

**JENADQ: A JENA EXTENSION TO SUPPORT DATA QUALITY
CONTEXT-AWARE ASSESSMENT OF LINKED DATA**



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

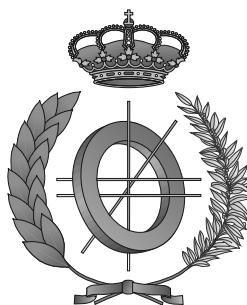
INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

JenaDQ: A Jena extension to Support Data Quality
Context-Aware Assessment of Linked Data

Raúl Reguillo Carmona

Septiembre, 2014



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA
Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

JenaDQ: A Jena extension to Support Data Quality
Context-Aware Assessment of Linked Data

Autor: Raúl Reguillo Carmona
Director: Dr. Ismael Caballero Muñoz-Reja

Septiembre, 2014

Raúl Reguillo Carmona

Ciudad Real – Spain

E-mail: Raul.Reguillo@alu.uclm.es

Teléfono: 638 175 608

Web site:

© 2014 Raúl Reguillo Carmona

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia esta incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

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 consiste 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 será, 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.

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 easily 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 into account or has been given 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 project involves the development of an extension for a Semantic Web and Linked Data framework: Apache Jena. The purpose of this extension is, having a set of Linked Data, to support a set of data quality measures taking into account the context in which these data should be considered.

Índice general

Resumen	XI
Abstract	XIII
Índice general	XV
Índice de cuadros	XIX
Índice de figuras	XXI
Índice de listados	XXIII
Listado de acrónimos	XXV
Agradecimientos	XXVII
1. Introducción	1
1.1. Descripción del problema	1
1.2. Estructura del documento	3
2. Objetivos	5
2.1. Objetivo principal	5
2.2. Objetivos parciales	5
3. Estado del Arte	7
3.1. Web Semántica	7
3.1.1. Concepto	7
3.1.2. Terminología de Web Semántica	8
3.1.3. Estándares	9
3.1.4. Arquitectura	11
3.1.5. Ontologías	13
3.1.6. Vocabularios	15

3.1.7.	Frameworks para el desarrollo de aplicaciones de Web Semántica	17
3.2.	Datos Enlazados (LD)	21
3.2.1.	Definición de LD	21
3.2.2.	Datos Abiertos (OD) y Datos Enlazados Abiertos (LOD)	21
3.2.3.	LOD en la actualidad	23
3.3.	Calidad de Datos	23
3.3.1.	Dimensiones de Calidad de Datos	25
3.3.2.	Completeness	25
3.3.3.	Accessibility	26
3.4.	Calidad de Datos en LD	26
4.	Metodología	29
4.1.	Proceso Unificado de Desarrollo (PUD)	29
4.1.1.	Características del Proceso Unificado de Desarrollo	29
4.1.2.	Ciclo de vida del Proceso Unificado de Desarrollo	32
4.2.	Planificación del Proyecto	35
4.2.1.	Requisitos Funcionales (RF) para JenaDQ	36
4.2.2.	Iteraciones del PUD	39
4.3.	Marco tecnológico de trabajo	49
4.3.1.	Frameworks de desarrollo de aplicaciones de Web (Semántica)	49
4.3.2.	Software de Desarrollo	50
4.3.3.	Edición	51
4.3.4.	Servidores	51
4.3.5.	Lenguajes de Programación	52
4.3.6.	Equipos de desarrollo	52
5.	Resultados	53
5.1.	Fase de Inicio	53
5.1.1.	Iteración 1: Requisitos, Casos de Uso y Planificación	53
5.2.	Fase de Elaboración	61
5.2.1.	Iteración 2: Diseño de la Arquitectura. C _D U 1 Realizar Evaluación para DQD Completeness	61
5.2.2.	Iteración 3: C _D U 2 Realizar Evaluación para DQD Accessibility	71
5.2.3.	Iteración 4: C _D U 3 Realizar evaluación de DQD y C _D U 4 Crear Plan de Evaluación	74
5.3.	Fase de Construcción	76
5.3.1.	Iteración 5: C _D U 9, 10, 11 y 12 sobre APISemDQ	77

5.3.2.	Iteración 6: CDU 5 y 13 Comparar Evaluación de DQD	80
5.3.3.	Iteración 7: CDU 6, 7 y 8 Carga de datos de evaluación	83
5.4.	Fase de Transición	86
5.4.1.	Iteración 8: Creación de LiDQA Tool	86
5.4.2.	Iteración 9: Entrega del PFC	92
6.	Conclusiones y Trabajo Futuro	95
6.1.	Conclusiones	95
6.1.1.	Sobre la consecución de objetivos	95
6.1.2.	Tecnologías utilizadas	98
6.2.	Propuestas de trabajo futuro	98
6.3.	Opinión personal	99
6.3.1.	Elección del PFC	99
6.3.2.	Documentación	100
6.3.3.	Elaboración	100
6.3.4.	Resultados finales y trabajo futuro	100
A.	Linked Data Quality Assessment Tool: Manual de Usuario	105
A.1.	Pantalla principal	105
A.2.	Realizar una evaluación	106
A.2.1.	Evaluación simple	106
A.2.2.	Plan de Evaluación	109
A.3.	Comparación de Modelos	111
A.4.	Gestión de errores	111
A.5.	Contenido	111
A.6.	Acceso a los datos a través de URI y HTTP	112
B.	Manual de Instalación de LiDQA Tool	115
B.1.	Instalación y Configuración de Apache Tomcat 7	115
B.2.	Despliegue de la herramienta	116
B.3.	Cambio de parámetros por defecto	116
C.	Vocabulario <i>Data Quality Assessment</i>	119
D.	Listados de código	125
D.1.	Algoritmo declarativo para la obtención de triplas por niveles	125
D.2.	Clase MeasurementResult	126
D.3.	Test de aceptación JUnit y Selenium	128

Índice de cuadros

3.1. Comparativa de frameworks de Web Semántica. Extraída de [W3Cc]	20
3.2. Categorías y dimensiones de DQ	25
4.1. Casos de uso según su prioridad	39
4.2. Planificación del PUD	40
4.3. Iteración 1	42
4.4. Iteración 2	43
4.5. Iteración 3	43
4.6. Iteración 4	44
4.7. Iteración 5	45
4.8. Iteración 6	45
4.9. Iteración 7	46
4.10. Iteración 8	47
4.11. Iteración 9	49
5.1. Iteración 1	54
5.2. Planificación del PUD	60
5.3. Casos de uso según su prioridad para JenaDQ	61
5.4. Iteración 2	61
5.5. Resultados para Completeness	71
5.6. Iteración 3	72
5.7. Resultados para Completeness	74
5.8. Iteración 4	74
5.9. Iteración 5	77
5.10. Iteración 6	81
5.11. Iteración 7	84
5.12. Planificación de sub-iteraciones para LiDQA Tool	87
6.1. Consecución de Objetivos	95

Índice de figuras

3.1. Esquema de tripla	10
3.2. Grafo basado en triplas. Extraído de [JEN14]	11
3.3. Tecnologías de Web Semántica	12
3.4. Arquitectura de la Web Semántica. Extraído de [San11]	13
3.5. Clasificación de Ontologías. Extraído de [Gua98]	15
3.6. Ejemplo de FOAF. Extraído de [JEN14]	17
3.7. Diagrama de la arquitectura de Jena. Extraída de [JEN14]	19
3.8. Razonamiento en Jena. Extraída de [JEN14]	20
3.9. Esquema LD DBpedia	22
3.10. Niveles de granularidad en la Web Semántica. Extraído de [DFP ⁺ 05]	22
3.11. Bus Gijón App	24
3.12. Bus Gurú App	24
4.1. Proceso Software [RJB]	29
4.2. Ciclo con fases e iteraciones [RJB].	32
4.3. Modelo del PUD extraído de [RJB].	33
4.4. Diferentes fases del PUD [RJB].	34
4.5. Diagrama de Casos de Uso: JenaDQ	39
4.6. Diagrama de Casos de Uso: JenaDQ - APISemDQ	41
4.7. Diagrama de Casos de Uso: LiDQA Tool	48
5.1. Diagrama de Casos de Uso: JenaDQ - APISemDQ	58
5.2. Diagrama de Casos de Uso: JenaDQ	58
5.3. Diagrama de Casos de Uso: LiDQA Tool	59
5.4. Diagrama del patrón <i>Builder</i>	62
5.5. Diagrama del patrón <i>Facade</i>	62
5.6. Diagrama de Clases: Jena DQ	63
5.7. Tipos de reglas de contexto	66
5.8. Diagrama de Interacción: Caso de Uso 1	70

5.9. Diagrama de Interacción: Caso de Uso 2	72
5.10. Diagrama de Interacción: Caso de Uso 10	79
5.11. Caso de Uso 9-11: Gestión de resultados (bottom-top)	80
5.12. Caso de Uso 12 - Escenario 1	80
5.13. Caso de Uso 12 - Escenario 2	81
5.14. Esquema de Comparación de resultados	82
5.15. Caso de Uso 5, 13: Comparación de resultados	83
5.16. Caso de Uso 6: Carga de datos a través de URI	85
5.17. Caso de Uso 7: Carga de datos desde fichero	85
5.18. Caso de Uso 8: Cargar datos de evaluación	86
5.19. Arquitectura de LiDQA Tool	88
5.20. Esquema de formulario para la indicación de datos de evaluación	89
5.21. Esquema de presentación de resultados	90
5.22. Casos de Uso de LiDQA Tool: 1, 2	90
5.23. Casos de Uso de LiDQA Tool: 4, 8	91
5.24. Caso de Uso de LiDQA Tool: 10	91
6.1. Jena integrada con JenaDQ. Inspirado en [JEN14]	97
A.1. Pantalla principal	105
A.2. Evaluación simple (I)	106
A.3. Evaluación simple (II)	107
A.4. Consulta sobre los datos cargados	107
A.5. Validación: HTML5	108
A.6. Validación: contra servidor	108
A.7. Evaluación simple (y III). Resultados.	109
A.8. Plan de evaluación (I)	110
A.9. Plan de evaluación (y II). Resultados	110
A.10. Comparación de resultados	111
A.11. Gestión de errores	112
A.12. Web de contenido	112
A.13. Fuseki: Servidor de triplas de resultados de evaluación	113
A.14. Fuseki: Página de consultas	114
B.1. Interfaz principal de Apache Tomcat	116
B.2. Despliegue de la aplicación	116

Índice de listados

3.1. Ejemplo de FOAF	17
3.2. Consulta SPARQL para identificación de literales perdidos (I)	27
3.3. Consulta SPARQL para identificación de literales perdidos (y II)	27
5.1. Ejemplo de regla de uso	67
5.2. Ejemplo de reglas de valores contextuales	67
5.3. Consulta SPARQL: obtención de nuevas triplas en nivel 1	68
5.4. Pseudocódigo de la evaluación para Completeness	70
5.5. Ejemplo de reglas contextuales para Accessibility	73
5.6. Pseudocódigo de la evaluación	73
5.7. Pseudocódigo del plan de evaluación	76
5.8. Fragmento de la clase fachada APISemDQ: DQAssessment	78
5.9. Fragmento de la clase fachada APISemDQ: DQAssessmentPlan	79
5.10. Pseudocódigo de la comparación entre modelos	82
5.11. Fragmento de la clase fachada APISemDQ: Comparison	83
5.12. Ejemplo de recuperación de modelos a través de URI y <i>endpoint</i>	86
B.1. Edición del archivo de configuración	115
C.1. Vocabulario utilizado para los resultados finales	119
D.1. Algoritmo por consultas sucesivas: triplas por niveles de profundidad	125
D.2. Clase MeasurementResult	126
D.3. Ejemplo de Test: comprobación de resultados correctos	128

Listado de acrónimos

AAA	<i>Anyone can say Anything about Any topic</i>
API	Application Programming Interface
CASE	Computer Aided Software Engineering
CdU	Caso de Uso
CSS	Cascade Style Sheet
CSV	Comma Separated Values
DQ	Calidad de Datos
DQD	Dimensión de Calidad de Datos
DVD	Digital Versatile Disc
DOAP	Description Of A Project
ER	Entidad-Interrelación
FOAF	Friend Of A Friend
GNU	GNU is Not Unix
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
ISO	International Organization for Standarization
JSP	Java Server Pages
LD	Datos Enlazados
LOD	Datos Enlazados Abiertos
OD	Datos Abiertos
OWL	Lenguaje de Ontología Web
PDF	Portable Document Format
PFC	Proyecto Fin de Carrera
PUD	Proceso Unificado de Desarrollo
RDF	Framework de Descripción de Recursos

RF	Requisitos Funcionales
RNF	Requisitos No 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
UML	Unified Modeling Language
URI	Identificador Uniforme de Recurso
URL	Localizador Uniforme de Recurso
W3C	Consortio para la World Wide Web
XML	Lenguaje de Marcas Extensible

Agradecimientos

En primer lugar quiero agradecer a mis padres sus esfuerzos, su paciencia, su cariño y apoyo incondicional. Sin ellos nada hubiese sido posible. Todo lo que tengo y todo lo que soy es gracias a ellos.

Agradezco a mi hermana el haber estado siempre ahí, porque siempre fue mi referente. Supo ayudarme y guiarme en el camino a través de la voz de la experiencia y la razón. Al igual que mis padres, es una parte fundamental en mi vida.

Quiero agradecerle a Ismael el haberme hecho esta propuesta de proyecto de la que día tras día me fui enamorando y por dejarme conocer al profesor, al compañero y al amigo que hay en él.

Finalmente, a todas aquellas personas que de una u otra manera han pasado por mi vida durante esta etapa. Las que aún siguen ahí y las que ya no están. De todas ellas he tenido el privilegio de aprender algo y el honor de formar parte de su historia.

Porque a fin de cuentas, se trata de contar historias.

Raúl Reguillo Carmona

A Papá, a Mamá y a Rebeca.

Capítulo 1

Introducción

1.1 Descripción del problema

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 identificar 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 tecnología semántica en general y Web Semántica en particular.

Considerando la no disponibilidad de operaciones de evaluación de calidad de datos en los frameworks de desarrollo de tecnologías semánticas, el objetivo del Proyecto Fin de Carrera (PFC) consiste en el desarrollo de una extensión para uno de estos frameworks. La finalidad de dicha extensión 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, considerando el contexto en el que los datos deben ser usados.

Tras llevar a cabo un estudio acerca del estado actual de los frameworks de desarrollo de Web Semántica y Linked Data, se ha escogido uno de ellos basado en Java, **Apache Jena**, y se ha extendido con la finalidad de ofrecer estos mecanismos para la evaluación de la calidad de datos semánticos. Jena se está convirtiendo en un referente en la comunidad debido al amplio abanico de tecnologías que implementa, que permite desde el manejo de datos semánticos hasta su almacenamiento y publicación.

El modo en que se ha diseñado dicha extensión facilitaría en un futuro el trabajo de reciclado y expansión del presente trabajo.

1.2 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 PFC 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: Metodología

Se expondrá detalladamente cómo se ha utilizado la metodología de desarrollo elegida para la realización del PFC, que será Proceso Unificado de Desarrollo (PUD). Se incluirá la planificación de las iteraciones y finalmente un resumen de todo el ecosistema tecnológico utilizado para el desarrollo.

Capítulo 5: Resultados

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

Capítulo 6: Conclusiones y Trabajo Futuro

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

Anexo A: Linked Data Quality Assessment Tool: Manual de Usuario

Este anexo se corresponde con el manual de usuario de la aplicación de prueba de concepto, LiDQA Tool.

Anexo B: Manual de Instalación de LiDQA Tool

Este anexo se corresponde con el manual de instalación de la aplicación de prueba de

concepto, LiDQA Tool.

Anexo C: Vocabulario *Data Quality Assessment*

Se corresponde con el listado de código del vocabulario creado para formalizar los resultados de evaluaciones de calidad de datos enlazados.

Anexo D: Listados de código

El último de los anexos muestra algunos fragmentos de código que pueden resultar de interés.

Contenido del DVD adjunto

En el DVD adjunto a este documento se incluyen los siguientes artefactos generados durante la elaboración del PFC:

- Documentación del PFC en formato PDF.
- Código de **JenaDQ** incluido como proyecto para Eclipse.
- Código de **LiDQA Tool** incluido como proyecto para Eclipse.
- Archivo `.war` preparado para despliegue de la herramienta LiDQA Tool: `SemanticAnalytics.war`.
- Vocabulario generado para la descripción de evaluaciones de calidad de datos en el fichero `DQA.owl`
- Vídeo de demostración de uso de LiDQA Tool.
- Documentación generada para la API de JenaDQ en formato HTML.

Capítulo 2

Objetivos

2.1 Objetivo principal

El objetivo de este proyecto fin de carrera (PFC) es la elaboración de una extensión para el framework Jena con el fin de dar soporte a primitivas de evaluación de calidad de datos.

Dicha extensión se generará como una API cuya finalidad será encapsular todas las primitivas desarrolladas y facilitar al usuario la utilización de dicha extensión.

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. 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 extensión del framework elegido, 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 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 (LiDQA Tool)

Elaborar una aplicación de Web Semántica que haga uso de la extensión previamente desarrollada. 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 PFC. La descripción incluye conceptos como Calidad de Datos (DQ), Datos Abiertos (OD), Datos Enlazados Abiertos (LOD) y Web Semántica. 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 *tripas* como una relación (*subjeto*, *predicado*, *objeto*) (véase figura 3.1 y Sección 3.1.2) en la que todos los componentes son URI.

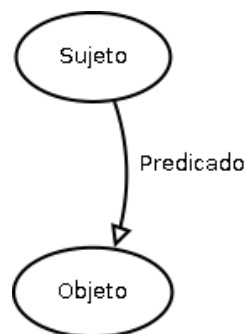


Figura 3.1: Esquema de tripla

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 referirse o derivarse de otras ontologías.

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

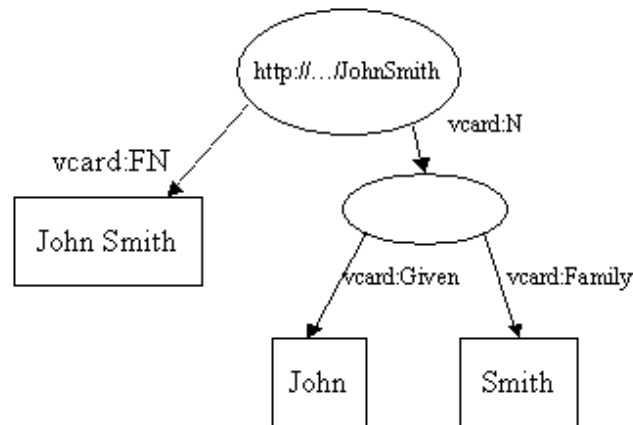


Figura 3.2: Grafo basado en triplas. Extraído de [JEN14]

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 [SPA] 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.

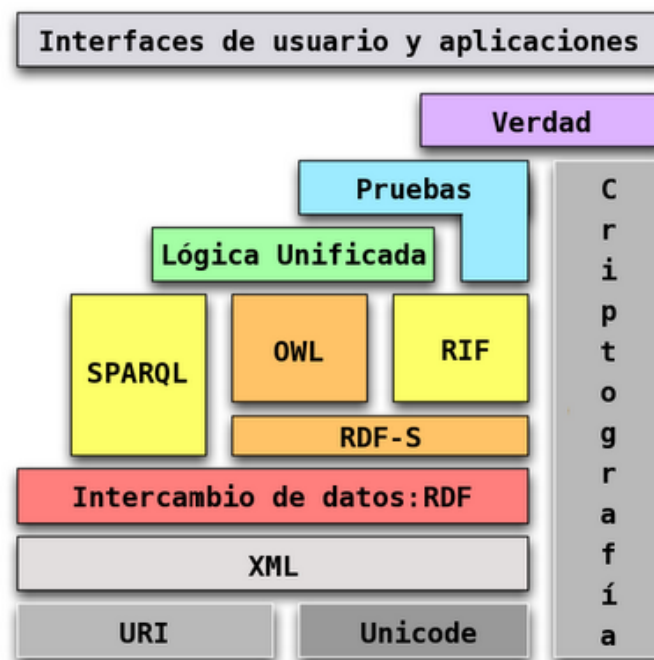


Figura 3.3: Tecnologías de Web Semántica

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.

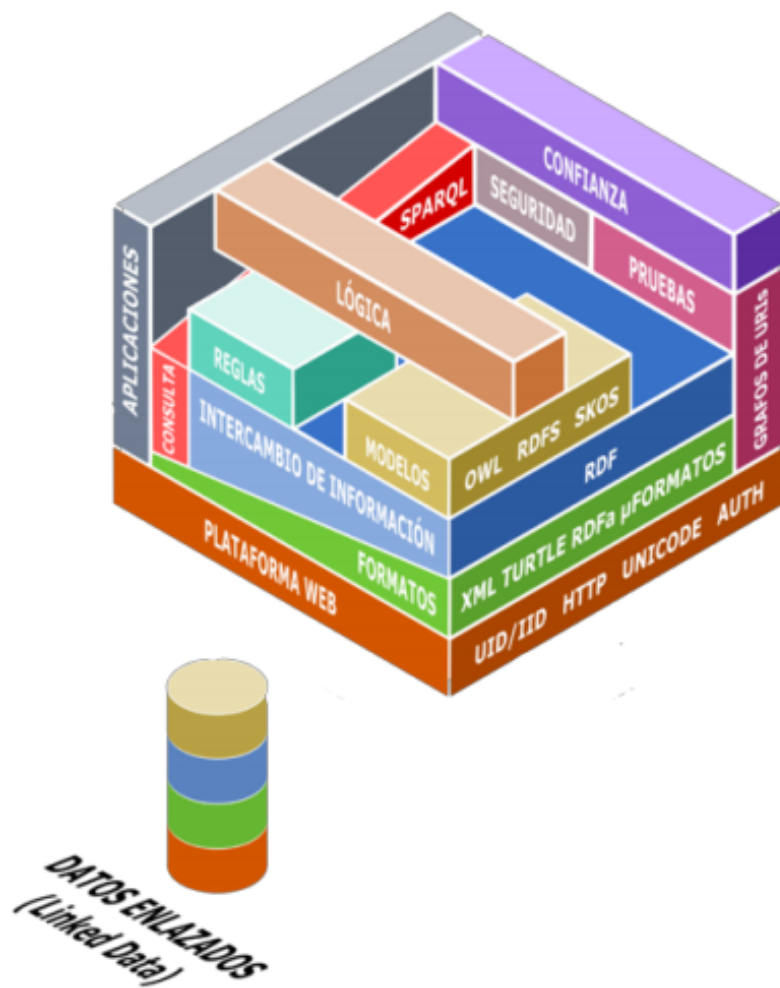


Figura 3.4: Arquitectura de la Web Semántica. Extraído de [San11]

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.

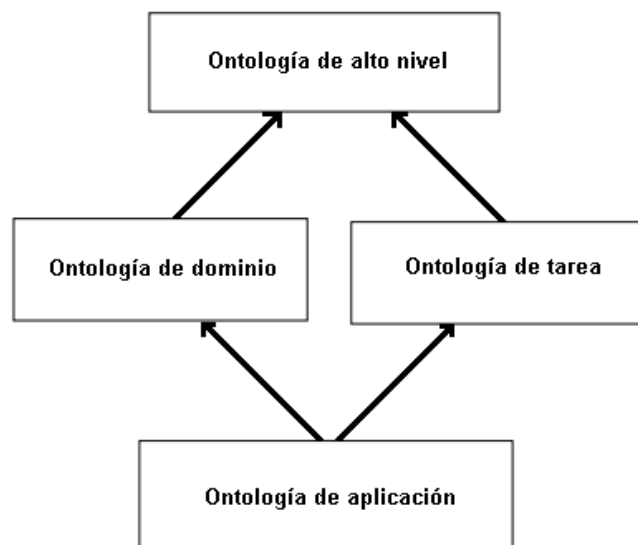


Figura 3.5: Clasificación de Ontologías. Extraído de [Gua98]

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 terminología 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.

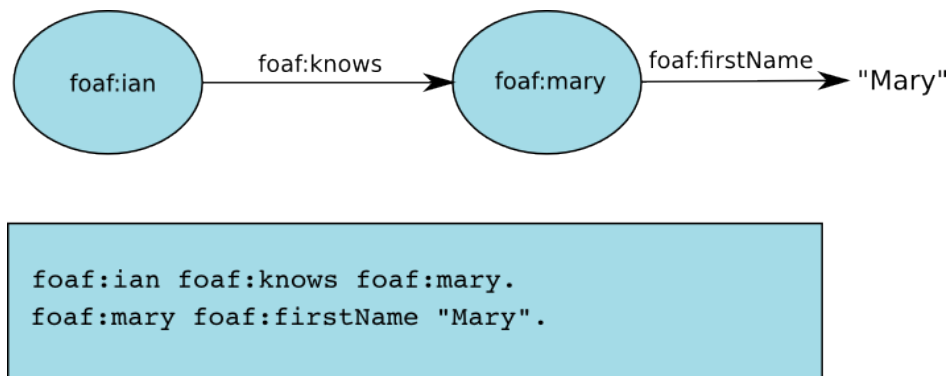


Figura 3.6: Ejemplo de FOAF. Extraído de [JEN14]

```

1 <rdf:RDF
2   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
3   xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'
4   xmlns:foaf='http://xmlns.com/foaf/0.1/'
5
6   <foaf:Person>
7     <foaf:name>Raul Reguillo Carmona</foaf:name>
8     <foaf:title>Mr</foaf:title>
9     <foaf:nick>Radulfr</foaf:nick>
10    <foaf:weblog rdf:resource='http://geeklandtryp.blogspot.com.es' />
11    <foaf:knows>
12      <foaf:Person>
13        <foaf:name>Ismael Caballero</foaf:name>
14      </foaf:Person>
15    </foaf:knows>
16  </foaf:Person>
17 </rdf:RDF>
```

Listado 3.1: Ejemplo de FOAF

3.1.7 Frameworks para el desarrollo de aplicaciones 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.

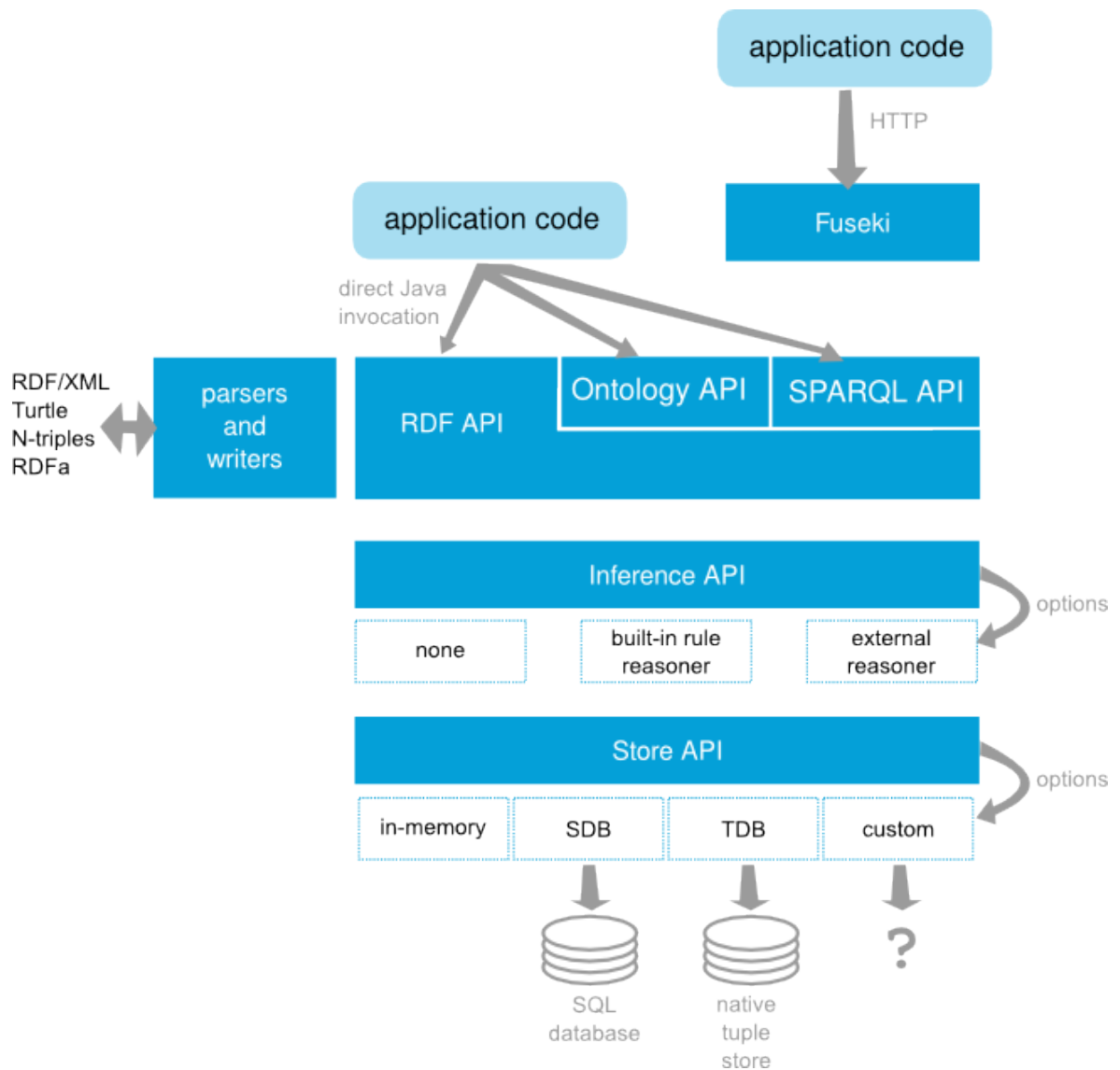


Figura 3.7: Diagrama de la arquitectura de Jena. Extraída de [JEN14]

4. Razonador de reglas genéricas: Se basa en reglas definidas, mediante el en-cadenamiento 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 procesamiento 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.

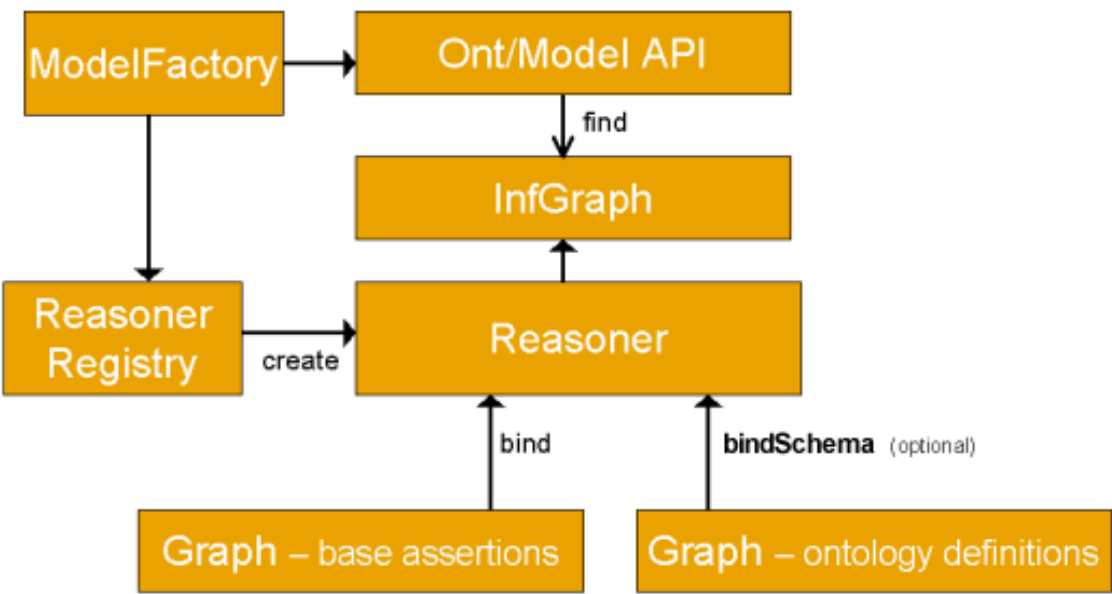


Figura 3.8: Razonamiento en Jena. Extraída de [JEN14]

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

Cuadro 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

- ★ 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 empujados 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 procesamiento** 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

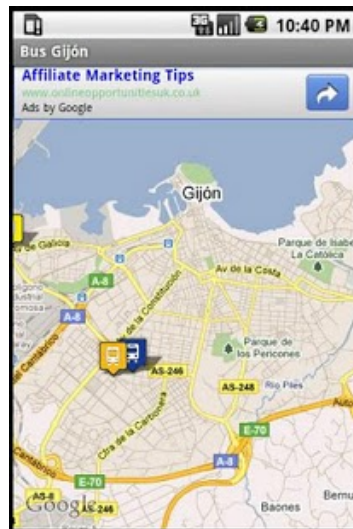


Figura 3.11: Bus Gijón App

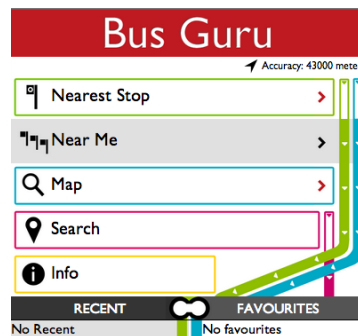


Figura 3.12: Bus Gurú App

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

Cuadro 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 5.2.1), 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 Accessibility

En entornos de Web Semántica y Linked Data tiene especial importancia el hecho de que dichos datos sean de fácil acceso. La palabra *accesibilidad* 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, en la Sección 5.2.2) como una particularización de otra dimensión de calidad bien distinta: *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.

```

1 SELECT ?s
2 WHERE { {
3   ?s a <class1> .
4   ?s <prop1> "" . }
5 UNION{
6   ?s a <class1> .
7   NOT EXISTS {
8     ?s prop1 > ?value}}}

```

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

```

1 SELECT ?s
2 WHERE { {
3   ?s a <class1> .
4   ?s <prop1> <value1>.
5   NOT EXISTS{
6     ?s <prop2> ?value2 .
7   }
8 }UNION{
9   ?s <prop1> <value1> .
10  ?s <prop2> "" .
11 }}

```

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 proprietary 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.

Metodología

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).

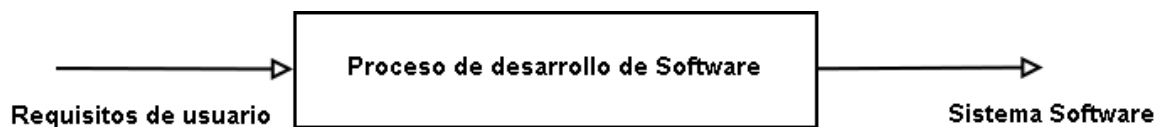


Figura 4.1: Proceso Software [RJB]

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. Trabaja 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.

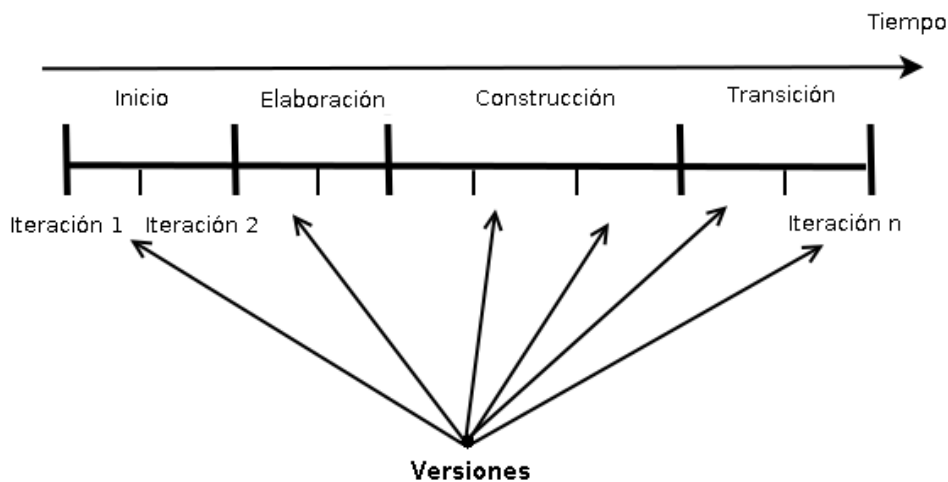


Figura 4.2: Ciclo con fases e iteraciones [RJB].

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).

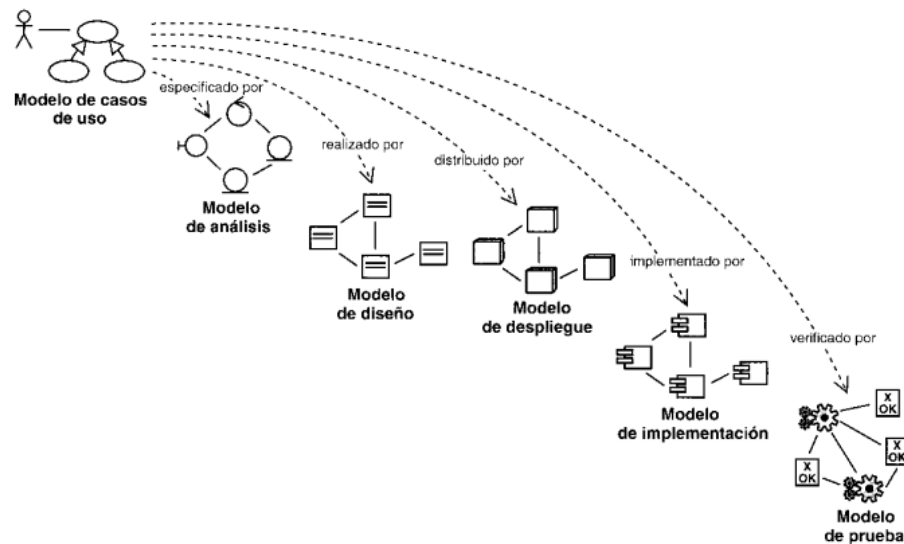


Figura 4.3: Modelo del PUD extraído de [RJB].

- **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.

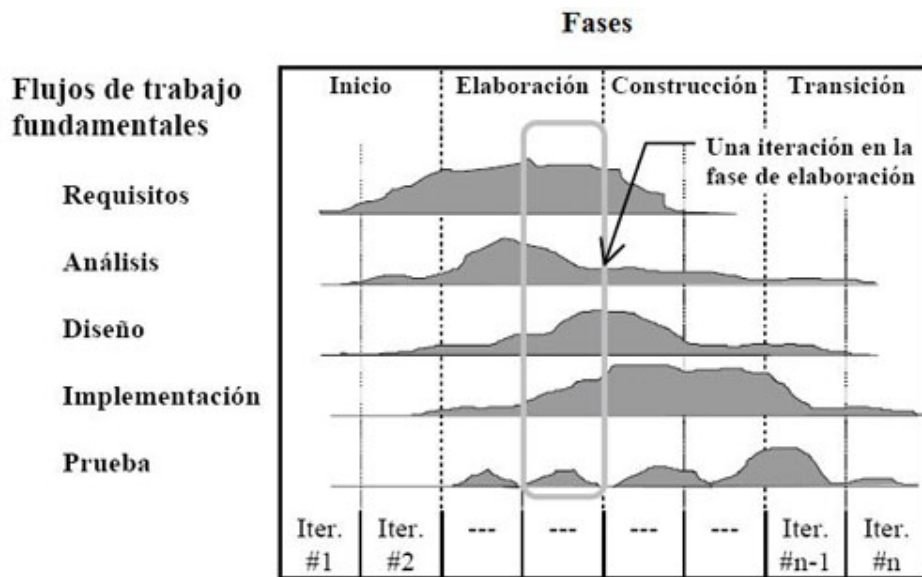


Figura 4.4: Diferentes fases del PUD [RJB].

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 detalle 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 PFC. 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.

JenaDQ: Extensión de Jena para la Evaluación de Calidad de Linked Data

La extensión tendrá por objetivo resolver el problema de ausencia de mecanismos de evaluación de calidad de los datos en frameworks de Web Semántica. Se integrará e interactuará con el framework Jena para ofrecer un conjunto de primitivas de evaluación de calidad de los datos. La extensión implementará una serie de primitivas cuyas responsabilidades serán:

- Generación de modelos (conjuntos de triplas) de calidad de datos (DQModel) a partir de datos enlazados.
- Evaluación de los niveles de calidad de datos (DQAssessment) de un conjunto de datos LOD usando las dimensiones de calidad (*DQDimension*) considerando el contexto de uso. Se han escogido las dimensiones expuestas a continuación por ser comúnmente valoradas en aspectos de medición de calidad de datos, como cita [WS96], y en especial por ser consideradas en aspectos de *Big Data* [Los13]. Las dimensiones a estudiar serán las siguientes:
 - Completeness: en la sección correspondiente del estado del arte (véase Sección 3.3.2) se encuentra la definición de esta dimensión de calidad.
 - Accessibility: en la sección homónima del estado del arte (véase Sección 3.3.3) se ha definido esta dimensión de calidad.
- Generación de planes de evaluación (DQAssessmentPlan) de de calidad de datos de LD para las dimensiones de calidad de datos elegidas y teniendo en cuenta el contexto de uso.
- Generación de resultados (MeasurementResult) de las evaluaciones de calidad de datos y de los planes de evaluación.
- Generar resultados LD de las evaluaciones con el fin de ser publicados.

Posteriormente, con el fin de ilustrar los resultados, se realizará una aplicación de prueba de concepto.

Aplicación Web prueba de concepto - Portal LiDQA Tool

Se va a desarrollar una aplicación de Web Semántica para evaluar la calidad de datos de conjuntos de datos semánticos enlazados. Consistirá en una aplicación Web destinada a ofrecer un servicio de evaluación haciendo uso de JenaDQ realizado como base de este PFC. Permitirá realizar planes de evaluación de calidad (véase definición en el listado de requisitos) de datos enlazados y almacenará los resultados de las evaluaciones, también como datos enlazados para que sean publicados. Facilitará el acceso al *endpoint* con el fin de ofrecer un servicio de consultas sobre las evaluaciones.

4.2.1 Requisitos Funcionales (RF) para JenaDQ

A continuación se exponen los requisitos funcionales identificados en la primera fase del PUD con el fin de formalizar el comportamiento del sistema y establecer el punto de partida para elaborar los casos de uso necesarios. Estos requisitos así como los diagramas de casos de uso y la planificación del PUD corresponde únicamente al desarrollo de **JenaDQ**.

RF 1. JenaDQ debe permitir al usuario el modelado del contexto, aceptando reglas en las evaluaciones

En la Sección 3.1.7 se explica el uso de los razonadores y de las reglas de Jena. Se deben incluir mecanismos de inferencia durante la evaluación de calidad de datos para considerar las distintas reglas que modelen el contexto de uso.

RF 2. JenaDQ debe permitir al usuario modelar la Evaluación de calidad

Como se detallará en la Sección 5.2.1, una evaluación de calidad se llevará a cabo sobre un archivo semántico, en un determinado servicio (*endpoint*), para un conjunto de reglas determinado y sobre un conjunto de dimensiones. A su vez, la evaluación constará de los siguientes pasos:

1. Cálculo de la(s) métrica(s) para cada DQD.
2. Aplicar el razonador junto con la(s) regla(s) de contexto para obtener un valor contextual de dicha métrica.
3. Generar resultados finales.

RF 3. JenaDQ debe permitir al usuario realizar evaluaciones de calidad de LD considerando el contexto, para la DQD *Completeness*

La extensión permitirá realizar una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Completeness* considerando la profundidad en los datos enlazados. Se detallará el algoritmo de la evaluación de esta dimensión en la Sección 5.2.1.

RF 4. JenaDQ debe permitir al usuario realizar evaluaciones de calidad de LD considerando el contexto, para la DQD *Accessibility*

La extensión permitirá realizar una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Accessibility* considerando el grado de enlaces presentado en un archivo semántico. Se detallará el algoritmo de la evaluación de esta dimensión en la Sección 5.2.2.

RF 5. JenaDQ debe permitir al usuario elaborar Planes de Evaluación

Un plan de evaluación será un conjunto de evaluaciones. Su finalidad es la de facilitar al usuario la ejecución de evaluaciones de calidad de datos sobre distintos conjuntos de datos (incluso en distintos servidores de datos) para diversas dimensiones y reglas. Se detallará en la Sección 5.2.3.

RF 6. JenaDQ debe permitir al usuario exportar los resultados de las evaluaciones como LOD

Los resultados de las evaluaciones se generarán durante las evaluaciones y posteriormente se podrán publicar como datos semánticos, aprovechando las funcionalidades ya incluidas en Jena (véase Sección 3.1.7). Se detallarán los resultados en la Sección 5.3.1.

RF 7. JenaDQ debe permitir al usuario recuperar los datos de las evaluaciones

Puesto que los datos se guardarán en un servidor de triplas, deben ser susceptibles de ser recuperados y tratados como un modelo igual a los modelos de datos que sirvieron de objeto de evaluación. La recuperación de los datos tiene que ser posible a través de protocolo HTTP por lo que se debe establecer un *endpoint* como servicio.

RF 8. JenaDQ debe permitir al usuario Realizar comparaciones sobre evaluaciones

Los usuarios podrán, dado un conjunto de evaluaciones, compararlas entre sí con el fin de comprobar que dadas dos evaluaciones sobre los mismos archivos y dimensiones, si los resultados son equivalentes, poder concluir que los contextos modelados a través de las reglas, también lo son. Se detallarán los algoritmos utilizados en la Sección 5.3.2.

Una vez definidos los requisitos funcionales de JenaDQ, se procede a listar los requisitos de la prueba de concepto, **LiDQA Tool**, siendo los funcionales los siguientes:

RF 1. LiDQA Tool debe permitir cargar modelos desde URI y endpoint

LiDQA debe permitir al usuario especificar el conjunto de datos que desea evaluar así como el servicio (endpoint) desde donde se traerán.

RF 2. LiDQA Tool debe permitir cargar la información necesaria para llevar a cabo evaluaciones de calidad de datos

Que incluirá:

- Parámetros de la evaluación a realizar.
- Reglas de contexto.

RF 3. LiDQA Tool debe permitir realizar evaluaciones de calidad de datos para la DQD
Completeness

La aplicación debe permitir la realización de una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Completeness* considerando la profundidad en los datos enlazados. Se detallará el algoritmo en la Sección 5.2.1.

RF 4. LiDQA Tool debe permitir realizar evaluaciones de calidad de datos para la DQD
Accessibility

La aplicación debe permitir la realización de una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Accessibility* considerando la profundidad en los datos enlazados. Se detallará el algoritmo en la Sección 5.2.2.

RF 5. LiDQA Tool debe permitir Mostrar los resultados de las evaluaciones realizadas

Los resultados de las evaluaciones deben ser mostrados tras haber llevado a cabo la evaluación.

RF 6. LiDQA Tool debe permitir realizar planes de evaluación de calidad de datos

Un plan de evaluación será un conjunto de evaluaciones. Su finalidad es la de facilitar al usuario la ejecución de evaluaciones de calidad de datos sobre distintos conjuntos de datos (incluso en distintos servidores de datos) para diversas dimensiones y reglas. Se detallará en la Sección 5.2.3.

RF 7. LiDQA Tool debe permitir descargar los resultados de las evaluaciones y planes de evaluación

Tras haber llevado a cabo la ejecución de la evaluación o del plan de evaluación, LiDQA Tool debe permitir descargar los resultados en forma de fichero.

RF 8. LiDQA Tool debe permitir realizar comparaciones entre los resultados de las evaluaciones

El usuario debe poder cargar dos archivos de resultados con el fin de ejecutar una comparación.

RF 9. Hacer persistentes los resultados

Los resultados de las evaluaciones deben ser almacenados automáticamente y de manera transparente para el usuario.

RF 10. Servidor de datos semánticos

LiDQA debe ofrecer los resultados de las evaluaciones a través de HTTP utilizando tecnología de Jena, en este caso, Jena Fuseki.

4.2.2 Iteraciones del PUD

La planificación del PFC se desarrollará mediante iteraciones, donde cada una de ellas pertenecerá a una o varias fases de PUD. De los componentes relatados anteriormente, la extensión para el framework será desarrollada durante la mayor parte del ciclo, relegando la aplicación Web al final del mismo.

Durante la *Iteración 1* se llevará a cabo un estudio exhaustivo del ámbito del proyecto y se planificará el desarrollo del PFC al completo. Para cada iteración se planteará una serie de objetivos y como consecuencia del trabajo realizado se obtendrá un conjunto de artefactos de salida. Así, cada una de las iteraciones se encontrará ubicada en una fase del PUD.

En las figuras 4.5 y 4.6 se muestran los diagramas de casos de uso utilizados para la elaboración de JenaDQ. En la figura 4.6 se puede contemplar como intermediario un sistema APISemDQ que servirá como interfaz de llamadas al conjunto de Jena y JenaDQ. Se explicará detalladamente en la Sección 5.2.1.

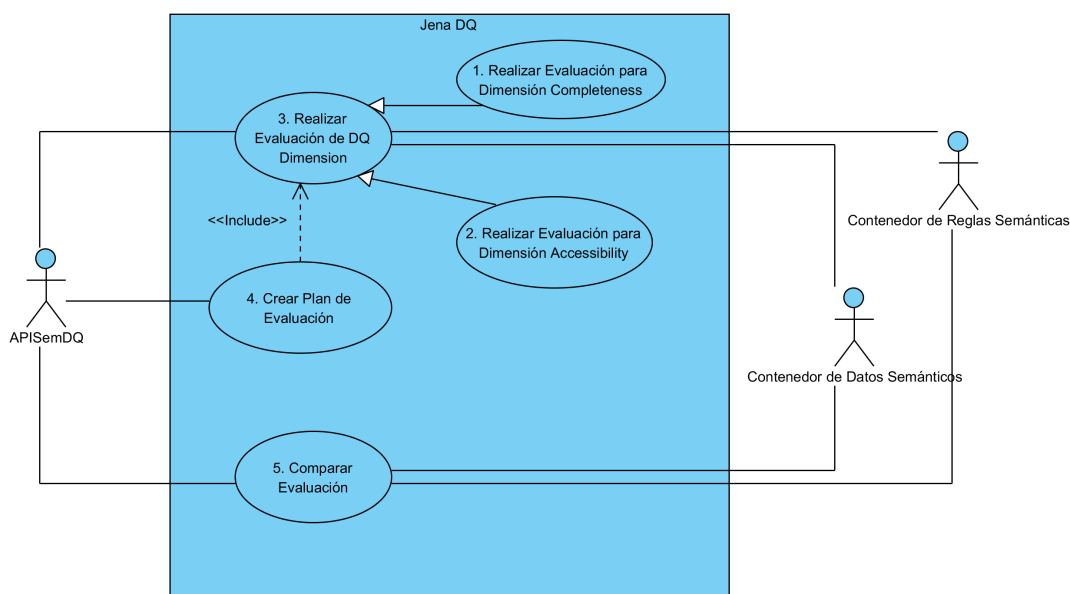


Figura 4.5: Diagrama de Casos de Uso: JenaDQ

Las prioridades asignadas a los casos de uso se pueden consultar en el cuadro 4.1, así como la planificación final queda resumida en el cuadro 4.2.

Prioridad	Caso de Uso
4 (más alta)	CdU1, CdU2, CdU3, CdU4
3	CdU9, CdU10, CdU11, CdU12
2	CdU5, CdU13
1	CdU6, CdU7, CdU8

Cuadro 4.1: Casos de uso según su prioridad

Fase del PUD	Iteraciones	Objetivos
Inicio	Iter. 1	<ul style="list-style-type: none"> ■ Estudiar el Estado del Arte ■ Determinación del alcance del proyecto ■ Establecer Requisitos ■ Realización de diagrama de casos de uso ■ Planificación de las iteraciones ■ Realización del documento “Anteproyecto Fin de Carrera” ■ Planificación del PFC
Elaboracion	Iter. 2	<ul style="list-style-type: none"> ■ Elaborar el diseño de la arquitectura del sistema ■ Establecer interfaz de comunicaciones para JenaDQ ■ Crear vocabulario para Evaluaciones de Calidad de Datos ■ CDU 1
	Iter. 3	<ul style="list-style-type: none"> ■ CDU 2
	Iter. 4	<ul style="list-style-type: none"> ■ CDU 3, 4
Construcción	Iter. 5	<ul style="list-style-type: none"> ■ CDU 9, 10, 11, 12
	Iter. 6	<ul style="list-style-type: none"> ■ CDU 5, 13
	Iter. 7	<ul style="list-style-type: none"> ■ CDU 6, 7, 8
Transición	Iter. 8	<ul style="list-style-type: none"> ■ Desarrollo de LiDQA Tool
	Iter. 9	<ul style="list-style-type: none"> ■ Finalizar memoria del PFC ■ Generar Manual de usuario de la herramienta Web ■ Generar la API para la extensión ■ Redactar Manual de despliegue de la herramienta

Cuadro 4.2: Planificación del PUD

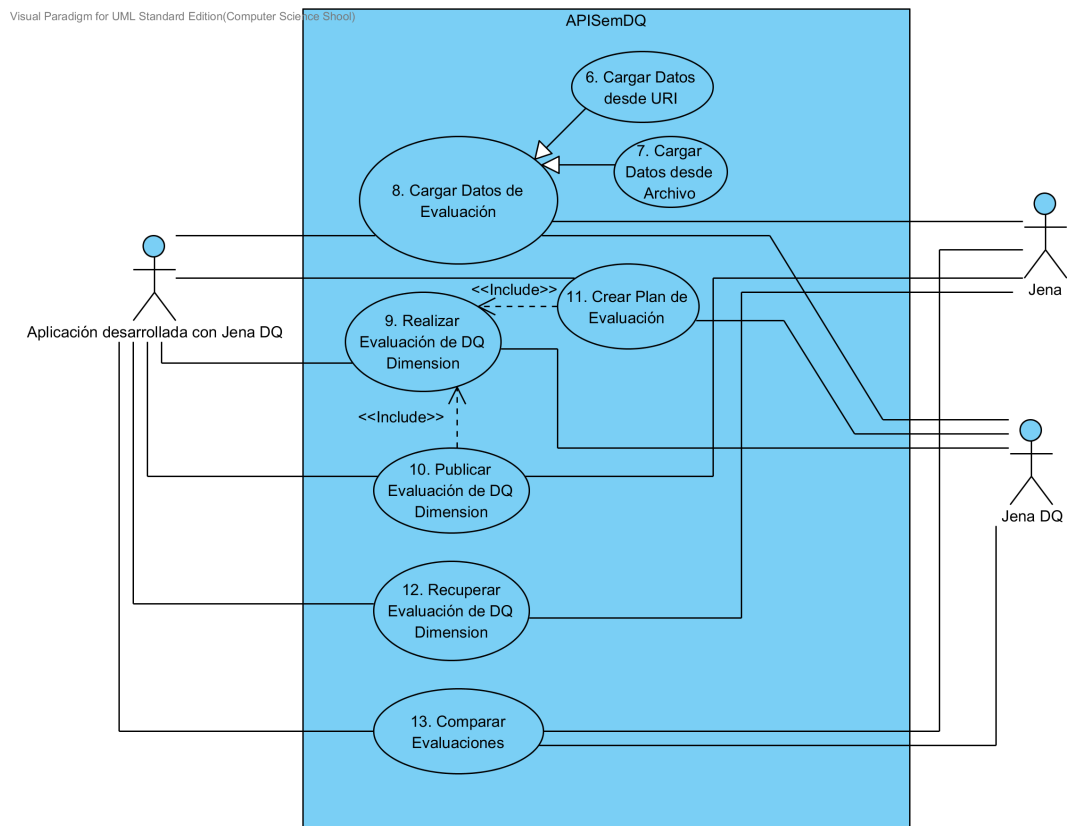


Figura 4.6: Diagrama de Casos de Uso: JenaDQ - APISemDQ

Iteración 1: Planificación, Casos de Uso y Requisitos

En esta iteración se pretende:

- Realizar un estudio detallado del Estado del Arte.
- Determinar el alcance del proyecto y llevar a cabo una planificación de las iteraciones.
- Generar diagrama de casos de uso preliminar.
- Realizar el documento “Anteproyecto Fin de Carrera”.

Es por lo tanto una de las iteraciones más relevantes en este marco de trabajo debido a la dependencia que todas las demás tendrán de ésta. Es en la primera iteración donde se lleva a cabo la realización del diagrama de casos de uso preliminar del sistema software que se pretende desarrollar así como se establecerán las prioridades de los diferentes casos. Este diagrama surgirá tras haber determinado el alcance del proyecto. Posteriormente, se priorizarán los casos de uso y se distribuirán en las distintas iteraciones en función de la prioridad asignada.

Paralelamente se realiza un estudio detallado del estado del arte, lo que englobará los aspectos de calidad de datos y tecnologías semánticas que se han podido revisar en el Capítulo 3.

Finalmente se escribirá el documento “Anteproyecto Fin de Carrera” donde se expondrá una introducción de este PFC así como los objetivos que se persiguen.

Una vez finalizada la iteración, comenzará el desarrollo tecnológico del PFC (véase cuadro 4.3).

Fase del PUD	Inicio
Flujo de trabajo del PUD	Análisis y Diseño
Fechas Inicio - Fin	1/09/2013 - 1/12/2013
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Estudiar el Estado del Arte ■ Determinación del alcance del proyecto ■ Establecer Requisitos ■ Realización de diagramas generales ■ Planificación de las iteraciones ■ Realización del documento “Anteproyecto Fin de Carrera” ■ Planificación del PFC 	<ul style="list-style-type: none"> ■ Revisión del Estado del Arte ■ Especificación de requisitos ■ Diagrama de casos de uso de JenaDQ ■ Documento “Anteproyecto Fin de Carrera” ■ Planificación del PFC

Cuadro 4.3: Iteración 1

Iteración 2: Diseño de la Arquitectura. Caso de Uso 1 Realizar Evaluación para DQD Completeness

En esta iteración se persiguen los siguientes objetivos (véase cuadro 4.4):

- Diseñar la arquitectura de JenaDQ.
- Establecer la interfaz de comunicaciones para JenaDQ.
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 1: Realizar Evaluación para DQD Completeness.

Iteración 3: Caso de Uso 2 Realizar Evaluación para DQD Accessibility

Una vez establecida la arquitectura y la interfaz de comunicaciones de JenaDQ, se procede en las siguientes iteraciones, correspondientes a la fase de Elaboración, a desarrollar los casos de uso más relevantes que conformarán el núcleo de JenaDQ (véase cuadro 4.5).

- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 2: Realizar Evaluación para DQD Accessibility.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/12/2013 - 1/02/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Elaborar el diseño de la arquitectura del sistema ■ Establecer interfaz de programación para JenaDQ ■ Crear vocabulario para Evaluaciones de Calidad de Datos ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 1: Realizar Evaluación para DQD: Completeness 	<ul style="list-style-type: none"> ■ Diseño de la Arquitectura del sistema ■ Interfaz de comunicaciones para JenaDQ ■ Vocabulario DQA para Evaluaciones de Calidad de Datos ■ Diagramas de Clases y de Secuencia, implementación y pruebas C_DU 1 ■ Módulo DQDimension Completeness

Cuadro 4.4: Iteración 2

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/02/2014 - 1/03/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 2: Realizar Evaluación para DQD: Accessibility 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_DU 2 ■ Módulo DQDimension Accessibility

Cuadro 4.5: Iteración 3

Iteración 4: Casos de Uso 3 y 4

Una vez establecidas las dimensiones de calidad, se procede a completar los casos de Uso relativos a las Evaluaciones y Planes de Evaluación que servirán de marco para dichas dimensiones (véase cuadro 4.6).

- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 3: Realizar Evaluación de DQD.
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 3: Crear Plan de Evaluación de DQD.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/03/2014 - 1/04/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 3: Realizar Evaluación de DQD. ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 4: Crear Plan de Evaluación de DQD. 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 3 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 4 ■ Módulos DQDimension, DQAssessment, DQAssessmentPlan, MeasurementResult.

Cuadro 4.6: Iteración 4

Iteración 5: Casos de Uso 9, 10, 11 y 12

Cuando el módulo JenaDQ está completo en lo que se refiere a las evaluaciones y su gestión, se procede a integrarlos en la API y al manejo de sus resultados (véase cuadro 4.7).

- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 9: Realizar evaluación de DQD (APISemDQ)
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 10: Publicar evaluación de DQD
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 11: Crear Plan de evaluación de DQD (APISemDQ)
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 12: Recuperar Evaluación de DQD

Iteración 6: Casos de Uso 5 y 13

Con la parte fundamental del análisis de calidad de datos considerando el contexto ya realizada, a continuación se procede a realizar comparaciones sobre las evaluaciones publicadas (véase cuadro 4.8).

- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 5: Comparar evaluación de DQD
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 13: Comparar Evaluación de DQD (APISemDQ)

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/03/2014 - 1/04/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 9: Realizar evaluación de DQD (APISemDQ) ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 10: Publicar evaluación de DQD ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 11: Crear Plan de evaluación de DQD (APISemDQ) ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 12: Recuperar Evaluación de DQD 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 9 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 10 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 11 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 12

Cuadro 4.7: Iteración 5

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/04/2014 - 20/04/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 5: Comparar evaluación de DQD ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 13: Comparar Evaluación de DQD (APISemDQ) 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 5 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 13

Cuadro 4.8: Iteración 6

Iteración 7: Casos de Uso 6, 7 y 8

Se completan algunos detalles como el acceso a los datos desde diferentes fuentes. Algunas primitivas, incorporadas ya en Jena, hay que refinarlas con el fin de derivar el modelo generado a un DQModel y así realizar las operaciones de calidad sobre éste (véase cuadro 4.9).

- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 6: Carga de datos desde URI y *endpoint*

- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 7: Carga de datos desde fichero
- Análisis, Diseño, Implementación y pruebas para el Caso de Uso 8: Carga de datos de Evaluación

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	20/04/2014 - 15/05/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y pruebas para el CDU 6: Carga de datos desde URI y <i>end-point</i> ■ Análisis, Diseño, Implementación y pruebas para el CDU 7: Carga de datos desde fichero ■ Análisis, Diseño, Implementación y pruebas para el CDU 8: Carga de datos de Evaluación 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 6 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 7 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 8

Cuadro 4.9: Iteración 7

Iteración 8: Desarrollo de LiDQA Tool

Finalmente con todos los módulos de la extensión desarrollados y las extensiones adicionales (Casos de Uso 6, 7, 8) plenamente funcionales, es momento de realizar la aplicación Web. El desarrollo de la aplicación se llevará a cabo siguiendo nuevamente el proceso unificado. Para hacer una distinción entre iteraciones, las iteraciones en el desarrollo de LiDQA Tool se nombrarán como *Partes* numeradas (véase cuadro 4.10).

Para realizar la aplicación, se llevará a cabo el diagrama de casos de uso de la misma, que se puede consultar en la figura 4.7.

La aplicación de prueba de concepto contará con los siguientes componentes:

Módulo de evaluaciones y planes de evaluación

Este módulo se encargará de llevar a cabo el proceso de evaluación, consistente en:

1. **Recolección de datos de evaluación:** controlar la correcta aparición de los parámetros de entrada, su formato y el manejo de excepciones que puedan surgir en este paso.
2. **Ejecución de evaluación:** llevar a cabo la evaluación de calidad de datos según los parámetros de entrada.

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	15/05/2014 - 15/07/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Fase de Inicio (parte I) <ul style="list-style-type: none"> ● Establecer requisitos de la aplicación ● Elaborar diagramas de casos de uso ● Planificar las iteraciones ● Diseñar la arquitectura del sistema ● Diseñar la interfaz de usuario 	<ul style="list-style-type: none"> ■ Requisitos de la aplicación ■ Diagrama de casos de uso ■ Planificación de las iteraciones ■ Diseño de la arquitectura del sistema ■ Diseño de la interfaz de usuario
<ul style="list-style-type: none"> ■ Fase de Elaboración (partes II, III y IV) <ul style="list-style-type: none"> ● (parte II) Casos de uso 1 y 2 ● (parte III) Casos de uso 3, 4 y 5 ● (parte IV) Casos de uso 6, 7 y 8 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 1, 2 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 3, 4 y 5 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 6, 7 y 8
<ul style="list-style-type: none"> ■ Fase de Construcción (partes V y VI) <ul style="list-style-type: none"> ● (parte V) Casos de uso 9 y 10 ● (parte VI) Casos de uso 11 y 12 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 9 y 10 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 11 y 12
<ul style="list-style-type: none"> ■ Fase de Transición (parte VII) <ul style="list-style-type: none"> ● Concretar la interfaz de usuario ● Generación de artefactos finales para la instalación e implantación 	<ul style="list-style-type: none"> ■ Interfaz de usuario mejorada y definitiva ■ Artefactos finales

Cuadro 4.10: Iteración 8

3. **Ofrecer resultados de evaluación al usuario:** ofrecer los resultados de manera amigable y entendible, permitiendo su descarga.

Módulo de almacenamiento de resultados

Al llevar a cabo una evaluación de calidad de datos, almacenar los resultados para su consulta vía HTTP, a través de la publicación en un servidor de triplas accesible como *end-*

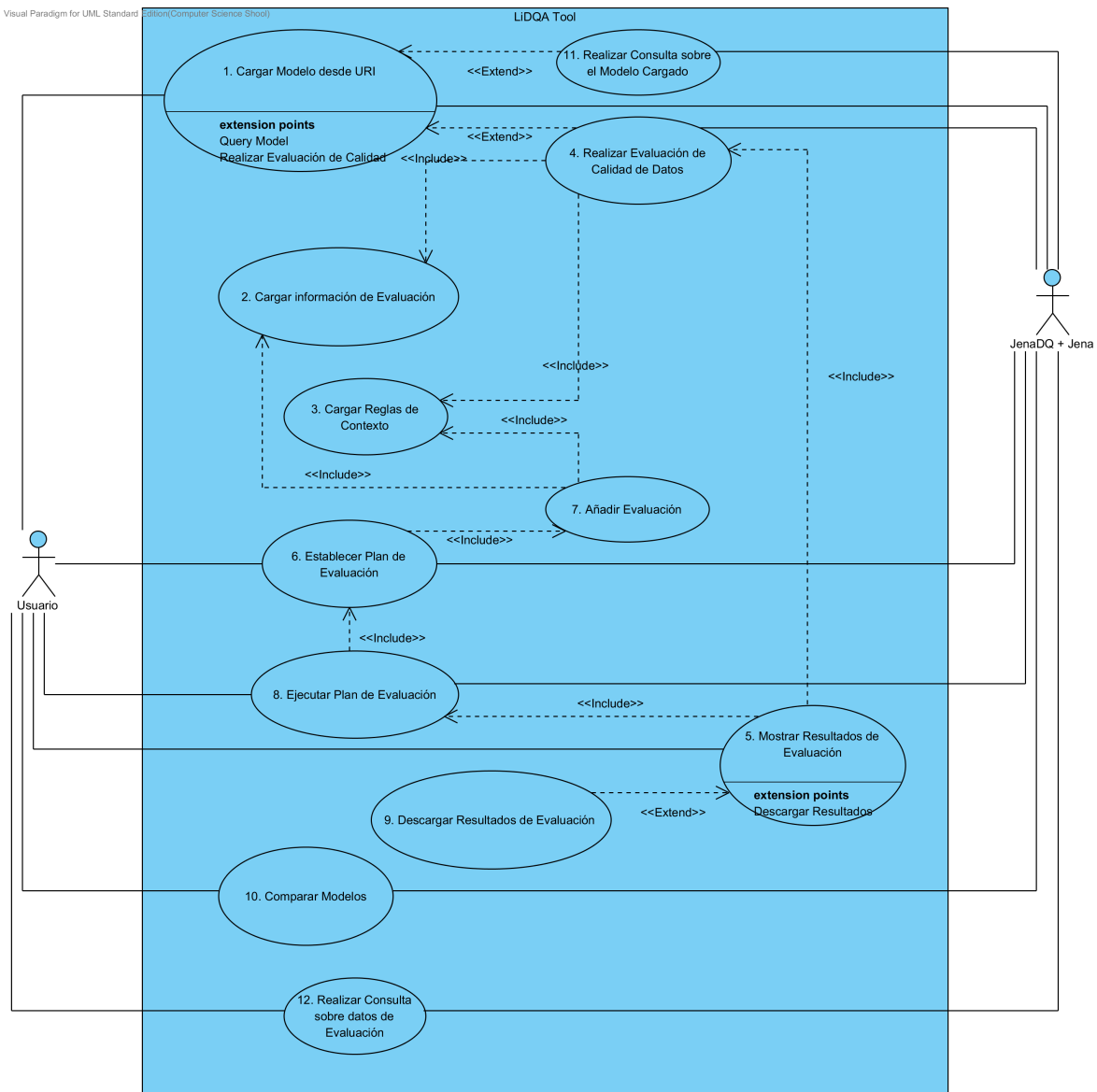


Figura 4.7: Diagrama de Casos de Uso: LiDQA Tool

point

Iteración 9: Entrega del PFC

Una vez realizada la aplicación Web y probada, se finaliza el trabajo redactando y finalizando la documentación pendiente (véase cuadro 4.11).

- Memoria del PFC
- Manual de usuario de LiDQA Tool
- Generación de la API para JenaDQ
- Manual de despliegue de LiDQA Tool

Fase del PUD	Transición
Flujo de trabajo del PUD	Documentación
Fechas Inicio - Fin	15/07/2014 - 15/08/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Finalizar memoria del PFC ■ Generar Manual de usuario de la herramienta Web ■ Generar la API para la extensión ■ Redactar Manual de despliegue de la herramienta 	<ul style="list-style-type: none"> ■ Memoria del PFC ■ Manual de usuario de la herramienta Web ■ Generación de la API para la extensión ■ Manual de despliegue de la herramienta

Cuadro 4.11: Iteración 9

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 PFC. 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 de aplicaciones de Web (Semántica)

Los distintos frameworks utilizados durante la elaboración de este PFC 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.

Jena se erige como la pieza clave dentro de este PFC debido a que la extensión elaborada, JenaDQ, basa su funcionamiento en el amplio abanico de herramientas que implementa Jena.

Para una consulta más detallada, véase la Sección 3.1.7.

Apache Struts 2

Para desarrollar la aplicación Web se ha utilizado este framework. Struts 2 (<http://struts.apache.org/2.x/index.html>) permite el desarrollo de aplicaciones de Web dinámicas utilizando tecnología Java.

4.3.2 Software de Desarrollo

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

Eclipse Juno

Eclipse (<https://www.eclipse.org>) es un Integrated Development Environment (IDE) compuesto por un conjunto de herramientas de código abierto para la programación. Eclipse es además multiplataforma e incluye gran cantidad de plugins para hacer de la labor de programación algo más fácil, principalmente con Java.

La versión utilizada para la realización de este PFC ha sido *Eclipse Juno*.

Selenium

Selenium (<http://www.seleniumhq.org/>) es un entorno de pruebas software para aplicaciones basadas en Web.

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.

Balsamiq Mockups

Balsamiq Mockups (<http://www.balsamic.com>) es una herramienta para la realización de borradores y bocetos de interfaces de usuario para múltiples plataformas y dispositivos.

Mockups se ha utilizado para realizar los prototipos de interfaces de usuario de la aplicación Web.

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.

L^AT_EX

L^AT_EX es 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 L^AT_EX.

Zotero

Zotero (<https://www.zotero.org>) es un programa de código abierto que permite a los usuarios la recolección y gestión de referencias. Facilita la búsqueda de documentos y la exportación a diferentes formatos (como BibTeX).

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 L^AT_EX para 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 rea-

lizar consultas como endpoint a través del protocolo HTTP. Se integra sobre distintos servidores pero principalmente sobre TDB.

4.3.5 Lenguajes de Programación

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

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 PFC se ha trabajado con la versión 1.7.

JavaScript

JavaScript es un lenguaje interpretado, imperativo, débilmente tipado, dinámico y orientado a objetos. Se utiliza principalmente en el lado del cliente en aplicaciones Web.

HyperText Markup Language (HTML)

HTML es un lenguaje interpretado para la elaboración de páginas Web. En este proyecto se ha utilizado algunas anotaciones pertenecientes a la última versión, HTML 5, con el fin de dotar de más dinamismo a la aplicación Web.

Java Server Pages (JSP)

JSP es el lenguaje de programación Web integrado con Java para realizar páginas Web dinámicas.

En el proyecto se ha utilizado junto con HTML para la realización de la aplicación Web.

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.

4.3.6 Equipos de desarrollo

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

- PC con *Microsoft Windows 7 Professional* procesador Intel Centrino doble núcleo de 2.1 GHz y 4 GB de RAM.
- PC con *Debian 7 Wheezy* con procesador Intel Atom doble núcleo de 1.2GHz y 2 GB de RAM.

Resultados

EN este capítulo se presentan los resultados obtenidos tras seguir el plan de trabajo que se ha presentado en el Capítulo 4 para el desarrollo del presente PFC. El proceso de desarrollo ha culminado con la obtención de JenaDQ y LiDQA Tool, que se pretendían desarrollar.

5.1 Fase de Inicio

En la fase de Inicio la única iteración que tiene lugar se destina a llevar a cabo una planificación detallada del esfuerzo, un diagrama general de los casos de uso de JenaDQ y de LiDQA que se van a implementar así como de los requisitos funcionales que deben cumplirse en la elaboración del PFC.

Los cuadros expuestos en el Capítulo 4 se exponen nuevamente por comodidad de la lectura.

5.1.1 Iteración 1: Requisitos, Casos de Uso y Planificación

A continuación se procede a detallar la primera de las iteraciones, resumida en el cuadro 5.1.

El resultado de la iteración ha cumplido con los objetivos propuestos, obteniendo los siguientes artefactos:

- Una revisión del Estado del Arte, que puede ser consultada en el Capítulo 3, que refleja el estado del arte en los principales campos relacionados con este PFC:
 - Calidad de Datos
 - Web Semántica
 - Calidad de Datos en Web Semántica
- Una especificación de requisitos, funcionales y no funcionales, tanto para JenaDQ como para LiDQA Tool, en su iteración correspondiente (véase Sección 5.2.1).
- Modelo general de Casos de Uso de JenaDQ y LiDQA Tool.
- El documento “Anteproyecto Fin de Carrera” [RCCMnR].

Fase del PUD	Inicio
Flujo de trabajo del PUD	Análisis y Diseño
Fechas Inicio - Fin	1/09/2013 - 1/12/2013
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Estudiar el Estado del Arte ■ Determinación del alcance del proyecto ■ Establecer Requisitos ■ Realización de diagramas generales ■ Planificación de las iteraciones ■ Realización del documento “Anteproyecto Fin de Carrera” ■ Planificación del PFC 	<ul style="list-style-type: none"> ■ Revisión del Estado del Arte ■ Especificación de requisitos ■ Diagrama de casos de uso de JenaDQ ■ Documento “Anteproyecto Fin de Carrera” ■ Planificación del PFC

Cuadro 5.1: Iteración 1

- Una planificación detallada del PFC dividida en fases e iteraciones siguiendo PUD.

Requisitos

Los requisitos funcionales del sistema software son aquellos que definen el comportamiento del sistema y establecen el punto de partida para elaborar el modelo de casos de uso.

Los requisitos no funcionales son aquellos orientados a aspectos del sistema que no tienen que ver con el comportamiento, como por ejemplo, el diseño, la implementación o la usabilidad.

A continuación se identifican los Requisitos Funcionales (RF) para **JenaDQ**:

RF 1. JenaDQ debe permitir al usuario el modelado del contexto, aceptando reglas en las evaluaciones

En la Sección 3.1.7 se explica el uso de los razonadores y de las reglas de Jena. Se deben incluir mecanismos de inferencia durante la evaluación de calidad de datos para considerar las distintas reglas que modelen el contexto de uso.

RF 2. JenaDQ debe permitir al usuario modelar la Evaluación de calidad

Como se detallará en la Sección 5.2.1, una evaluación de calidad se llevará a cabo sobre un archivo semántico, en un determinado servicio (*endpoint*), para un conjunto de reglas determinado y sobre un conjunto de dimensiones. A su vez, la evaluación constará de los siguientes pasos:

1. Cálculo de la(s) métrica(s) para cada DQD.

2. Aplicar el razonador junto con la(s) regla(s) de contexto para obtener un valor contextual de dicha métrica.
3. Generar resultados finales.

RF 3. JenaDQ debe permitir al usuario realizar evaluaciones de calidad de LD considerando el contexto, para la DQD *Completeness*

La extensión debe permitir dar soporte a la realización de una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Completeness* considerando la profundidad en los datos enlazados. Se detallará el algoritmo de la evaluación de esta dimensión en la Sección 5.2.1.

RF 4. JenaDQ debe permitir al usuario realizar evaluaciones de calidad de LD considerando el contexto, para la DQD *Accessibility*

La extensión permitirá realizar una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Accessibility* considerando el grado de enlaces presentado en un archivo semántico. Se detallará el algoritmo de la evaluación de esta dimensión en la Sección 5.2.2.

RF 5. JenaDQ debe permitir al usuario elaborar Planes de Evaluación

Un plan de evaluación será un conjunto de evaluaciones. Su finalidad es la de facilitar al usuario la ejecución de evaluaciones de calidad de datos sobre distintos conjuntos de datos (incluso en distintos servidores de datos) para diversas dimensiones y reglas. Se detallará en la Sección 5.2.3.

RF 6. JenaDQ debe permitir al usuario exportar los resultados de las evaluaciones como LOD

Los resultados de las evaluaciones se generarán durante las evaluaciones y posteriormente se podrán publicar como datos semánticos, aprovechando las funcionalidades ya incluidas en Jena (véase 3.1.7). Se detallarán los resultados en la Sección 5.3.1.

RF 7. JenaDQ debe permitir al usuario recuperar los datos de las evaluaciones

Puesto que los datos se guardarán en un servidor de triplas, deben ser susceptibles de ser recuperados y tratados como un modelo igual a los modelos de datos que sirvieron de objeto de evaluación. La recuperación de los datos tiene que ser posible a través de protocolo HTTP por lo que se debe establecer un *endpoint* como servicio.

RF 8. JenaDQ debe permitir al usuario Realizar comparaciones sobre evaluaciones

Los usuarios podrán, dado un conjunto de evaluaciones, compararlas entre sí con el fin de comprobar que dadas dos evaluaciones sobre los mismos archivos y dimensiones, si los resultados son equivalentes, poder concluir que los contextos modelados a través de las reglas, también lo son. Se detallarán los algoritmos utilizados en la Sección 5.3.2.

Seguidamente se exponen los Requisitos No Funcionales (RNF) para **JenaDQ**:

RNF 1. Utilización de un framework para desarrollo de tecnologías semánticas (Apache Jena)

Se utilizará un framework de tecnología semántica para la obtención de modelos semánticos y el tratamiento de los mismos. Este framework será Apache Jena.

RNF 2. Generación de la documentación de la API y ponerla a disposición del usuario

Es imprescindible adjuntar con la extensión una API que permita a los programadores extender y utilizar el módulo JenaDQ.

Una vez definidos los requisitos funcionales y no funcionales de JenaDQ, se procede a listar los requisitos de **LiDQA Tool**, siendo los funcionales los siguientes:

RF 1. LiDQA Tool debe permitir cargar modelos desde URI y endpoint

LiDQA debe permitir al usuario especificar el conjunto de datos que desea evaluar así como el servicio (endpoint) desde donde se traerán.

RF 2. LiDQA Tool debe permitir cargar la información necesaria para llevar a cabo evaluaciones de calidad de datos

Que incluirá:

- Parámetros de la evaluación a realizar.
- Reglas de contexto.

RF 3. LiDQA Tool debe permitir realizar evaluaciones de calidad de datos para la DQD *Completeness*

La aplicación debe permitir la realización de una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Completeness* considerando la profundidad en los datos enlazados. Se detallará el algoritmo en la Sección 5.2.1.

RF 4. LiDQA Tool debe permitir realizar evaluaciones de calidad de datos para la DQD *Accessibility*

La aplicación debe permitir la realización de una evaluación sobre LD considerando el contexto de uso para cada usuario, obteniendo como resultado una métrica y un valor contextual de dicha métrica. Se evaluará la métrica *Accessibility* considerando la profundidad en los datos enlazados. Se detallará el algoritmo en la Sección 5.2.2.

RF 5. LiDQA Tool debe permitir Mostrar los resultados de las evaluaciones realizadas

Los resultados de las evaluaciones deben ser mostrados tras haber llevado a cabo la evaluación.

RF 6. LiDQA Tool debe permitir realizar planes de evaluación de calidad de datos

Un plan de evaluación será un conjunto de evaluaciones. Su finalidad es la de facilitar

al usuario la ejecución de evaluaciones de calidad de datos sobre distintos conjuntos de datos (incluso en distintos servidores de datos) para diversas dimensiones y reglas. Se detallará en la Sección 5.2.3.

RF 7. LiDQA Tool debe permitir descargar los resultados de las evaluaciones y planes de evaluación

Tras haber llevado a cabo la ejecución de la evaluación o del plan de evaluación, LiDQA Tool debe permitir descargar los resultados en forma de fichero.

RF 8. LiDQA Tool debe permitir realizar comparaciones entre los resultados de las evaluaciones

El usuario debe poder cargar dos archivos de resultados con el fin de ejecutar una comparación.

RF 9. LiDQA Tool debe hacer persistentes los resultados

Los resultados de las evaluaciones deben ser almacenados automáticamente y de manera transparente para el usuario.

RF 10. LiDQA Tool debe ofrecer un servidor de datos semánticos

LiDQA ofrecerá los resultados de las evaluaciones a través de HTTP utilizando tecnología de Jena, en este caso, Jena Fuseki.

Finalmente se enumeran los requisitos no funcionales de **LiDQA Tool**:

RF 1. Arquitectura Web

Se debe realizar una aplicación Web como herramienta que utilice la tecnología desarrollada y ofrecerse como servicio.

RF 2. Utilización de framework para desarrollo de la aplicación Web (Struts 2)

LiDQA Se desarrollará tomando como base Struts 2.

RNF 3. Tecnologías del lado del cliente

Con el fin de asegurar la una interacción amigable con la herramienta, se deben utilizar tecnologías que faciliten al usuario el manejo de la aplicación, tales como HTML5, CSS3 o JavaScript.

Modelo General de Casos de Uso

Partiendo de los requisitos funcionales de JenaDQ y de LiDQA Tool, se ha realizado un modelo de casos de uso que describe el funcionamiento del sistema software que se pretende implementar. Los modelos de casos de uso están divididos en dos secciones:

1. En primer lugar, los modelos de casos de uso de JenaDQ partiendo de sus requisitos funcionales. Véanse figuras 5.1 y 5.2.
2. Para concluir, el modelo de casos de uso de LiDQA Tool a desarrollar, a partir de sus requisitos funcionales. Véase figura 5.3.

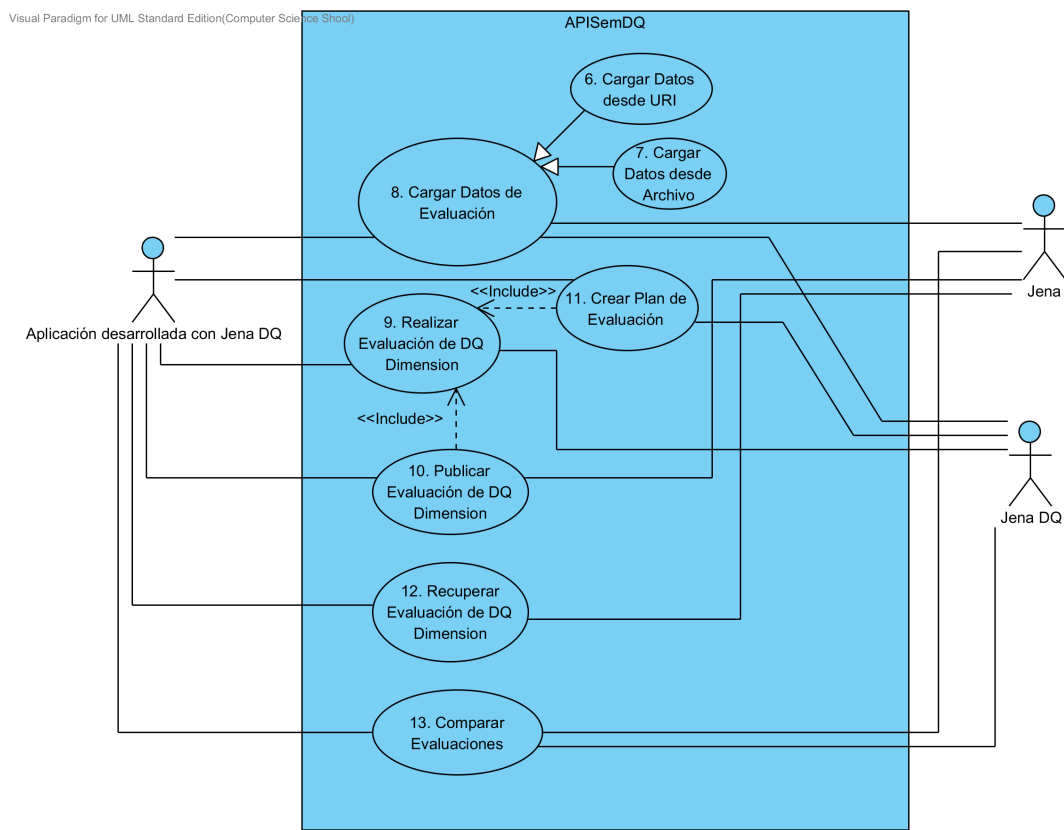


Figura 5.1: Diagrama de Casos de Uso: JenaDQ - APISemDQ

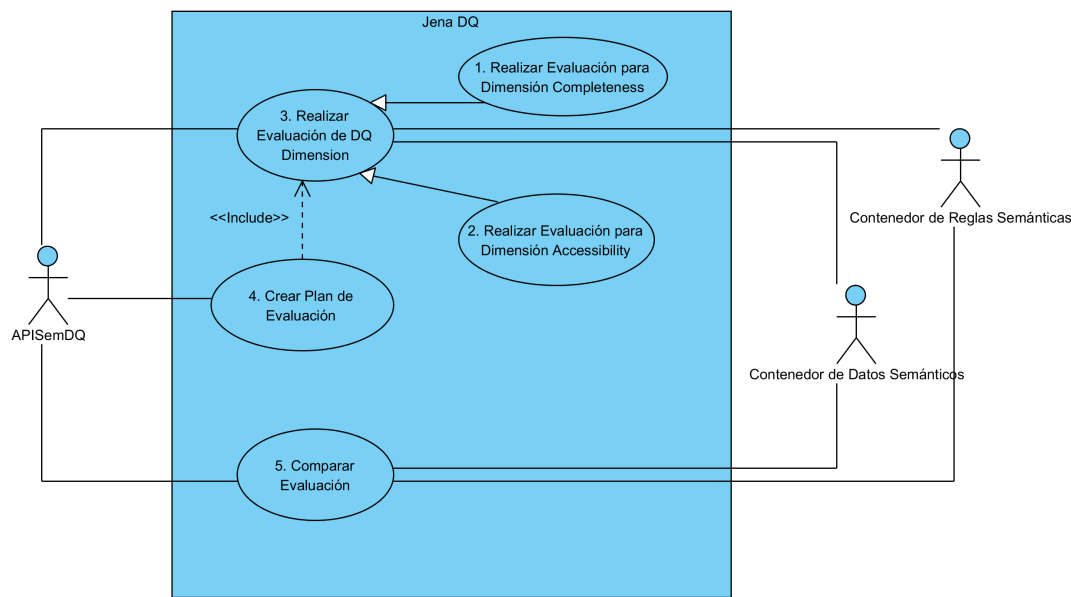


Figura 5.2: Diagrama de Casos de Uso: JenaDQ

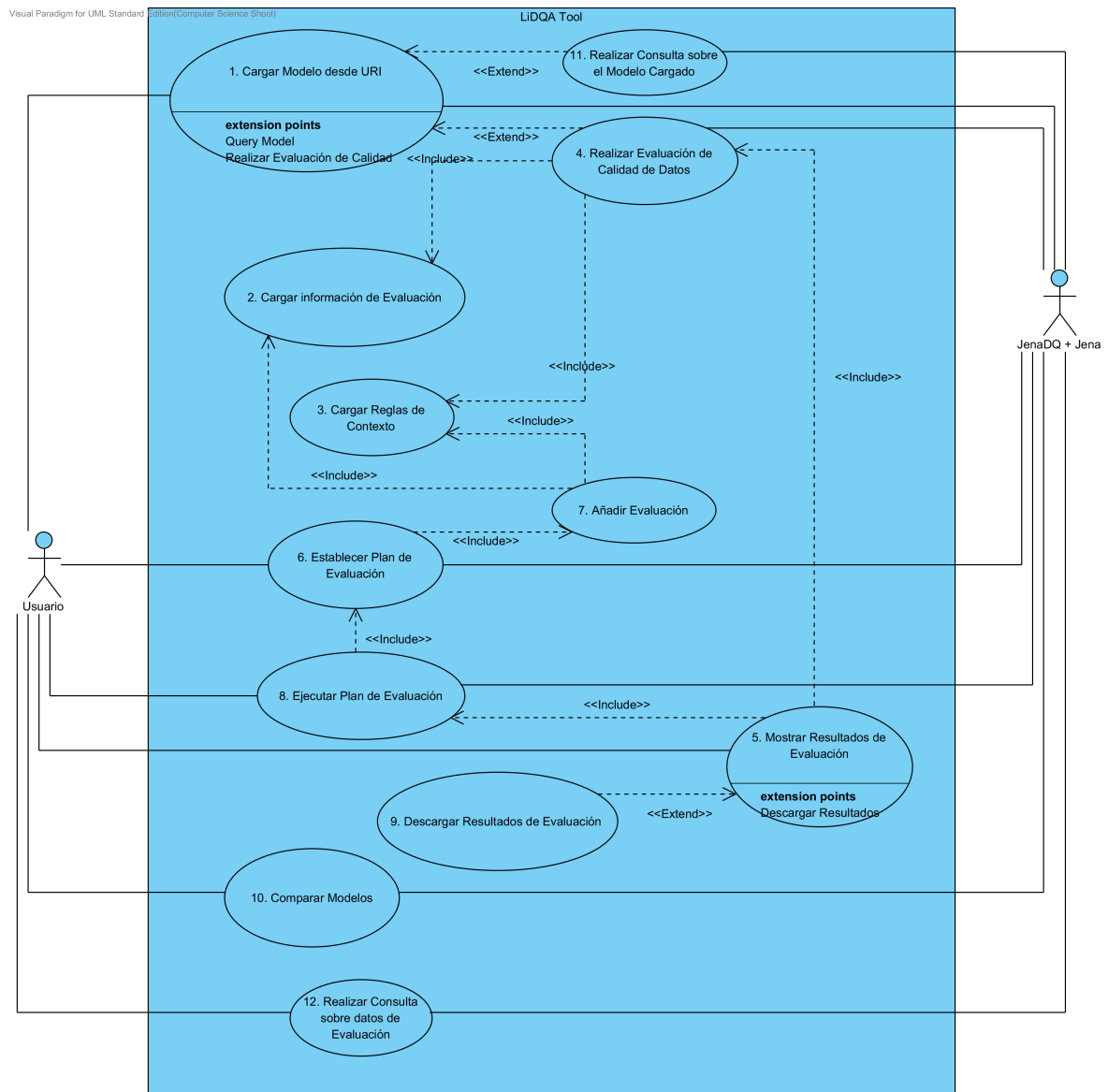


Figura 5.3: Diagrama de Casos de Uso: LiDQA Tool

Planificación

Para llevar a cabo una correcta planificación, primero se han priorizado los casos de uso para JenaDQ según aparece en el cuadro 5.3. Se han asignado prioridades considerando aquellos casos de uso que resultan más importantes para el desarrollo ordenado del proyecto. Así, se comienza con los que tienen que ver con la generación de los datos de las métricas concretamente para después prestar atención a aquellos casos de uso que gestionan los resultados obtenidos y generan los documentos finales.

Se ha planificado el PFC siguiendo el PUD tal y como se indicó en el Capítulo 4, obteniendo como resultado una plan de desarrollo resumido en el cuadro 5.2.

Fase del PUD	Iteraciones	Objetivos
Inicio	Iter. 1	<ul style="list-style-type: none"> ■ Estudiar el Estado del Arte ■ Determinación del alcance del proyecto ■ Establecer Requisitos ■ Realización de diagrama de casos de uso ■ Planificación de las iteraciones ■ Realización del documento “Anteproyecto Fin de Carrera” ■ Planificación del PFC
Elaboracion	Iter. 2	<ul style="list-style-type: none"> ■ Elaborar el diseño de la arquitectura del sistema ■ Establecer interfaz de comunicaciones para JenaDQ ■ Crear vocabulario para Evaluaciones de Calidad de Datos ■ CDU 1
	Iter. 3	<ul style="list-style-type: none"> ■ CDU 2
	Iter. 4	<ul style="list-style-type: none"> ■ CDU 3, 4
Construcción	Iter. 5	<ul style="list-style-type: none"> ■ CDU 13, 14, 15, 16
	Iter. 6	<ul style="list-style-type: none"> ■ CDU 5, 17
	Iter. 7	<ul style="list-style-type: none"> ■ CDU 6, 7, 8
Transición	Iter. 8	<ul style="list-style-type: none"> ■ Desarrollo de LiDQA Tool
	Iter. 9	<ul style="list-style-type: none"> ■ Finalizar memoria del PFC ■ Generar Manual de usuario de la herramienta Web ■ Generar la API para la extensión ■ Redactar Manual de despliegue de la herramienta

Cuadro 5.2: Planificación del PUD

Prioridad	Caso de Uso
4 (más alta)	CdU1, CdU2, CdU3, CdU4
3	CdU9, CdU10, CdU11, CdU12
2	CdU5, CdU13
1	CdU6, CdU7, CdU8

Cuadro 5.3: Casos de uso según su prioridad para JenaDQ

5.2 Fase de Elaboración

En la fase de Elaboración, tiene lugar el desarrollo del núcleo de la extensión de Jena, **JenaDQ**, objetivo de este PFC.

5.2.1 Iteración 2: Diseño de la Arquitectura. CdU 1 Realizar Evaluación para DQD Completeness

La Iteración 2 se resume en el cuadro 5.4. Este cuadro se incluyó en el Capítulo 4 y es reproducido aquí por conveniencia. En esta iteración se persigue diseñar la arquitectura, establecer una interfaz de comunicaciones para JenaDQ y llevar a cabo el proceso de desarrollo al completo para el primero de los casos de uso.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/12/2013 - 1/02/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Elaborar el diseño de la arquitectura del sistema ■ Establecer interfaz de programación para JenaDQ ■ Crear vocabulario para Evaluaciones de Calidad de Datos ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 1: Realizar Evaluación para DQD: Completeness 	<ul style="list-style-type: none"> ■ Diseño de la Arquitectura del sistema ■ Interfaz de comunicaciones para JenaDQ ■ Vocabulario DQA para Evaluaciones de Calidad de Datos ■ Diagramas de Clases y de Secuencia, implementación y pruebas CdU 1 ■ Módulo DQDimension Completeness

Cuadro 5.4: Iteración 1

Arquitectura

Puesto que JenaDQ es una extensión de Jena, hereda la arquitectura que está extendiendo. La arquitectura de Jena se puede consultar en la Sección 3.1.7.

Para incluir JenaDQ en el framework original, se ha hecho uso de dos patrones de diseño.

Ambos se pueden consultar en [GHJV96]:

1. **Patrón Builder:** el patrón de creación *builder*, permite la creación de variedades de objetos complejos desde un objeto fuente. Durante el diseño se ha utilizado el patrón builder para la gestión de los métodos de evaluación de las dos dimensiones de calidad existentes en JenaDQ. Se puede consultar una ilustración de este patrón en la figura 5.4. Concretamente se ha aplicado sobre las clases *DQDimension*, *_dimAccessibility* y *_dimCompleteness* (véase figura 5.6), para la ejecución de las medidas y generación de resultados.
2. **Patrón Facade:** el patrón fachada es de tipo estructural y se aplica cuando existe una necesidad de estructurar un entorno para reducir su complejidad, debido a la cantidad de subsistemas existentes. De manera que pueden reducirse las comunicaciones y dependencias entre componentes. Se puede ver una ilustración de este patrón en la figura 5.5.

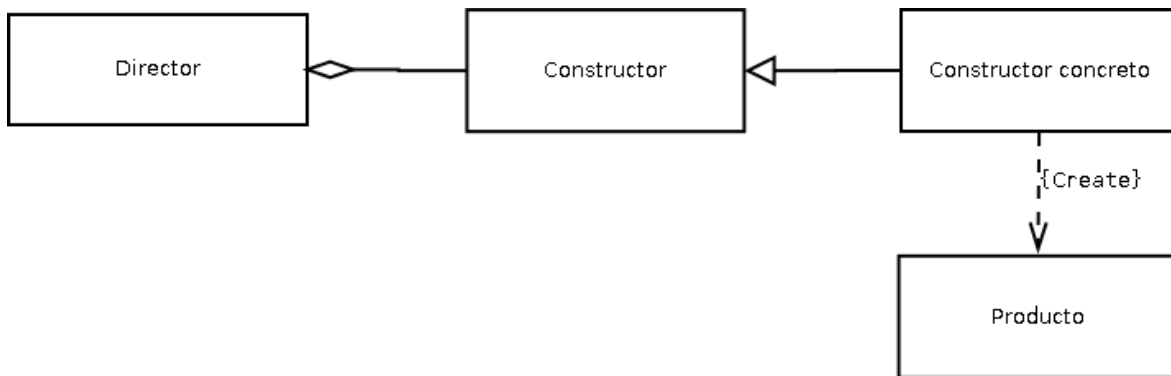


Figura 5.4: Diagrama del patrón *Builder*

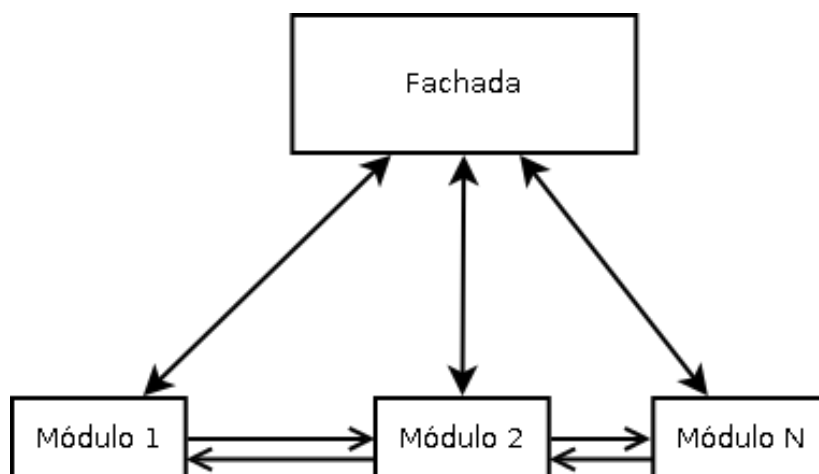


Figura 5.5: Diagrama del patrón *Facade*

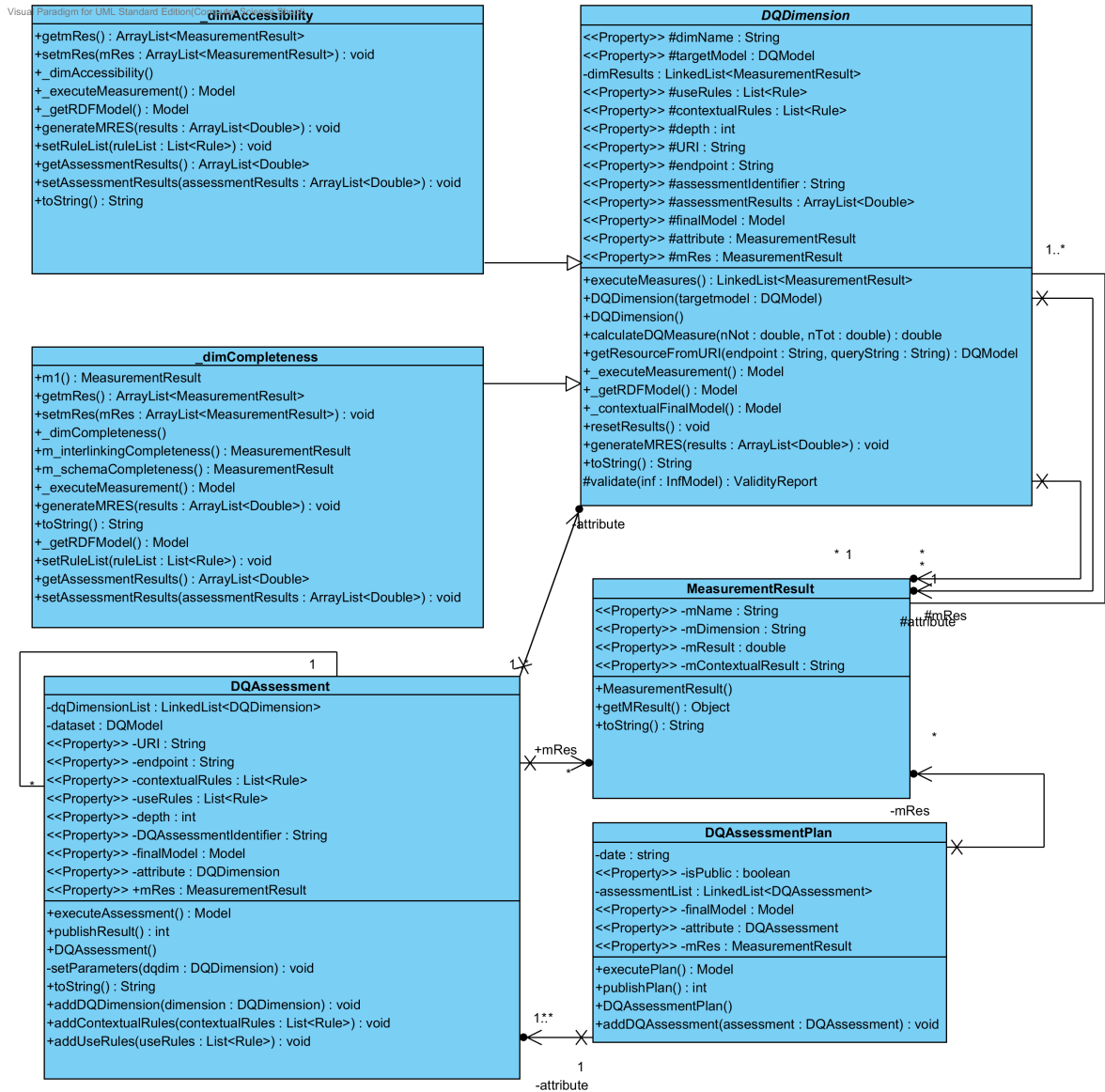


Figura 5.6: Diagrama de Clases: Jena DQ

En la figura 5.6 se expone el diagrama de clases desarrollado para la parte central del proyecto. Consta de lo siguiente:

- Clase **DQDimension**: de la que heredarán todas las DQD que se incluyan en el proyecto actual o en los sucesivos proyectos. Deberán sobrescribirse al menos las siguientes funciones:
 - `executeMeasurement()`: destinada a llevar a cabo la ejecución de la medición para esa dimensión en concreto, devolviendo un modelo semántico de resultados.
 - `_getRDFModel()`: destinada a generar los resultados en formato LOD y mediante un objeto `MeasurementResult`.
- Clases **_dimDQDName**: que particularizará la anterior para las medidas que deba realizar

así como la formación de los resultados.

- Clase `MeasurementResult`: clase de apoyo que guardará los resultados de las mediciones así como los contextuales con el fin de que éstos sean más fácilmente accesibles para el programador, sin necesidad de llevar a cabo consultas sobre el modelo semántico generado.
- Clase `DQAssessment`: esta clase definirá una evaluación, para lo cual deberá incorporar los siguientes datos:
 - **URI**: la URI objetivo de medición.
 - ***endpoint***: dirección del servicio donde se encontrarán alojados los datos.
 - ***contextualValuesRule List***: lista de reglas para llevar a cabo la evaluación contextual sobre las métricas.
 - ***useRule List***: lista de reglas de uso. Necesarias para algunas DQD. En este caso, las reglas de uso fueron necesarias para la dimensión *Completeness*.
 - ***depth***: profundidad a la que se llegará expandiendo nodos enlazados.
 - **Identificador**: un identificador para la evaluación.
- Clase `DQAssessmentPlan`: que definirá un plan de evaluación como un conjunto de evaluaciones. El resultado final de la ejecución de un plan será un único modelo englobando todas las evaluaciones.

Interfaz

La interfaz de comunicaciones para JenaDQ quedará definida por el patrón fachada que implementará las funciones necesarias para el acceso a la API.

APISemDQ

El patrón fachada se implementa a través de la clase `APISemDQ`, donde se delegan las responsabilidades de las llamadas a las diferentes clases del sistema. Consta de las siguientes funciones listadas a continuación. Se detallará en secciones posteriores correspondientes a su desarrollo en el ámbito del PUD:

- `createAssessmentPlan`
- `executeAssessmentPlan`
- `addDQAssessmentToPlan`
- `createDQAssessment`
- `executeDQAssessment`
- `executeDQAssessmentGetModel`
- `executeDQAssessmentGetMeasurementResult`

- addDQDimensionToAssessment
- createDQDimensionCompleteness
- createDQDimensionAccessibility
- createDQmodel
- modelComparison

DQModel

También se implementa el patrón fachada en esta clase, como fachada intermedia, para facilitar las comunicaciones con Jena en lo que se refiere a los siguientes aspectos:

- Acceso y recuperación de los datos mediante URI y endpoint.
- Acceso y recuperación de los datos mediante fichero.

Vocabulario DQA para la Evaluación de Calidad de Datos

Se ha desarrollado un vocabulario para publicar las evaluaciones. Para ello se ha utilizado la herramienta *Protégé*. El vocabulario puede ser consultado en el Anexo C.

Caso de Uso 1: Realizar Evaluación para DQD: Completeness

Análisis

Tras haber estudiado el estado del arte (véase Sección 3.3.2), se llevó a cabo el análisis con el fin de encontrar una única métrica que reflejase esta dimensión. Se obtuvo como resultado la siguiente definición adoptada en este proyecto para la DQD Completeness: *dado un nivel de profundidad n en el grafo del conjunto de datos enlazados, se define completeness como el grado en el que para un determinado conjunto de propiedades existen valores que las describan, para cada nivel comprendido entre 0 (conjunto de datos origen) y n (n -ésima expansión de los enlaces, evitando ciclos) .*

De esta manera, si P es un conjunto de propiedades que deben poseer valor y n es el nivel de profundidad hasta donde interesa evaluar esta dimensión, se obtendrán los siguientes resultados:

- *Nivel 0 del conjunto de datos original:* el valor de la medida Completeness en este punto será siempre dicotómico: $\{0, 1\}$, puesto que en el momento en que una de esas propiedades se encuentre sin valor, condicionará el resultado a cero.
- *Niveles 1 - N :* el valor oscilará entre $[0,1]$, obteniendo así una métrica capa por capa según la profundidad en la que se esté evaluando.

La medida vendrá determinada por la ecuación 5.1:

$$DQ_{Measure} = 1 - \frac{NumberOfDataUnitsNotSatisfyingADQCriterion}{TotalNumberOfDataUnits} \quad (5.1)$$

Siendo *NumberOfDataUnitsNotSatisfyingADQCriterion* el número total de URIs en un mismo nivel, que al ser desreferenciadas y obteniendo de ellas el modelo que describen, no cumplen con la propiedad indicada en la regla. Del mismo modo, *TotalNumberOfDataUnits* se referirá al total de de las URIs en ese nivel.

Por otra parte, la definición de **reglas de contexto** se definirán mediante una serie de predicados declarativos que permitirán realizar operaciones de inferencia sobre los modelos de Jena (véase Sección 3.1.7). Se contemplan en este proyecto dos tipos de reglas de contexto (véase figura 5.7):

- **Reglas de uso:** que indicarán para qué predicados o relaciones dentro de un modelo se esperan valores no nulos. Puede verse un ejemplo en el listado 5.1. En este caso se puede comprobar la inclusión en el modelo de una nueva tripla sobre el sujeto que cumpla la tripla: no exista valor para la propiedad `rdf:type`.
- **Reglas de valores contextuales:** una vez calculado los valores de las medidas, establecerán unos niveles de calidad en base a éstos. Se puede consultar un ejemplo en el listado 5.2. En el ejemplo se puede comprobar la adición de una nueva tripla sobre el sujeto `?x` del resultado que, tras haber calculado su medida, se compruebe que ésta sea mayor o igual que el valor indicado (`ge(?m, 0.5)`). La nueva tripla indicará que el sujeto tiene un valor contextual alto en ese caso.

Durante la implementación se generan reglas para cubrir todo el intervalo $[0,1]$, delegando en el usuario la asignación de los valores para que defina así su contexto.

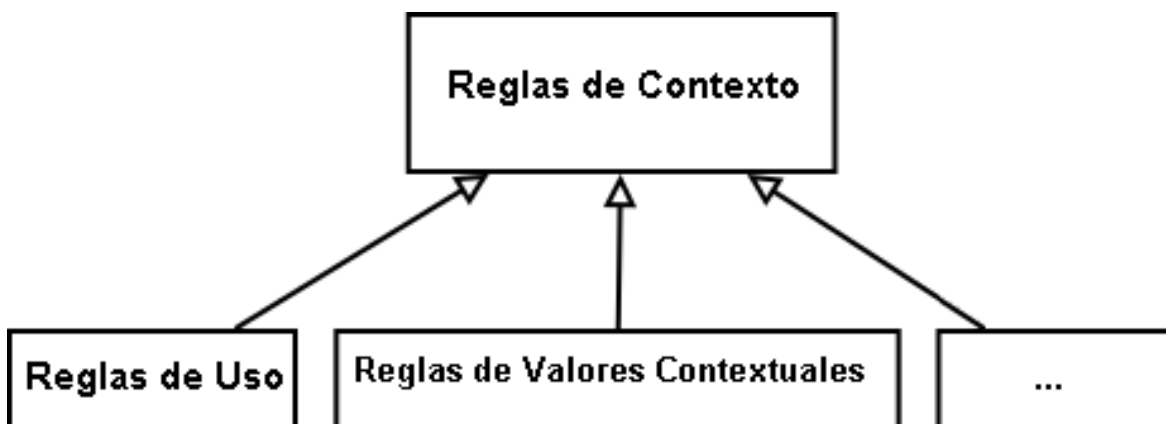


Figura 5.7: Tipos de reglas de contexto


```

1 @prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
2 @prefix dqa: http://www.example/ontology#
3 [rule: noValue(?x rdf:type ?y) -> (?x rdf:type dqa:NoCompleteness) ]

```

Listado 5.1: Ejemplo de regla de uso

```

1 @prefix dqa: http://www.example/ontology#
2 [ruleLOW: (?x dqa:CompletenessMeasure ?m) lessThan(?m, 0.5 ) -> (?x
3   dqa:ContextualValue dqa:LOW) ]
4 [ruleHIGH: (?x dqa:CompletenessMeasure ?m) ge(?m, 0.5 ) -> (?x
5   dqa:ContextualValue dqa:HIG) ]

```

Listado 5.2: Ejemplo de reglas de valores contextuales

Finalmente, el **razonador** escogido para llevar a cabo la inferencia en base a las reglas ha sido el *Generic Reasoner* de Jena (véase Sección 3.1.7).

Diseño

La fase de diseño ha consistido en la elaboración de los algoritmos necesarios para llevar a cabo los cálculos. En esta etapa se han desarrollado dos algoritmos para alcanzar este fin: uno iterativo y otro declarativo, que se exponen a continuación.

- El **algoritmo iterativo** consiste en la expansión en anchura de los recursos tipo URI que se encuentren en el modelo. De esta manera, se genera una estructura de datos como una lista de colecciones, donde el primero de los elementos será la colección que contenga las URIs del modelo original. Después, para cada URI en ese conjunto, se expandirá el nodo al que hace referencia y se volverán a almacenar todas las URIs encontradas en el nuevo modelo, en el siguiente nivel de la estructura de datos.

La escalabilidad de este método permite llegar a profundidades elevadas, a costa de tiempos de cómputo exageradamente altos debido a la alta interacción entre el cliente y el servidor donde se alojan los recursos. Se podrá comprobar posteriormente en los resultados obtenidos.

- El **algoritmo declarativo** aprovecha la potencia declarativa del lenguaje SPARQL para obtener las diversas sentencias nivel a nivel con consultas sencillas y de automatización trivial, como las que se pueden consultar en el listado 5.3, donde se puede comprobar cómo la consulta devolverá todos aquellos recursos que tengan relación con cualquier recurso que se haga referencia en el modelo original, es decir, devolvería la capa número uno (considerando el modelo original como la capa cero).

Para profundidades no muy elevadas (considerando n alrededor de 3) los tiempos de respuesta son asumibles. No obstante, a partir de ese umbral, el servidor comienza a balancear la carga y los tiempos de respuesta se incrementan.

```
1 SELECT DISTINCT ?obj WHERE { <TargetUri> ?pi ?o1. ?o1 ?pj ?obj }
```

Listado 5.3: Consulta SPARQL: obtención de nuevas triplas en nivel 1

Finalmente tras llevar a cabo el cálculo de la métrica (que se explicará en el siguiente punto) e independientemente del algoritmo de expansión de nodos utilizados, se generarán los resultados como un modelo de Jena para que puedan ser publicados.

Por otra parte con el fin de mantener una estructura en memoria que haga los resultados más accesibles para el programador, se diseña una estructura de datos `MeasurementResult` que guardará los resultados de las evaluaciones a medida que éstos se vayan generando. Dicha estructura será una lista de registros que contará con los siguientes campos:

- `DQDDimension`: nombre de la DQD.
- `Dimension metric`: nombre de la métrica de la DQD que comparando con otros autores tal y como se contempló en la Sección 3.4 y con el fin de crear una estructura suficientemente versátil, se da la opción de para cada dimensión construir diversas métricas, aunque en el presente trabajo únicamente se trabaja con una única métrica.
- `Value`: valor de la métrica.
- `Contextual Value`: valor contextual de la evaluación.

Se ha tomado la decisión de añadir esta estructura de datos para facilitar al programador, como se ha dicho, los datos obtenidos mediante una evaluación. De no ser así la única forma de acceder a dichos datos sería realizando consultas sobre el modelo de Jena generado con los resultados, lo que implica a su vez llevar a cabo consultas SPARQL que en ocasiones no es trivial automatizar.

Una cuestión que merece ser respondida en este punto es la decisión de diseño que se ha tomado a la hora de realizar las evaluaciones. Existen dos maneras básicas de trabajar con datos almacenados en un servidor de triplas:

1. Si se posee el propio servidor con todos los datos, trabajar localmente contra él.
2. Si el servidor es remoto, trabajar remotamente a través de HTTP.

Supóngase que se necesita realizar un análisis de calidad de datos sobre datasets que están repartidos en diversos servidores. En primer lugar deberían descargarse cada uno de los datasets (que usualmente vienen en bloque de tamaño considerable). Acto seguido guardar todas las triplas en un único servidor y después llevar a cabo el análisis.

JenaDQ evita ese paso, indicando los datasets que queremos evaluar por medio de su URI así como el endpoint donde los datos están accesibles. Al iniciar la evaluación,

los datos se van cargando dinámicamente mediante sucesivas consultas a los distintos servidores mediante HTTP y los resultados se engloban en un único archivo final RDF.

■ **Ventajas:**

- No es necesario descargar todo el dataset para cada análisis. No se descargará más información de la estrictamente necesaria.
- El programador tampoco tendrá que realizar la carga de dichos conjuntos de datos en un servidor propio ni establecer arquitectura alguna para llevar a cabo el análisis (aunque sí deberá hacerlo si desea guardar los resultados). Dicha carga podría resultar costosa en tiempo y recursos.
- En el caso de que el usuario desee realizar análisis en local, puede habilitar un servidor y la base de datos de triplas y realizar el análisis contra dicho servidor en cualquier momento, pues la ejecución es transparente en ese sentido.

■ **Inconvenientes:**

- Sobre ciertos análisis (los que requieren de una profundidad, como se verá en los resultados de este caso de uso) el tráfico de información así como el tiempo de respuesta del servidor pueden ralentizar los cálculos de manera notable. Por otro lado, casi cualquier dataset puede ser evaluado considerando que una profundidad elevada carecería de sentido, siendo los resultados prácticamente idénticos a una profundidad acotada a no más de 4 niveles.
- La dependencia de un servidor de datos en ocasiones puede significar que el servicio (el endpoint) se encuentre inactivo por labores de mantenimiento, fallos en la red o cualquier causa que afecte a las comunicaciones entre los dos extremos. Si los datos no están replicados en otro servidor, el análisis resultaría inviable.

Implementación

En la fase de implementación se ha traducido a código los algoritmos especificados en la etapa anterior y se han integrado mínimamente con Jena (la elaboración de la fachada y la integración de JenaDQ en Jena se desarrollará en secciones posteriores), siguiendo la interfaz desarrollada en esta misma iteración.

El resultado de dicha implementación se resume en el algoritmo reflejado en el listado 5.4, donde se puede comprobar cómo se consideran recursos válidos a aquellos que cumplen con las reglas de uso. Finalmente los resultados son guardados en dos estructuras de datos:

1. Un modelo de Jena utilizando como base el vocabulario desarrollado, que puede consultarse en el Anexo C. Este modelo a partir de este momento es susceptible de ser publicado o tratado como cualquier otro modelo de Jena.

2. Una estructura de datos `MeasurementResult` que guarda en memoria dinámica los datos de la evaluación con el fin de hacerlos más accesibles al usuario.

```

1 for(level i:DataStructure){
2   for(statement st:i){
3     !isValidResource(st, rule){
4       notValid <- notValid + 1
5     }
6   }
7   measure = calculateDQMeasure(notValid, i.size())
8   notValid = 0
9   store(measure)
10 }

```

Listado 5.4: Pseudocódigo de la evaluación para Completeness

Pruebas

Durante la etapa de pruebas se han llevado a cabo test unitarios utilizando `jUnit` hasta que los resultados fueron satisfactorios. Pueden consultarse ejemplos de código de pruebas en el Anexo D.

En la figura 5.8 se puede apreciar la interacción existente entre el módulo desarrollado y los distintos actores: quien solicita la evaluación y quien sirve los datos que serán objeto de ser evaluados.

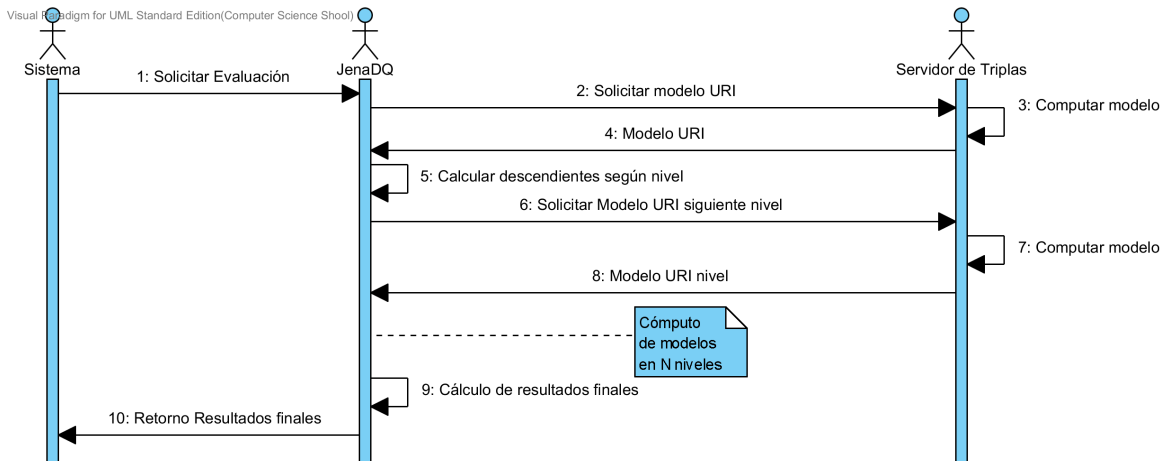


Figura 5.8: Diagrama de Interacción: Caso de Uso 1

Como resultado, se obtiene una implementación funcional del primer caso de uso para la DQD Completeness en la que se evalúa la compleción de un conjunto de datos semánticos considerando el contexto y la profundidad en el grafo de datos enlazados.

En el cuadro 5.5 se muestran algunos resultados en términos de tiempo obtenidos por los diversos algoritmos utilizados. Los conjuntos de datos y el *endpoint* pertenecen al dominio DBPedia:

- URI: <http://dbpedia.org/resource/...>
- Endpoint: <http://dbpedia.org/sparql>

URI	Profundidad	Nodos/Nivel	T. Nodos	T. Total	Algoritmo
:Manakkara	3	25 34 70	0.615s	11.942s	Declarativo
:Manakkara	3	25 34 70	5.517s	16.613s	Iterativo
:Life_of_Pi	3	154 1254 10052	17.454s	183.831s	Declarativo
:Life_of_Pi	2	154 1254	1.903s	120.085s	Declarativo
:Metallica	2	229 3668	5.863s	330.864s	Declarativo
:Metallica	1	229	0.4s	25.821s	Declarativo
:Metallica	1	229	0.342s	22.672s	Iterativo

Cuadro 5.5: Resultados para evaluaciones de Completeness

Como se puede comprobar, los tiempos comienzan a incrementarse a medida que el número de nodos por nivel aumenta. Del mismo modo se observa que el tiempo de recuperación de nodos por nivel es parecido cuando la profundidad a la que se desea evaluar la calidad se mantiene baja, mientras que a profundidades altas una consulta declarativa contra el *endpoint* resuelve antes el problema.

No obstante se puede observar cómo si la carga hacia el servidor es alta, el tiempo de respuesta (por política del servidor) crece, pudiendo darse el caso que el servidor ante la cantidad de cómputo que se le exige y por motivos de seguridad, bloquee la consulta y no devuelva resultados.

En cualquiera de los dos algoritmos el tráfico es elevado, puesto que para cada recurso dentro de la estructura de datos, se debe solicitar al servidor que devuelva el conjunto de datos asociado a esa URI. No obstante, nuevamente el tráfico se disminuye con el algoritmo declarativo puesto que en una única consulta devuelve todos los datos necesarios para cada capa, lo que ahorra tiempo e intercambio de información a través de la red.

5.2.2 Iteración 3: CdU 2 Realizar Evaluación para DQD Accessibility

En el cuadro 5.6 se resumen los objetivos y resultados obtenidos en esta iteración. En la figura 5.9 se muestra la interacción existente entre el módulo desarrollado y los distintos actores, al igual que en caso de uso anterior. En este caso al no existir búsqueda por niveles, el cómputo es más rápido y la mecánica de funcionamiento se reduce a evaluar:

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/02/2014 - 1/03/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> Análisis, Diseño, Implementación y Pruebas para Caso de Uso 2: Realizar Evaluación para DQD: Accessibility 	<ul style="list-style-type: none"> Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 2 Módulo DQDimension Accessibility

Cuadro 5.6: Iteración 3

1. En primer lugar, cuántos de los objetos dentro de una tripla (recuérdese, una tripla es un trío formado por *sujeto*, *predicado*, *objeto*) son enlaces.
2. Una vez calculados todos los enlaces, se evalúan buscando enlaces caídos.
3. Se contabilizan enlaces totales, triplas totales y enlaces válidos, obteniendo así la métrica.

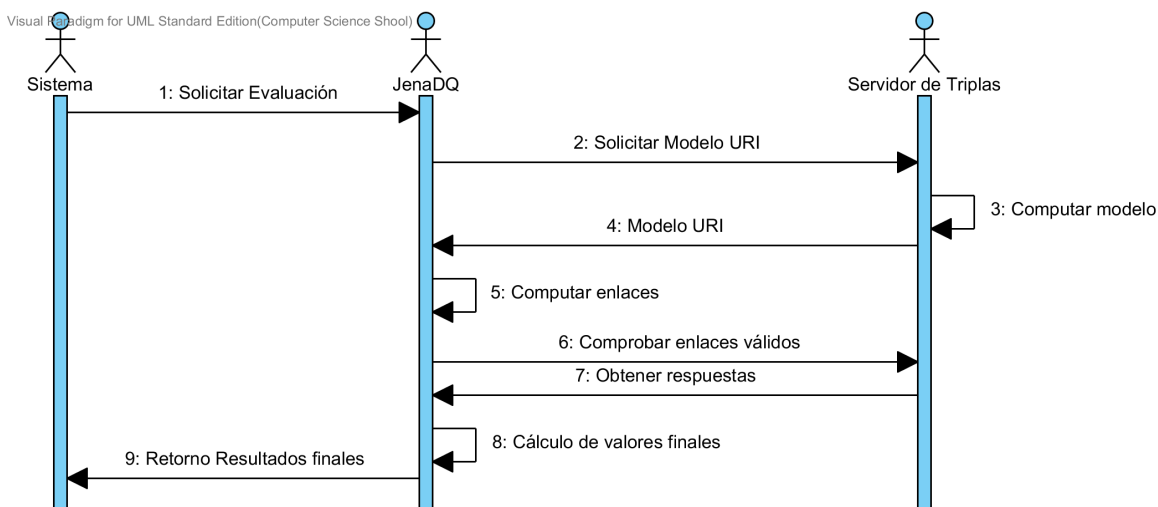


Figura 5.9: Diagrama de Interacción: Caso de Uso 2

Análisis

Habiendo estudiado el estado del arte (véase Sección 3.3.3), se llevó a cabo el análisis buscando la métrica para esta dimensión. Como resultado, se ha establecido la siguiente definición para la DQD Accessibility en este proyecto: *grado de interconexiones en Linked Data para un determinado conjunto de datos*.

De igual modo, la medida se calculará mediante la ecuación 5.1.

Las **reglas de contexto** utilizadas en este caso serán únicamente de **valores contextuales**. Se puede ver un ejemplo de las reglas en el listado 5.5. En este caso las nuevas propiedades hacen referencias a medidas de accesibilidad.

```

1    @prefix dqa: http://www.example/ontology#
2    [ruleLOW: (?x dqa:AccessibilityMeasure ?m) lessThan(?m, 0.8 ) ->
      (?x
3      dqa:ContextualValue dqa:LOW) ]
4    [ruleMED: (?x dqa:AccessibilityMeasure ?m) ge(?m, 0.8 ) lessThan
      (?m, 0.9) -> (?x dqa:ContextualValue dqa:MED) ]
5    [ruleHIGH: (?x dqa:AccessibilityMeasure ?m) ge(?m, 0.9 ) le(?m,
      1.0) -> (?x dqa:ContextualValue dqa:HIG) ]

```

Listado 5.5: Ejemplo de reglas contextuales para Accessibility

Diseño

En comparación con Completeness, el cálculo de la métrica no requiere de uso de estructuras de datos para almacenar conjuntos de triplas, puesto que no hay búsqueda por niveles. Así pues el algoritmo puede resumirse según sigue: para cada tripla en el conjunto de datos proporcionado, comprobar si el objeto que es descrito es una URI válida. Se puede comprobar el pseudocódigo en el listado 5.6.

```

1    for(Statement st:ModelStatements){
2        if(!isURIResource(st.getObject())){
3            notValid <- notValid + 1
4        }
5        else{
6            if(isFileURI() || !isValidURI()){
7                notValid <- notValid + 1
8            }
9        }
10    }
11    measure = calculateDQMeasure(notValid, ModelStatements.size())
12    notValid = 0
13    store(measure)

```

Listado 5.6: Pseudocódigo de la evaluación

`isFileURI` comprobará si la URI hace referencia a un enlace de algún tipo de archivo (foto, vídeo, documento de texto, ...).

Por otro lado, `isValidURI` realizará comprobaciones respecto de la validez y estado de la URI proporcionada, tales como comprobar que el enlace no está caído.

Implementación

Para la fase de Implementación se ha traducido a código el algoritmo anteriormente descrito siguiendo un procedimiento muy similar al caso de uso desarrollado anteriormente.

Los resultados después de una ejecución de la evaluación generarán, al igual que para Completeness:

- Un modelo de Jena con los resultados.
- Una estructura de datos MeasurementResult con los datos de la evaluación y sus resultados, para hacerlos más accesibles.

Pruebas

Durante la etapa de pruebas, se han realizado test unitarios utilizando junit hasta que los resultados fueron satisfactorios. Pueden consultarse ejemplos de código de pruebas en el Anexo D.

En el cuadro 5.7 se muestran los **resultados** obtenidos para esta dimensión de calidad en términos de tiempo y número de nodos. También se añaden los resultados obtenidos por el algoritmo tras llevar a cabo el cálculo.

URI	Nodos	Tiempo Total	Resultado
:Manakkara	25	1.053s	0.73
:Life_of_Pi	154	1.391s	0.619
:Metallica	229	1.465s	0.772

Cuadro 5.7: Resultados para evaluaciones de Accessibility

5.2.3 Iteración 4: CDU 3 Realizar evaluación de DQD y CDU 4 Crear Plan de Evaluación

En el cuadro 5.8 se muestra un resumen del trabajo durante esta iteración.

Fase del PUD	Elaboración
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/03/2014 - 1/04/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 3: Realizar Evaluación de DQD. ■ Análisis, Diseño, Implementación y Pruebas para Caso de Uso 4: Crear Plan de Evaluación de DQD. 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 3 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 4 ■ Módulos DQDimension, DQAssessment, DQAssessmentPlan, MeasurementResult.

Cuadro 5.8: Iteración 4

Caso de Uso 3: Realizar Evaluación de DQD

Análisis

En este punto se lleva la integración de ambas dimensiones de calidad desarrolladas mediante el patrón *Builder*. Se define una evaluación como:

1. Una **URI** objetivo que contendrá los datos semánticos que se desean evaluar.
2. Un **endpoint**, servidor donde se alojan los datos y desde donde se recuperarán para su análisis.
3. Un conjunto de **Dimensión de Calidad de Datos (DQD)** para las cuales se evaluará la calidad de los datos.
4. Una serie de **reglas de contexto** que a su vez pueden ser:
 - **Reglas de valores contextuales**: siempre son requeridas. Se aplicarán sobre las métricas una vez calculadas para obtener los valores contextuales de calidad.
 - **Reglas de uso**: requeridas en algunas dimensiones (en este caso, únicamente Completeness necesita reglas de uso).
5. Una **profundidad** a la que se desea evaluar el conjunto de datos. Únicamente Completeness necesita este parámetro.
6. Un **identificador** de evaluación.

Diseño

Se hace uso del patrón *Builder* para integrar ambas dimensiones y que éstas funcionen de la misma manera. Para ello se definen las funciones abstractas en la clase *DQDDimension*:

- **executeMeasurement**: que ejecutará la medida de la dimensión devolviendo como resultado un modelo de Jena.
- **getRDFModel**: generará internamente para cada dimensión el modelo de Jena y la estructura *MeasurementResult*. El modo en que estas estructuras se generan varía para cada dimensión.

Implementación

En la etapa de Implementación se llevan los algoritmos al código, aplicando el patrón de diseño necesario.

Pruebas

Se han llevado a cabo las pruebas unitarias con *jUnit* hasta que el resultado ha sido satisfactorio.

Caso de Uso 4: Crear Plan de Evaluación

Análisis

Un Plan de Evaluación consiste en un conjunto de evaluaciones, que puede ser sobre

un mismo conjunto de datos en momentos diferentes de tiempo. Al ejecutar el plan, se ejecutarán independientemente cada una de las evaluaciones, que pueden ser sobre los mismos conjuntos de datos o distintos, así como las reglas, endpoints y otros valores pueden coincidir o no.

El resultado final es, al igual que en una evaluación simple, un modelo semántico de datos enlazados con el resultado de cada una de las evaluaciones. Es posible que todas las evaluaciones tengan identificadores distintas o compartan un único identificador.

Dicho identificador es a nivel del propio modelo, por lo que no generará conflictos con otros datos almacenados.

Diseño

Se ha diseñado la clase `DQAssessmentPlan`, que estará parametrizada por:

- Lista de `DQAssessment`
- Modelo final
- Lista de `MeasurementResult` final

Cada una de las evaluaciones (`DQAssessment`) tendrá su propio identificador, que puede coincidir o no con cualquier otro en la lista de `DQAssessmentPlan`.

Para la generación de resultados se utilizan operaciones de unión de conjuntos (evitando así triplas repetidas) de manera que al final se obtiene un único modelo que resulta de la unión de todos los modelos parciales para cada dimensión evaluada.

Implementación

La implementación finaliza con la creación de la clase `DQAssessmentPlan`, que lleva al código los conceptos anteriormente señalados. En el listado 5.7 puede verse el pseudocódigo de ejecución de un plan.

Pruebas

Se han realizado las pruebas pertinentes con `jUnit` hasta que el resultado del funcionamiento ha sido satisfactorio.

```
1  for(DQAssessment dqa:DQAssessmentList){
2      dqa.executeAssessment();
3      finalModel = finalModel.union(dqa.getAssessmentResult());
4  }
5  return finalModel;
```

Listado 5.7: Pseudocódigo del plan de evaluación

5.3 Fase de Construcción

En la fase de Construcción se desarrolla el código relativo a las evaluaciones simples y los planes de evaluación así como la propia API que se facilitará para su uso posterior. Una vez

desarrollada la parte central de la extensión en las iteraciones previas, es hora de incluir los conjuntos de funciones necesarias para hacer del código algo usable y reutilizable.

5.3.1 Iteración 5: C_{DU} 9, 10, 11 y 12 sobre APISemDQ

En el cuadro 5.9 se muestra un resumen de los objetivos para esta iteración así como de los resultados obtenidos.

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/03/2014 - 1/04/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 9: Realizar evaluación de DQD (APISemDQ) ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 10: Publicar evaluación de DQD ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 11: Crear Plan de evaluación de DQD (APISemDQ) ■ Análisis, Diseño, Implementación y pruebas para el C_{DU} 12: Recuperar Evaluación de DQD 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 9 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 10 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 11 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para C_{DU} 12

Cuadro 5.9: Iteración 5

Caso de Uso 9: Realizar Evaluación de DQD sobre APISemDQ

Para este caso de uso se han añadido los conjuntos de funciones necesarias a la clase fachada para completar la API. Estas funciones facilitan la creación y gestión de una Evaluación de DQ permitiendo añadir dimensiones de calidad, reglas y, en general, cualquiera de los parámetros necesarios para la puesta a punto y ejecución del proceso de evaluación. Pueden verse ejemplos en el listado 5.8.

También incluye funciones de rescate de resultados. Estos resultados se presentan igualmente en forma de modelo de datos semánticos de Jena, con lo que son susceptibles de ser publicados como LOD.

La generación de los resultados se basa en el concepto de *unión* de conjuntos. Cada resultado parcial, es decir, cada resultado generado para cada evaluación de dimensión, se considera a efectos prácticos un conjunto. Jena incluye operaciones de conjuntos sobre los

```

1 public DQAssessment createdQAssessment(
2     LinkedList<DQDimension> dqDimensionList, String uRI,
3     String endpoint, List<Rule> contextualRules, int depth,
4     String dqAssessmentIdentifier){

6     DQAssessment dqa = new DQAssessment(dqDimensionList, uRI, endpoint,
7         contextualRules, depth, dqAssessmentIdentifier);
8     return dqa;
9 }

11 public Model executeDQAssessmentGetModel(DQAssessment dq) {
12     return dq.executeAssessment();
13 }

```

Listado 5.8: Fragmento de la clase fachada APISemDQ: DQAssessment

modelos que se carguen, tales como *unión* o *diferencia*.

De esta manera, gracias a la operación de *unión* se obtiene un único modelo sin redundancia que engloba a todas las dimensiones de calidad evaluadas.

En la figura 5.11 se puede ver un esquema de la generación de resultados parciales y finales utilizando la aproximación *bottom-top* en el ámbito de un plan de evaluación.

Caso de Uso 10: Publicar Evaluación

Con el fin de hacer JenaDQ independiente de cualquier tecnología, se da libertad al usuario de escoger su propio contenedor semántico (véase Sección 3.1.7). Si bien, es recomendable utilizar Jena TDB, que ya viene integrado con Jena y es altamente escalable y eficiente.

Más adelante, en el desarrollo de LiDQA Tool, generada como prueba de concepto, podrá corroborarse el uso de este sistema para el almacenamiento de los resultados de la evaluación.

En la figura 5.10 puede observarse un escenario de funcionamiento de la aplicación almacenando los datos. Es muy importante recalcar que JenaDQ está completamente desacoplado del almacenamiento, para que el usuario sea libre de guardar o no los resultados y tratar con ellos a su gusto usando cualquier tecnología que desee integrar con JenaDQ.

Cabe destacar que si en efecto se utiliza como contenedor semántico Jena TDB, el flujo de interacción final volvería al actor *Extended Jena*.

Caso de Uso 11: Crear Plan de Evaluación sobre APISemDQ

Al igual que para el CDU 9, se han añadido los conjuntos de funciones necesarias para completar la API en la clase fachada APISemDQ. Estas funciones facilitan la labor del programador para mover y ajustar los parámetros necesarios a la hora de definir un plan de evaluación.

Un plan de evaluación es un conjunto de evaluaciones independientes pero que generan

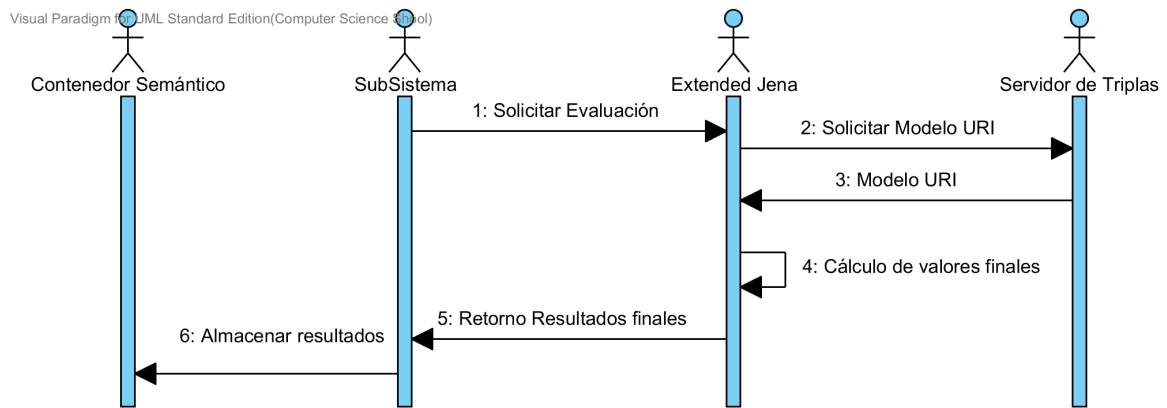


Figura 5.10: Diagrama de Interacción: Caso de Uso 10

un resultado común. Este resultado se traduce en un único modelo de datos que, en función de las evaluaciones, puede contener uno o varios identificadores de evaluación.

La generación de este modelo se obtiene mediante operaciones de conjuntos respecto de los modelos parciales generados en cada evaluación. De esta manera, al combinar dos modelos por medio de la operación *unión*, se descartan las triplas repetidas, luego no existe redundancia y el modelo final es la combinación de los modelos parciales, como se puede ver en la figura 5.11. Puede verse un ejemplo del código desarrollado en el listado 5.9.

```

1 public DQAssessmentPlan createAssessmentPlan() {
2     DQAssessmentPlan dqplan = new DQAssessmentPlan();
3     LinkedList<DQAssessment> dqplanlist =
4         new LinkedList<DQAssessment>();
5     dqplan.setAssessmentList(dqplanlist);
6     return dqplan;
7 }

9 public Model executeAssessmentPlan(DQAssessmentPlan dqplan) {
10     return dqplan.executePlan();
11 }
  
```

Listado 5.9: Fragmento de la clase fachada APISemDQ: DQAssessmentPlan

Caso de Uso 12: Recuperar Evaluación de DQ desde el repositorio de datos

En este caso se han definido dos escenarios para ilustrar el tipo de interacción existente entre una aplicación haciendo uso del framework extendido y:

1. **Escenario 1:** Un contenedor de datos semánticos externo a Jena.
2. **Escenario 2:** Jena TDB, integrado dentro del propio framework.

En el **escenario 1** (ver figura 5.12) puede comprobarse como la obtención del modelo es transparente en el momento que se tiene la URI y el *endpoint*. Únicamente hay que solici-

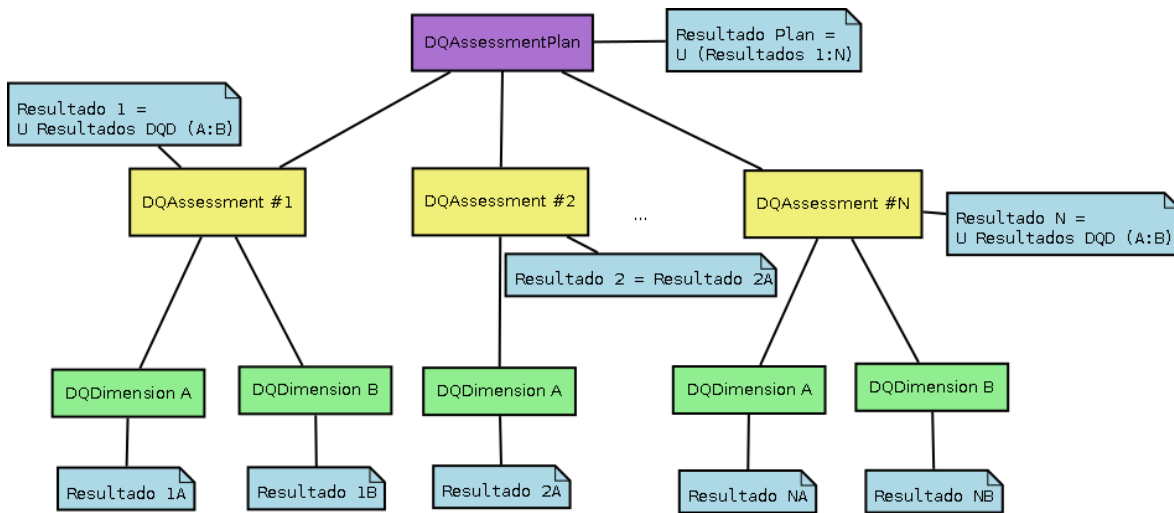


Figura 5.11: Caso de Uso 9-11: Gestión de resultados (bottom-top)

tarle al framework extendido esos dos parámetros y puesto que el contenido del modelo es independiente del modo de obtenerlo, basta con invocar a las funciones habituales.

En el **escenario 2** (ver figura 5.13) se puede comprobar que si se está haciendo uso de TDB como contenedor semántico gestionado por Jena, la llamada no tiene por qué ser a través de protocolo HTTP, pues el contenedor semántico será local al servidor. Únicamente hay que indicarle el nombre con el que se ha guardado el modelo y el framework extendido se encargará de devolverlo.

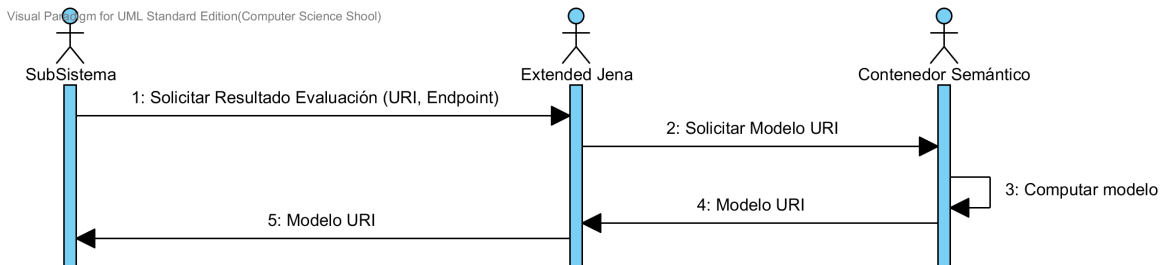


Figura 5.12: Caso de Uso 12 - Escenario 1

5.3.2 Iteración 6: CdU 5 y 13 Comparar Evaluación de DQD

En esta iteración el esfuerzo se ha centrado en obtener una comparación entre dos evaluaciones, como se puede apreciar en el cuadro 5.10.

Caso de Uso 5: Comparación de Evaluaciones

Análisis

La comparación de evaluaciones se fundamenta en la comparación de conjuntos. Puesto que Jena incluye operaciones de conjuntos sobre los modelos de datos, únicamente

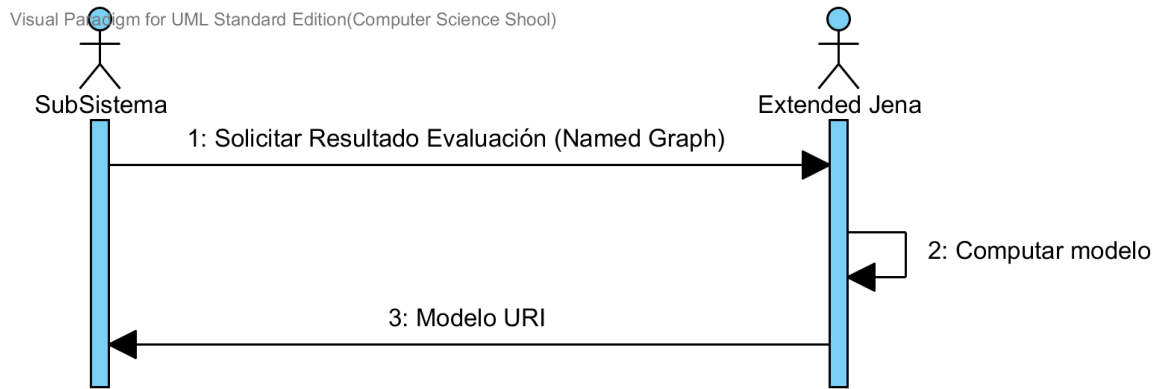


Figura 5.13: Caso de Uso 12 - Escenario 2

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	1/04/2014 - 20/04/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> Análisis, Diseño, Implementación y pruebas para el CDU 5: Comparar evaluación de DQD Análisis, Diseño, Implementación y pruebas para el CDU 13: Comparar Evaluación de DQD (APISemDQ) 	<ul style="list-style-type: none"> Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 5 Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 13

Cuadro 5.10: Iteración 6

hay que especificar qué operaciones debemos realizar.

En la figura 5.14 se muestra un ejemplo en el que dos modelos tienen una parte común (que bien puede ser una evaluación sobre el mismo conjunto de datos que presenta los mismos resultados) y una parte disjunta. La comparación de los modelos comprueba los siguientes aspectos:

1. Porcentaje de afinidad de los modelos: del total de triplas de ambos modelos de datos, cuántos son iguales.
2. Modelo resultante de la comparación: la comparación entre modelos busca responder a la pregunta: *¿En qué se diferencian los modelos A y B?* Dados dos modelos cualesquiera A y B, siendo R el modelo resultado, la comparación de modelos se computa según la ecuación 5.2.

$$R = (A \cup B) - (A \cap B) \quad (5.2)$$

Diseño

En el listado 5.10 se puede ver el pseudocódigo obtenido en la fase de diseño, que ilustrará el resultado final.

```
1 Model IntersectModel = ModelA.intersection(ModelB);  
2 Model ResultModel = ModelA.union(ModelB);  
3 ResultModel = ResultModel - IntersectModel;  
4 return ResultModel;
```

Listado 5.10: Pseudocódigo de la comparación entre modelos

Implementación

Se ha llevado a cabo la implementación del pseudocódigo satisfactoriamente, incluyendo dos funciones:

- `modelComparison`: que devolverá el modelo resultante de la comparación entre ambos modelos.
- `affinity`: que devolverá un valor de afinidad entre los dos modelos, siguiendo los mismos principios.

Pruebas

Se han llevado a cabo las pruebas con jUnit hasta que los resultados han sido satisfactorios. Pueden consultarse ejemplos de código de pruebas en el Anexo D.

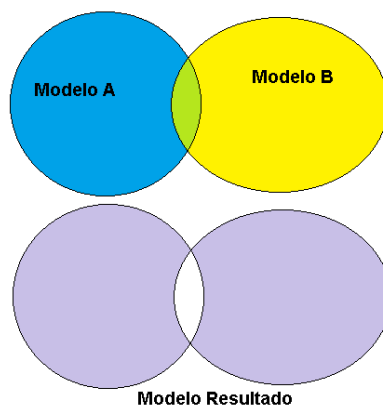


Figura 5.14: Esquema de Comparación de resultados

Caso de Uso 13: Comparación de Evaluaciones sobre APISemDQ

Como se puede comprobar en la figura 5.15, el sistema que realiza la petición es el encargado de hacer llegar al framework extendido los modelos que se deseen comparar a través del patrón fachada APISemDQ, cuyo fragmento de código para este caso de uso puede consultarse en el listado 5.11.

Dicha comparación se realiza obteniendo como resultado un modelo de comparación que será devuelto al usuario.

Nuevamente se remarca que los orígenes de los datos deben estar lo más desacoplados posible de la tecnología, delegando la labor de generar y facilitar los datos al usuario o sistema encargado de realizar la comparación.

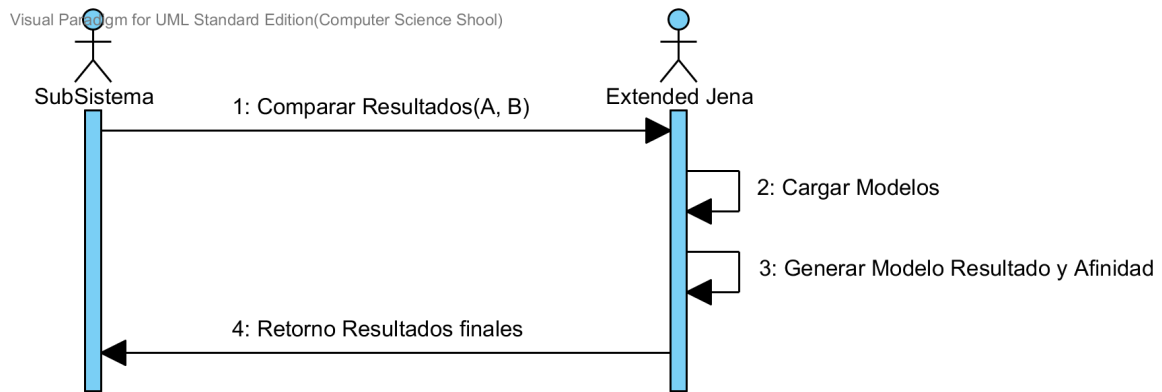


Figura 5.15: Caso de Uso 5, 13: Comparación de resultados

```

1 public Model modelComparison(DQModel modelA , DQModel modelB){
2     return modelA.compareModelWith(modelB);
3 }
  
```

Listado 5.11: Fragmento de la clase fachada APISemDQ: Comparison

5.3.3 Iteración 7: CdU 6, 7 y 8 Carga de datos de evaluación

En el cuadro 5.11 se muestra un resumen del trabajo durante esta iteración, que se ha centrado en la adquisición de datos.

Caso de Uso 6: Carga de datos a través de una URI

Análisis

Esta iteración se ha destinado a encapsular algunas de las funcionalidades de Jena en objetos que permitan un acceso más amigable. Los tres casos de uso tratados se orientan a la carga de datos para su posterior uso. Esto se ha conseguido generando un patrón fachada que encapsula las funcionalidades de Jena al respecto: DQModel.

DQModel permite una carga de datos más sencilla al programador. Estos datos se pueden cargar de dos maneras:

1. A través de un fichero
2. A través de una URI y un *endpoint*

Fase del PUD	Construcción
Flujo de trabajo del PUD	Análisis, Diseño, Implementación y Pruebas
Fechas Inicio - Fin	20/04/2014 - 15/05/2014
Objetivos	Artefactos de Salida
<ul style="list-style-type: none"> ■ Análisis, Diseño, Implementación y pruebas para el CDU 6: Carga de datos desde URI y <i>endpoint</i> ■ Análisis, Diseño, Implementación y pruebas para el CDU 7: Carga de datos desde fichero ■ Análisis, Diseño, Implementación y pruebas para el CDU 8: Carga de datos de Evaluación 	<ul style="list-style-type: none"> ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 6 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 7 ■ Diagramas de Clases y de Secuencia, implementación y pruebas para CDU 8

Cuadro 5.11: Iteración 7

La funcionalidad de `DQModel` no es solamente la de la carga de datos, sino también la de la impresión por pantalla de los modelos cargados en un formato determinado o bien la detección del formato en el que se presentan las triplas de manera automática.

Diseño

Se ha generado `DQModel` como una fachada para la obtención de modelos mediante las dos formas mencionadas. Para ello, se ha delegado la responsabilidad de rescatar los archivos en una clase `DataPicker` que es llamada desde `DQModel`.

La carga de datos a través de una dirección recibe dos parámetros:

- **URI:** la dirección a la cual está asignada el modelo de datos objetivo de la evaluación. Dichos datos se encontrarán alojados en un servidor de triplas.
- **Endpoint:** la dirección del servidor donde se encuentran alojados los datos. Este servidor a través de las operaciones SPARQL correspondientes vía HTTP permitirá recuperar los modelos de datos especificados.

En la figura 5.16 puede contemplarse la interacción existente entre los distintos sistemas hasta obtener el modelo deseado.

Implementación

Se ha trasladado a código el diseño referente a la carga desde URI y endpoint.

Pruebas

Se han realizado pruebas unitarias con `jUnit` hasta que el resultado ha sido satisfactorio.

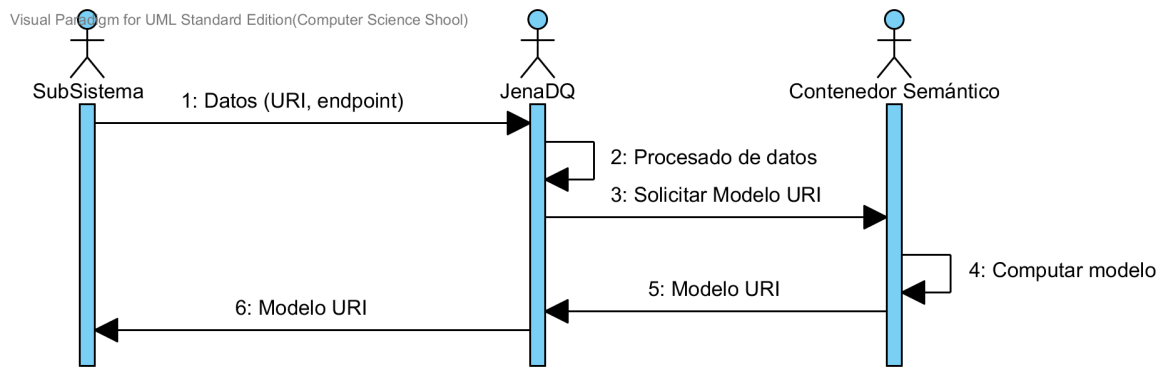


Figura 5.16: Caso de Uso 6: Carga de datos a través de URI

Caso de Uso 7: Carga de datos desde fichero

Como se puede apreciar en la figura 5.17 en caso de cargar desde fichero no existe necesidad de llevar a cabo ninguna llamada externa, puesto que las triplas ya se encuentran en este fichero.

Existe a su vez una variante en la clase `DQModel` que permite indicar la URL del fichero donde se encuentran los datos, cargándolos de manera transparente.

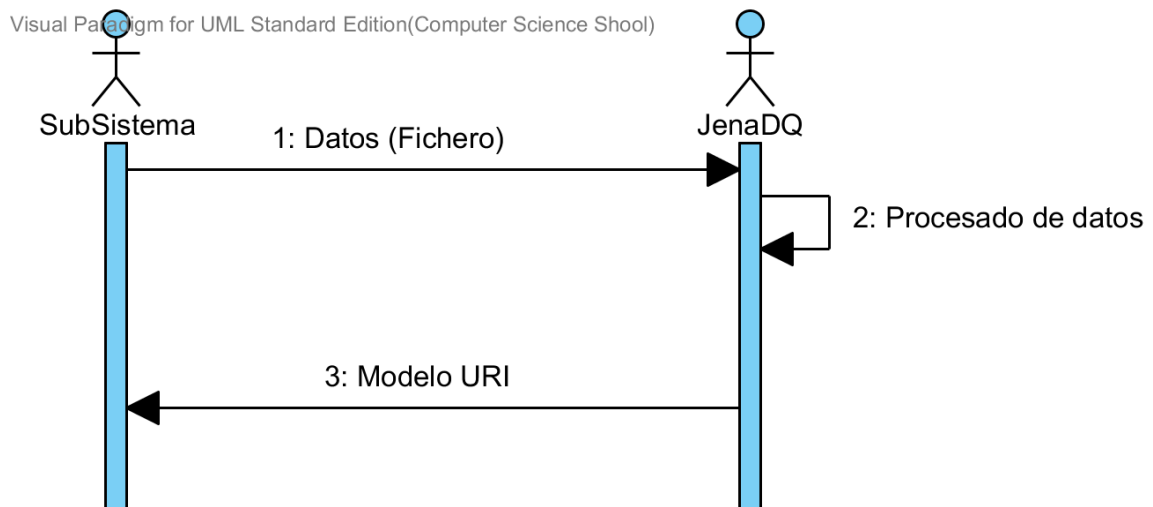


Figura 5.17: Caso de Uso 7: Carga de datos desde fichero

Caso de Uso 8: Cargar datos de evaluación

En la figura 5.18 se puede apreciar el resultado final del diseño de clases para cargar este escenario de uso. Para ello se han desarrollado una clase principal y dos clases auxiliares:

- `DQModel`: clase principal encargada de encapsular las llamadas necesarias a Jena para obtener los modelos de datos a partir de las fuentes.
- `DataPicker`: clase auxiliar en la que delega `DQModel` para encapsular las llamadas a

Jena de recuperación de datos en base a los parámetros. Incluye un reconocimiento de extensiones para ficheros.

- **DataWriter**: clase auxiliar en la que delega **DQModel** para encapsular las llamadas a Jena encargadas de mostrar por pantalla los datos según el formato y generar los modelos finales.

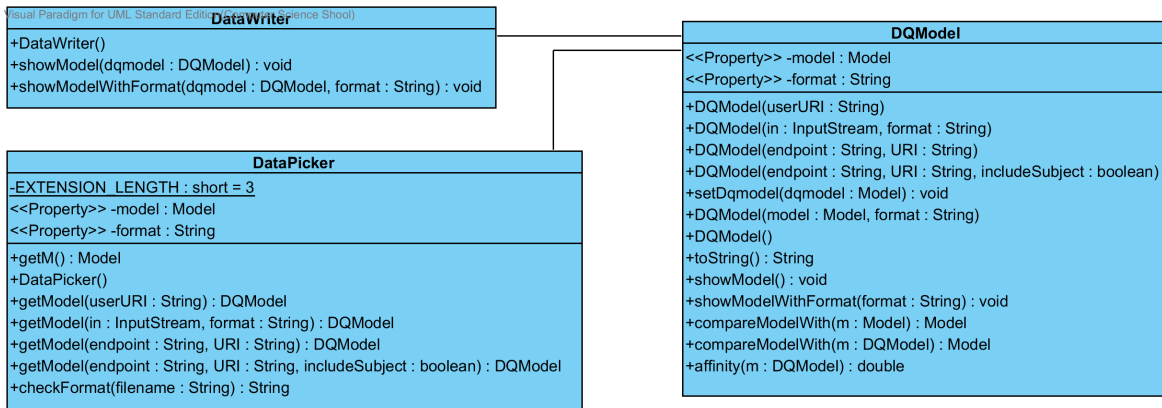


Figura 5.18: Caso de Uso 8: Cargar datos de evaluación

En **DQModel** se aprecia que ya en los constructores es posible obtener los modelos. Un ejemplo de uso podría ser el mostrado por el listado 5.12.

```

1 String uri = "http://dbpedia.org/resource/Pink_Floyd";
2 String endpoint = "http://dbpedia.org/sparql";
3 DQModel dq = new DQModel(endpoint, uri);
4 dq.showModelWithFormat("N3");
  
```

Listado 5.12: Ejemplo de recuperación de modelos a través de URI y *endpoint*

5.4 Fase de Transición

Finalmente en la fase de transición tienen lugar dos iteraciones. En primer lugar, en una iteración se realizará la aplicación de prueba de concepto que ponga de manifiesto el uso de la extensión, como el segundo y último componente que requiere este PFC.

La última iteración será la de desarrollo de toda la documentación necesaria.

5.4.1 Iteración 8: Creación de LiDQA Tool

En el cuadro 5.12 se puede contemplar una planificación detallada para esta iteración del PUD. En ella, la iteración presente se ha descompuesto en flujos de trabajo relativos a la elaboración de esta herramienta. El diagrama de casos de uso en el cual se ha basado esta planificación se puede consultar en la figura 5.3.

Fase del PUD	Iteraciones	Objetivos
Inicio	Iter. 1	<ul style="list-style-type: none"> ■ Establecer Requisitos ■ Realización de diagrama de casos de uso ■ Planificación de las iteraciones ■ Elaborar el diseño de la arquitectura del sistema ■ Elaborar diseño de interfaz de usuario
Elaboracion	Iter. 2	<ul style="list-style-type: none"> ■ Casos de Uso 1, 2
	Iter. 3	<ul style="list-style-type: none"> ■ CdU 3, 4, 5
	Iter. 4	<ul style="list-style-type: none"> ■ CdU 6, 7, 8
Construcción	Iter. 5	<ul style="list-style-type: none"> ■ CdU 9, 10
	Iter. 6	<ul style="list-style-type: none"> ■ CdU 11, 12
Transición	Iter. 7	<ul style="list-style-type: none"> ■ Mejoras en la interfaz de usuario ■ Generación de artefactos finales para la instalación e implantación

Cuadro 5.12: Planificación de sub-iteraciones para LiDQA Tool

Iteración 8: fase de Inicio (parte I)

Los resultados para esta parte de la iteración son los siguientes:

- Requisitos: Los requisitos de LiDQA Tool ya han sido especificados y se pueden consultar en la Sección 5.1.1.
- El diagrama de casos de uso para la aplicación Web se puede consultar en la figura 5.3.
- La planificación de las iteraciones resultantes se puede consultar en el cuadro 5.12.
- El diseño de la arquitectura del sistema será cliente-servidor se puede consultar en la figura 5.19.
- El diseño de la interfaz de usuario puede consultarse en las figuras 5.20 y 5.21.

Iteración 8: fase de Elaboración (parte II)

En esta parte de la iteración se llevaron a cabo las implementaciones de los primeros casos de uso de LiDQA Tool, consistentes en cargar los datos semánticos desde una URI y la infor-

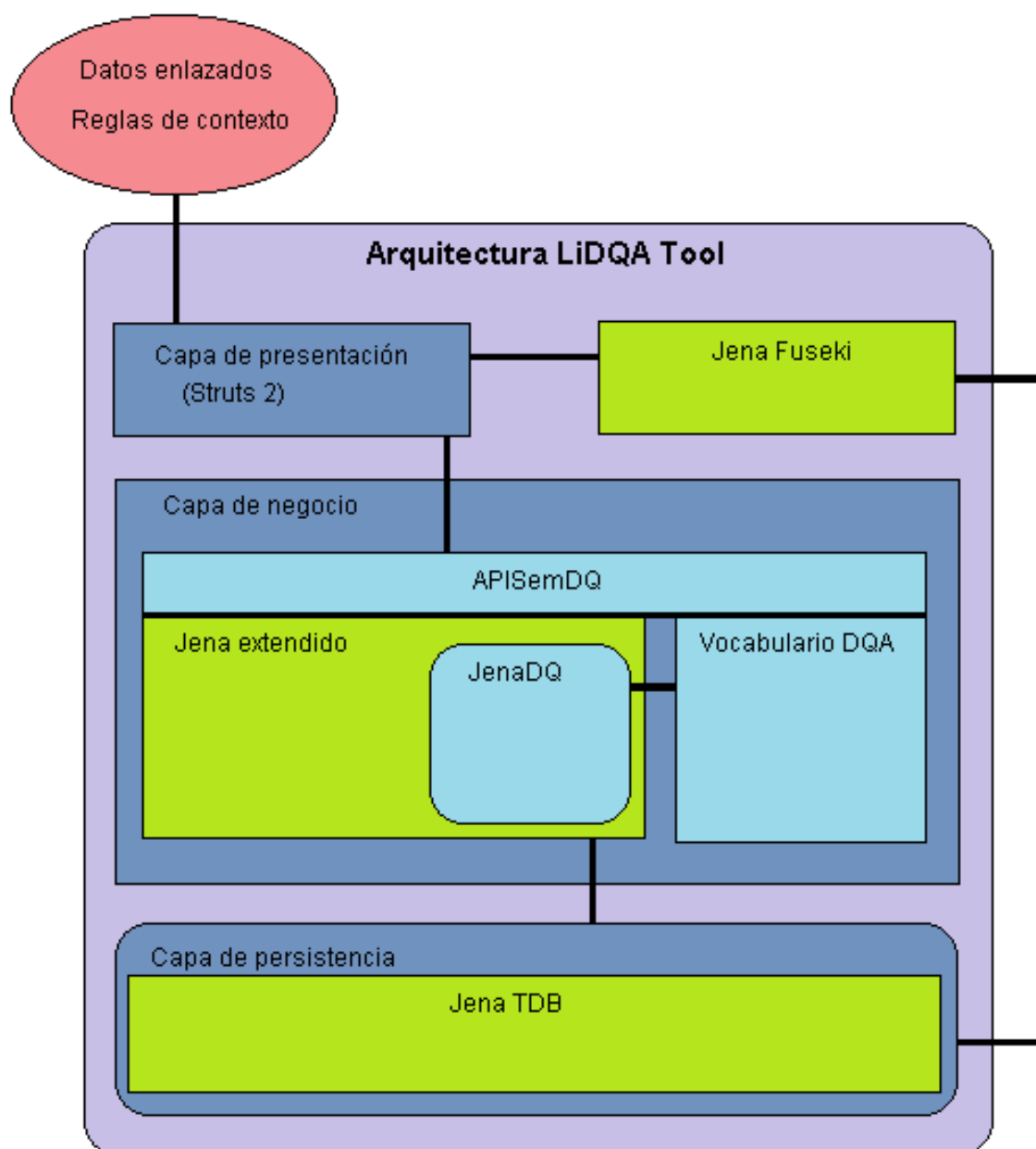


Figura 5.19: Arquitectura de LiDQA Tool

mación de la evaluación. Se pueden resumir en el diagrama de interacción correspondiente a la figura 5.22.

Iteración 8: fase de Elaboración (parte III)

En este caso se distinguen tres casos de uso con escenarios distintos:

- **Caso de Uso 3 Cargar reglas de contexto:** se contempla la carga de reglas de contexto

The screenshot shows a web browser window titled "LIDQA Tool" with the address bar displaying "http://hostname/SemanticAnalytics/". The page layout includes a top navigation bar with a search icon and a "Logos and banners" section. On the left, a sidebar menu lists "Single Assessment", "Assessment Plan", and "Model Comparison". The main content area is titled "Assessment Plan Form" and contains several input fields: "Your URI", "Your Endpoint", "Your Rules", and three fields labeled "Another relevant data goes here". Below these fields are two buttons: "< Add Assessment" and "< Execute Plan". A top navigation menu includes "Index", "Endpoint", "Javadoc", "About", and "...".

Figura 5.20: Esquema de formulario para la indicación de datos de evaluación

para aquellas dimensiones que las requieran. Existen dos tipos de reglas de contexto, (I) de valores contextuales que son siempre necesarias y (II) de uso, que se necesitan únicamente para evaluaciones de la DQD Completeness. En este caso de uso se controla esta necesidad a través de validadores.

- **Caso de Uso 4 Realizar evaluación:** en la figura 5.23 se puede observar la interacción entre los distintos actores durante la evaluación. Para entrar en detalle de cómo la aplicación Web realiza dichos cálculos, consúltese el diagrama 5.10. El diagrama 5.23 también es válido para la ejecución de planes de evaluación.
- **Caso de Uso 5 Mostrar resultados:** en la figura 5.23 se incluye esta interacción.

Iteración 8: fase de Elaboración (parte IV)

Para esta parte de la iteración se desarrollan los siguientes casos de Uso:

- **Caso de Uso 6 Establecer Plan de evaluación:** al establecer el plan de evaluación se crean las estructuras de datos necesarias y se redirecciona al usuario a la página correspondiente para que rellene el formulario de evaluación.

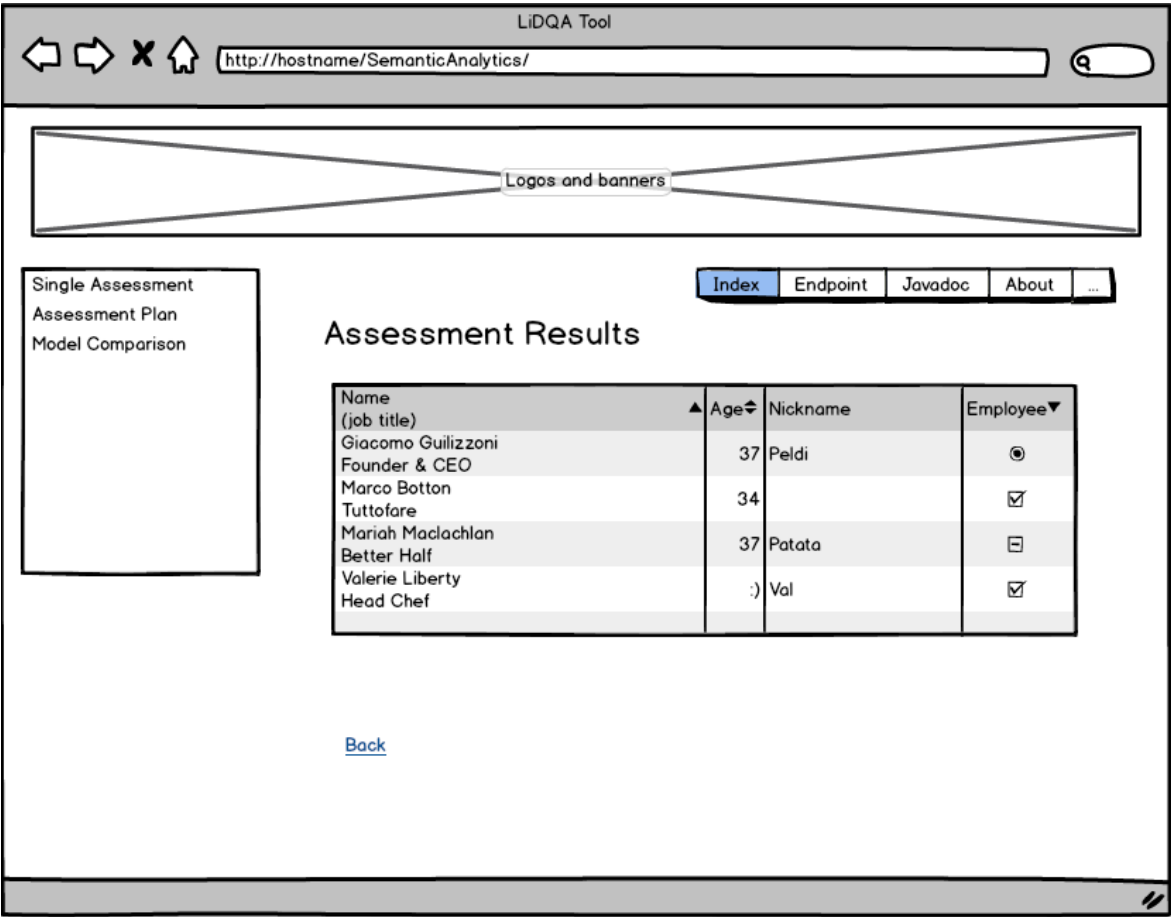


Figura 5.21: Esquema de presentación de resultados

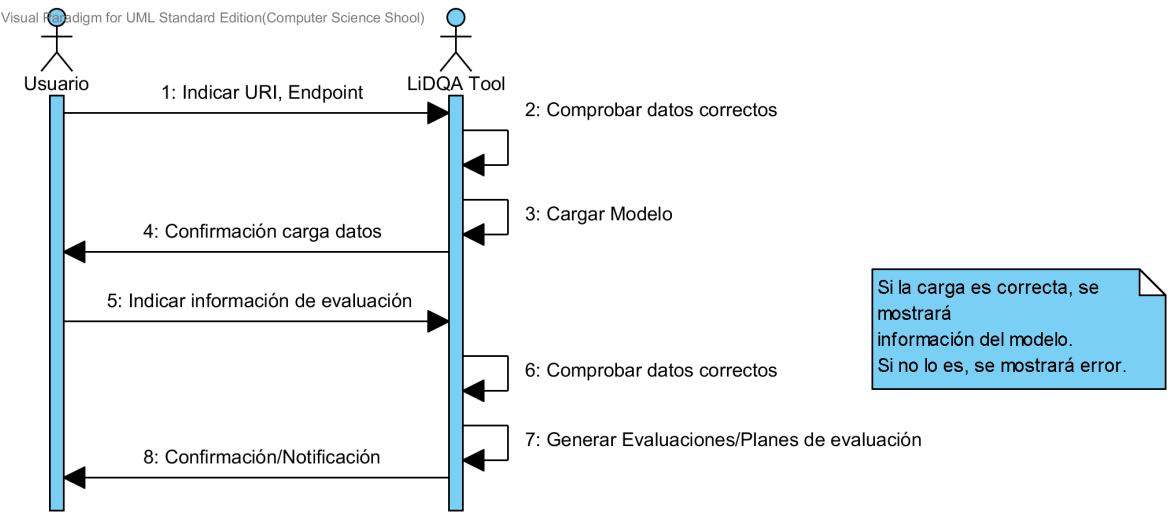


Figura 5.22: Casos de Uso de LiDQA Tool: 1, 2

- **Caso de Uso 7 Añadir evaluación:** En este caso de uso se rellenan los datos de la evaluación y se envían, redirigiendo a la misma página hasta que el usuario decida ejecutar el plan.

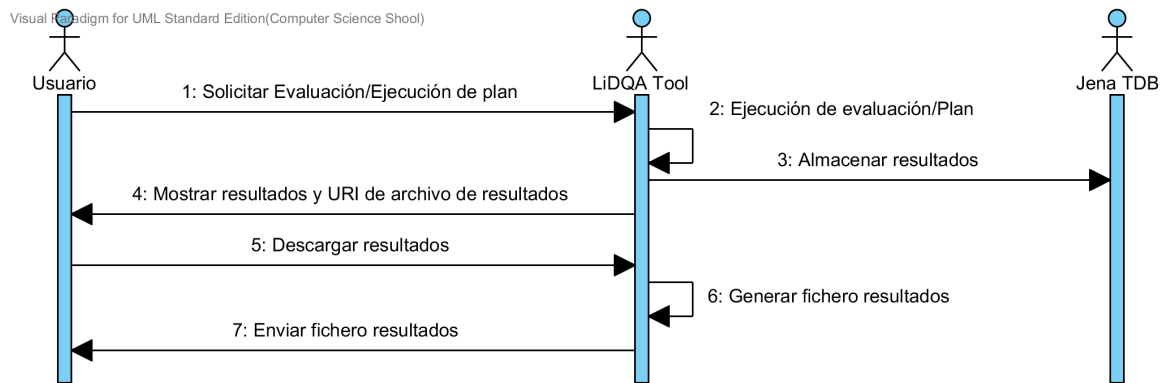


Figura 5.23: Casos de Uso de LiDQA Tool: 4, 8

- **Caso de Uso 8 Ejecutar plan de evaluación:** al igual que para el CDU 4, el diagrama resultante de la interacción entre el usuario y el sistema es el reflejado en la figura 5.23.

Iteración 8: fase de Construcción (parte V)

- **Caso de Uso 9 Descargar resultados de evaluación:** en la figura 5.23 se refleja la interacción para descargar los resultados tras la evaluación.
- **Caso de Uso 10 Comparar modelos de resultados:** tras cargar los modelos que se desean comparar, el sistema realiza dicha comparación y muestra al usuario los resultados, dando la opción de descargarlos, tal y como se indica en la figura 5.24

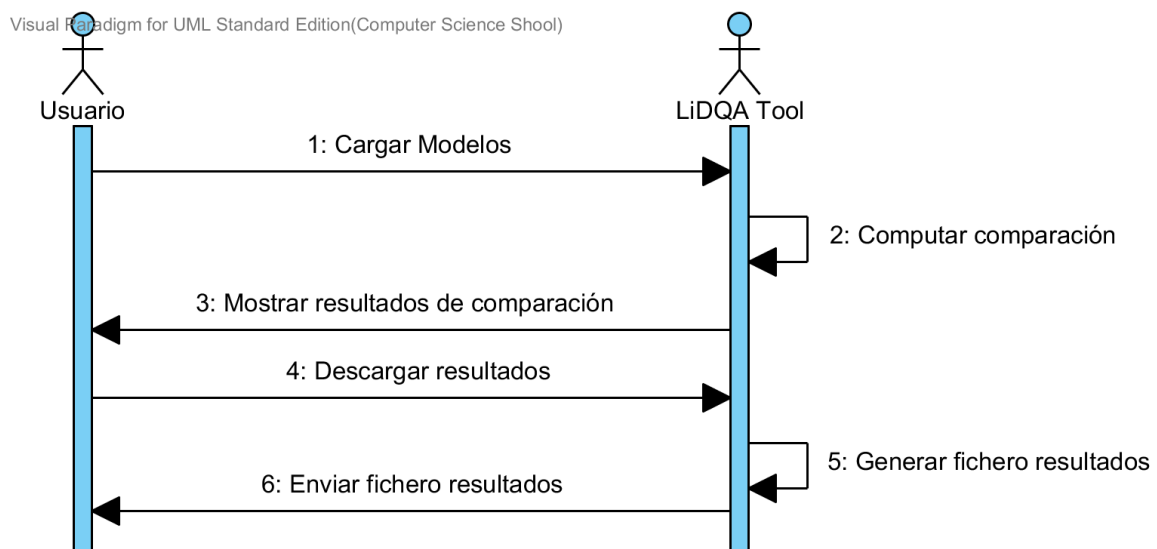


Figura 5.24: Caso de Uso de LiDQA Tool: 10

Iteración 8: fase de Construcción (parte VI)

- **Caso de Uso 12 Realizar consulta sobre el modelo cargado:** en este caso la consulta se hace sobre el modelo que se ha cargado de manera independiente. Debe rellenarse el campo con la *query* correspondiente y enviar la consulta. Estos resultados no se pueden descargar, puesto que es información que está implícita en el propio modelo cargado.
- **Caso de Uso 12 Realizar consulta sobre datos de evaluación:** para este caso de uso se incluye un enlace al *endpoint* que gestionará las consultas almacenadas. Esta parte queda relegada al subsistema *Fuseki*.

Iteración 8: fase de Transición (parte VII)

Finalmente, en la fase de transición, se llevan a cabo las siguientes tareas:

- **Mejoras en la interfaz de usuario:** utilizando elementos de lenguajes como HTML5, CSS3 y JavaScript, así como una buena gestión de los contenidos, se ha generado una interfaz amigable.
- **Generación de artefactos finales para la instalación e implantación:** tras finalizar el proyecto se ha generado el fichero `.war` para cargarse en cualquier servidor de aplicaciones compatible.

Se puede consultar el manual de usuario de la herramienta junto con ilustraciones en el Anexo A.

Para la generación de pruebas se ha hecho uso de *Selenium* (véase Sección 4.3). Un ejemplo del código generado por esta herramienta, exportado a `JUnit`, puede verse en el Anexo D.

5.4.2 Iteración 9: Entrega del PFC

Habiendo cumplido los objetivos propuestos para esta iteración, se han obtenido los siguientes artefactos:

- API de la extensión de Jena obtenida: JenaDQ.
- Vocabulario DQA para la Evaluación de Calidad de Datos. El vocabulario desarrollado puede consultarse en el Anexo C.
- Manual de usuario de la herramienta Web. El manual de usuario de LiDQA Tool puede consultarse en el Anexo A.
- Manual de despliegue de la herramienta. El manual de despliegue de LiDQA Tool puede consultarse en el Anexo B
- Memoria del PFC.

Durante la última iteración se ha finalizado la elaboración de toda la documentación necesaria para la entrega de este PFC. Todo el trabajo ha sido de redacción y documentación, habiendo acabado la labor de programación en la etapa anterior.

Capítulo 6

Conclusiones y Trabajo Futuro

EN este capítulo se van a exponer las conclusiones obtenidas tras el trabajo realizado en la elaboración de este PFC. Incluye además un apartado de propuestas de trabajo futuro, donde se describen posibles trabajos adicionales tomando como base el desarrollo de este proyecto, con el fin de abrir nuevas líneas de investigación o profundizar en el ámbito en el que se enmarca.

6.1 Conclusiones

A continuación se exponen las conclusiones obtenidas sobre los distintos objetivos parciales y una panorámica sobre el total de la elaboración del PFC.

6.1.1 Sobre la consecución de objetivos

Seguidamente se enumera la consecución de objetivos, que queda resumida en el cuadro 6.1.

Objetivo	Referencia	Conseguido
O1. Representación del contexto mediante reglas	Sección 5.2.1 y Sección 5.2.2	✓
O2. Desarrollo de un vocabulario para resultados	Sección 5.2.1 y Anexo C	✓
O3. Diseño e implementación de primitivas de evaluación	Sección 5.2.1 y Sección 5.2.2	✓
O4. Elección de un razonador de reglas	Sección 5.2.1	✓
O5. Desarrollo de una aplicación de prueba de concepto	Sección 5.4.1	✓

Cuadro 6.1: Consecución de Objetivos

O1. Representación del contexto de evaluación de calidad de datos

Mediante el uso de reglas se ha conseguido dar forma al contexto, permitiendo al usuario indicar parámetros o niveles de calidad así como atributos, en el caso de Completeness, que considere necesarios para el desarrollo de su tarea. Se puede consultar al respecto en las secciones 5.2.1 y 5.2.2.

O2. Desarrollo de un vocabulario para los resultados de evaluaciones de calidad de datos

El vocabulario desarrollado, que puede consultarse en el Anexo C, es suficientemente flexible y representativo como para ofrecer resultados de evaluaciones sencillas como de planes de evaluación (véase Sección 5.2.1).

O3. Diseño e implementación de las primitivas de evaluación de calidad de datos

Los algoritmos desarrollados demuestran ser eficaces y completos. Los resultados se ajustan a las decisiones sobre la medición tomadas en este proyecto. Se pueden consultar tanto algoritmos como todo el proceso de desarrollo en las secciones 5.2.1 y 5.2.2.

O4. Elección de un razonador de reglas para datos semánticos

Los razonadores incorporados en Jena permiten añadir nuevas triplas a los modelos de evaluación en función de las reglas que tomen como base, luego sirven al propósito de llevar a cabo evaluaciones considerando el contexto definido en esas reglas.

El razonador genérico de reglas cumple adecuadamente con la labor de comprobación de resultados y de valores para añadir nuevas triplas a los modelos de resultados. Se puede consultar este resultado en la Sección 5.2.1.

O5. Desarrollo de una aplicación de prueba de concepto

LiDQA Tool es una aplicación que engloba toda la funcionalidad desarrollada en JenaDQ, permitiendo a los usuarios realizar evaluaciones de forma dinámica y amigable, así como consultar los resultados de dichas evaluaciones y compararlos.

El desarrollo y finalización de LiDQA Tool puede consultarse en la Sección 5.4.1.

Todos los objetivos se han alcanzado satisfactoriamente. Se ha logrado extender un framework de desarrollo de tecnologías semánticas con el fin de facilitar una serie de operaciones necesarias para la gestión de la calidad de datos enlazados. Además, la extensión realizada aporta un diseño que permitirá incluir en el futuro nuevas dimensiones de calidad de manera ágil y sencilla. Se puede consultar un esquema de la arquitectura de Jena junto con JenaDQ, inspirado en [JEN14], en la figura 6.1.

La representación del contexto usando las reglas resulta efectiva y cumple perfectamente el objetivo propuesto: dos evaluaciones sobre los mismos datos en las mismas dimensiones pueden tener valores contextuales diferentes en función de las reglas. Esto permite que, para cada usuario, es posible saber exactamente qué datos son los que más valor le aportan.

Por otro lado se ha proporcionado un vocabulario que es susceptible de ser enriquecido a medida que la extensión exija nuevos términos en el momento que se añadan nuevas DQD, como se propondrá más adelante. Este vocabulario se obtuvo en las primeras etapas del desarrollo (véase Sección 5.2.1).

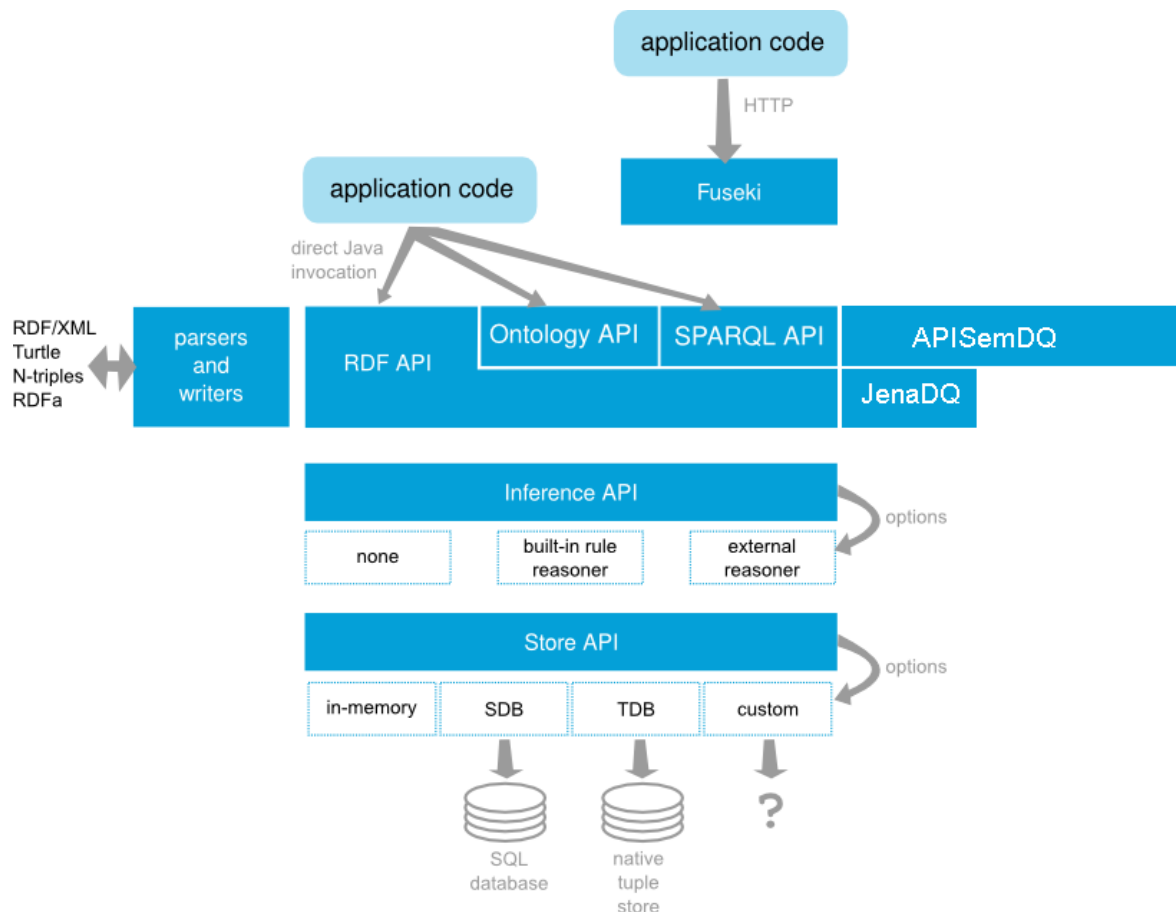


Figura 6.1: Jena integrada con JenaDQ. Inspirado en [JEN14]

La implementación de los mecanismos de evaluación sin lugar a dudas ha sido el conjunto de iteraciones que más esfuerzo ha requerido. La implementación de los mecanismos de evaluación de las dimensiones de calidad de datos es consistente con los modelos tradicionales y abre a su vez una nueva vía de investigación: replantear las diferentes dimensiones de calidad para datos enlazados se traduce en una necesidad para seguir progresando en esta dirección.

Se han abordado ambas perspectivas de calidad en las dimensiones elaboradas:

- *Meeting requirements* como perspectiva para elaborar las métricas.
- *Fitness for Use* como intérprete de esas métricas para el usuario.

Finalmente, el desarrollo de la prueba de concepto ha sido muy satisfactorio.

Como objetivo del PFC, *LiDQA Tool* ha puesto de manifiesto que el uso de la extensión de Jena realizada es viable y tiene multitud de aplicaciones en todos aquellos campos de negocio donde ya se estén usando datos semánticos.

6.1.2 Tecnologías utilizadas

Apache Jena es un framework con mucha comunidad tras él y en continuo desarrollo. Su arquitectura, como se ha podido comprobar en los Capítulos 3 y 4, es de las más completas que existen actualmente y su tendencia es seguir creciendo.

Al estar desarrollado en Java, el uso de esta tecnología puede extenderse a cualquier otro framework (en este caso ha sido Struts 2) que igualmente utilice este lenguaje de programación. No obstante cabe destacar que una de las principales ventajas de las tecnologías semánticas es la interoperabilidad que ofrecen a la hora de integrar aplicaciones heterogéneas, con lo cual sería posible igualmente integrar Jena y esta extensión con otras aplicaciones realizadas en otros lenguajes, lo que constituye a su vez una gran ventaja.

6.2 Propuestas de trabajo futuro

Adición de nuevas DQD

Incrementar el número de dimensiones de calidad sería una labor sumamente interesante de cara a completar un Modelo de Calidad de Datos. En este proyecto el desarrollo se ha centrado en dos de ellas, que han sido consideradas de crucial importancia para el contexto en el que se enmarcan, pero existen otras muchas cuya adición aportaría mucho valor a la extensión de Jena que se ha realizado.

Repositorio de reglas contextuales

Crear y mantener un repositorio de reglas de contexto es una opción tremendamente interesante desde el punto de vista de la calidad de datos. De esta manera se podrían incluir en los archivos de evaluación referencias ontológicas a estas reglas con el fin de mejorar la interconexión entre conceptos.

Vocabulario de reglas de contexto

Realizar un vocabulario para las reglas también es una opción de interés, puesto que así las reglas también serían susceptibles de ser publicadas como cualquier otro conjunto de datos semánticos y vinculadas con las evaluaciones realizadas.

Formalizar y estandarizar la ontología de evaluaciones

La manera más productiva de contribuir al presente trabajo es extender las dimensiones de calidad de datos. Esto requerirá de un refinamiento del vocabulario de evaluación propuesto en este PFC.

En el momento que se tenga un vocabulario para las reglas de contexto sería conveniente tratar con esta ontología para hacer referencia a las reglas utilizadas en cada evaluación, con el fin de tener unos datos de evaluación más concretos y ricos en contenido.

Refinamiento de los umbrales contextuales

Actualmente los umbrales contextuales se delimitan mediante literales numéricos. Una

opción muy interesante a la hora de contribuir a este proyecto sería realizar un estudio acerca de dichos umbrales y su representación en las reglas. Utilizando principios de conjuntos difusos pueden evitarse definir estos umbrales y hacer del análisis contextual algo más cercano a la comprensión humana.

Poder codificar umbrales difusos en reglas entendibles por Jena sería un avance muy significativo en el ámbito de la evaluación contextual de calidad de datos, en general y de datos enlazados en particular.

Comparación de contextos

Partiendo de dos evaluaciones contextuales sobre conjuntos de datos, si éstas son similares implicaría que los contextos de uso también lo son. Comparar estos contextos requeriría una aportación que englobara algunos de los apartados anteriores: definición de reglas, repositorio de reglas e incluso la creación de un razonador específico para ello.

Paralelización de evaluaciones. Uso de tecnologías Map-Reduce o Big Data

La cantidad de datos que puede albergar un servidor es potencialmente alta. Contenedores semánticos como los utilizados en los ejemplos (DBpedia) almacenan del orden de cientos de millones de triplas.

Si a esto le añadimos una búsqueda en profundidad, la complejidad computacional de los cálculos crece exponencialmente, tal y como se puede ver en el Capítulo 5.

Es por esto por lo que surge como una necesidad el uso de tecnologías para paralelizar el análisis de calidad de datos cuando existen búsquedas en profundidad o se entiende que dicho análisis requerirá de volúmenes de información muy elevados.

Cuando se trata con tanta información, surgen nombres como *Map-Reduce* o *Big Data* para facilitar las labores de cálculo.

Quizás una de las propuestas más interesantes que se puede ofrecer es la de incluir tecnologías *Big Data* con el fin de tratar con estos volúmenes de datos de una manera más acertada.

6.3 Opinión personal

En este apartado abordaré una serie de opiniones en relación a la elección y desarrollo del PFC.

6.3.1 Elección del PFC

En mi opinión, las tecnologías semánticas están cada vez más implantadas en Internet. Junto con los datos abiertos, las empresas y los investigadores se están percatando del potencial que encierra estos nuevos movimientos. De la misma forma, la Web Semántica y sus tecnologías asociadas se integran en cada vez más lugares. Empresas tales como Facebook

o BBC ya están haciendo uso de tecnologías semánticas considerándolas parte vital de sus procesos de negocio.

Cuando elegí este PFC no tenía en absoluto claro los conceptos pero sí que hubo una serie de palabras clave que me persuadieron para sumergirme en este ámbito. Datos enlazados, semántica y calidad son conceptos que o bien están tomando fuerza o bien son ya muy valorados por las empresas y los consumidores.

Pienso que tanto la Web Semántica como la Calidad de los Datos tienen mucho que hacer tanto juntas como por separado, y es por ello por lo que no me arrepiento para nada de la elección que tomé.

6.3.2 Documentación

Durante el desarrollo de este PFC he encontrado suficiente información como para que el desarrollo posterior haya surgido sin complicaciones. No obstante no son temas, bajo mi punto de vista, que a día de hoy estén en boca de todos, pero sí pienso que lo estarán en unos años, a medida que las tecnologías avancen y demuestren el potencial del que disponen.

En cuanto a la documentación, se encuentra poco material que relacione Web Semántica o LD con Calidad de Datos en comparación con lo que se encuentra por separado de cada uno de estos temas. Es por ello que pienso que hay cierto movimiento pero que en efecto, como indicaba Fürber en [FH11], se le ha prestado poca atención a evaluar la calidad de los datos semánticos.

6.3.3 Elaboración

Como autor del PFC esta ha sido la parte en la que más he disfrutado y aprendido. El poder aunar en un solo proyecto conceptos tales como Álgebra de Conjuntos, Programación Declarativa e Inteligencia Artificial por una parte con Calidad por la otra, ha resultado muy satisfactorio y enriquecedor. Se ha dedicado mucho tiempo a la elaboración tanto de la extensión de Jena como de la aplicación Web, pero no ha resultado una tarea que haya desmotivado, ni mucho menos: a medida que avanzaba en conocimientos y en implementación, crecían las ilusiones por tener algo consistente y funcional y finalmente llegaron los resultados.

Cuando personalmente se disfruta tanto de una tarea significa que la elección ha sido correcta y es, en definitiva, cuando más y mejor se aprende.

6.3.4 Resultados finales y trabajo futuro

Estoy muy satisfecho con los resultados finales, tanto de JenaDQ como de LiDQA Tool. Pienso que han sido los frutos de trabajo que hasta ahora me han traído más satisfacción en el ámbito académico. Es por ello que no descarto continuar en dicho ámbito con este trabajo.

Como se ha comentado antes, existen muchas vías por donde continuar el presente proyecto y prometen mucho al tratar con tecnologías vanguardistas como Big Data. Pienso que pueden lograrse objetivos tan útiles como interesantes si se sigue investigando en este campo.

ANEXOS

Anexo A

Linked Data Quality Assessment Tool: Manual de Usuario

Este capítulo está destinado a servir de manual de usuario para la utilización de la aplicación Web desarrollada.

A.1 Pantalla principal

La ventana principal está compuesta por tres áreas, como se puede ver en la figura A.1:

- Un área de imágenes y logotipos, situada en toda la parte superior.
- Un área de menú, en el margen izquierdo.
- El área de trabajo, situada en la parte central. Esta parte es donde se muestra todo el contenido incluido en la Web y donde se desarrolla toda la funcionalidad. Incluye una pequeña barra de enlaces en la parte superior derecha a través de la cual se puede acceder a diversas secciones y servicios.

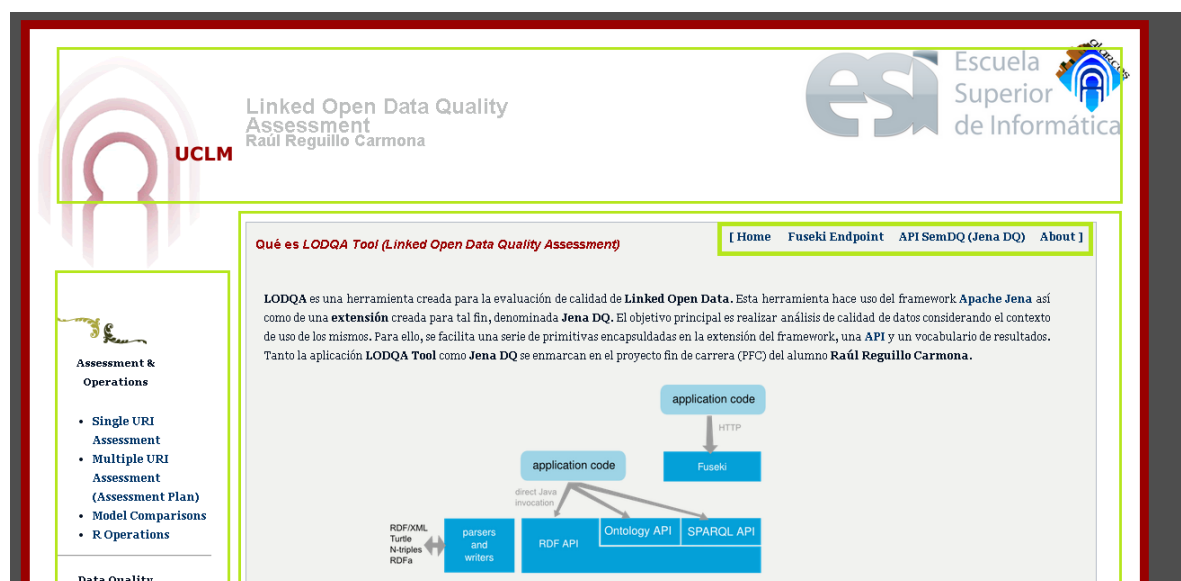
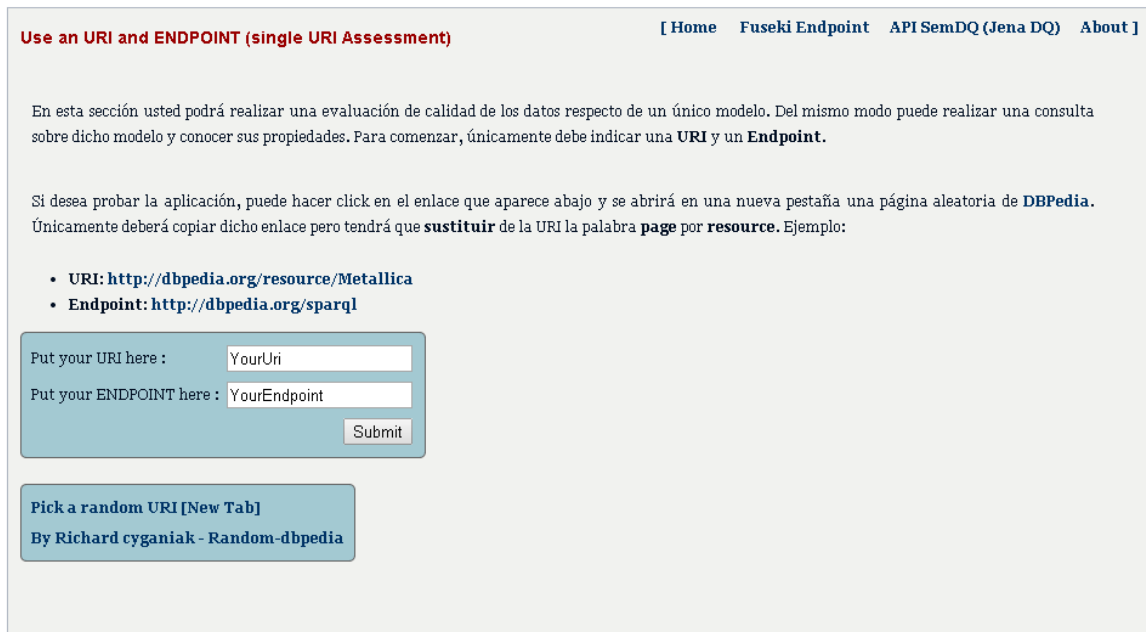


Figura A.1: Pantalla principal

A.2 Realizar una evaluación

A.2.1 Evaluación simple

Para realizar una evaluación basta con pinchar en el enlace del margen izquierdo **Single URI Assessment**. En ese momento el navegador redireccionará a la ventana que ilustra la figura A.2.



Use an URI and ENDPOINT (single URI Assessment) [Home Fuseki Endpoint API SemDQ (Jena DQ) About]

En esta sección usted podrá realizar una evaluación de calidad de los datos respecto de un único modelo. Del mismo modo puede realizar una consulta sobre dicho modelo y conocer sus propiedades. Para comenzar, únicamente debe indicar una **URI** y un **Endpoint**.

Si desea probar la aplicación, puede hacer click en el enlace que aparece abajo y se abrirá en una nueva pestaña una página aleatoria de **DBPedia**. Únicamente deberá copiar dicho enlace pero tendrá que **sustituir** de la URI la palabra **page** por **resource**. Ejemplo:

- URI: <http://dbpedia.org/resource/Metallica>
- Endpoint: <http://dbpedia.org/sparql>

Put your URI here :

Put your ENDPOINT here :

By Richard cyganiak - Random-dbpedia

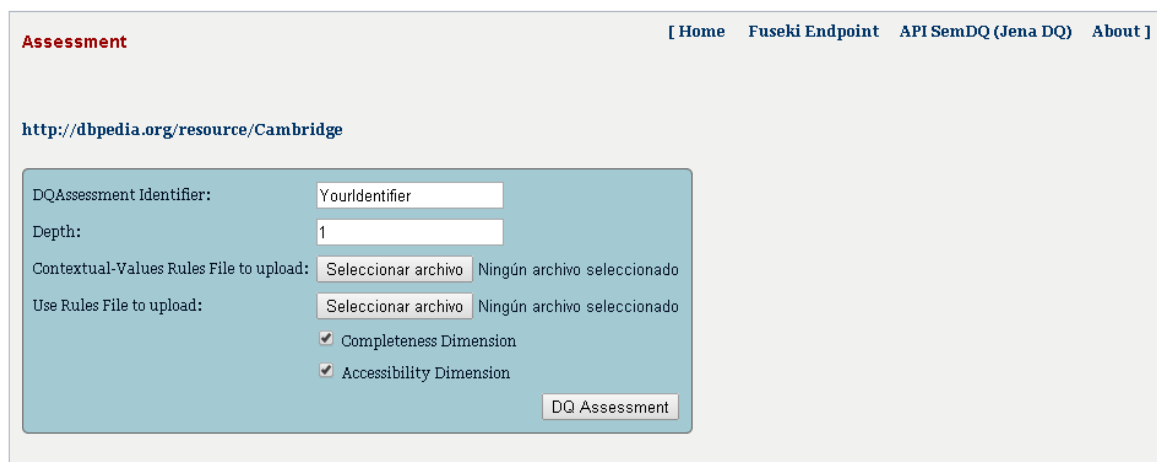
Figura A.2: Evaluación simple (I)

En esta ventana se introducirán los datos pertinentes, URI y *endpoint*. También se facilita un generador de URI aleatorias y el *endpoint* de DBPedia.

Una vez se hayan facilitado los datos, el navegador los cargará y pasará a la pantalla que se indica en la figura A.3.

En esta ventana se pueden llevar a cabo dos operaciones:

1. Cargar los datos restantes para realizar una evaluación, consistentes en:
 - Un **identificador** de la evaluación.
 - Una **profundidad** a la que se explorará el grafo (solamente para análisis de Completeness).
 - Archivos de reglas contextuales:
 - **Reglas de valores contextuales**: que definirán el contexto del usuario (siempre son requeridas).
 - **Reglas de uso**: que indicarán para qué propiedades se esperan valores no nulos (solamente para análisis de Completeness).
 - Selección de las **dimensiones** a evaluar.



Assessment [Home Fuseki Endpoint API SemDQ (Jena DQ) About]

<http://dbpedia.org/resource/Cambridge>

DQAssessment Identifier:

Depth:

Contextual-Values Rules File to upload: Ningún archivo seleccionado

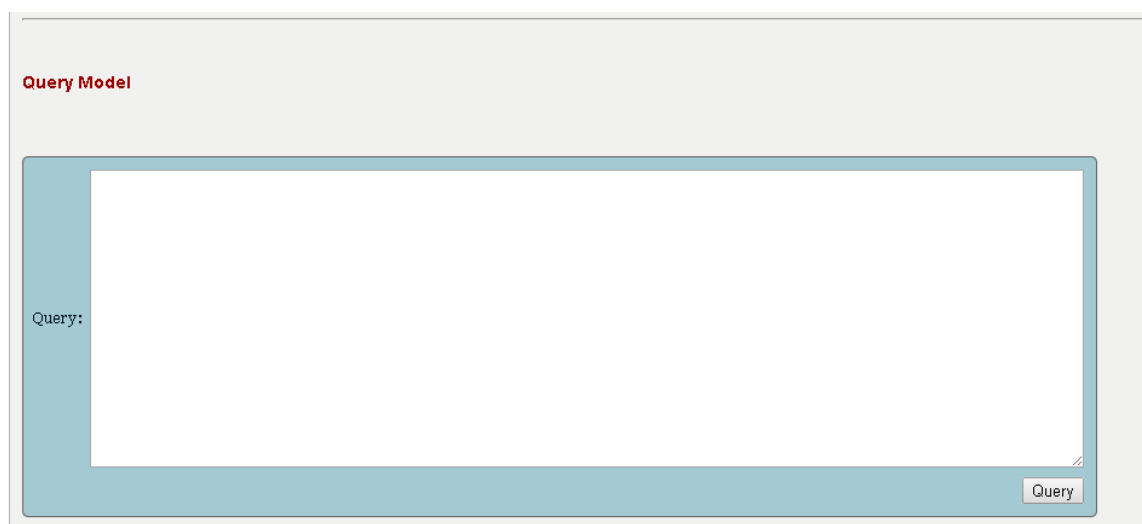
Use Rules File to upload: Ningún archivo seleccionado

☒ Completeness Dimension

☒ Accessibility Dimension

Figura A.3: Evaluación simple (II)

2. Realizar una consulta sobre los datos recientemente cargados, como se muestra en la figura A.4.



Query Model

Query:

Figura A.4: Consulta sobre los datos cargados

Todos los campos presentan validación tanto de HTML5 como validación contra el servidor, de manera que avisará si se introducen campos erróneos, como se puede ver en las figuras A.5 y A.6.

• **URI:** <http://dbpedia.org/resource/Metallica>

• **Endpoint:** <http://dbpedia.org/sparql>

Put your URI here :

Put your ENDPOINT here :

! Utiliza un formato que coincida con el solicitado

Pick a random URI [New Tab]
By Richard cyganiak - Random-dbpedia

Figura A.5: Validación: HTML5

DQAssessment Identifier:

Depth:

Check the correct rules files

Contextual-Values Rules File to upload: Ningún archivo seleccionado

Check the correct rules files

Use Rules File to upload: Ningún archivo seleccionado

At least one dimension has to be selected

An assessment identifier is required

☒ Completeness Dimension

☒ Accessibility Dimension

DQ Assessment

Figura A.6: Validación: contra servidor

Finalmente, tras haber solicitado la evaluación, la aplicación mostrará los resultados tal y como muestra la imagen A.7.

Como se puede comprobar, los resultados presentan tanto valores para las métricas como los valores contextuales generados según las reglas. Igualmente indica qué dimensión se está tratando y dentro de ésta, qué métrica es la que se ha calculado (en caso de haberla).

Results of the DQ Assessment [\[Home Fuseki Endpoint API SemDQ \(Jena DQ\) About \]](#)

DQ Dimension	Dimension metric	Value	Contextual Value
[Accessibility, Completeness]	http://dbpedia.org/resource/Cambridge	-1.0	Assessment Data
Accessibility	Interlinking	0.2954545454545454	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#LOW
Completeness	Lvl 0	1.0	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#HIGH
Completeness	Lvl 1	0.7871287128712872	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#HIGH

URI for results (fuseki Server): http://212.122.105.160:3030/db/AssessmentPlan_1407425844372

Download results

RDF/XML N3

Back






Figura A.7: Evaluación simple (y III). Resultados.

Se facilita una URI gracias a la cual es posible acceder a los resultados de la evaluación a través de protocolo HTTP en el *endpoint* de Fuseki, enlazado en la barra de enlaces superior.

Los resultados pueden ser descargados en este preciso momento en diversos formatos, haciendo click en la sección correspondiente.

A.2.2 Plan de Evaluación

Para llevar a cabo un plan de evaluación basta con pinchar en el enlace del margen izquierdo **Multiple URI Assessment**. En ese momento el navegador redireccionará a la ventana que ilustra la figura A.8.

Al igual que en el caso de las evaluaciones simples, se debe rellenar un formulario con los mismos campos. En este caso, la URI y el *endpoint* se indican junto con el resto de campos y no es posible realizar consultas sobre los modelos que se cargan.

Para añadir los datos de la evaluación, se debe pulsar el botón *Add Assessment*. Esta acción conducirá de nuevo a la página actual donde se podrá volver a rellenar el formulario y añadir evaluaciones de forma indefinida.

Cabe destacar dos aspectos en este punto:

- Se pueden nombrar las diferentes evaluaciones con el mismo identificador o con distinto, según interese el formato final de los archivos de datos enlazados.
- Se pueden realizar evaluaciones sobre distintos *endpoints* o sucesivamente sobre los

Si desea probar la aplicación, puede hacer click en el enlace que aparece abajo y se abrirá en una nueva pestaña una página aleatoria de **DBPedia**. Únicamente deberá copiar dicho enlace pero tendrá que **sustituir** de la URI la palabra **page** por **resource**. Ejemplo:

- URI: http://dbpedia.org/resource/Iron_Maiden
- Endpoint: <http://dbpedia.org/sparql>

URI:

Endpoint:

DQAssessment Identifier:

Depth:

Contextual-Values Rules File to upload: Ningún archivo seleccionado

Use Rules File to upload: Ningún archivo seleccionado

☒ Completeness Dimension

☒ Accessibility Dimension

Figura A.8: Plan de evaluación (I)

mismos conjuntos de datos. No hay restricción al respecto.

Una vez finalizada la generación del plan, basta con pulsar el botón *Execute DQ Assessment Plan* para obtener los resultados, como se muestra en la figura A.9.

Results of the DQ Assessment [\[Home Fuseki Endpoint API SemDQ \(Jena DQ\) About \]](#)

DQ Dimension	Dimension metric	Value	Contextual Value
[Accessibility]	http://dbpedia.org/resource/Cambridge	-1.0	Assessment Data
Accessibility	Interlinking	0.29261363636363635	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#LOW
[Accessibility, Completeness]	http://dbpedia.org/resource/Gladiator_(1992_soundtrack)	-1.0	Assessment Data
Accessibility	Interlinking	0.6785714285714286	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#MED
Completeness	Lvl 0	1.0	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#HIGH
Completeness	Lvl 1	0.8837209302325582	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#HIGH

URI for results (fuseki Server): http://212.122.105.160:3030/db/AssessmentPlan_1407427851625

Download results

Back

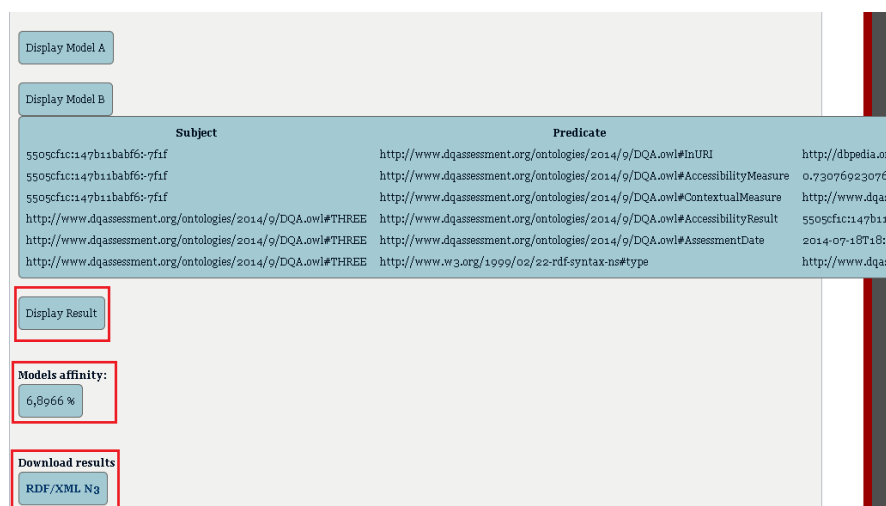
W3C HTML 4.01 ☒ W3C CSS ☒

Figura A.9: Plan de evaluación (y II). Resultados

Como se puede comprobar, en la tabla de resultados se indican qué URI están asociadas a qué dimensiones, así como sus resultados. Al igual que en la evaluación simple, se facilita la URI del archivo de resultados y se permite la descarga de los mismos directamente.

A.3 Comparación de Modelos

Para llevar a cabo una comparación de resultados basta con pinchar en el enlace **Model Comparisons**, lo que nos llevará a la pantalla de comparación. En ésta, se podrán cargar dos modelos y al pulsar sobre el botón de envío, se obtendrán los resultados como se ilustra en la figura A.10.



Subject	Predicate	
5505fc1c147b11babf6c-7f1f	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#InURI	http://dbpedia.org
5505fc1c147b11babf6c-7f1f	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#AccessibilityMeasure	0.73076923076
5505fc1c147b11babf6c-7f1f	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#ContextualMeasure	http://www.dqas
http://www.dqassessment.org/ontologies/2014/9/DQA.owl#THREE	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#AccessibilityResult	5505fc1c147b11
http://www.dqassessment.org/ontologies/2014/9/DQA.owl#THREE	http://www.dqassessment.org/ontologies/2014/9/DQA.owl#AssessmentDate	2014-07-18T18:0
http://www.dqassessment.org/ontologies/2014/9/DQA.owl#THREE	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.dqas

Display Result

Models affinity:
6,8966 %

Download results
RDF/XML N3

Figura A.10: Comparación de resultados

En esta pantalla se podrán desplegar los modelos cargados y el modelo resultado de la comparación. Se mostrará a su vez un porcentaje de afinidad entre ambos modelos y se permitirá la descarga como en los casos anteriores.

A.4 Gestión de errores

La aplicación gestiona los errores mediante mensajes derivados de las excepciones ocurridas. Así, cuando algo salga realmente mal, mostrará un mensaje de error en la página correspondiente, pidiendo que se revisen los parámetros (véase figura A.11).

A.5 Contenido

La Web expone para cada operación suficiente contenido explicativo como para que su usabilidad quede garantizada. De igual modo, explica la teoría subyacente definiendo los conceptos más importantes y sugiriendo algunos ejemplos, como se puede ver en la figura A.12.

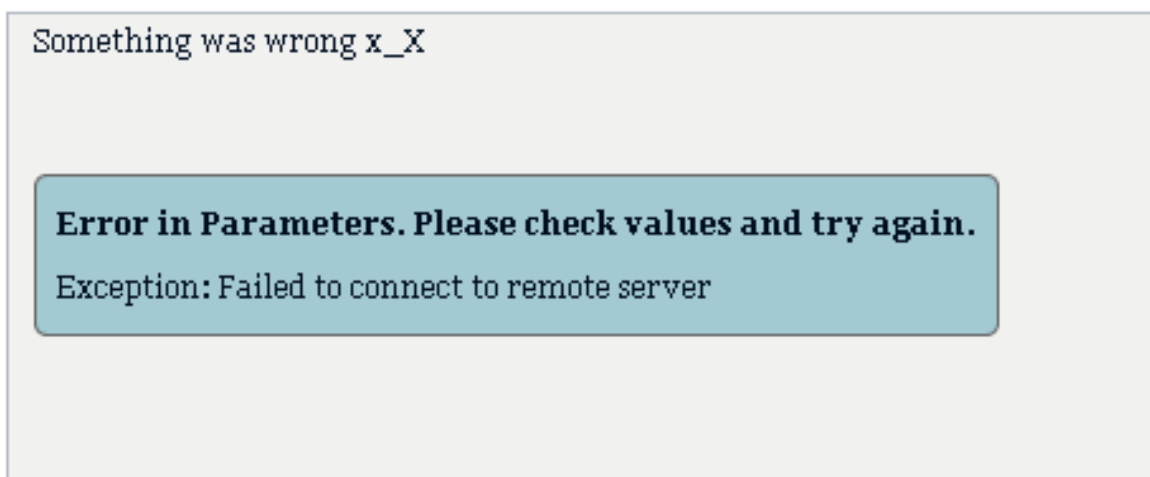


Figura A.11: Gestión de errores

En el presente trabajo, se utilizan dos tipos de reglas contextuales:

- **Reglas de Uso:** Aquellas que definen qué propiedades se esperan en los conjuntos de datos.
- **Reglas de Contexto (valores de contexto):** Definen rangos de valores respecto de propiedades en el conjunto de datos.

Mostrar ejemplo de reglas de uso

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
[rule1: ( ?x rdf:type ?y ) -> print(?x 'has RDF TYPE property' ?y) ]
```

Mostrar ejemplo de reglas de valores de contexto

Estas reglas son aplicadas a las dos dimensiones de calidad implementadas en este proyecto: **Accessibility** y **Completeness**. Si desea conocer más información acerca de los detalles de cada dimensión, diríjase a la sección correspondiente.

W3C HTML 4.01 CSS

Figura A.12: Web de contenido

A.6 Acceso a los datos a través de URI y HTTP

Para acceder a los datos publicados de la evaluación, basta con hacer click en el enlace **Fuseki Endpoint** situado en la barra de enlaces, lo que automáticamente dirigirá a la página del servidor de triplas, indicado en la figura A.13.

A continuación se debe hacer click en el enlace **Control Panel** y tras ello seleccionar la base de datos existente. Finalmente se accederá a una nueva página donde se podrán realizar

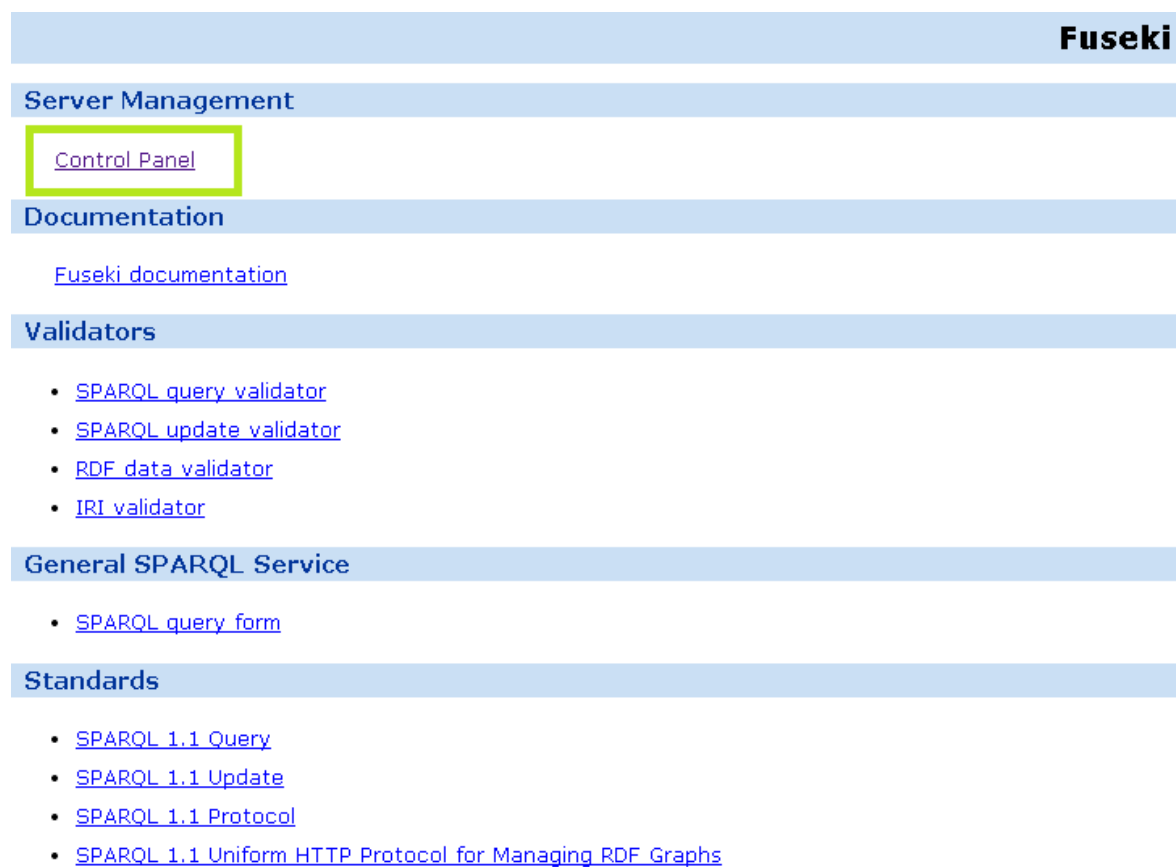


Figura A.13: Fuseki: Servidor de triplas de resultados de evaluación

operaciones de consulta y actualización, como muestra la figura A.14.

Fuseki Query

Dataset: /db

SPARQL Query

Output: Text ▼

If XML output, add XSLT style sheet (blank for none): ▼

☐ Force the accept header to text/plain regardless.

Get Results

SPARQL Update

Perform update

Figura A.14: Fuseki: Página de consultas

Anexo B

Manual de Instalación de LiDQA Tool

En este apéndice se va a relatar cómo instalar en un sistema compatible la herramienta Web desarrollada.

B.1 Instalación y Configuración de Apache Tomcat 7

1. Descargar Apache Tomcat: <http://tomcat.apache.org>.
2. Descomprimir en un directorio.
3. Configurar las siguientes variables de entorno:
 - CATALINA_HOME: ruta del directorio donde se ha descomprimido.
 - JAVA_HOME: ruta de la instalación Java.
4. Configurar un rol `manager-gui` que habilite la administración a través de interfaz Web, así como un usuario y contraseña asignado a ese rol. El archivo que se debe editar se encuentra en el directorio Tomcat `conf/tomcat-users.xml`. Ver listado B.1.
5. Una vez configurado, se arranca el servidor a través de los ejecutables:
 - Desde un sistema Windows: `bin/startup.bat`.
 - Desde un sistema GNU/Linux: `bin/startup.sh`.
6. En un navegador, introducir la dirección `localhost:8080`
7. Si todo es correcto, aparecerá la ventana de administración de Tomcat. Ver figura B.1.

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="YourID" password="y0urP4ssw0rd" roles="manager-gui"/
  >
</tomcat-users>
```

Listado B.1: Edición del archivo de configuración

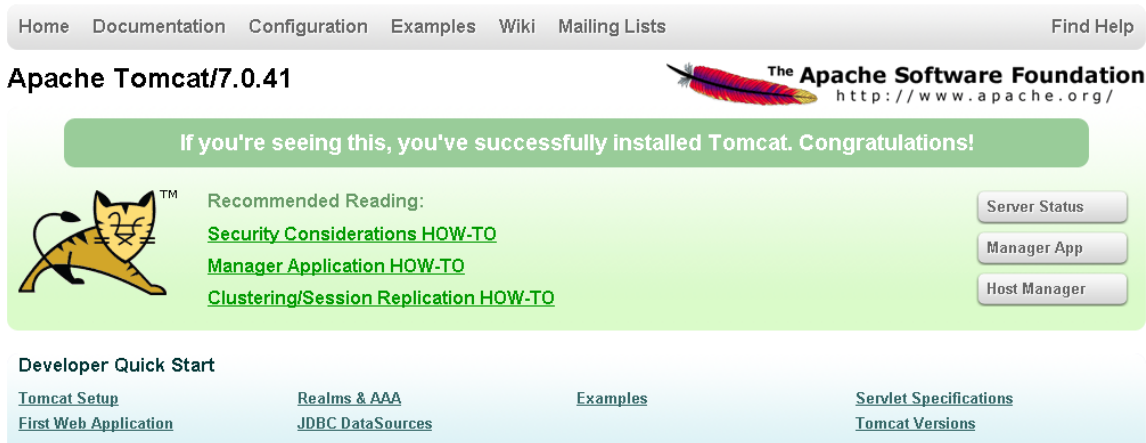


Figura B.1: Interfaz principal de Apache Tomcat

B.2 Despliegue de la herramienta

Para realizar el despliegue de la herramienta basta con entrar en la administración de Tomcat y en la sección correspondiente, seleccionar el archivo .war adjunto a este PFC, tal y como aparece en la figura B.2.

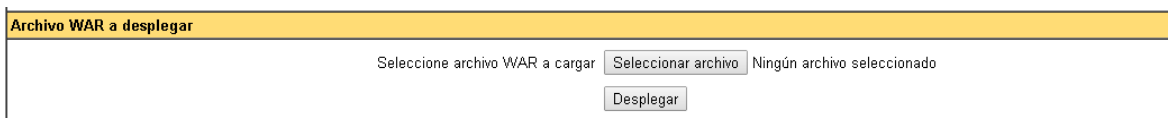


Figura B.2: Despliegue de la aplicación

B.3 Cambio de parámetros por defecto

Existen dos parámetros que el archivo .war compilado trae por defecto:

- Ruta por defecto para el almacenamiento de resultados de evaluación.
- URI por defecto para los resultados de evaluación.

Para cambiar estos parámetros basta con seguir estos pasos:

1. Cargar el proyecto SemanticAnalytics adjunto en el PFC en Eclipse.
2. Abrir el archivo TDBNames.java, que se podrá encontrar en la carpeta Java Resources → ProjectSources → NAMES package.
3. Editar las variables:
 - DIR: ruta para el almacenamiento de resultados.
 - URIBASE: URI por defecto para los resultados. A esta URI se le añadirá un código numérico posterior que la hará unívoca en el sistema, sin que el desarrollador tenga que tocar nada más en este aspecto.

4. Exportar el proyecto como `.war`.
5. Finalmente, cargar el archivo en Tomcat, para lo cual se deben seguir los pasos enumerados en la sección anterior.

Anexo C

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" >
5  <!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#"
10  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#">
15  <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#" />
20  <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
25  a jena extension to support data quality context-aware assessment of
    linked data</Literal>
  </Annotation>
  <Declaration>
    <Class IRI="AccessibilityAssessment" />
  </Declaration>
30  <Declaration>
    <Class IRI="CompletenessAssessment" />
  </Declaration>
  <Declaration>
    <ObjectProperty IRI="AccessibilityMeasure" />
35  </Declaration>
  <Declaration>
    <ObjectProperty IRI="AccessibilityResult" />
  </Declaration>
  <Declaration>
```

```

40      <ObjectProperty IRI="AssessmentDate"/>
    </Declaration>
    <Declaration>
      <ObjectProperty IRI="CompletenessMeasure"/>
    </Declaration>
45    <Declaration>
      <ObjectProperty IRI="CompletenessResult"/>
    </Declaration>
    <Declaration>
      <ObjectProperty IRI="ContextualMeasure"/>
50    </Declaration>
    <Declaration>
      <ObjectProperty IRI="Identifier"/>
    </Declaration>
    <Declaration>
55      <ObjectProperty IRI="InLevel"/>
    </Declaration>
    <Declaration>
      <ObjectProperty IRI="InURI"/>
    </Declaration>
60    <Declaration>
      <DataProperty IRI="AccessibilityResult"/>
    </Declaration>
    <Declaration>
      <DataProperty IRI="AssessmentDate"/>
65    </Declaration>
    <Declaration>
      <DataProperty IRI="CompletenessResult"/>
    </Declaration>
    <SubObjectPropertyOf>
70      <ObjectProperty IRI="AccessibilityMeasure"/>
      <ObjectProperty IRI="AccessibilityResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
      <ObjectProperty IRI="AccessibilityResult"/>
75      <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
      <ObjectProperty IRI="CompletenessMeasure"/>
      <ObjectProperty IRI="CompletenessResult"/>
80    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
      <ObjectProperty IRI="CompletenessResult"/>
      <ObjectProperty abbreviatedIRI="owl:topObjectProperty"/>
    </SubObjectPropertyOf>
85    <SubObjectPropertyOf>
      <ObjectProperty IRI="ContextualMeasure"/>
      <ObjectProperty IRI="AccessibilityResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
90      <ObjectProperty IRI="ContextualMeasure"/>
      <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
      <ObjectProperty IRI="InLevel"/>
95      <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>

```

```

    <ObjectProperty IRI="InURI"/>
    <ObjectProperty IRI="AccessibilityResult"/>
100 </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="InURI"/>
        <ObjectProperty IRI="CompletenessResult"/>
    </SubObjectPropertyOf>
105 <ObjectPropertyDomain>
    <ObjectProperty IRI="AccessibilityMeasure"/>
    <Class IRI="AccessibilityAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
110 <ObjectProperty IRI="AccessibilityResult"/>
    <Class IRI="AccessibilityAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="AssessmentDate"/>
115 <Class IRI="AccessibilityAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="AssessmentDate"/>
    <Class IRI="CompletenessAssessment"/>
120 </ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="CompletenessMeasure"/>
    <Class IRI="CompletenessAssessment"/>
</ObjectPropertyDomain>
125 <ObjectPropertyDomain>
    <ObjectProperty IRI="CompletenessResult"/>
    <Class IRI="CompletenessAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
130 <ObjectProperty IRI="ContextualMeasure"/>
    <Class IRI="AccessibilityAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="ContextualMeasure"/>
135 <Class IRI="CompletenessAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="Identifier"/>
    <Class abbreviatedIRI="owl:Thing"/>
140 </ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="InLevel"/>
    <Class IRI="CompletenessAssessment"/>
</ObjectPropertyDomain>
145 <ObjectPropertyDomain>
    <ObjectProperty IRI="InURI"/>
    <Class IRI="AccessibilityAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
150 <ObjectProperty IRI="InURI"/>
    <Class IRI="CompletenessAssessment"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty IRI="AccessibilityMeasure"/>
155 <DataExactCardinality cardinality="1">

```

```

        <DataProperty IRI="AccessibilityResult"/>
        <Datatype abbreviatedIRI="xsd:double"/>
    </DataExactCardinality>
</ObjectPropertyRange>
160 <ObjectPropertyRange>
    <ObjectProperty IRI="AccessibilityResult"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="AccessibilityResult"/>
        <Datatype abbreviatedIRI="xsd:anyURI"/>
165    </DataExactCardinality>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="AssessmentDate"/>
    <DataExactCardinality cardinality="1">
170    <DataProperty IRI="AssessmentDate"/>
        <Datatype abbreviatedIRI="xsd:dateTime"/>
    </DataExactCardinality>
</ObjectPropertyRange>
<ObjectPropertyRange>
175    <ObjectProperty IRI="CompletenessMeasure"/>
    <DataExactCardinality cardinality="1">
        <DataProperty IRI="CompletenessResult"/>
        <Datatype abbreviatedIRI="xsd:double"/>
    </DataExactCardinality>
180 </ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="CompletenessResult"/>
    <DataMinCardinality cardinality="1">
        <DataProperty IRI="CompletenessResult"/>
185        <Datatype abbreviatedIRI="xsd:anyURI"/>
    </DataMinCardinality>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="ContextualMeasure"/>
190    <DataExactCardinality cardinality="1">
        <DataProperty IRI="CompletenessResult"/>
        <Datatype abbreviatedIRI="xsd:anyURI"/>
    </DataExactCardinality>
</ObjectPropertyRange>
195 <ObjectPropertyRange>
    <ObjectProperty IRI="Identifier"/>
    <DataMinCardinality cardinality="1">
        <DataProperty abbreviatedIRI="owl:topDataProperty"/>
        <Datatype abbreviatedIRI="xsd:string"/>
200    </DataMinCardinality>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="InLevel"/>
    <DataExactCardinality cardinality="1">
205    <DataProperty IRI="CompletenessResult"/>
        <Datatype abbreviatedIRI="xsd:int"/>
    </DataExactCardinality>
</ObjectPropertyRange>
<ObjectPropertyRange>
210    <ObjectProperty IRI="InURI"/>
    <DataMinCardinality cardinality="1">
        <DataProperty IRI="CompletenessResult"/>
        <Datatype abbreviatedIRI="xsd:anyURI"/>

```



```
215         </DataMinCardinality>
        </ObjectPropertyRange>
    </Ontology>

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

Listado C.1: Vocabulario utilizado para los resultados finales

Anexo D

Listados de código

D.1 Algoritmo declarativo para la obtención de triplas por niveles

```
public static ArrayList<ArrayList<RDFNode>> getResourcesInDepthQuery(  
    String endpoint, String uri, int depth) {  
    // Init  
    ArrayList<ArrayList<RDFNode>> resources =  
        new ArrayList<ArrayList<RDFNode>>();  
    DQModel dq = new DQModel(endpoint, uri, false);  
    Query query;  
    QueryEngineHTTP qexec;  
    ArrayList<RDFNode> lvlCollection; // LVL 0  
  
    resources.add(getURIResourceList(dq.getModel()));  
    if (depth > 0) {  
  
        String headerQuery = "SELECT DISTINCT ?obj WHERE { <" + uri  
            + "> ?p1 ?o1 .";  
        String qAux = "";  
        String finalQuery = "";  
  
        for (int i = 1; i < depth; i++) {  
            lvlCollection = new ArrayList<RDFNode>();  
            // Creating query  
            qAux += "\n ?o" + i + " ?p" + (i + 1);  
            finalQuery = headerQuery + qAux  
                + " ?obj . FILTER (!sameTERM(?obj,?o" + i + ")) .}";  
            System.out.println(i + ": " + finalQuery);  
            // Executing query  
            query = QueryFactory.create(finalQuery);  
            qexec = QueryExecutionFactory.createServiceRequest(  
                endpoint, query);  
  
            ResultSet resultsQuery = qexec.execSelect();  
            // store results  
            while (resultsQuery.hasNext())  
                lvlCollection.add(resultsQuery.next().get("?obj"));  
  
            resources.add(i, lvlCollection);  
        }  
    }  
    return resources;  
}
```

Listado D.1: Algoritmo por consultas sucesivas: triplas por niveles de profundidad

D.2 Clase MeasurementResult

```

package JenaDQ;

/**
 * Almacena resultados de evaluacion para que sean mas accesibles,
 * como una
 * alternativa a realizar consultas sobre el modelo resultante
 *
 * @author Raul Reguillo Carmona
8  *
 */
public class MeasurementResult {

    /**
     *
     * @param mName
     *         nombre de la metrica
     * @param mDimension
     *         nombre de la dimension de calidad
18  */
    public MeasurementResult(String mName, String mDimension) {
        super();
        setMName(mName);
        setMDimension(mDimension);
        setmContextualResult(null);
    }

    /**
     *
28  * @param mName
     *         nombre de la metrica
     * @param mDimension
     *         nombre de la dimension de calidad
     * @param mResult
     *         resultado numerico de la evaluacion
     */
    public MeasurementResult(String mName, String mDimension, double
        mResult) {
        super();
        setMName(mName);
38  setMDimension(mDimension);
        setMResult(mResult);
    }

    /**
     *
     * @param mName
     *         nombre de la metrica
     * @param mDimension
     *         nombre de la dimension de calidad
48  * @param mResult

```

```
*          resultado numerico de la evaluacion
* @param cResult
*          resultado contextual de la evaluacion
*/
public MeasurementResult(String mName, String mDimension, double
    mResult,
    String cResult) {

    super();
    setMName(mName);
58    setMDimension(mDimension);
    setMResult(mResult);
    setmContextualResult(cResult);
}

/**
 * Constructor vacio
 */
68 public MeasurementResult() {
}

private String mName;
private String mDimension;
private double mResult;
private String mContextualResult;

public String getMName() {
    return this.mName;
}

public void setMName(String mName) {
    this.mName = mName;
}

public String getMDimension() {
    return this.mDimension;
}

public void setMDimension(String mDimension) {
88    this.mDimension = mDimension;
}

public Object getMResult() {
    return this.mResult;
}

public void setMResult(double mResult) {
    this.mResult = mResult;
}

public String getmContextualResult() {
    return mContextualResult;
}

public void setmContextualResult(String mContextualResult) {
    this.mContextualResult = mContextualResult;
}
```

```

    @Override
108    public String toString() {
        return "MeasurementResult [mName=" + mName + ", mDimension="
            + mDimension + ", mResult=" + mResult + ",
                mContextualResult="
            + mContextualResult + "]";
    }
}

```

Listado D.2: Clase MeasurementResult

D.3 Test de aceptación junit y Selenium

```

package tests;

import java.util.regex.Pattern;
import java.util.concurrent.TimeUnit;
5 import org.junit.*;
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.openqa.selenium.*;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.Select;

public class Test {
    private WebDriver driver;
    private String baseUrl;
15 private boolean acceptNextAlert = true;
    private StringBuffer verificationErrors = new StringBuffer();

    @Before
    public void setUp() throws Exception {
        driver = new FirefoxDriver();
        baseUrl = "http://localhost:8080/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    }

25 @Test
    public void test() throws Exception {
        driver.get(baseUrl + "/SemanticAnalytics/index.jsp");
        driver.findElement(By.cssSelector("a")).click();
        driver.findElement(By.id("uriEndpoint_uri")).click();
        driver.findElement(By.id("uriEndpoint_uri")).clear();
        driver.findElement(By.id("uriEndpoint_uri")).sendKeys("http://
            dbpedia.org/resource/Metallica");
        driver.findElement(By.id("uriEndpoint_endpoint")).click();
        driver.findElement(By.id("uriEndpoint_endpoint")).clear();
        driver.findElement(By.id("uriEndpoint_endpoint")).sendKeys("http
            ://lod.openlinksw.com/sparql");
35 driver.findElement(By.id("uriEndpoint_0")).click();
        driver.findElement(By.id("dqassessment_file")).clear();
        driver.findElement(By.id("dqassessment_file")).sendKeys("D:\\
            Dropbox\\PFC_RaulReguillo\\rules\\contextual.rules");
        driver.findElement(By.id("dqassessment_dq")).click();
    }
}

```

```

        assertFalse(driver.findElement(By.cssSelector("BODY")).getText().
            matches("Error*"));
    }

    @After
    public void tearDown() throws Exception {
        driver.quit();
45    String verificationErrorString = verificationErrors.toString();
        if (!"".equals(verificationErrorString)) {
            fail(verificationErrorString);
        }
    }

    private boolean isElementPresent(By by) {
        try {
            driver.findElement(by);
            return true;
55    } catch (NoSuchElementException e) {
            return false;
        }
    }

    private boolean isAlertPresent() {
        try {
            driver.switchTo().alert();
            return true;
        } catch (NoAlertPresentException e) {
65    return false;
        }
    }

    private String closeAlertAndGetItsText() {
        try {
            Alert alert = driver.switchTo().alert();
            String alertText = alert.getText();
            if (acceptNextAlert) {
                alert.accept();
75    } else {
                alert.dismiss();
            }
            return alertText;
        } finally {
            acceptNextAlert = true;
        }
    }
}

```

Listado D.3: Ejemplo de Test: comprobación de resultados correctos

Bibliografía

- [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/>.
- [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>.

- [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>.
- [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/>.
- [GHJV96] E. Gamma, R. Helm, R. Johnson, y J. Vlissides. *Design Patterns*. Addison-Wesley, 1996.
- [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.
- [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.
- [JEN14] Apache Jena - Home, 2014. url: <http://jena.apache.org/>.
- [Los13] David Loshin. *Big data analytics: from strategic planning to enterprise integration with tools, techniques, NoSQL, and graph*. 2013.
- [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. Anteproyecto Fin de Carrera: Extending Jena to Support Data Quality Context-Aware Assessment of Linked Open 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.
- [San11] Juan Antonio Pastor Sanchez. *Tecnologías 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>.
- [SPA] 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 **arco-pfc** que se puede encontrar en:
https://bitbucket.org/arco_group/arco-pfc

[Respetar esta atribución al autor]

