puede procesar más eficientemente los datos. Las métricas para esta dimensión son:
  - *No usage of slash*-URIs: uso de URIs abreviadas cuando existen grandes cantidades de datos.
  - *Low latency*: si una petición HTTP es contestada en un tiempo medio de un segundo.
  - *High throughput*: número de peticiones HTTP contestadas por segundo.
  - *Scalability of a data source*: detección de si el tiempo en responder una cantidad de diez peticiones, dividido entre diez, no es mayor que el tiempo en responder una petición.
  - *No use of prolix* RDF *features*: detección del uso de primitivas RDF tales como contenedores y colecciones.
- **Seguridad**: Grado en el que los datos pueden ser restringidos y de esta manera protegidos contra alteraciones ilegales y uso no permitido. Las métricas propuestas son las siguientes:
  - *Access to data is secure*: uso para login de credenciales o uso de protocolos específicos.
  - *Data is of propietary nature*: el propietario de los datos permite el acceso únicamente a ciertos usuarios.
- **Tiempo de respuesta**: Medición del retardo, generalmente en segundos, entre el envío de una consulta por el usuario y la recepción de los resultados. Existe una métrica para esta dimensión:
  - *Delay in response time*: retardo entre el tiempo de envío de una petición por el usuario y la recepción de dicha petición por el sistema.

3.5 Big Data

3.5.1 Concepto

Desde los comienzos del siglo XXI han tiendo lugar una serie de cambios muy significativos en la industria de las tecnologías de la información, tales como el nacimiento del Internet de las Cosas, Computación en la Nube y la aparición de las Redes Sociales.

El desarrollo y explotación de estas tecnologías ha propiciado que la cantidad de los datos que se generan por unidad de tiempo se haya incrementado a una velocidad sin precedentes, trayendo consigo la necesidad de cómputo de tales volúmenes de información heterogénea, en tiempos razonables y de manera escalable.

Big Data es por lo tanto un concepto que surge debido a la necesidad del uso de herramientas y técnicas específicas para procesar esta información, imposible para los sistemas transaccionales comunes [BAR].

De esta manera, el concepto *Big Data* se puede caracterizar según la definición de las tres V's:

- **Volumen**: en referencia al tamaño del conjunto de datos. No hay un número estándar que defina a partir de qué tamaño del conjunto de datos se pueda empezar a hablar de Big Data. No obstante, cuando el conjunto de datos no es susceptible de ser procesado en un único nodo, y requiere la presencia de varios de éstos, puede decirse que es Big Data. Luego el orden de magnitud puede ir desde los cientos de GB hasta los Petabytes.
- **Velocidad**: referenciando a las necesidades relativas al consumo o proceso de datos en tanto se aproximen al tiempo real (*real time*) o próximos al tiempo real (*near real time*).
- **Variedad**: en relación ala heterogeneidad de los datos. Esto incluye representaciones diversas de datos estructurados, como diversas fuentes relacionales, datos semiestructurados o datos no estructurados, como pudieran ser imágenes o vídeo.

Adicionalmente se ha extendido esta definición a medida que las tecnologías y la problemática ha cambiado, encontrando:

- **Valor**: considerando el beneficio que se puede extraer de los datos bajo la premisa que tanto la complejidad de la extracción del valor, como dicho beneficio en sí mismo se incrementa junto con el volumen.
- **Veracidad**: entendiendo como el grado de confianza que se establece sobre los datos.

Cada dimensión deja de manifiesto un problema concreto que hay que afrontar en un contexto Big Data. No obstante estas definiciones, a medida que las tecnologías evolucionan, se adaptan y renuevan, pudiendo hablar sobre la existencia de hasta 9Vs.

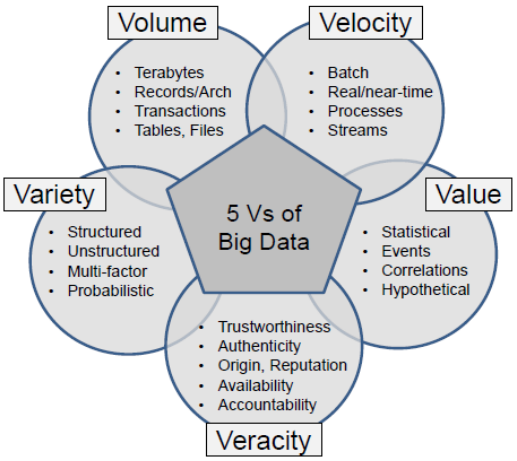

Figura 3.13: 5vs del Big Data

En [DG04] se define por primera vez un modelo de procesamiento y generación de grandes conjuntos de datos, a través de funciones *map* y *reduce*, apoyándose en que muchos problemas del mundo real son expresables en estos términos.

En adelante se desarrollaron multitud de frameworks y herramientas que basándose en estos principios optimizarán y permitirán el procesado escalable de grandes volúmenes de información, cumpliendo con el paradigma de las 5Vs. Entre ellos, se encuentra el ecosistema Hadoop que se describirá con detalle en la sección Ecosistema Apache Hadoop.

Este modelo consiste en el particionado de los datos de entrada y procesado en paralelo de dichas particiones por varios nodos, gestionados por un nodo maestro que ejerce de árbitro y orquestador. En la etapa final, la unificación de los resultados parciales en torno a una solución final se lleva a cabo por otro conjunto de nodos que combinarán el resultado en uno o varios archivos de salida.

Al cierre del 2015, los miembros de la ITU (Unión Internacional de Telecomunicaciones), aprobaron la primera norma de Big Data, estableciendo la primera definición oficial [ITU]:

Paradigma para hacer posible la recopilación, el almacenamiento, la gestión, el análisis y la visualización, potencialmente en condiciones de tiempo real, de grandes conjuntos de datos con características heterogéneas.

3.5.2 Desafíos

Cuando la volumetría y la disparidad de los datos está presente, existe una serie de desafíos que deben afrontarse a la hora de acometer un proyecto [CZ14]:

- **Captura y almacenamiento**: debido al ritmo de crecimiento de los datos así como a su origen disperso, el modo en el que se acceden, capturan y almacenan representa una

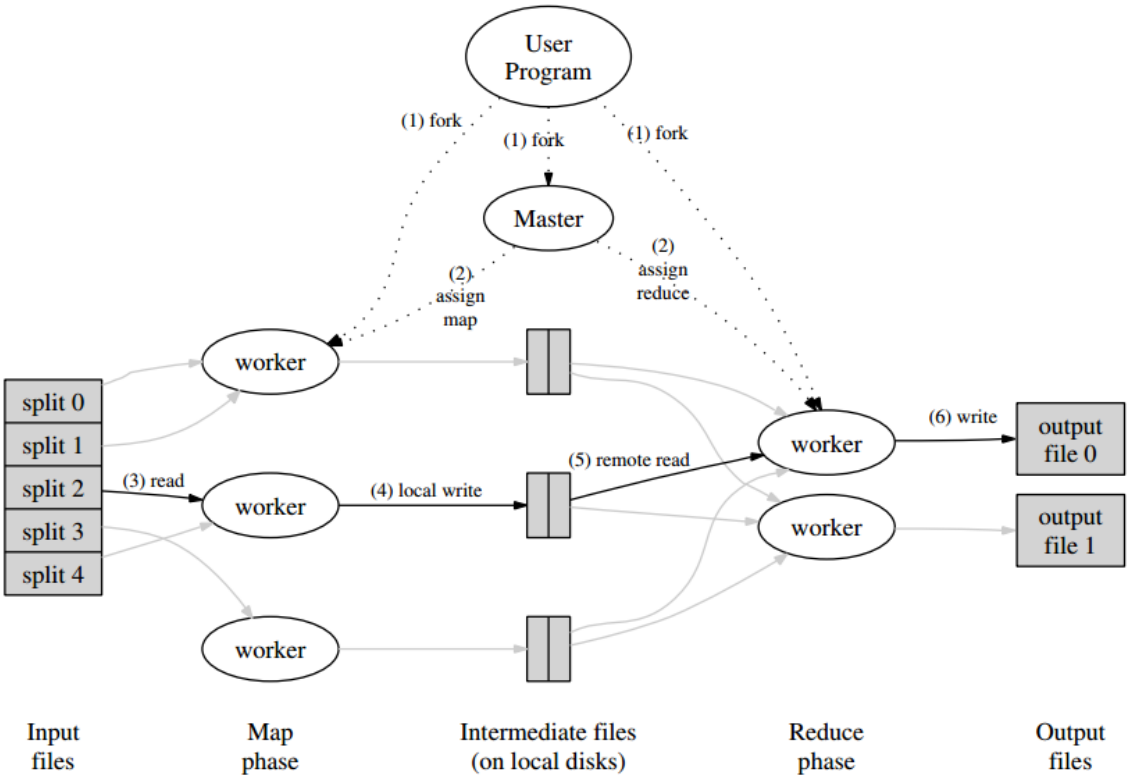

Figura 3.14: MapReduce: ejecución [DG04]

primera barrera a superar.

- **Descubrimiento y limpieza**: referente a tratamiento de la información previo a su procesamiento, análisis exploratorio, gestión de datos perdidos o corruptos.

- **Transmisión**: el ancho de banda es un factor limitante cuando se habla en un contexto de Big Data y puede representar el cuello de botella de un sistema si no se gestiona correctamente el movimiento de tales volúmenes de datos.

- **Análisis y visualización**: llevar a cabo un análisis de volúmenes masivos de datos implica desarrollar técnicas tanto a nivel hardware como software para permitir dichos análisis. Igualmente la visualización resulta un factor crucial a la hora de obtener lecturas fiables de cara a la interpretabilidad de dichos datos.

3.5.3 Frameworks

Existe una gran cantidad de frameworks y ecosistemas dedicados al procesamiento distribuido de grandes volúmenes de información. Entre ellos caben destacar el ecosistema Hadoop y las soluciones de Amazon Web Services, que serán descritos a continuación.

Ecosistema Apache Hadoop

El ecosistema Hadoop se ha convertido en el estándar *de facto* a la hora de llevar a cabo proyectos Big Data. Se organiza a través de una serie de módulos encargados de diversas operaciones en el contexto de un cluster Big Data [HAD].

Entre sus principales componentes caben destacar los siguientes:

- **Hadoop Distributed File System (HDFS)**: como sistema de archivos distribuido y escalable.
- **Hadoop MapReduce**: framework para escribir aplicaciones que puedan procesar grandes volúmenes de información en paralelo en el contexto de un cluster, de una manera tolerante a fallos, siguiendo el modelo MapReduce.
- **YARN**: Yet ANother Resource Negotiator es un framework para la gestión de recursos de un cluster, encargado de balancear la carga, asignar nodos a los distintos trabajos y liberar recursos.
- **Pig**: plataforma de alto nivel para crear programas que se ejecutan sobre HDFS, a través de un lenguaje de programación de scripts que permiten realizar operaciones ETL (Extract, Transform and Load) sobre datos guardados en HDFS.
- **Hive**: proyecto que se ejecuta sobre HDFS con el fin de establecer una plataforma de consulta y análisis a través de una interfaz SQL.
- **Oozie**: orquestador de tareas dentro del ecosistema Hadoop.
- **Sqoop**: herramienta para el traspaso eficiente de datos entre HDFS y otras soluciones de almacenamiento, tales como bases de datos relacionales o ficheros de texto plano.
- **Flume**: solución de recolección y distribución de grandes volúmenes de datos entre distintas fuentes.
- **HBase**: base de datos distribuida, escalable y no relacional.
- **Zookeeper**: servicio centralizado para el mantenimiento, la configuración, nombrado y sincronización de los distintos servicios ejecutándose en el contexto de un cluster Hadoop.

Posteriormente se han ido agregando a proyectos apache nuevos frameworks destinados a cubrir necesidades específicas o a mejorar el rendimiento de los actuales:

- **Spark**: [SPAa] es un framework para el procesado de grandes volúmenes de información. La ventaja principal respecto de Hadoop MapReduce es que el procesamiento se realiza en memoria en lugar de en disco, con lo cual la velocidad se incrementa en varios órdenes de magnitud. Se basa en el concepto de Resilient Distributed Dataset (RDD) como estructura de datos distribuida básica, asentando sobre ésta otras abstracciones como son los Dataframes, que en versiones recientes del framework se

han convertido en la recomendación de uso. Análogos a los Dataframes de lenguajes como R, son estructuras de datos similares a tablas de bases de datos relacionales y ofrecen primitivas de consulta similares. Spark además tiene como complemento una serie de herramientas de alto nivel:

- **SparkSQL**: módulo para el procesamiento distribuido de datos estructurados. Ofrece una interfaz parecida a consultas SQL y mantiene información estructural acerca de los datos distribuidos.
- **MLlib**: librería de machine learning que incluye una serie de algoritmos distribuidos, supervisados y no supervisados, pipelines, persistencia y utilidades para dar soporte al proceso de aprendizaje automático de manera distribuida.
- **GraphX**: es uno de los componentes más recientes de Spark. Incluye una abstracción *Graph* que trabaja directamente sobre los dataset distribuidos permitiendo operaciones básicas en el manejo de grafos.
- **Spark Streaming**: es una extensión del núcleo de Spark, que introduce el concepto de mini-batch como unidad operativa básica y ofrece un procesamiento distribuido y tolerante a fallos en streaming. Los datos pueden ser ingestados desde distintas fuentes, tales como Kafka, Kinesis o sockets TCP.

- **Kafka**: plataforma de streaming distribuida, ideada para dar soporte a aplicaciones real-time.
- **Mahout**: framework que ofrece una serie de algoritmos distribuidos de Machine Learning.
- **NiFi**: Apache NiFi [NIF] es un proyecto multipropósito, pero principalmente enfocado a la automatización del flujo de datos entre distintos componentes software en un entorno Big Data. Se basa en el concepto de topologías y *Flow Files* y permite conectar un gran número de componentes entre sí con una interfaz *drag and drop*, programar tareas y aplicar transformaciones a los datos a medida que éstos pasan por la topología.

Soluciones Amazon Web Services

Amazon Web Services es una colección de servicios en la nube, homólogos a los que se encuentran en el ecosistema Hadoop, que establece una plataforma de computación.

Entre los principales caben destacar:

- **S3**: Simple Storage Service, equivalente a HDFS, ofrece un sistema de archivos distribuido, altamente escalable e integrado en los demás servicios.
- **ElasticSearch y Kibana**: ElasticSearch es un indexador de documentos frecuentemente usado como capa de baja latencia para consumir información y aplicar agregaciones sencillas. Kibana, apoyado sobre ElasticSearch, es una herramienta de visualización de datos, altamente configurable.

- **EC2**: Elastic Compute Cloud, ofrece capacidad de cómputo adaptable a través de servidores que pueden realizar autoescalado en los centros de datos de amazon, versátiles para hospedar sistemas software.
- **EMR**: Ealstic Map Reducec proporciona un marco de Hadoop hospedado que permite hospedar datos en instancias EC2 escaladas dinámicamente.
- **Kinesis**: equivalente a Kafka, permite la recopilación, procesamiento y el análisis de datos streaming en tiempo real.

3.5.4 Datos Semánticos en entornos Big Data

Si bien los frameworks anteriormente descritos ofrecen un conjunto de herramientas de procesado de datos de propósito general, no existe a día de hoy una versión madura de éstos que se haya convertido en un estándar para el procesado de grandes volúmenes de datos semánticos. Si bien existen un par de referencias que están comenzando a abrir esta vía:

- **Jena Elephas**: como parte del stack de Jena, Elephas es un conjunto de librerías que proveen de herramientas para el procesado de datos RDF sobre Hadoop. No obstante el módulo se encuentra actualmente en una fase Beta [JEN].
- **Sansa Stack**: Scalable Semantic Analytics Stack, es un motor de procesamiento tolerante a fallos que permite computación distribuida sobre datos RDF. Actualmente en la versión 0.2 y en desarrollo [SAN].

Capítulo 4

Método de Trabajo

4.1 Proceso Unificado de Desarrollo (PUD)

En [RJB] se define el Proceso Unificado de Desarrollo (PUD) como un proceso de desarrollo de software. A su vez, un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de usuario en un sistema software (véase figura 4.1).

Figura 4.1: Proceso Software [RJB]

El PUD está basado en *componentes*, lo que significa que el software en construcción está formado por componentes interconectados a través de interfaces bien definidas.

Para diseñar y desarrollar todos los artefactos de un sistema software, el PUD utiliza un lenguaje unificado de desarrollo denominado Unified Modeling Language (UML). Se puede consultar más información sobre UML en [RJB04].

[RJB] define las principales características del PUD en:

1. Dirigido por casos de uso.
2. Centrado en la arquitectura.
3. Iterativo e incremental.

4.1.1 Características del Proceso Unificado de Desarrollo

Dirigido por casos de uso

La finalidad de un sistema software es proporcionar servicio a los usuarios [RJB]. Estos usuarios pueden ser tanto humanos como otros sistemas, luego el término *usuario* se refiere a toda aquella entidad que interactúa con el sistema que se está desarrollando. Es una interacción concreta la que da nombre al concepto Caso de Uso (CDU).

En [RJB] se define **caso de uso** como un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante.

La función de los casos de uso es representar los requisitos funcionales del sistema. Al conjunto de todos los casos de uso se le denomina **modelo de casos de uso**. [RJB] define al modelo de casos de uso a aquel formado por casos de uso, actores y sus relaciones. Un modelo de casos de uso define la funcionalidad de todo el sistema, así pues representa todos los escenarios de interacción posible entre el usuario y el sistema.

[RJB] advierte que si bien es cierto que los casos de uso guían el proceso no deben ser desarrollados de manera aislada, pues hay que considerar la arquitectura del sistema. *Los casos de uso guían la arquitectura del sistema y la arquitectura del sistema influye en la selección de los casos de uso*.

Esto sugiere que existe un proceso de **maduración** tanto de arquitectura como del modelo de casos de uso a medida que avanza el ciclo de desarrollo.

Centrado en la arquitectura

Al igual que en la construcción de edificios, el papel de la arquitectura de software se puede contemplar desde diversos puntos de vista, tales como estructura o servicios [RJB]. En una arquitectura software la visión incluye aspectos estáticos y dinámicos. Así pues una arquitectura surge de las necesidades y refleja los casos de uso, pero también se ve influida por otros muchos factores (plataforma, marcos de trabajo, interfaces gráficas, requisitos de rendimiento o fiabilidad, etc.).

[RJB] denota que la relación existente entre casos de uso y arquitectura debe ser la siguiente:

1. Los casos de uso deben encajar en la arquitectura cuando se llevan a cabo.
2. La arquitectura debe permitir el desarrollo de todos los casos de uso requeridos y permitir una evolución en paralelo.

De esta manera los casos de uso representan la función del sistema mientras que la arquitectura define la forma de dicho sistema.

Según [RJB] el arquitecto:

1. Realizará una versión inicial de la arquitectura, comenzando por la sección del sistema que no es específica de los casos de uso pero debe mantener en perspectiva de éstos antes de comenzar la creación del esquema arquitectónico.
2. Trabajará con un subconjunto de los casos de uso que represente las funciones clave del sistema. Estos casos de uso deben especificarse en detalle y se realizará en términos de subsistemas, clases y componentes.

3. A medida que los casos de uso se especifican y desarrollan se descubren nuevos aspectos de la arquitectura, lo que implica una maduración de los casos de uso.

Iterativo e incremental

Cuando se aborda el desarrollo de un producto software se asume que supondrá un gran esfuerzo en términos económicos y temporales. Así pues resulta práctico dividir el trabajo en partes más pequeñas que se denominan iteraciones. Cada iteración obtendrá como resultado un incremento y dicho incremento se traducirá en una pequeña mejora o avance en el desarrollo del producto. En [RJB] se remarca la necesidad de establecer iteraciones *controladas*, es decir, iteraciones que deben ser previamente seleccionadas y ejecutadas de forma planificada como si fuesen proyectos más pequeños.

Para llevar a cabo este control, el desarrollador basa la selección en dos factores [RJB]:

1. La iteración tratará un grupo de casos de uso que en conjunto ampliarán la utilidad del producto desarrollado.
2. La iteración tratará los riesgos más importantes.

Puesto que cada iteración se considera un mini-proyecto, tomarán como punto de partida los casos de uso y después desarrollará el trabajo según los flujos:

1. Análisis
2. Diseño
3. Implementación
4. Pruebas

[RJB] aclara que al final de cada iteración no se tiene por qué tener un incremento necesariamente aditivo puesto que en las primeras fases del ciclo de vida los desarrolladores pueden tener que realizar tareas de reemplazo de diseño (los más superficiales por los más sofisticados). No obstante en las fases posteriores los incrementos suelen ser aditivos.

Según [RJB] el desarrollador durante una iteración del PUD debe realizar las siguientes tareas:

1. Identificar y especificar los casos de uso más relevantes.
2. Crear un diseño siguiendo la arquitectura seleccionada.
3. Implementar el diseño mediante componentes.
4. Verificar que los componentes satisfacen los casos de uso.

Todo ello ciñéndose al orden lógico de las iteraciones para economizar recursos y lograr el objetivo. Es habitual que surjan problemas inesperados y sea necesario realizar adaptaciones a los nuevos problemas, pero siguiendo un plan los beneficios son significativos:

1. El coste del riesgo se reduce a solo un incremento del proceso. Si una iteración debe repetirse, la organización sólo perdería el esfuerzo empleado en la iteración en concreto.
2. Se reduce el riesgo de retrasos en la entrega del producto mediante la identificación de riesgos en las fases más tempranas.
3. Puesto que los desarrolladores trabajan de manera más eficiente se consigue incrementar el ritmo del esfuerzo de desarrollo.
4. El cambio de requisitos por parte de los usuarios no es acentuado, debido a que los requisitos se van refinando durante las iteraciones.

4.1.2 Ciclo de vida del Proceso Unificado de Desarrollo

Modelo del PUD

El PUD se repite a lo largo de una serie de ciclos que constituyen la vida de un sistema y cada ciclo concluye con una versión del producto [RJB].

Cada ciclo está constituido por cuatro fases y cada fase a su vez se divide en iteraciones como ilustra la figura 4.2.

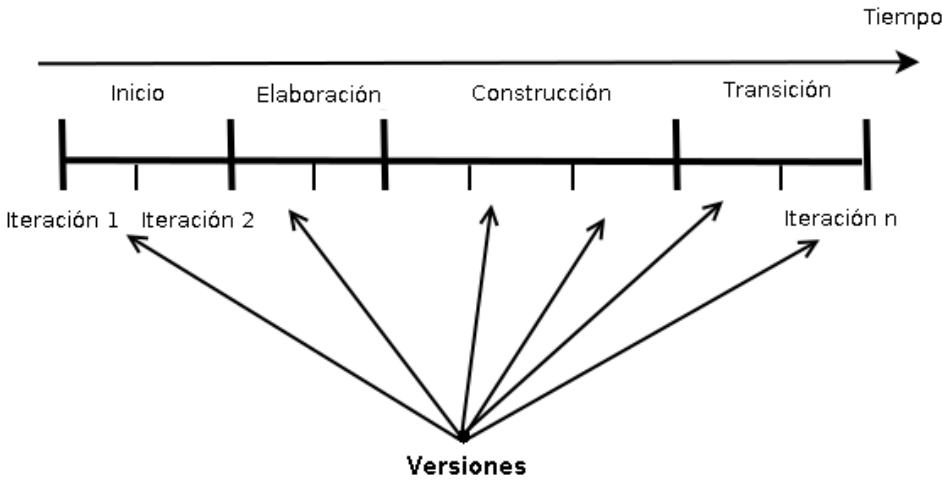

Figura 4.2: Ciclo con fases e iteraciones [RJB].

En cada ciclo se produce una nueva versión del sistema, siendo una versión un producto preparado para su entrega. El entregable a su vez consta de un cuerpo de código fuente además de manuales y productos asociados a su funcionamiento. El producto terminado incluirá los requisitos, casos de uso, especificaciones no funcionales y casos de prueba, incluyendo el modelo de la arquitectura y el visual.

Para llevar a cabo los ciclos del PUD los desarrolladores necesitan todas las representaciones del producto software. Estas representaciones del producto se denominan **Modelo del Proceso Unificado** (véase figura 4.3).

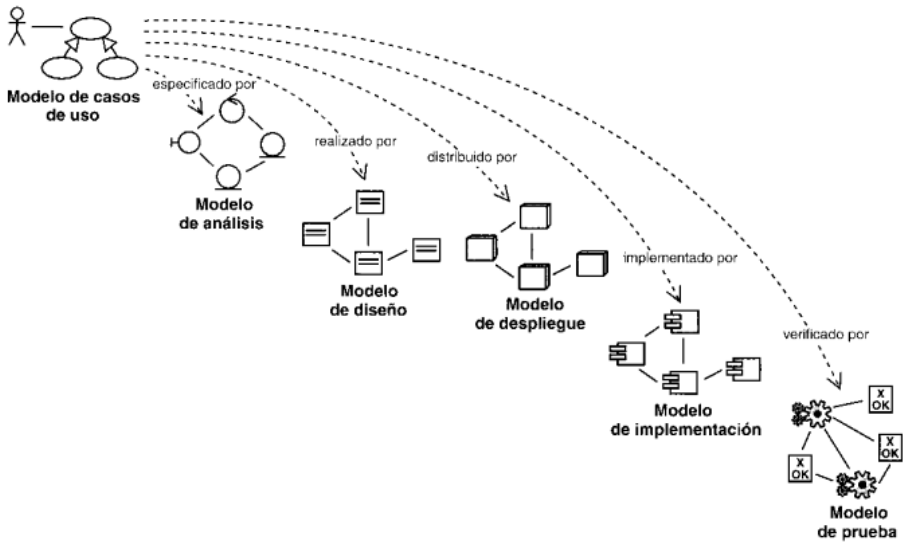

Figura 4.3: Modelo del PUD extraído de [RJB].

- **Modelo de casos de uso**: Todos los casos de uso del producto software.
- **Modelo de análisis**: Refina los casos de uso añadiendo más detalles y asigna funcionalidad a los objetos del producto software que proporcionan comportamiento.
- **Modelo de diseño**: Define la estructura estática del sistema (subsistemas, clases e interfaces) que representan los casos de uso.
- **Modelo de implementación**: Incluye los componentes, que representan al código fuente así como la correspondencia entre las clases y esos dichos componentes.
- **Modelo de despliegue**: Define los nodos físicos y su correspondencia con los componentes.
- **Modelo de prueba**: Especifica los casos de prueba que verifican los casos de uso.
- **Representación de la arquitectura**: Una representación de la arquitectura del sistema software desarrollado.
- El sistema debe tener un modelo de dominio o modelo del negocio. Su objetivo es describir el contexto del negocio en el que se encuentra el sistema.

Fases de un ciclo del Proceso Unificado de Desarrollo

[RJB] cita y explica las cuatro fases de cada ciclo del PUD:

1. Inicio
2. Elaboración
3. Construcción
4. Transición

A su vez, cada fase se puede descomponer a su vez en iteraciones (véase figura 4.4), acabando cada iteración en un hito. En la siguiente figura se representa el esfuerzo que se realiza en cada una de las fases.

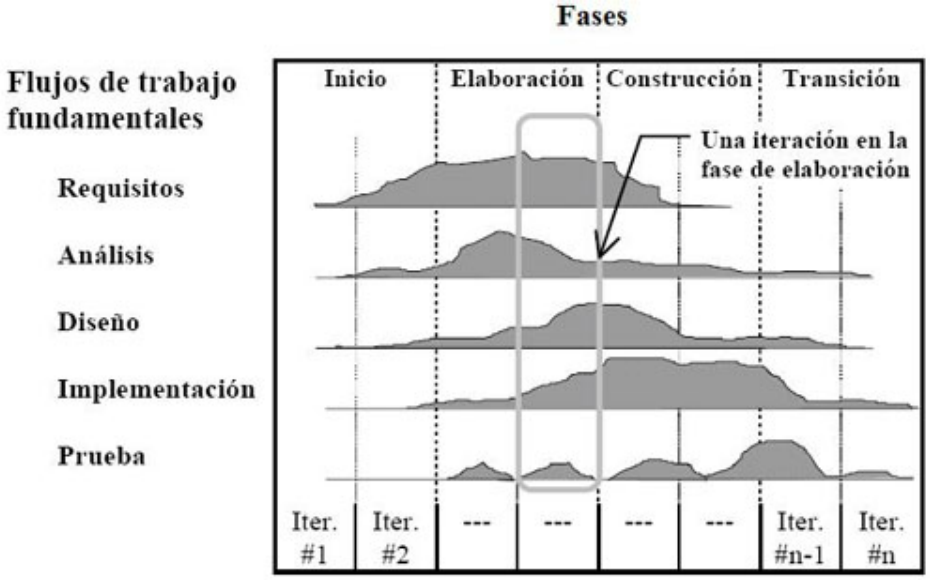

Figura 4.4: Diferentes fases del PUD [RJB].

Fase de Inicio: En la fase de inicio se realiza una descripción del producto final, es decir, se realiza el análisis de negocio para el producto esperado. En esta fase se responde a las siguientes cuestiones:

- Cuáles son las principales funciones del sistema para sus usuarios más importantes: lo que dará como resultado el modelo de casos de uso simplificado, conteniendo aquellos más críticos.
- Cómo podría ser la arquitectura del sistema: una vez se tiene el modelo, se comienza a esbozar la arquitectura con los subsistemas más importantes.
- Cuál es el plan de proyecto y cuánto costará desarrollar el producto: del paso anterior se planifican los riesgos y comienza a estimarse el coste del proyecto.

Fase de Elaboración: En esta fase se describe con más de talle los casos de uso y se diseña de manera definitiva la arquitectura y el sistema al completo. Finalmente, el director del proyecto debe ser capaz de planificar las actividades y estimar los recursos necesarios para terminar el proyecto.

Fase de Construcción: Fase de desarrollo del producto. El producto final contiene todos los casos de uso que se han acordado para el sistema. La línea base crecerá hasta convertirse en el sistema al completo. Se tiene una arquitectura estable pero aún susceptible de mejorar. Puede no estar completamente libre de defectos, que serán detectados y atajados en la fase de transición.

Fase de Transición: Corresponde al periodo en el cual el producto pasa a ser una versión *beta*. Se corrigen problemas y se realizan mejoras del sistema. Además esta fase contiene otras actividades complementarias, tales como formación del cliente, ayuda y asistencia, soporte técnico y corrección de defectos. Estos defectos se pueden dividir en dos categorías:

1. Los que tienen suficiente impacto para justificar un incremento.
2. Los que pueden corregirse en una versión usual.

4.2 Planificación del Proyecto

En esta sección se muestra la planificación que se llevará a cabo en la realización del TFM. En el Capítulo 5 se detallarán todos los productos resultantes de cada una de las iteraciones.

El sistema software a desarrollar se describe a continuación.

4.2.1 SparkDQ: Un stack de Spark para la evaluación de la calidad de datos de triplas semánticas

Este capa software tendrá por objetivo resolver el problema del tratamiento de datos semánticos en entornos Big Data. Se integrará e interactuará con el framework Apache Spark para ofrecer un conjunto de primitivas de lectura de datos semánticos y evaluación de la calidad de los mismos.

Este artefacto software tendrá las siguientes responsabilidades:

- Permitir la lectura de triplas semánticas y almacenarlas en una estructura de datos conveniente de forma distribuida.
- Evaluación de los niveles de calidad de datos de un conjunto LOD para la dimensión de calidad *Completeness* desde la perspectiva de dos de sus métricas, descritas en su sección correspondiente del capítulo 3 (3.4):
  - *Interlinking Completeness*
  - *Schema Completeness*

4.2.2 Prueba de Concepto para SparkDQ

Para materializar el stack elaborado en el punto anterior, se elaborará una aplicación software que aproveche las métricas implementadas en un caso de uso real, en el que se tendrá una perspectiva *end to end* de un problema de calidad de datos, persiguiendo los siguientes objetivos:

1. Ingesta de datos semánticos en un entorno Big Data.
2. Evaluación de la calidad de los datos ingestados de manera distribuida.
3. Almacenamiento de resultados de la evaluación.

4. Consumo de los resultados de la evaluación.

4.2.3 Requisitos funcionales para SparkDQ

Seguidamente se exponen los requisitos funcionales que se han identificado en la primera fase del PUD. Estos requisitos corresponden al desarrollo del stack **SparkDQ**.

1. **SparkDQ debe permitir la carga de triplas semánticas en un entorno distribuido**
   La premisa inicial es que el volumen de datos es suficientemente grande como para que se requiera un almacenamiento y procesamiento en distribuido de los mismos. Por lo tanto, SparkDQ debe ser capaz de leer las triplas almacenadas en entornos distribuidos y cargarlas en un modelo igualmente distribuido para su posterior procesamiento.
2. **SparkDQ debe permitir al usuario modelar la evaluación de calidad**
   Se entiende como evaluación de calidad el cálculo de unas métricas y su posterior interpretación, así pues SparkDQ debe exponer mecanismos de evaluación de calidad de datos para ser llevados a cabo sobre un juego de datos distribuido.
3. **SparkDQ debe permitir llevar a cabo evaluaciones de calidad de datos enlazados para la métrica *SchemaCompleteness***
   Se evaluará esta métrica considerando un esquema deseable para los datos obteniendo como resultado un grado de alineamiento de los datos con dicho esquema.
4. **SparkDQ debe permitir llevar a cabo evaluaciones de calidad de datos enlazados para la métrica *InterlinkingCompleteness***
   Se evaluará esta métrica considerando un nivel de profundidad, siendo esto una agregación del nivel de interconectividad estratificado.

4.2.4 Requisitos funcionales para la Proof of Concept (PoC)

Finalmente se exponen los requisitos funcionales identificados para la aplicación de Prueba de Concepto.

1. **PoC debe permitir la adquisición de datos semánticos**
   Como parte del *end to end* el primer paso de la prueba de concepto será seleccionar una fuente de datos y llevar a cabo un proceso de ingesta dentro de la arquitectura propuesta.
2. **PoC debe permitir el almacenamiento de datos semánticos en un entorno distribuido**
   Los datos deben ingestarse en un entorno distribuido para su posterior consumo, considerando que su volumen y variedad pueden cambiar a lo largo del tiempo.
3. **PoC debe permitir cargar datos semánticos desde un entorno distribuido y procesarlos**
   Los datos almacenados deben ser fácilmente accesibles y consumidos, entendiendo el

consumo de estos datos como la adquisición, procesamiento y obtención de resultados en base a éstos.

4. **PoC debe mantener unos niveles de seguridad adecuados**
   Los datos ingestados y almacenados deben estar protegidos de accesos no deseados.

5. **PoC debe poder almacenar el resultado de las evaluaciones de calidad de datos en un entorno distribuido**
   Los resultados de las evaluaciones deben ser convenientemente almacenados bajo las mismas directrices de disponibilidad y seguridad que los datos inicialmente ingestados.

6. **PoC debe permitir el consumo de los resultados de las evaluaciones llevadas a cabo**
   Los resultados de las evaluaciones deben ser accesibles y fácilmente consumibles por parte de los usuarios.

4.2.5 Iteraciones del PUD

La planificación del TFM se desarrollará mediante iteraciones que pertenecerán a diversas fases del Proceso Unificado de Desarrollo. En las iteraciones más tempranas tendrá lugar la concepción del proyecto en lo que se refiere a estudio, enlace con el PFC y diseño de la solución. En las iteraciones centrales el desarrollo y puesta a punto de los artefactos software generados para finalmente, en la fase de transición del PUD, la elaboración de la prueba de conecpto y la entrega del presente trabajo.

Fase del PUD	Iteraciones	Objetivos
Inicio	Iteracion 1	■ Estudiar el Estado del Arte. ■ Determinación del alcance del proyecto, requisitos y planificación. ■ Realización de diagrama de casos de uso.
Elaboracion	Iteración 2	■ Actualización técnica de JenaDQ. ■ Adaptar prueba de concepto.
	Iteración 3	■ Diseño de la arquitectura de SparkDQ: Dependencia con SparkRDF y Jena.
Construcción	Iteración 4	■ Análisis, Diseño, Implementación y Pruebas para SparkRDF.
	Iteración 5	■ Análisis, Diseño, Implementación y Pruebas para SparkDQ: *Interlinking*.
	Iteración 6	■ Análisis, Diseño, Implementación y Pruebas para SparkDQ: *SchemaCompleteness*.
Transición	Iteración 7	■ Análisis, Diseño, Implementación y Pruebas para PoC.
	Iteración 8	■ Entrega TFM

Tabla 4.1: Resumen de iteraciones para PUD

Iteración 1: Planificación, casos de uso y requisitos
Los objetivos de esta iteración son:

1. Contextualizar el trabajo llevado a cabo en el PFC con respecto a las tecnologías actuales y los avances en el campo.
2. Establecer una serie de casos de uso iniciales.
3. Fijar una serie de requisitos en base al conjunto de casos de uso especificado.
4. Establecer una arquitectura de la solución preliminar.

Planteando este TFM como una evolución del pasado PFC (véase [RCCMnR]), resulta imprescindible conocer en qué punto se halla todo el trabajo realizado tras varios años de evolución tecnológica. Es por lo tanto una de las iteraciones cruciales para definir el rumbo del desarrollo así como de actualización del estado de la cuestión.
Inicialmente se retoman las tecnologías utilizadas, Jena y JenaDQ, comprobando su evolución en los últimos años. Seguidamente se establece un marco contextual en el que se estudia el estado de la cuestión y se detectan carencias en el ámbito actual: ausencia de frameworks que permitan el procesamiento distribuido de triplas semánticas y el análisis de calidad de datos de las mismas en entornos Big Data.
Esto ayuda a especificar una serie de necesidades y unos casos de uso para cubrirlas. En base a los casos de uso se generan los requisitos dirigidos siempre hacia una arquitectura.

Fase del PUD	Inicio
Flujo de trabajo del PUD	Análisis y Diseño
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Contextualizar trabajo PFC ▪ CDU iniciales y requisitos ▪ Arquitectura preliminar	▪ Plan de actualización artefactos PFC ▪ CDU y RF ▪ Arquitectura preliminar

Tabla 4.2: Iteración 1

Iteración 2: Actualización de JenaDQ
Una vez establecidos los requisitos, se dispone a actualizar el estado de JenaDQ para dotarle de mayor nivel de madurez y alinearlo al trabajo que va a ser desarrollado, de manera que pueda compararse en ejecución con los desarrollos que van a tener lugar.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Actualización JenaDQ	▪ JenaDQ actualizado

Tabla 4.3: Iteración 2

Iteración 3: Diseño de arquitectura de SparkDQ
Tras analizar la problemática comienza el diseño de las soluciones. En este caso, el objetivo final es elaborar un stack que permita el procesamiento distribuido de triplas semánticas y la evaluación de la calidad de datos de las mismas de manera escalable. Para ello se requerirán de dos artefactos:

1. **SparkRDF**: como interfaz entre un conjunto suficientemente grande de triplas semánticas y una representación escalable que permita la transformación de éstas hacia modelos de procesamiento distribuido, en este caso y como se verá en el capítulo de resultados, mediante un grafo dirigido.

2. **SparkDQ**: Una vez se tenga la representación distribuida de las triplas, se procederá a la creación de primitivas que permitan la evaluación de la calidad de los datos para las métricas *SchemaCompleteness* e *Interlinking*.

El objetivo de esta iteración será diseñar los componentes necesarios para dar soporte al desarrollo de estos dos artefactos.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis y Diseño
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Diseñar nueva arquitectura de framework ▪ Diserñar arquitectura PoC	▪ Diseño de SparkRDF y SparkDQ ▪ Diseño de PoC preliminar

Tabla 4.4: Iteración 3

Iteración 4: Desarrollo de SparkRDF
Una vez diseñada la arquitectura, comienza el desarrollo con su primera dependencia: obtención de datos semánticos y representación en estructuras de datos distribuídas propias del framework de desarrollo.
Para ello se abordan las etapas tradicionales de desarrollo: Análisis, Diseño, Implementación y Pruebas. Se iterará hasta que se estime un estado óptimo del artefacto.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de SparkRDF ▪ Primitivas de lectura de triplas y exportar a modelo distribuido ▪ QA	▪ Artefacto SparkRDF

Tabla 4.5: Iteración 4

Iteración 5: Desarrollo de SparkDQ: Evaluación *Interlinking*
Teniendo el punto de entrada de los datos, lo siguiente es comenzar con la elaboración del artefacto de evaluación de calidad de datos mediante métricas.
La primera iteración sobre este artefacto tendrá por objetivo la elaboración de el conjunto de funciones necesarias para implementar la métrica *Interlinking*.
Se abordará mediante las etapas tradicionales: Análisis, Diseño, Implementación y Pruebas, iterando hasta que el resultado sea óptimo.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de SparkDQ ▪ Métrica Interlinking ▪ QA	▪ Artefacto SparkDQ preliminar con primera métrica

Tabla 4.6: Iteración 5

Iteración 6: Desarrollo de SparkDQ: Evaluación *SchemaCompleteness*
La segunda iteración sobre este artefacto tendrá por objetivo la elaboración de el conjunto de funciones necesarias para implementar la métrica *SchemaCompleteness*.
Se abordará mediante las etapas tradicionales: Análisis, Diseño, Implementación y Pruebas, iterando hasta que el resultado sea óptimo.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de SparkDQ ▪ Métrica SchemaCompleteness ▪ QA	▪ Artefacto SparkDQ preliminar con segunda métrica

Tabla 4.7: Iteración 6

Iteración 7: Desarrollo de la PoC
Finalmente tras haber finalizado el artefacto de evaluación de la calidad, se aplicará sobre una prueba de concepto. Esta prueba a su vez se dividirá en una serie de fases de Análisis, Diseño, Implementación y Pruebas, no relativas únicamente al artefacto software a desarrollar, sino a la arquitectura global que se ofrecerá finalmente.

Fase del PUD	Transición
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
■ Desarrollo de PoC ■ Implementación de arquitectura PoC ■ QA	■ Artefacto PoC ■ Arquitectura final

Tabla 4.8: Iteración 7

Iteración 8: Entrega de TFM
Una vez realizada la prueba de concepto, se finaliza el trabajo pendiente redactando y detallando la documentación:

1. Memoria del TFM

2. Documentación sobre los distintos artefactos generados:

   - SparkRDF
   - SparkDQ
   - spark-sem-dq-assessment
   - Microservicio de ingesta
   - Microservicio de transformación tweets a triplas semánticas
   - Ontologías desarrolladas

Fase del PUD	Transición
Flujo de trabajo del PUD	Documentación
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Elaboración de la documentación para los distintos artefactos (SparkRDF, SparkDQ) ▪ Elaboración de la documentación para la PoC ▪ Documentación TFM	▪ Documentación artefactos ▪ Documentación PoC ▪ Documentación TFM

Tabla 4.9: Iteración 8

4.3 Marco tecnológico de trabajo

En esta sección se describen las herramientas y tecnologías que se han elegido para dar soporte al desarrollo de este TFM. Dichas herramientas y tecnologías son utilizadas tanto para el modelado del sistema y desarrollo de diagramas como para la implementación del código y la documentación necesaria.

4.3.1 Frameworks de desarrollo

Los distintos frameworks utilizados durante la elaboración de este TFM se exponen a continuación.

Apache Jena

Jena (`http://jena.apache.org`) es un framework de código abierto, basado en Java para la construcción de aplicaciones basadas en tecnología Semántica y datos enlazados.

En el Capítulo 3 se han explicado algunas de las características más importantes de este framework.

Apache Spark

Spark (`http://spark.apache.org`) se ha convertido en el estándar *de facto* a la hora del procesamiento distribuido de alta velocidad.

En el Capítulo 3 se han explicado las características más interesantes de dicho framework así como la importancia que tiene en el actual ecosistema Big Data.

Docker

Docker (`https://www.docker.com/`) es una plataforma Open Source para virtualizar y encapsular servicios para su despliegue y gestión de una manera más rápida y eficiente.

4.3.2 Software de Desarrollo

Como software de soporte al desarrollo del TFM se han utilizado las siguientes herramientas.

IntelliJ

IntelliJ (`https://www.jetbrains.com/idea/`) es un Integrated Development Environment (IDE) que ha ganado en popularidad durante los últimos años debido a su extensibilidad y facilidad de uso. Puede enriquecerse con *plugins* que dan soporte a tecnologías actuales y facilitan la experiencia de desarrollo.

Maven

Maven (`https://maven.apache.org/`) es un proyecto apache cuya finalidad es la de servir de gestor de proyectos en diversos ámbitos. Con Maven es fácil gestionar las distintas dependencias entre artefactos, ejecutar código, obtener informes de resultados de pruebas y empaquetar los resultados.

Visual Paradigm

Visual Paradigm (`http://www.visual-paradigm.com`) es una herramienta Computer Aided Software Engineering (CASE) que permite facilitar todo el proceso de desarrollo, incluyendo herramientas de edición de diagramas UML de todo tipo y generación de código automática a partir de diagramas y viceversa.

Visual Paradigm se ha utilizado para la realización de todos los diagramas.

Git Git (`http://git-scm.com/`) ha sido el software de control de versiones utilizado para el desarrollo.

jUnit

jUnit (`http://junit.sourceforge.net`) es un framework que permite realizar ejecución de pruebas unitarias de manera controlada para evaluar el correcto funcionamiento de determinado código.

Protégé

Protégé (`http://protege.stanford.edu`) es un editor de ontologías de código abierto. Se ha utilizado en este proyecto para la creación del vocabulario de evaluación de calidad de datos.

4.3.3 Edición

Para las diferentes labores de edición, tanto de texto como de imagen, se han utilizado las siguientes herramientas.

LaTeX

LaTeXes un sistema de composición de textos orientado principalmente a la creación de documentos científicos y técnicos.

Toda la documentación se ha generado con esta tecnología.

BibTeX

BibTeX es un sistema gestor de referencias bibliográficas que se integra con LaTeX.

Emacs

Emacs (`www.gnu.org/software/emacs/`) es un editor de texto multifuncional que incluye una gran cantidad de librerías y extensiones para facilitar cualquier tipo de edición que se lleve a cabo sobre él.

En este proyecto, se ha utilizado Emacs integrado con LaTeXpara la elaboración de toda la documentación. También para la edición de otros archivos menores como reglas o scripts.

Dia Dia (`https://wiki.gnome.org/Apps/Dia`) es un programa editor de diagramas de diferentes tipos (UML, flujo de datos, ...).

Gimp

Gimp (`www.gimp.org`) es un editor de imagen de código abierto.

4.3.4 Servidores

Se han utilizado dos tipos de servidores: los de la aplicación Web y los de datos. Se enumeran a continuación.

Apache Tomcat

Apache Tomcat (`http://tomcat.apache.org`) es un servidor Web de código abierto para aplicaciones Java.

Jena TDB

Dentro del proyecto Jena, TDB es sistema de almacenamiento altamente escalable y de alta velocidad empleado para las triplas.

Jena Fuseki

Dentro del proyecto Jena, Fuseki es el servidor de triplas sobre el que se pueden realizar consultas como endpoint a través del protocolo HTTP. Se integra sobre distintos servidores pero principalmente sobre TDB.

4.3.5 Cloud computing

Se han utilizado diversas soluciones *cloud* para llevar a cabo el desarrollo de la prueba de concepto. En este caso, el proveedor elegido ha sido Amazon Web Services, utilizando los siguientes servicios:

- **Amazon S3**: *Simple Storage Service* o S3 es un servicio de almacenamiento distribuido y escalable basado en *buckets* como directorios.
- **Amazon Elastic Search**: Solución de Amazon para un servicio autogestionado del stack ElasticSearch y Kibana. ElasticSearch es un indexador de documentos basado en índices, mientras que Kibana, montado sobre éste, permite visualizar los documentos indexados a través de agregaciones sencillas.

4.3.6 Lenguajes de Programación

A continuación se indican aquellos lenguajes de programación que han tenido más peso en el desarrollo del TFM.

Java Java (`https://www.java.com/`) es el lenguaje de programación para el desarrollo de la parte central del proyecto. Se ha escogido Java debido a su gran extensión y soporte por parte de la comunidad.

Para este TFM se ha trabajado con la versión 1.8.

Scala

Scala(`https://www.scala-lang.org/`) es un lenguaje de programación funcional, que está convirtiéndose en una auténtica revolución en el mundo del Big Data debido a que corre sobre la máquina virtual de Java, permite reutilizar todas las librerías disponibles para éste y su orientación funcional lo hace idóneo para la resolución de problemas y elaboración de algoritmos en estos ámbitos. Es el lenguaje con el que está desarrollado Spark y ha sido el lenguaje de programación con el que se ha desarrollado el grueso del proyecto. La versión utilizada para el desarrollo de este TFM es la 2.11.

Datalog Jena

Para la parte relacionada con la inferencia, Jena posee su propio lenguaje de definición de reglas: Datalog Jena. Es el lenguaje utilizado para definir las reglas de contexto que luego se han utilizado para llevar a cabo los razonamientos (véase Sección 3.1.7).

SPARQL

SPARQL es el lenguaje declarativo para llevar a cabo operaciones de consulta y actualización sobre conjuntos de datos semánticos.

4.3.7 Equipos de desarrollo

Para la realización del TFM se han utilizado las siguientes máquinas:

- PC con *GNU/Linux Ubuntu 16.04* procesador Intel i7 de ocho núcleos de 2.6 GHz y 16 GB de RAM.
- PC con *GNU/Linux Ubuntu 16.04* procesador Intel i5 de cuatro núcleos de 2.7 GHz y 16 GB de RAM.

Capítulo 5

Resultados

En este capítulo se presentan los resultados obtenidos tras llevar a cabo el plan de trabajo presentado en el capítulo anterior. El proceso ha desembocado en la obtención de varios artefactos, principalmente SparkDQ y su prueba de concepto, tal y como se pretendía desarrollar.

5.1 Fase de Inicio

Durante la fase inicial, se destina una iteración a planificar el desarrollo del TFM, establecer requisitos y en función de éstos extraer una serie de casos de uso.
Las tablas expuestas en el capítulo anterior se van a exponer de nuevo con el fin de facilitar la lectura.

5.1.1 Iteración 1: Planificación, casos de uso y requisitos

La primera iteración tiene como objetivo establecer una planficación, detallar requisitos y establecer una serie de casos de uso.
Se han obtenido como resultados los siguientes artefactos:

- Una revisión del estado del arte que puede consultarse en el Capítulo 3, reflejando la situación actual en los principales campos relacionados con el TFM
    - Web Semántica
    - Procesamiento en distribuido
    - Calidad de Datos
    - Calidad de Linked Data
- Una especificación de requisitos, funcionales y no funcionales, tanto para SparkDQ como para la prueba de concepto a realizar a partir de ésta.
- Modelo general de Casos de Uso de SparkDQ y PoC
- Planificación detallada dividida en fases e iteraciones del PUD

Fase del PUD	Inicio
Flujo de trabajo del PUD	Análisis y Diseño
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
■ Contextualizar trabajo PFC ■ CDU iniciales y requisitos ■ Arquitectura preliminar	■ Plan de actualización artefactos PFC ■ CDU y RF ■ Arquitectura preliminar

Tabla 5.1: Iteración 1

Requisitos

Los requisitos funcionales de cualquier sistema software son aquellos que definen el comportamiento del sistema y establecen un punto de partida para elaborar un modelo de casos de uso (ver figura 5.1), mientras que los no funcionales son aquellos orientados a aspectos del sistema que notienen relación con el comportamiento, por ejemplo, el diseño, la implementación on la usabilidad.

Seguidamente se exponen los requisitos funcionales que se han identificado en la primera fase del PUD. Estos requisitos corresponden al desarrollo del stack **SparkDQ**.

1. **RF 1. SparkDQ debe permitir la carga de triplas semánticas en un entorno distribuido**
   La premisa inicial es que el volumen de datos es suficientemente grande como para que se requiera un almacenamiento y procesamiento en distribuido de los mismos. Por lo tanto, SparkDQ debe ser capaz de leer las triplas almacenadas en entornos distribuidos y cargarlas en un modelo igualmente distribuido para su posterior procesamiento.
2. **RF 2. SparkDQ debe permitir al usuario modelar la evaluación de calidad**
   Se entiende como evaluación de calidad el cálculo de unas métricas y su posterior interpretación, así pues SparkDQ debe exponer mecanismos de evaluación de calidad de datos para ser llevados a cabo sobre un juego de datos distribuido.
3. **RF 3. SparkDQ debe permitir llevar a cabo evaluaciones de calidad de datos enlazados para la métrica *SchemaCompleteness***
   Se evaluará esta métrica considerando un esquema deseable para los datos obteniendo como resultado un grado de alineamiento de los datos con dicho esquema.
4. **RF 4. SparkDQ debe permitir llevar a cabo evaluaciones de calidad de datos enlazados para la métrica *InterlinkingCompleteness***
   Se evaluará esta métrica considerando un nivel de profundidad, siendo esto una agregación del nivel de interconectividad estratificado.

Una vez definidos los funcionales, se indican los no funcionales para SparkDQ.

- **RNF 1. Utilización de un framework para desarrollo de tecnologías semánticas: Apache Jena**
  Se retomará el uso de Apache Jena como framework de desarrollo de tecnologías semánticas para utilizar en el contexto del TFM
- **RNF 2. Utilización de un framework de procesamiento en distribuido: Apache Spark**
  Para el procesamiento distribuido se utilizará Apache Spark, que queda explicado en la sección 3.5.3
- **RNF 3. Generación de la documentación de la API**
  Finalmente se generará la documentación de la API que permita a los desarrolladores extender y utilizar el módulo creado.

Finalmente se exponen los requisitos funcionales y no funcionales identificados para la aplicación de Prueba de Concepto.

1. **PoC debe permitir la adquisición de datos semánticos**
   Como parte del *end to end* el primer paso de la prueba de concepto será seleccionar una fuente de datos y llevar a cabo un proceso de ingesta dentro de la arquitectura propuesta.
2. **PoC debe permitir el almacenamiento de datos semánticos en un entorno distribuido**
   Los datos deben ingestarse en un entorno distribuido para su posterior consumo, considerando que su volumen y variedad pueden cambiar a lo largo del tiempo.
3. **PoC debe permitir cargar datos semánticos desde un entorno distribuido y procesarlos**
   Los datos almacenados deben ser fácilmente accesibles y consumidos, entendiendo el consumo de estos datos como la adquisición, procesamiento y obtención de resultados en base a éstos.
4. **PoC debe mantener unos niveles de seguridad adecuados**
   Los datos ingestados y almacenados deben estar protegidos de accesos no deseados.
5. **PoC debe poder almacenar el resultado de las evaluaciones de calidad de datos en un entorno distribuido**
   Los resultados de las evaluaciones deben ser convenientemente almacenados bajo las mismas directrices de disponibilidad y seguridad que los datos inicialmente ingestados.
6. **PoC debe permitir el consumo de los resultados de las evaluaciones llevadas a cabo**
   Los resultados de las evaluaciones deben ser accesibles y fácilmente consumibles por parte de los usuarios.

Siendo los requisitos no funcionales los siguientes:

- **RNF 1. Arquitectura en Cloud**
  Se debe realizar una prueba de concepto utilizando tecnologías Cloud recientes y de uso extendido.
- **RNF 2. Utilización de parte del ecosistema Apache**
  Siendo SparkDQ un incremento sobre un framework del ecosistema Hadoop, se buscará integrarse también con otras tecnologías del mismo ecosistema.

Modelo general de Casos de Uso
Uno de los primeros pasos consiste en la elaboración del modelo de Casos de Uso para el proyecto.
Se han identificado dos sistemas, SparkRDF y SparkDQ, con dependencia de éste último sobre el primero, conteniendo ambos los casos que se ilustran y se detallarán en las iteraciones pertinentes (secciones 5.3.1 y 5.3.2).
El modelo general se puede consultar en la figura 5.1.

5.2 Fase de Elaboración

En la fase de Elaboración, se han sentado las bases para el desarrollo propiamente dicho de los artefactos software que se han indicado en la fase inicial. Concretamente se ha trabajado en dos vertientes:

- Actualización de JenaDQ como artefacto software
- Diseño del stack SparkDQ

5.2.1 Iteración 2: Actualización de JenaDQ

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Actualización JenaDQ	▪ JenaDQ actualizado

Tabla 5.2: Iteración 2

En esta iteración el objetivo principal es actualizar JenaDQ y alinearlo con las tecnologías que van a ser usadas en este proyecto.
Concretamente, se ha generado un proyecto Maven (véase Sección4.3.2) con el que gestionar más fácilmente las dependencias y el ciclo de vida de las aplicación.
Posteriormente se ha hecho una revisión del código y se han actualizado aquellos puntos de

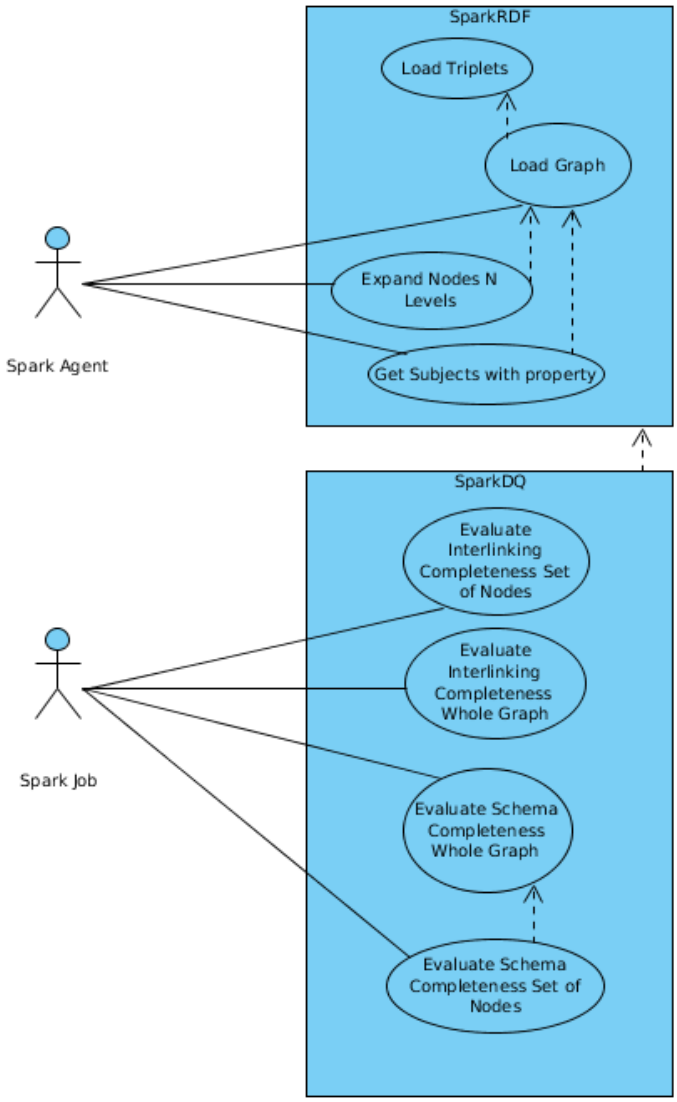

Figura 5.1: Diagrama general de Casos de Uso

JenaDQ que se han creído oportunos:

1. Se han añadido más pruebas unitarias.

2. Refactorizado de código.

3. Test de integración.

Como resultado se ha obtenido una nueva versión de JenaDQ, que ha mantenido las métricas pero cuyo nivel de calidad como artefacto software y mantenibilidad se ha visto incrementado.

5.2.2 Iteración 3: Diseño de la arquitectura de SparkDQ

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis y Diseño
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Diseñar nueva arquitectura de framework ▪ Diserñar arquitectura PoC	▪ Diseño de SparkRDF y SparkDQ ▪ Diseño de PoC preliminar

Tabla 5.3: Iteración 3

La iteración ha tenido como objetivo:

- Establecer un diseño de la arquitectura de SparkDQ
- Establecer un diseño de la arquitectura de la prueba de concepto

En primer lugar se ha concluido que SparkDQ contará con dos elementos:

- **SparkRDF**: un artefacto encargado únicamente de la captura de datos semánticos y su transformación a una estructura de datos distribuida. En este caso, dicha estructura será la de un grafo dirigido. Este grafo se construirá sobre la librería Spark GraphX, orientada específicamente a trabajar con este tipo de estructuras.
- **SparkDQ**: artefacto encargado de llevar a cabo evaluaciones de calidad de datos sobre datos semánticos. Tendrá una dependencia con SparkRDF, delegando en éste la adquisición de datos y tomando en sus funciones como puntos de entrada los datos extraídos por SparkRDF. Se plantean a su vez dos métricas para evaluar: *Interlinking* y *SchemaCompleteness*, descritas en la sección 3.4.

Para la prueba de concepto, se ha establecido:

1. Generación de un artefacto que lleve a cabo cálculos de calidad de datos semánticos sobre un conjunto suficientemente grande. Dicho artefacto se llamará spark-sem-dq-assessment y tendrá una dependencia de SparkDQ. Podrá configurarse para llevar a cabo una evaluación de la calidad de los datos sobre cualquiera de las métricas que ofrezca SparkDQ.
2. Elaboración de un diseño de arquitectura en cloud: involucrará los siguientes elementos:

   a) Arquitectura de microservicios: para llevar a cabo la captura y procesamiento de datos.

b) Almacenamiento en cloud - Amazon S3: como repositorio de datos iniciales y directorio final de resultados de cálculos.

c) Visualización - Amazon ElasticSearc and Kibana stack: para la visualización de los resultados de las evaluaciones.

Se ha obtenido como resultado el diagrama de clases de SparkRDF, diagrama de clases de SparkDQ (figura 5.2) y diagrama de secuencia de la prueba de concepto (figura 5.3).

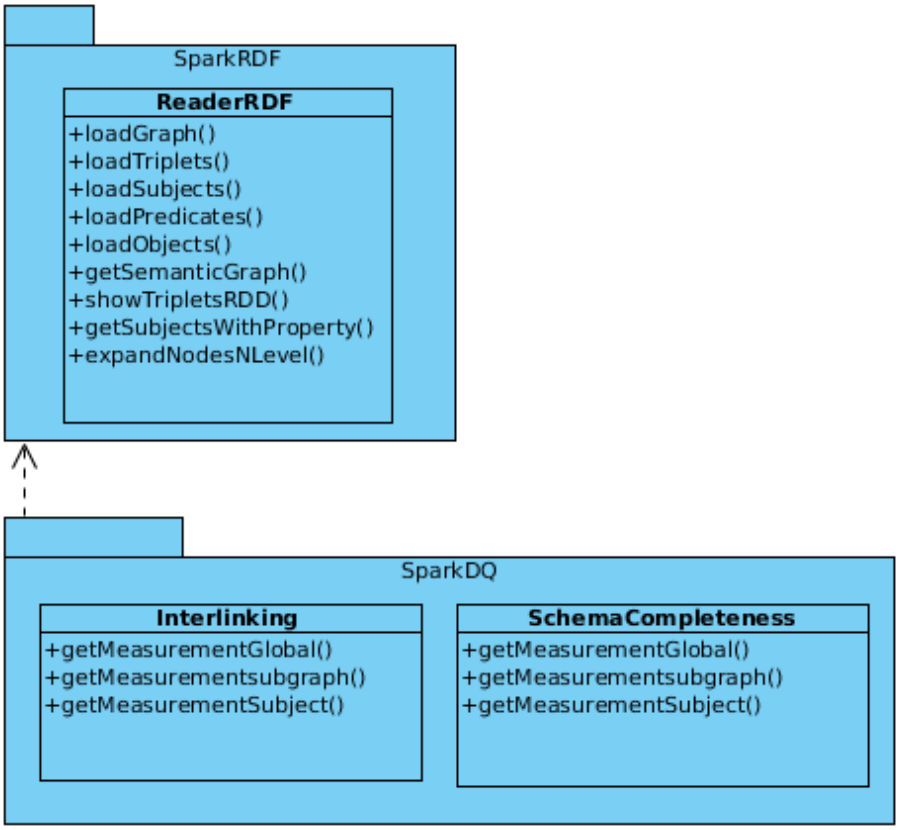

Figura 5.2: Diagrama de clases SparkDQ

5.3 Fase de Construcción

Una vez asentados los diseños de los artefactos a desarrollar, en la fase de construcción tendrá lugar el desarrollo del núcleo de la aplicación, esto es, los artefactos clave de desarrollo: SparkRDF y SparkDQ, en las iteraciones 4 y 5 respectivamente. Se considera el ciclo completo de desarrollo en estos puntos: Análisis, Diseño, Implementación y Pruebas. Dichas pruebas tendrán lugar en la fase final como parte de un paso de Quality Assurance (QA), y serán indicativas sobre la repetición del ciclo en caso de que los resultados no sean los esperados.

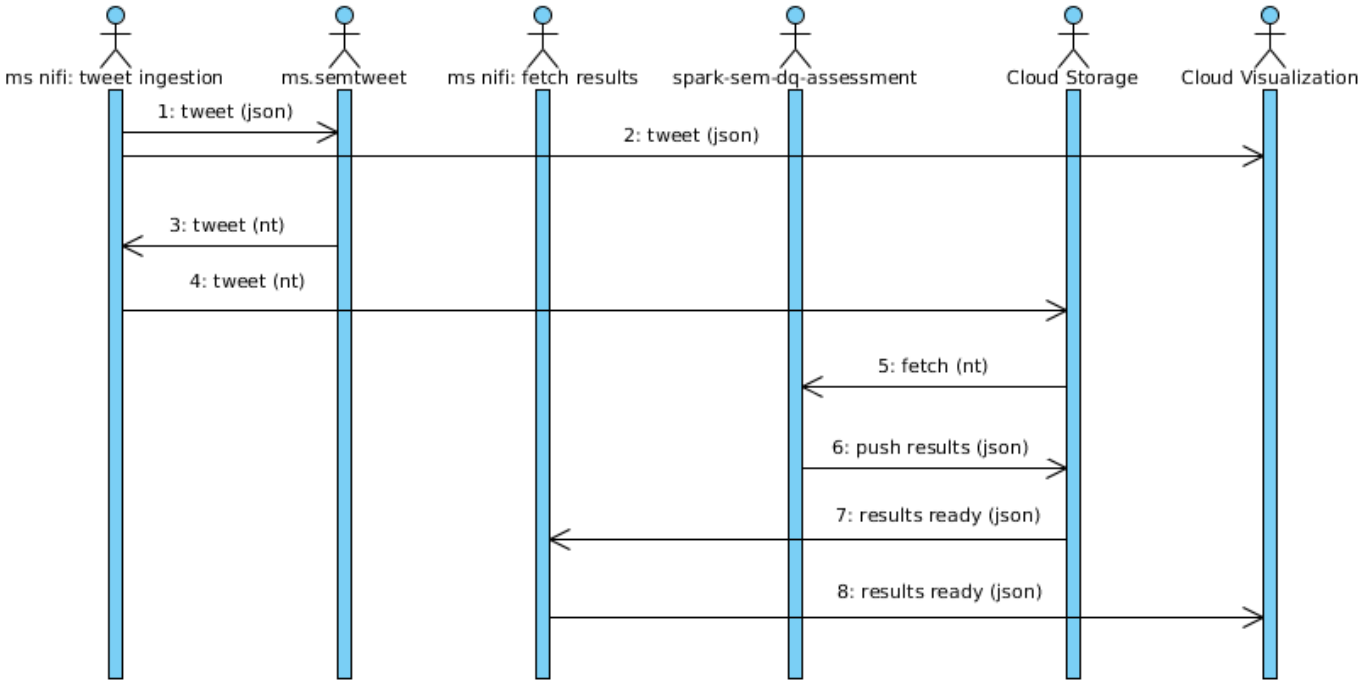

Figura 5.3: Diagrama de secuencia para la PoC

5.3.1 Iteración 4: Desarrollo de SparkRDF

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de SparkRDF ▪ Primitivas de lectura de triplas y exportar a modelo distribuido ▪ QA	▪ Artefacto SparkRDF

Tabla 5.4: Iteración 4

SparkRDF utiliza un sub-módulo de Apache Jena (riot) para la lectura de documentos en formato `TTL` o `NT`, es decir, en forma de triplas, en las que cada línea del fichero tiene tres campos separados por espacio, siendo dichos campos sujeto, predicado y objeto, respectivamente. El listado 5.1 muestra un fragmento de código encargado de la lectura de triplas y

almacenamiento en un RDD.

```
def loadTriplets(sparkSession: SparkSession, path: String): RDD[Triple] = {
  val tripletsRDD = sparkSession.sparkContext.textFile(path)
  .filter(line => !line.trim().isEmpty & !line.startsWith(‘‘#’’))
  .map(line =>
          RDFDataMgr.createIteratorTriples(new
          ByteArrayInputStream(line.getBytes), Lang.NTRIPLES, null).next())
          tripletsRDD
}
```

Listado 5.1: Fragmento del lector de triplas

Una vez cargadas las triplas y puesto que la información semántica es inherentemente un grafo, se utiliza la extensión GraphX de Spark para generar un grafo dirigido de triplas semánticas, en el que los nodos son sujetos y objetos y los arcos son las relaciones entre los nodos. El listado 5.2 ilustra esta funcionalidad.

```
private def getSemanticGraph(tripleRDD: RDD[Triple]): org.apache.spark.graphx.Graph[Node, Node] = {
//Generate hashCodes for graphx representation
    val extTripleRDD = tripleRDD.map(triple => (triple,
        triple.getSubject().hashCode().toLong,
        triple.getObject().hashCode().toLong))
    val subjects: RDD[(VertexId, Node)] = tripleRDD.map(triple =>
        (triple.getSubject().hashCode().toLong, triple.getSubject()))
    val predicates: RDD[Edge[Node]] = extTripleRDD.map(line =>
        Edge(line._2, line._3, line._1.getPredicate()))
    val objects: RDD[(VertexId, Node)] = tripleRDD.map(triple =>
        (triple.getObject().hashCode().toLong, triple.getObject()))
    val graph = org.apache.spark.graphx.Graph(subjects.union(objects).distinct(),
         predicates.distinct())
    graph
}
```

Listado 5.2: Conversión de RDD de Triplas a Grafo de nodos

El tratamiento de los datos RDF y por ende las responsabilidades de este artefacto no terminan solamente con la lectura y transformación a una estructura de grafo distribuida, sino que cualquier operación de tratamiento y consulta de estos datos debe delegar en esta capa de abstracción. Así pues, además de las nombradas, existe una serie de funciones dedicadas a llevar a cabo operaciones auxiliares sobre un conjunto de triplas, tales como listar sujetos, predicados, objetos o funciones de consulta como las de buscar nodos que cumplan cierta propiedad.

Como se verá en la siguiente sección, existe una función clave para el desarrollo de este proyecto y el cálculo de las métricas, que queda delegada a SparkRDF:

```
def expandNodesNLevel(nodes: VertexRDD[Node],
                      graph: org.apache.spark.graphx.Graph[Node, Node],
                      levels: Int = 1): Dataset[Row]
```

Listado 5.3: Conversión de RDD de Triplas a Grafo de nodos

Su funcionalidad es la de, dada una colección de nodos, expandir en N capas esos nodos. En la siguiente sección se explicará en detalle la necesidad de esta funcionalidad.

5.3.2 Iteración 5: Desarrollo de SparkDQ para la métrica *Interlinking*

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de SparkDQ ▪ Métrica Interlinking ▪ QA	▪ Artefacto SparkDQ preliminar con primera métrica

Tabla 5.5: Iteración 5

Se define la métrica *Interlinking* como el índice en el que las instancias en el conjunto de datos están interconectadas, entendiendo interconexión como el *grado* para un determinado nodo del grafo. No obstante, se particulariza en el ámbito de los datos enlazados, como el índice de conexiones que tiene dicho nodo hacia nodos que no sean hoja, es decir, sigan siendo nodos desreferenciables y así tengan un grado mayor a cero. Así, para un nodo, se pueden establecer métricas de interconexión por niveles:

- Nivel 0: los vecinos directos del nodo: V1
- Nivel 1: Para cada nodo perteneciente a V1, el conjunto unión de todos sus vecinos: V2
- etc.

De esta manera se obtiene una métrica estratificada por niveles. Así, para cada nivel, se tendría el número de nodos que son URIs respecto del total de vecinos directos.
Para abordar esta métrica se distinguen dos fases:

1. Para todos los nodos, es decir, sujetos semánticos candidatos a calcular la métrica, expandir por niveles para conocer los nodos con los que está relacionado tal y como se ha explicado en el punto anterior. Se ilustra el algoritmo en el listado 5.4.
2. Formar una tabla con las siguientes entradas (ver tabla XX):

- Id del nodo sujeto (`Nodo`)
- Profundidad N (`Int`)
- Nodo vecino en profundidad N (`Nodo`)
- Nodo vecino en profundidad N es URI (`Boolean`)

3. Sobre esta estructura, agrupar por Id del nodo/profundidad contando el número de nodos que son URI y dividiendo entre el total por el nivel. De esta manera se obtiene un número entre 0 y 1 correspondiente a la métrica para ese sujeto. Véase listado 5.5.

```
  def expandNodesNLevel(nodes: VertexRDD[Node],
                        graph: org.apache.spark.graphx.Graph[Node, Node],
                        levels: Int = 1): Dataset[Row] = {
    import processSparkSession.implicits._

    val edges = graph.edges.map(l => (l.srcId, l.dstId)).toDF(Seq(''srcId'',
    ''dstId''): _*).cache()
    var edgesR = graph.edges.map(l => (l.srcId, l.dstId,
    0)).toDF(Seq(''source'', ''level'', ''depth''): _*).cache()
    val nodesR = nodes.map(l => l._1).toDF(Seq(''nodeId''): _*)

    var results = edgesR.distinct()

    for (level <- 1 until levels) {
      val res = edges.join(edgesR.drop(''depth''), $''dstId'' === $''source'',
      ''leftouter'').orderBy($''srcId'')
      edgesR = res.select($''srcId'' as ''source'', $''level'' as
      ''level'').withColumn(''depth'', lit(level))
      results = results.union(edgesR.distinct())
    }
    results = results.join(nodesR, $''source'' ===
    $''nodeId'').drop($''nodeId'').na.drop().distinct().orderBy($''depth'',
    $''source'')
    results
  }
}
```

Listado 5.4: Expansión de nodos en N niveles para un conjunto de sujetos

```
  def getMeasurementSubgraph(subjects: VertexRDD[Node], graph: Graph[Node,
    Node], depth: Int ): Dataset[Row] = {
    val expanded = expandNodesNLevel(subjects, graph, depth)
    import processSparkSession.implicits._
    val subs = subjects
      .filter(ll => ll._2.isURI())
      .map(l => (l._1, l._2.getURI())).toDF(Seq(''vertexId'', ''vertexURI''):
    _*)
```

```
    val filteredNodes = graph.vertices.map(l => (l._1,
    l._2.isURI())).toDF(Seq(''nodeId'', ''isURI''): _*)
    val nodesTF = expanded.join(filteredNodes, $''level'' ===
    $''nodeId'').drop($''nodeId'').drop($''level'').orderBy($''source'',
    $''depth'')
    val partResultTrue = nodesTF.groupBy($''source'',
    $''depth'').agg(count(when($''isURI'' === true, true)) as
    ''countT'').orderBy($''source'', $''depth'')
    val partResultFalse = nodesTF.groupBy($''source'',
    $''depth'').agg(count(when($''isURI'' === false, true)) as
    ''countF'').orderBy($''source'', $''depth'')
      .toDF(Seq(''sourceF'', ''depthF'', ''countF''): _*)
    val result = partResultTrue.join(partResultFalse, $''source'' ===
    $''sourceF'' and $''depth'' ===
    $''depthF'').drop($''sourceF'').drop($''depthF'').orderBy($''source'',
    $''depth'')
      .withColumn(''measurement'', getRatio($''countT'',
    $''countF'')).join(subs, $''source'' === $''vertexId'').drop($''vertexId'')
    result
  }
```

Listado 5.5: Cálculo de la métrica en colección expandida de nodos

Finalmente se acaba la iteración con las pruebas correspondientes a la funcionalidad desarrollada.

Ejemplo
[EJEMPLO GRÁFICO AQUÍ]

5.3.3 Iteración 6: Desarrollo de SparkDQ para la métrica *SchemaCompleteness*

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de SparkDQ ▪ Métrica SchemaCompleteness ▪ QA	▪ Artefacto SparkDQ preliminar con segunda métrica

Tabla 5.6: Iteración 6

Se define la métrica *SchemaCompleteness* como el grado en el que las clases y propiedades de una ontología están representadas.
Por lo tanto, dado un conjunto de propiedades o relaciones entre sujetos y objetos, la métrica se computa como el ratio de relaciones que un determinado sujeto tiene con otras entidades.
En este particular no tiene sentido especificar un grado de esta métrica estratificado por niveles del grafo puesto que cada nodo vecino del inicial ya será una entidad en sí mismo, y la métrica aplicada para ese nodo en ese nivel corresponde únicamente a ese sujeto. Se puede enunciar formalmente así:
Sea G un grafo formado por un conjunto de nodos N y un conjunto de arcos A. Sea PD un conjunto de propiedades deseables. El valor de la métrica vendrá dado por el número de propiedades de PD que se encuentran reflejados en A, dividido entre el número total de propiedades deseables.
El cálculo de esta métrica requiere de una serie de pasos que se van a ilustrar en los siguientes listados.

- En primer lugar, se define una función que calcula el valor de la métrica en sí mismo, esto es, el ratio entre el total de relaciones para un nodo que están incluídas en la lista de propiedades. Esta función se muestra en el listado 5.6. Cabe destacar la secuencia de control de la métrica para valores mayores que 1. Pueden darse en el caso de que un nodo esté conectado con otros a través de la misma propiedad más de una vez, por ello es importante considerar que para la métrica tiene efecto la aparición una única vez de la propiedad.
- Seguidamente, se generaliza el cálculo de la métrica para todo el grafo, ilustrado en el listado 5.7. Para cada propiedad deseable, se obtiene un conjunto de arcos que representan dicha propiedad. La unión de estos conjuntos es el conjunto de arcos de todo el grafo, que cumple con todas las propiedades deseables.
- El conjunto total de arcos es combinado con el conjunto de arcos que cumplen las propiedades. Agregando los resultados y contando el total de arcos y de propiedades.
- Aplicando la función inicial para cada nodo del grafo, se obtiene la métrica global para todos los nodos.
- En el paso final, se filtran los nodos del grafo para los nodos candidatos que quieran evaluarse, como se muestra en el listado 5.8.

```
def getRatio = udf((totalTrues: Int, total: Int) => {
  var res = totalTrues.toDouble/total.toDouble
  if (res > 1.0)
    res = 1.0
  res
})
```

Listado 5.6: Cálculo del ratio

//TODO IN CODE: filtro en propIdsRDD para hacer distinct de propiedades.

```
def getMeasurementGlobal(graph: Graph[Node, Node], properties: Seq[String]):
Dataset[Row] = {
  import processSparkSession.implicits._
  val edgeRDD = graph.edges.filter(l => true)

  val propIdsRDD = properties.map(p => edgeRDD.filter(l =>
  l.attr.hasURI(p))).reduce(_ union _)
  //TODO toDF().distinct by srcId, attr.URI(bla)
    .map(l => l.srcId).cache()



  val nonPropsDF = edgeRDD.map(l => l.srcId).toDF(Seq(''source''): _*)
  nonPropsDF.join(propIdsRDD
    .toDF(Seq(''sourceProp''): _*)
    .groupBy($''sourceProp'').agg(count($''sourceProp'') as ''propCount'')
    .withColumn(''totalProperties'', lit(properties.length))
    .withColumn(''meas'', getRatio($''propCount'', $''totalProperties''))
    .drop($''propCount'')
    .drop($''totalProperties'')
    .drop($''srcId''), $''source'' === $''sourceProp'', ''leftouter'')
    .drop($''sourceProp'')
      .withColumn(''measurement'', when(
        col(''meas'').isNull, 0.0
      ).otherwise(col(''meas'')))
    .drop($''meas'')
    .distinct()
}
```

Listado 5.7: Cálculo de la métrica en el grafo

```
def getMeasurementSubgraph(subjects: VertexRDD[Node], graph: Graph[Node,
  Node], properties: Seq[String]): Dataset[Row] = {
  import processSparkSession.implicits._
  val subjectsDF = subjects
    .filter(ll=> ll._2 != null)
    .filter(l => l._2.isURI())
    .map(l => (l._1, l._2.toString()))
    .toDF(Seq(''srcId'', ''uri''): _*)
  getMeasurementGlobal(graph, properties).join(subjectsDF, $''source'' ===
  $''srcId'')
    .drop($''srcId'')
```

```
}
```

Listado 5.8: Cálculo de la métrica en colección de nodos

Finalmente se acaba la iteración con las pruebas correspondientes a la funcionalidad desarrollada.

Ejemplo
[EJEMPLO GRÁFICO AQUÍ]

5.4 Fase de Transición

En esta fase del PUD se abordarán dos iteraciones. Por un lado, el desarrollo de la prueba de concepto, que hará uso de los artefactos generados hasta el momento. Finalmente, la última de las iteraciones consistirá en la elaboración de todos los documentos necesarios para la entrega del presente trabajo, incluyendo manuales, anexos de código y el documento TFM.

5.4.1 Iteración 7: Desarrollo de la PoC

Fase del PUD	Transición
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Desarrollo de PoC ▪ Implementación de arquitectura PoC ▪ QA	▪ Artefacto PoC ▪ Arquitectura final

Tabla 5.7: Iteración 7

El objetivo de esta PoC es ilustrar la capacidad de SparkDQ a través de un escenario de uso real.
En este escenario se tomará como fuente de datos un conjunto de tweets extraídos en tiempo real de su API. Se transformarán dichos tweets en datos semánticos y sobre éstos se llevarán a cabo análisis de calidad de los datos para las métricas implementadas.
Finalmente se generarán resultados y éstos estarán expuestos en una plataforma cloud listos para su visualización y consumo.
[INSERTA IMAGEN AQUI ARQUITECTURA]
El desarrollo de la prueba de concepto ha involucrado varias fases:

- Desarrollo del artefacto Spark encargado de realizar los cálculos de las métricas, aprovechando el stack SparkDQ previamente desarrollado. Este artefacto se ha llamado spark-sem-dq-assessment.
- Implementación de microservicio encargado de coordinar el flujo de datos entre los distintos sistemas.
- Desarrollo de una ontología para la representación de tweets.
- Implementación de microservicio encargado de transformar tweets recibidos a través de una api REST en archivos semánticos en formato `NT`.
- Servicios cloud: almacenamiento en la nube Amazon S3.
- Servicios cloud: consumo de resultados a través de Amazon ElasticSearch y Kibana.

Cálculos de las métricas: spark-sem-dq-assessment
Spark-sem-dq-assessment es un job de Spark que incluye dependencias con SparkDQ y está orientado a generar métricas de calidad.
[EJEMPLO SUBMIT]
La entrada y salida de datos de spark-sem-dq-assessment puede configurarse de distintas formas:

- Directorios locales a una máquina
- Directorios distribuidos en HDFS
- Directorios distribuidos cloud en Amazon S3

La salida para cada ejecución será:

- Un resultado para la métrica con la que se programe la ejecución
- Un resultado de estadísticas relativas a la ejecución del propio job de Spark, incluyendo (ver listado 5.9):
  - Nombre de la métrica evaluada.
  - Fecha de cálculo.
  - Tiempo de generación del grafo.
  - Tiempo de proceso de métrica.
  - Tiempo total de procesamiento.
  - Número total de nodos.
  - Número total de arcos.
  - Tiempo medio por nodo.
  - Tiempo medio por arco.
  - Tiempo medio de procesamiento por nodo.
  - Tiempo medio de procesamiento por arco.
- Se guardarán los resultados de la evaluación y las métricas estadísticas en formato `JSON`.

```
val statisticsDF = sparkSession.sparkContext.parallelize(Seq((
  calculationDate,
  ''InterlinkingAssessment'',
  loadGraphTime,
  processTime,
  totalTime,
  nNodes,
  nEdges,
  avgReadTimePerNode,
  avgReadTimePerEdge,
  avgProcessTimePerNode,
  avgProcessTimePerEdge))).toDF(Seq(
  ''calculationDate'',
  ''metric'',
  ''loadGraphTime'',
  ''processTime'',
  ''totalTime'',
  ''nNodes'',
  ''nEdges'',
  ''avgReadTimePerNode'',
  ''avgReadTimePerEdge'',
  ''avgProcessTimePerNode'',
  ''avgProcessTimePerEdge''
): _*)
```

Listado 5.9: Campos de salida para las métricas estadísticas sobre la evaluación de calidad

Microservicio de coordinación: DockerNiFi

Se ha utilizado un contenedor de Docker junto con NiFi (véase Sección 3.5.3) para coordinar tres flujos de datos:

1. Adquisición de tweets: NiFi ofrece un procesador específico que consulta la API de Twitter, pudiéndose configurar por conceptos, cuentas o coordenadas para tweets geo-localizados. Tan solo es necesario introducir las credenciales de la aplicación y NiFi comienza a consumir información. Desde ese momento, se genera un FlowFile por cada tweet consumido. Sobre estos tweets se aplica una serie de transformaciones (TODO) y finalmente se envían a ms.semtweet para que sea transformado de JSON a NT. (REFERENCIA IMAGEN)

2. Enrutado de sem-tweets a almacenamiento distribuido: La segunda topología levanta un endpoint al que ms.semtweet enviará los resultados de su procesamiento y NiFi los enrutará hacia el sistema de almacenamiento elegido (directorio local, HDFS o S3).(REFERENCIA IMAGEN)

3. Lectura desde almacenamiento y envío a sistema de visualización de resultados: NiFi

permanecerá a la escucha en la ruta en la que los archivos de resultado de spark-sem-dq-assesment fuesen depositados, recuperándolos en el momento en que se produzcan cambios y enviándolos al sistema de visualización: Amazon ElasticSearch y Kibana. NiFi obtendrá desde el path de lectura de los archivos el índice de ElasticSearch al que debe mandar cada dato, de manera automática y transparente. Los índices que se han creado para estos servicios y el nombre de carpetas de S3 donde son almacenados los resultados de spark-sem-dq-assessment se comentarán en la siguiente sección. Se puede encontrar un ejemplo ilustrativo en la imagen (REFERENCIA IMAGEN).

En el anexo C se puede encontrar el DockerFile necesario para instalar este servicio.

Ontología para representación de tweets: Ontotweet

Se ha desarrollado una ontología para representar la información que contiene un tweet en el momento que es consumido desde la API. Esta ontología modela las siguientes entidades:

- Tweet
- User
- Coordinates
- Entity
  - Hashtag

El resultado se puede consultar en el anexo A.

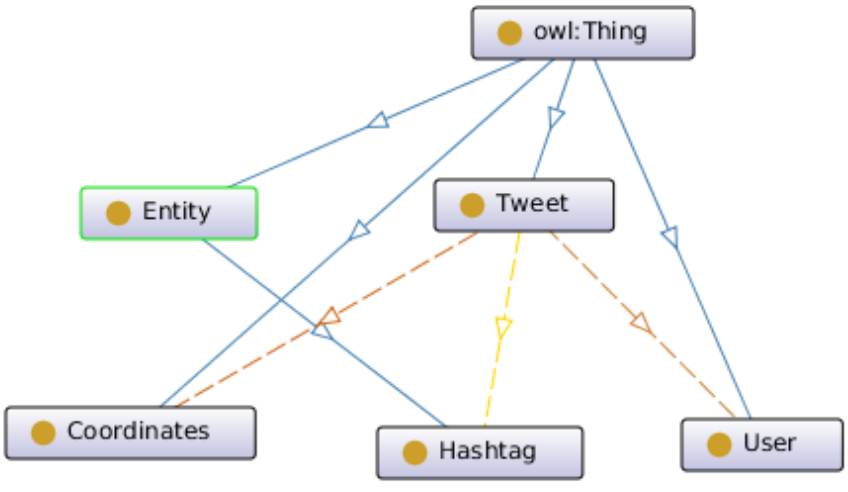

Figura 5.4: Esquema de la ontología de Twitter en Protégé

Microservicio de transformación de tweets: ms.semtweet

La finalidad de este microservicio es dado un JSON con información de un tweet, obtener como resultado un conjunto de triplas conforme a la ontología desarrollada en el punto ante-

rior.
Para ello se expone una API REST que recibirá el tweet y enviará una vez transformado a triplas, el resultado a un endpoint que se configurará como parámetro en el microservicio.
En el listado 5.10 se puede encontrar una relación entre clases en Scala para la transformación.
En el anexo D puede consultarse parte del código para generar las triplas, mientras que en el anexo E se muestra el DockerFile para lanzar el microservicio.

```
package org.uclm.alarcos.rrc.ms.models
case class User(id: Int, id_str: String, name: String, verified: Boolean,
                followers_count: Long, friends_count: Long,
                favourites_count: Long, created_at: String,
                lang: String, time_zone: Option[String])
case class Coordinate(coordinates: Array[Double], `type`: String)
case class Hashtag(indices: Array[Int], text: String)
case class Entity(hashtags: Array[Hashtag])
case class Tweet (created_at: String, id: Int, id_str: String, text: String,
                  source: String, truncated: Boolean,
                  retweet_count: Long, favorite_count: Long, lang: String,
                  timestamp_ms: String, user: User, coordinates:
                  Option[Coordinate], entities: Entity)
```

Listado 5.10: Entidades de transformación en Scala

Servicios cloud
Se han contratado para la PoC los siguientes servicios de Amazon Web Services:

- Amazon S3: como almacenamiento escalable y distribuído. Existirán tres directorios principales:
  - `poc-tfm-nifi`: como directorio para el almacenamiento de tweets semánticos, es decir, tras haber sido procesados por ms.semtweet.
  - `spark-sem-dq-assessment`: destino de los archivos de salida del job de Spark de la PoC, organizado como sigue:
    - SchemaAssessment: resultados para las evaluaciones de *SchemaCompleteness*.
    - InterlinkingAssessment: resultados para las evaluaciones de *Interlinking*.
    - DQAssessmentStatistics: resultados estadísticos para cualquiera de las dos evaluaciones anteriores.
- ElasticSearch y Kibana: como capa de baja latencia para el consumo de resultados de evaluaciones y visualización. Se han creado los siguientes índices:
  - `twittertfm`: indexará todos los tweets en bruto tal y como se obtienen de la API de Twitter, sin procesar.

- `interlinkingassessment`: resultados para las evaluaciones de *Interlinking*.
- `schemaasssessment`: resultados para las evaluaciones de *SchemaCompleteness*.
- `dqassessmentstatistics`: resultados estadísticos para cualquiera de las dos evaluaciones anteriores.

5.4.2 Iteración 8: Entrega de TFM

Fase del PUD	Transición
Flujo de trabajo del PUD	Documentación
Fechas Inicio - Fin	
Objetivos	Artefactos de Salida
▪ Elaboración de la documentación para los distintos artefactos (SparkRDF, SparkDQ) ▪ Elaboración de la documentación para la PoC ▪ Documentación TFM	▪ Documentación artefactos ▪ Documentación PoC ▪ Documentación TFM

Tabla 5.8: Iteración 8

La última de las iteraciones ha consistido en un trabajo de documentación detallado de todos los pasos que han tenido lugar para la elaboración del presente trabajo, dando como resultado los siguientes elementos:

- Memoria del TFM.
- ScalaDoc de SparkRDF y SparkDQ.
- Anexos.
  - Ontología de tweets.
  - Ontología DQ.
  - Anexo de QA.
- Vídeo demostración de la PoC.

La totalidad de la memoria ha sido realizada en LaTeX, mientras que la documentación de las diferentes APIs se ha generado con ScalaDoc, análogo a JavaDoc.
Todos los archivos de ontologías han sido incluidos en la documentación.

Capítulo 6

Conclusiones

Breve resumen de lo más destacable del trabajo con la solución propuesta. Análisis del logro del objetivo general y objetivos parciales propuestos. Concluir con posibles mejoras, ampliaciones o trabajos relacionados que quedan por hacer y que tienen interés para el tema tratado.

A modo de referencia, este capítulo tendrá una longitud aproximada de entre 2 y 10 páginas.

6.1 Conclusiones
6.2 Propuestas de trabajos futuros
6.3 Publicaciones
6.4 Opinión personal

ANEXOS

Anexo A

Ontología de tweets: Ontotwitter

```
<?xml version=''1.0''?>
<rdf:RDF
xmlns=''http://www.semanticweb.org/rrc/ontologies/2017/7/untitled-ontology-2#''
     xml:base=''http://www.semanticweb.org/rrc/ontologies/2017/7/untitled-ontology-2''
     xmlns:semtweet=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#''
     xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
     xmlns:owl=''http://www.w3.org/2002/07/owl#''
     xmlns:xml=''http://www.w3.org/XML/1998/namespace''
     xmlns:xsd=''http://www.w3.org/2001/XMLSchema#''
     xmlns:untitled-ontology-2=''http://www.semanticweb.org/rrc/ontologies/2017/7/untitled-ontology
          -2#''
     xmlns:rdfs=''http://www.w3.org/2000/01/rdf-schema#''>
    <owl:Ontology
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet''/>
    <!--
    ////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ////////////////////////////////////////////////////////////////////////////////
     -->
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasCoordinates -->

    <owl:ObjectProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasCoordinates''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coordinates''/>
    </owl:ObjectProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasHashtag
    -->
    <owl:ObjectProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasHashtag''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hashtag''/>
    </owl:ObjectProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUser -->
    <owl:ObjectProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUser''>
        <rdfs:domain
```

```
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
    </owl:ObjectProperty>
    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasCreationDate
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasCreationDate''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#dateTime''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasFavouriteCount
    -->

    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasFavouriteCount''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#long''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasFollowersCount
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasFollowersCount''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#long''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasHashtagValue
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasHashtagValue''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hashtag''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#string''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasId -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasId''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#long''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasLang -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasLang''>
```

```
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#language''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasLatitude
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasLatitude''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coordinates''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#double''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasLongitude
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasLongitude''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coordinates''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#double''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasName -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasName''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#string''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasRetweetCount
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasRetweetCount''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#long''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasText -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasText''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#string''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasTextSource
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasTextSource''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range
        rdf:resource=''http://www.w3.org/2001/XMLSchema#normalizedString''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasTimeStampMs -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasTimeStampMs''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
```

```
        <rdfs:range
        rdf:resource=''http://www.w3.org/2001/XMLSchema#unsignedLong''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasTimeZone
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasTimeZone''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#string''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserCreationDate
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserCreationDate''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#dateTime''/>
    </owl:DatatypeProperty>
    <!--
    http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserFavouriteCount
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserFavouriteCount''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#long''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserId -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserId''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#long''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserLang
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#hasUserLang''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#language''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#isTruncated
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#isTruncated''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#boolean''/>
    </owl:DatatypeProperty>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#isVerified
    -->
    <owl:DatatypeProperty
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#isVerified''>
        <rdfs:domain
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
```

```
        <rdfs:range rdf:resource=''http://www.w3.org/2001/XMLSchema#boolean''/>
    </owl:DatatypeProperty>
    <!--
    ////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ////////////////////////////////////////////////////////////////////////////////////
     -->
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coordinates
    -->
    <owl:Class
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coordinates''/>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Entity -->
    <owl:Class
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Entity''/>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hashtag -->
    <owl:Class
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hashtag''>
        <rdfs:subClassOf
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Entity''/>
    </owl:Class>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet -->
    <owl:Class
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User -->
    <owl:Class
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
    <!--
    ////////////////////////////////////////////////////////////////////////////////////
    //
    // Individuals
    //
    ////////////////////////////////////////////////////////////////////////////////////
     -->
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coord1 -->
    <owl:NamedIndividual
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coord1''>
        <semtweet:hasLatitude
        rdf:datatype=''http://www.w3.org/2001/XMLSchema#double''>40.05701649</semtweet:hasLatitude>
        <semtweet:hasLongitude
        rdf:datatype=''http://www.w3.org/2001/XMLSchema#double''>-75.14310264</semtweet:hasLongitude
            >
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hash1 -->
    <owl:NamedIndividual
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hash1''>
        <semtweet:hasHashtagValue
        rdf:datatype=''http://www.w3.org/2001/XMLSchema#string''>#TestHashtag</
            semtweet:hasHashtagValue>
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet1 -->
    <owl:NamedIndividual
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet1''>
        <semtweet:hasCoordinates
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coord1''/>
        <semtweet:hasUser
        rdf:resource=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User1''/>
```

```
        <semtweet:hasId
        rdf:datatype=''http://www.w3.org/2001/XMLSchema#long''>819797</semtweet:hasId>
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User1 -->
    <owl:NamedIndividual
    rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User1''>
        <semtweet:hasUserId
        rdf:datatype=''http://www.w3.org/2001/XMLSchema#long''>819797</semtweet:hasUserId>
        <semtweet:isVerified
        rdf:datatype=''http://www.w3.org/2001/XMLSchema#boolean''>true</semtweet:isVerified>
    </owl:NamedIndividual>
    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // General axioms
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->
    <rdf:Description>
        <rdf:type
        rdf:resource=''http://www.w3.org/2002/07/owl#AllDisjointClasses''/>
        <owl:members rdf:parseType=''Collection''>
            <rdf:Description
            rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Coordinates''/>
            <rdf:Description
            rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Hashtag''/>
            <rdf:Description
            rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#Tweet''/>
            <rdf:Description
            rdf:about=''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet#User''/>
        </owl:members>
    </rdf:Description>
</rdf:RDF>
```

Listado A.1: Ontología de tweets: Ontotwitter

Anexo B

Vocabulario *Data Quality Assessment*

```
<?xml version="1.0"?>
<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<Ontology xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.dqassessment.org/ontologies/2014/9/DQA.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     ontologyIRI="http://www.dqassessment.org/ontologies/2014/9/DQA.owl#">
    <Prefix name="" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
    <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
    <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#"/>
    <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#"/>
    <Annotation>
        <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
        <Literal datatypeIRI="&rdf;PlainLiteral">an ontology or vocabulary that describes various
            data quality context-aware assessment through various data quality dimensions.

pfc - raul reguillo carmona
a jena extension to support data quality context-aware assessment of linked data</Literal>
    </Annotation>
    <Declaration>
        <Class IRI="AccessibilityAssessment"/>
    </Declaration>
    <Declaration>
        <Class IRI="CompletenessAssessment"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="AccessibilityMeasure"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="AccessibilityResult"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="AssessmentDate"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="CompletenessMeasure"/>
```

```
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="CompletenessResult"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="ContextualMeasure"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="Identifier"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="InLevel"/>
    </Declaration>
    <Declaration>
        <ObjectProperty IRI="InURI"/>
    </Declaration>
    <Declaration>
        <DataProperty IRI="AccessibilityResult"/>
    </Declaration>
    <Declaration>
        <DataProperty IRI="AssessmentDate"/>
    </Declaration>
    <Declaration>
        <DataProperty IRI="CompletenessResult"/>
    </Declaration>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="AccessibilityMeasure"/>
        <ObjectProperty IRI="AccessibilityResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="AccessibilityResult"/>
        <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="CompletenessMeasure"/>
        <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="CompletenessResult"/>
        <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="ContextualMeasure"/>
        <ObjectProperty IRI="AccessibilityResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="ContextualMeasure"/>
        <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="InLevel"/>
        <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="InURI"/>
        <ObjectProperty IRI="AccessibilityResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
```

```
        <ObjectProperty IRI="InURI"/>
        <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="AccessibilityMeasure"/>
        <Class IRI="AccessibilityAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="AccessibilityResult"/>
        <Class IRI="AccessibilityAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="AssessmentDate"/>
        <Class IRI="AccessibilityAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="AssessmentDate"/>
        <Class IRI="CompletenessAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="CompletenessMeasure"/>
        <Class IRI="CompletenessAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="CompletenessResult"/>
        <Class IRI="CompletenessAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="ContextualMeasure"/>
        <Class IRI="AccessibilityAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="ContextualMeasure"/>
        <Class IRI="CompletenessAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="Identifier"/>
        <Class abbreviatedIRI="owl:Thing"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="InLevel"/>
        <Class IRI="CompletenessAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="InURI"/>
        <Class IRI="AccessibilityAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyDomain>
        <ObjectProperty IRI="InURI"/>
        <Class IRI="CompletenessAssessment"/>
    </ObjectPropertyDomain>
    <ObjectPropertyRange>
        <ObjectProperty IRI="AccessibilityMeasure"/>
        <DataExactCardinality cardinality="1">
            <DataProperty IRI="AccessibilityResult"/>
            <Datatype abbreviatedIRI="xsd:double"/>
        </DataExactCardinality>
    </ObjectPropertyRange>
```

```
    <ObjectPropertyRange>
        <ObjectProperty IRI="AccessibilityResult"/>
        <DataExactCardinality cardinality="1">
            <DataProperty IRI="AccessibilityResult"/>
            <Datatype abbreviatedIRI="xsd:anyURI"/>
        </DataExactCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="AssessmentDate"/>
        <DataExactCardinality cardinality="1">
            <DataProperty IRI="AssessmentDate"/>
            <Datatype abbreviatedIRI="xsd:dateTime"/>
        </DataExactCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="CompletenessMeasure"/>
        <DataExactCardinality cardinality="1">
            <DataProperty IRI="CompletenessResult"/>
            <Datatype abbreviatedIRI="xsd:double"/>
        </DataExactCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="CompletenessResult"/>
        <DataMinCardinality cardinality="1">
            <DataProperty IRI="CompletenessResult"/>
            <Datatype abbreviatedIRI="xsd:anyURI"/>
        </DataMinCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="ContextualMeasure"/>
        <DataExactCardinality cardinality="1">
            <DataProperty IRI="CompletenessResult"/>
            <Datatype abbreviatedIRI="xsd:anyURI"/>
        </DataExactCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="Identifier"/>
        <DataMinCardinality cardinality="1">
            <DataProperty abbreviatedIRI="owl:topDataProperty"/>
            <Datatype abbreviatedIRI="xsd:string"/>
        </DataMinCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="InLevel"/>
        <DataExactCardinality cardinality="1">
            <DataProperty IRI="CompletenessResult"/>
            <Datatype abbreviatedIRI="xsd:int"/>
        </DataExactCardinality>
    </ObjectPropertyRange>
    <ObjectPropertyRange>
        <ObjectProperty IRI="InURI"/>
        <DataMinCardinality cardinality="1">
            <DataProperty IRI="CompletenessResult"/>
            <Datatype abbreviatedIRI="xsd:anyURI"/>
        </DataMinCardinality>
    </ObjectPropertyRange>
</Ontology>
```

```
<!-- Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net -->
```

Listado B.1: Vocabulario utilizado para los resultados finales

Anexo C

Microservicio DockerNiFi: DockerFile

```
FROM centos:latest
MAINTAINER Raul Reguillo raul.reguillo {at} gmail.com

#install yum repos
RUN yum update -y &&\
    yum install -y which wget git java-1.8.0-openjdk python && yum clean all

ENV JAVA_HOME /usr/lib/jvm/jre-1.8.0-openjdk
RUN export JAVA_HOME

ENV NIFI_VERSION=1.3.0 \
        NIFI_HOME=/opt/nifi \
        MIRROR_SITE=http://apache.rediris.es

# Picked the recommended mirror from Apache for the distribution.
# Import the Apache NiFi release keys, download the release, and set up a user
# to run NiFi.
RUN set -x \
        && curl -Lf https://dist.apache.org/repos/dist/release/nifi/KEYS -o
# /tmp/nifi-keys.txt \
        && gpg --import /tmp/nifi-keys.txt \
        && curl -Lf
# ${MIRROR_SITE}/nifi/${NIFI_VERSION}/nifi-${NIFI_VERSION}-bin.tar.gz -o
# /tmp/nifi-bin.tar.gz \
        && mkdir -p ${NIFI_HOME} \
        && tar -z -x -f /tmp/nifi-bin.tar.gz -C ${NIFI_HOME}
# --strip-components=1 \
        && rm /tmp/nifi-keys.txt \
        && sed -i -e ''s|^nifi.ui.banner.text=.*$$|nifi.ui.banner.text=Docker
# NiFi ${NIFI_VERSION}|'' ${NIFI_HOME}/conf/nifi.properties \
        && groupadd nifi \
        && useradd -r -g nifi nifi \
        && bash -c ''mkdir -p
# ${NIFI_HOME}/{database_repository,flowfile_repository,content_repository,provenance_repository}''
# \
        && chown nifi:nifi -R ${NIFI_HOME}

# These are the volumes (in order) for the following:
# 1) user access and flow controller history
# 2) FlowFile attributes and current state in the system
# 3) content for all the FlowFiles in the system
# 4) information related to Data Provenance
# You can find more information about the system properties here -
```

```
# https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html#system_properties
VOLUME [''${NIFI_HOME}/database_repository'', \
        ''${NIFI_HOME}/flowfile_repository'', \
        ''${NIFI_HOME}/content_repository'', \
        ''${NIFI_HOME}/provenance_repository'']

# Open port 8081 for the HTTP listen
USER nifi
WORKDIR ${NIFI_HOME}
EXPOSE 8080 8081
CMD [''bin/nifi.sh'', ''run'']
```

Listado C.1: Microservicio DockerNiFi: DockerFIle

Anexo D

Microservicio ms.semtweet

```
package org.uclm.alarcos.rrc.ms.services.semantic
import java.io.{File, IOException, PrintWriter, StringWriter}
import java.util.logging.Logger

import akka.actor.ActorSystem
import akka.stream.Materializer
import org.apache.jena.datatypes.xsd.XSDDatatype
import org.apache.jena.ontology.{Individual, OntModel, OntModelSpec}
import org.apache.jena.rdf.model.{Model, ModelFactory, Property}
import org.apache.jena.util.FileManager
import org.uclm.alarcos.rrc.ms.models.{Coordinate, Tweet}

import scala.concurrent.ExecutionContextExecutor

trait Semtweet {

  private[this] val logger = Logger.getLogger(getClass().getName())

  implicit val system: ActorSystem
  implicit def executor: ExecutionContextExecutor
  implicit val materializer: Materializer

  var ontFile = ''ontotwitter/semtweets.owl''
  val nameSpace = ''http://www.semanticweb.org/rrc/ontologies/2017/7/semtweet''
  val prefix = ''semtweet:''
  val TWEET = nameSpace + ''#Tweet''
  val USER = nameSpace + ''#User''
  val COORD = nameSpace + ''#Coordinates''
  val HASH = nameSpace + ''#Hashtag''

  def modelToString(model: Model, format: String): String = {
    val out: StringWriter = new StringWriter()
    model.write(out, format)
    out.toString()
  }

  def generateFile(tweet: Tweet, model: Model): File = {
    val outputFile = new File(''outputs/'' + tweet.id_str + ''.rdf'')
    val pw = new PrintWriter(outputFile)
    model.write(pw)
    outputFile
  }
```

```
  def semTweet(tweet: Tweet): Model = {
    var resultModel = ModelFactory.createDefaultModel()
    val ontTweet  = getOntologyModel(ontFile)

    //Classes
    var cTweet = ontTweet.getOntClass(TWEET)
    var cUser = ontTweet.getOntClass(USER)
    var cCoords = ontTweet.getOntClass(COORD)
    var cHash = ontTweet.getOntClass(HASH)
    //Data Properties
    // - tweet
    val hasId: Property = ontTweet.getOntProperty(nameSpace + ''#hasId'')
    val hasCreationDate = ontTweet.getOntProperty(nameSpace + ''#hasCreationDate'')
    val hasFavouriteCount = ontTweet.getOntProperty(nameSpace + ''#hasFavouriteCount'')
    val hasLang = ontTweet.getOntProperty(nameSpace + ''#hasLang'')
    val hasRetweetCount = ontTweet.getOntProperty(nameSpace + ''#hasRetweetCount'')
    val hasText = ontTweet.getOntProperty(nameSpace + ''#hasText'')
    val hasTextSource = ontTweet.getOntProperty(nameSpace + ''#hasTextSource'')
    val hasTimeStampMs = ontTweet.getOntProperty(nameSpace + ''#hasTimeStampMs'')
    val isTruncated = ontTweet.getOntProperty(nameSpace + ''#isTruncated'')
    // -- object properties
    val hasCoordinates = ontTweet.getOntProperty(nameSpace + ''#hasCoordinates'')
    val hasUser = ontTweet.getOntProperty(nameSpace + ''#hasUser'')
    val hasHashtag = ontTweet.getOntProperty(nameSpace + ''#hasHashtag'')

    // - user
    val hasFollowersCount = ontTweet.getOntProperty(nameSpace + ''#hasFollowersCount'')
    val hasName = ontTweet.getOntProperty(nameSpace + ''#hasName'')
    val hasUserCreationDate = ontTweet.getOntProperty(nameSpace + ''#hasUserCreationDate'')
    val hasUserFavouriteCount = ontTweet.getOntProperty(nameSpace + ''#hasUserFavouriteCount'')
    val hasUserId = ontTweet.getOntProperty(nameSpace + ''#hasUserId'')
    val hasUserLang = ontTweet.getOntProperty(nameSpace + ''#hasUserLang'')
    val hasTimeZone = ontTweet.getOntProperty(nameSpace + ''#hasTimeZone'')
    val isVerified = ontTweet.getOntProperty(nameSpace + ''#isVerified'')

    // - hashtag
    val hasHashtagValue = ontTweet.getOntProperty(nameSpace + ''#hasHashtagValue'')

    // - coordinates
    val hasLatitude = ontTweet.getOntProperty(nameSpace + ''#hasLatitude'')
    val hasLongitude = ontTweet.getOntProperty(nameSpace + ''#hasLongitude'')

    val t1: Individual = ontTweet.createIndividual(nameSpace + ''#tweet_'' + tweet.id_str, cTweet)
    val u1: Individual = ontTweet.createIndividual(nameSpace + ''#user_'' + tweet.user.id_str, cUser
        )
    val c1: Individual = ontTweet.createIndividual(nameSpace + ''#coord_'' + tweet.id_str, cCoords)
    val h1: Individual = ontTweet.createIndividual(nameSpace + ''#hasht_'' + tweet.id_str, cHash)
    //Tweet Properties
    t1.addProperty(hasId, tweet.id.toString(), XSDDatatype.XSDlong)
    t1.addProperty(hasCreationDate, tweet.created_at, XSDDatatype.XSDdate)
    t1.addProperty(hasFavouriteCount, tweet.favorite_count.toString(), XSDDatatype.XSDlong)
    t1.addProperty(hasLang, tweet.lang, XSDDatatype.XSDlanguage)
    t1.addProperty(hasRetweetCount, tweet.retweet_count.toString(), XSDDatatype.XSDlong)
    t1.addProperty(hasText, tweet.text)
    t1.addProperty(hasTextSource, tweet.source)
    t1.addProperty(hasTimeStampMs, tweet.timestamp_ms.toString(), XSDDatatype.XSDunsignedLong)
    t1.addProperty(isTruncated, tweet.truncated.toString(), XSDDatatype.XSDboolean)
```

```
      // - object properties
      //t1.addProperty(hasCoordinates, c1) see below in Coordinates
      t1.addProperty(hasUser, u1)
      t1.addProperty(hasHashtag, h1)

      //User
      u1.addProperty(hasFollowersCount, tweet.user.followers_count.toString(), XSDDatatype.XSDlong)
      u1.addProperty(hasName,tweet.user.name)
      u1.addProperty(hasTimeZone,tweet.user.time_zone.getOrElse(''no_timezone''))
      u1.addProperty(hasUserCreationDate, tweet.user.created_at, XSDDatatype.XSDdate)
      u1.addProperty(hasUserFavouriteCount, tweet.user.favourites_count.toString(), XSDDatatype.
          XSDlong)
      u1.addProperty(hasUserId, tweet.user.id.toString(), XSDDatatype.XSDlong)
      u1.addProperty(hasUserLang, tweet.user.lang, XSDDatatype.XSDlanguage)
      u1.addProperty(isVerified, tweet.user.verified.toString(), XSDDatatype.XSDboolean)

      //Hashtag
      tweet.entities.hashtags.map(x => h1.addProperty(hasHashtagValue, x.text))

      //Coordinates
      val coords: Coordinate = tweet.coordinates.getOrElse(Coordinate(Array(0.0,0.0), ''null''))
      if (!coords.'type'.equals(''null'')){
        c1.addProperty(hasLongitude, coords.coordinates(0).toString(), XSDDatatype.XSDdouble)
        c1.addProperty(hasLatitude, coords.coordinates(1).toString(), XSDDatatype.XSDdouble)
        t1.addProperty(hasCoordinates, c1)
      }
      ontTweet
    }


    def getOntologyModel(ontFile: String): OntModel = {
      val ontoModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM, null)
      try{
        val in = FileManager.get().open(ontFile)
        ontoModel.read(in, null)
      }
      catch {
        case e: Exception => e.printStackTrace()
      }
      ontoModel
    }
  }
```

Listado D.1: ms.semtweet: transformación a triplas

Anexo E

Microservicio ms.semtweet: DockerFile

```
FROM centos:latest
MAINTAINER Raul Reguillo raul.reguillo at gmail.com
RUN yum update -y && \
yum install -y which wget git java-1.8.0-openjdk && yum clean all
ENV JAVA_HOME /usr/lib/jvm/jre-1.8.0-openjdk
ENV SEMTWEET_HTTP_INTERFACE 0.0.0.0
ENV SEMTWEET_HTTP_PORT 9003
ENV SEMTWEET_OUTPUT_HTTP_INTERFACE 172.17.0.2
ENV SEMTWEET_OUTPUT_HTTP_PORT 9091
ENV SEMTWEET_OUTPUT_HTTP_SERVICE datavault/v1/listener

ADD /target/ms.semtweet-1.0-SNAPSHOT-allinone.jar ms.semtweet-1.0-SNAPSHOT-allinone.jar
ADD /ontotwitter/semtweets.owl ontotwitter/semtweets.owl

EXPOSE 9003
ENTRYPOINT [‘‘java’’, ‘‘-jar’’, ‘‘ms.semtweet-1.0-SNAPSHOT-allinone.jar’’]
```

Listado E.1: ms.semtweet: DockerFile

Anexo F

SparkDQ: Casos de prueba para *SchemaCompleteness*

```
package org.uclm.alarcos.rrc.dq

import org.junit.runner.RunWith
import org.scalamock.scalatest.MockFactory
import org.scalatest.junit.JUnitRunner
import org.uclm.alarcos.rrc.CommonTest
import org.uclm.alarcos.rrc.dataquality.completeness.{Interlinking,
  SchemaCompleteness}
import org.uclm.alarcos.rrc.spark.SparkSpec

RunWith(classOf[JUnitRunner])
class MeasurementsTest extends CommonTest with SparkSpec with MockFactory {
  //Nodes IDS
  val A = 293150257L
  val B = 293150288L
  val C = 293150319L
  val D = 293150350L
  val E = 293150381L
  val F = 293150412L
  val G = 293150443L

  ''Execute getMEasurementSugraph SchemaCompleteness'' should ''be succesfully''
  in {
    val testPath = ''src/test/resources/dataset/tinysampleSC.nt''
    object MockedTripleReader extends SchemaCompleteness(spark, testPath)
    val step = MockedTripleReader
    val graph = step.loadGraph(spark, testPath)
    val properties = Seq(
      ''http://xmlns.com/foaf/0.1/name'',
      ''http://xmlns.com/foaf/0.1/exprop1''
    )
    val verts = graph.vertices.filter(l => l._2.isURI())
    verts.collect().foreach(println(_))
    val result = step.getMeasurementSubgraph(verts, graph, properties)
    val results = result.collect()
    result.show(1000, truncate=false)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === A & l.get(1) ===
    1.0) == 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === B & l.get(1) ===
    1.0) == 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === C & l.get(1) ===
    1.0) == 1)
```

```
    assert(results.count(l => l.get(0).asInstanceOf[Long] === D & l.get(1) ===
46  0.5) == 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === E & l.get(1) ===
    0.5) == 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === F & l.get(1) ===
    0.0) == 1)
51  assert(results.count(l => l.get(0).asInstanceOf[Long] === G & l.get(1) ===
    0.0) == 1)

}
```

Listado F.1: SparkDQ: Fragmento de tests unitarios para Schema Completeness

Anexo G

SparkDQ: Casos de prueba para *Interlinking*

```
package org.uclm.alarcos.rrc.dq

import org.junit.runner.RunWith
import org.scalamock.scalatest.MockFactory
import org.scalatest.junit.JUnitRunner
import org.uclm.alarcos.rrc.CommonTest
import org.uclm.alarcos.rrc.dataquality.completeness.{Interlinking,
  SchemaCompleteness}
import org.uclm.alarcos.rrc.spark.SparkSpec

RunWith(classOf[JUnitRunner])
class MeasurementsTest extends CommonTest with SparkSpec with MockFactory {
  //Nodes IDS
  val A = 293150257L
  val B = 293150288L
  val C = 293150319L
  val D = 293150350L
  val E = 293150381L
  val F = 293150412L
  val G = 293150443L

  ''Execute getMeasurementSubgraph Interlinking Completeness'' should ''be
  succesfully'' in {
    val testPath = ''src/test/resources/dataset/tinysampleI.nt''
    object MockedTripleReader extends Interlinking(spark, testPath)
    val step = MockedTripleReader
    val graph = step.loadGraph(spark, testPath)
    val depth = 4
    val result = step.getMeasurementSubgraph(graph.vertices.filter(l =>
    l._2.isURI()), graph, depth)
    val results = result.collect()
    result.show(1000, truncate=false)
    //    A -> B -> D -> F -> G
    //         | \  |
    //         v  \,v
    //         C -> E

//      +---------+-----+------+------+-------------------+
//      |   source|depth|countT|countF|        measurement|
//      +---------+-----+------+------+-------------------+
//      |293150257|    0|     1|     3|               0.25|
//      |293150257|    1|     3|     3|                0.5|
//      |293150257|    2|     2|     9|0.18181818181818182|
//      |293150257|    3|     1|     6|0.14285714285714285|
```

```
//      |293150288|    0|     3|     3|                0.5|
//      |293150288|    1|     2|     9|0.18181818181818182|
//      |293150288|    2|     1|     6|0.14285714285714285|
//      |293150288|    3|     0|     3|                0.0|
//      |293150319|    0|     1|     3|               0.25|
//      |293150319|    1|     0|     3|                0.0|
//      |293150350|    0|     2|     3|                0.4|
//      |293150350|    1|     1|     6|0.14285714285714285|
//      |293150350|    2|     0|     3|                0.0|
//      |293150381|    0|     0|     3|                0.0|
//      |293150412|    0|     1|     3|               0.25|
//      |293150412|    1|     0|     3|                0.0|
//      |293150443|    0|     0|     3|                0.0|
//      +---------+-----+------+------+-------------------+
    //Check some cases in DF
    assert(results.count(l => l.get(0).asInstanceOf[Long] === A & l.get(1) === 0
    & l.get(2) === 1 & l.get(3) === 3 & l.get(4) === 0.25) === 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === A & l.get(1) === 1
    & l.get(2) === 3 & l.get(3) === 3 & l.get(4) === 0.5) === 1)

    assert(results.count(l => l.get(0).asInstanceOf[Long] === B & l.get(1) === 0
    & l.get(2) === 3 & l.get(3) === 3 & l.get(4) === 0.5) === 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === B & l.get(1) === 3
    & l.get(2) === 0 & l.get(3) === 3 & l.get(4) === 0.0) === 1)

    assert(results.count(l => l.get(0).asInstanceOf[Long] === F & l.get(1) === 0
    & l.get(2) === 1 & l.get(3) === 3 & l.get(4) === 0.25) === 1)
    assert(results.count(l => l.get(0).asInstanceOf[Long] === F & l.get(1) === 1
    & l.get(2) === 0 & l.get(3) === 3 & l.get(4) === 0.0) === 1)

    assert(results.count(l => l.get(0).asInstanceOf[Long] === G & l.get(1) === 0
    & l.get(2) === 0 & l.get(3) === 3 & l.get(4) === 0.0) === 1)
  }
}
```

Listado G.1: SparkDQ: Fragmento de tests unitarios para Interlinking

Referencias

[AH08] Dean Allemang y James Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[APP] Your City Needs These 7 Open Data Apps. url: http://mashable.com/2012/11/07/open-data-city-apps/.

[BAR] ¿Qué es Big Data? url: https://www.ibm.com/developerworks/ssa/local/im/que-es-big-data/.

[BH12] Peter Benson y Melissa Hildebrand. *Managing Blind: A Data Quality and Data Governance Vade Mecum*. ECCMA, Bethlehem (Pensylvania), 2012.

[BHBL09] Christian Bizer, Tom Heath, y Tim Berners-Lee. Linked Data - The Story So Far:. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009. url: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/jswis.2009081901.

[BL09] Tim Berners-Lee. Linked-data design issues. W3C design issue document, June 2009. http://www.w3.org/DesignIssue/LinkedData.html. url: http://www.w3.org/DesignIssues/LinkedData.html.

[BLHL01] Tim Berners-Lee, James Hendler, y Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, Mayo 2001. url: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[CMC08] Ismael Caballero, M.A. Moraga, y Coral Calero. Integración de Aspectos de Calidad de Datos en Sistemas de Información. 2008.

[CUB] CubicWeb Semantic Web Framework. url: http://www.cubicweb.org/.

[CVCP08] Ismael Caballero, Eugenio Verbo, Coral Calero, y Mario Piattini. DQRDFS - Towards a Semantic Web Enhanced with Data Quality. En José Cordeiro, Joaquim Filipe, y Slimane Hammoudi, editors, *WEBIST (1)*, páginas 178–183. INSTICC Press, 2008. url: http://dblp.uni-trier.de/db/conf/webist/webist2008-1.html#CaballeroVCP08.

[CZ14] C. L. Philip Chen y Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.*, 275:314–347, 2014. url: https://doi.org/10.1016/j.ins.2014.01.015.

[DFP+05] Li Ding, Tim Finin, Yun Peng, Paulo Pinheiro Da Silva, y Deborah L. McGuinness. Tracking rdf graph provenance using rdf molecules. En *Proc. of the 4th International Semantic Web Conference (Poster)*, página 42, 2005. url: ftp://www.ksl.stanford.edu/local/pub/KSL_Reports/KSL-05-06.pdf.

[DG04] Jeffrey Dean y Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. En *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, Berkeley, CA, USA, 2004. USENIX Association. url: http://dl.acm.org/citation.cfm?id=1251254.1251264.

[DMVH+00] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, y Ian Horrocks. The semantic web: The roles of XML and RDF. *Internet Computing, IEEE*, 4(5):63–73, 2000. url: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=877487.

[FH10] Christian Fürber y Martin Hepp. Using Semantic Web Resources for Data Quality Management. En Philipp Cimiano y Helena Sofia Pinto, editors, *EKAW*, volume 6317 of *Lecture Notes in Computer Science*, páginas 211–225. Springer, 2010. url: http://dblp.uni-trier.de/db/conf/ekaw/ekaw2010.html#FurberH10.

[FH11] Christian Fürber y Martin Hepp. Towards a Vocabulary for Data Quality Management in Semantic Web Architectures. En *Proceedings of the 1st International Workshop on Linked Web Data Management*, LWDM '11, páginas 1–8, New York, NY, USA, 2011. ACM. url: http://doi.acm.org/10.1145/1966901.1966903.

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

[Gru] Tom Gruber. Ontology (Computer Science) - definition in Encyclopedia of Database Systems. url: http://tomgruber.org/writing/ontology-definition-2007.htm.

[Gua98] Nicola Guarino. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*, volume 46. IOS press, 1998.

[HAD] Hadoop ecosystem. url: http://hadoop.apache.org.

[HBLM02] James Hendler, Tim Berners-Le, y Eric Miller. Integrating Applications on the Semantic Web, 2002. url: http://www.w3.org/2002/07/swint.

[HT06] Harry Halpin y Henry S. Thompson. One document to bind them: combining XML, web services, and the semantic web. página 679. ACM Press, 2006. url: http://portal.acm.org/citation.cfm?doid=1135777.1135877.

[ISH85] What is Total Quality Control?: The Japanese Way: Amazon.es: Kaoru Ishikawa: Libros en idiomas extranjeros, 1985.

[ISO] ISO25012. ISO/IEC 25012 x ISO/IEC 25012 Software-Engineering - Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) - Modell der Datenqualität. Technical report.

[ITU] Y.3600:Big data - Requisitos y capacidades basados en la computación en la nube. url: https://www.itu.int/rec/T-REC-Y.3600-201511-I/es.

[JEN] Apache Jena Elephas. url: https://jena.apache.org/documentation/hadoop/.

[JEN14] Apache Jena - Home, 2014. url: http://jena.apache.org/.

[NIF] Apache NiFi. url: http://nifi.apache.org.

[NMo01] Natalya F. Noy, Deborah L. McGuinness, y others. *Ontology development 101: A guide to creating your first ontology*. Stanford knowledge systems laboratory technical report KSL-01-05 and Stanford medical informatics technical report SMI-2001-0880, 2001. url: http://liris.cnrs.fr/~amille/enseignements/Ecole_Centrale/What%20is%20an%20ontology%20and%20why%20we%20need%20it.htm.

[OPE] openRDF.org: Home. url: http://www.openrdf.org/.

[OWL] OWL - Semantic Web Standards. url: http://www.w3.org/OWL/.

[PLW02] Leo L. Pipino, Yang W. Lee, y Richard Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002. url: http://dl.acm.org/citation.cfm?id=506010.

[RCCMnR] Raúl Reguillo Carmona y Ismael Caballero Muñoz Reja. Proyecto Fin de Carrera: JenaDQ: A Jena Extension to Support Data Quality Context-Aware Assessment of Linked Data.

[RDF] RDF - Semantic Web Standards. url: http://www.w3.org/RDF/.

[RJB] James Rumbaugh, Ivar Jacobson, y Grady Booch. *EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE*. Addison Wesley.

[RJB04] James Rumbaugh, Ivar Jacobson, y Grady Booch. *The Unified Modeling Language Reference Manual,*. Addison Wesley Pub Co Inc, Boston, edición Second, Julio 2004.

[SAN] SANSA-Stack. url: http://sansa-stack.net/.

[San11] Juan Antonio Pastor Sanchez. Tecnologias de Web Semantica, 2011.

[SBF98] Rudi Studer, V.Richard Benjamins, y Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, Marzo 1998. url: http://linkinghub.elsevier.com/retrieve/pii/S0169023X97000566.

[SBLH06] Nigel Shadbolt, Tim Berners-Lee, y Wendy Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, Mayo 2006. url: http://dx.doi.org/10.1109/MIS.2006.62.

[SLW97] Diane M. Strong, Yang W. Lee, y Richard Y. Wang. Data Quality in Context. *Commun. ACM*, 40(5):103–110, Mayo 1997. url: http://doi.acm.org/10.1145/253769.253804.

[SPAa] Apache Spark. url: http://spark.apache.org.

[SPAb] SPARQL Query Language for RDF. url: http://www.w3.org/TR/rdf-sparql-query/.

[W3Ca] Semantic Web - W3C. url: http://www.w3.org/standards/semanticweb/.

[W3Cb] Semantic Web - W3C. url: http://www.w3.es/Divulgacion/GuiasBreves/WebSemantica.

[W3Cc] Tools - Semantic Web Standards. url: http://www.w3.org/2001/sw/wiki/Tools.

[Wan98] Richard Y. Wang. A Product Perspective on Total Data Quality Management. *Commun. ACM*, 41(2):58–65, Febrero 1998. url: http://doi.acm.org/10.1145/269012.269022.

[WS96] Richard Y. Wang y Diane M. Strong. Beyond Accuracy: What Data Quality Means to Data Consumers. *J. Manage. Inf. Syst.*, 12(4):5–33, Marzo 1996. url: http://dl.acm.org/citation.cfm?id=1189570.1189572.

[XML] Extensible Markup Language (XML). url: http://www.w3.org/XML/.

[ZRM+13] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, Soren Auer, y Pascal Hitzler. Quality assessment methodologies for linked open data. *Submitted to Semantic Web Journal*, 2013. url: http://semantic-web-journal.net/system/files/swj414.pdf.

Este documento fue editado y tipografiado con LaTeX empleando
la clase **esi-tfm** (versión 0.20170930) que se puede encontrar en:
`https://bitbucket.org/arco_group/esi-tfg`