GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

**TRABAJO FIN DE GRADO**

***gENERACIón, EVALUACIÓN Y EXPLOTACIÓN DE open LINKED DATA A PARTIR DE DATOS PUBLICADOS por OPEN DATA EUSKADI***

**Curso:** 2017-2018

**Alumna**: Uchuari Vera, Mishel

**Director:** Oyarzabal Casquero, Oskar

**Fecha:** Bilbao, 23 de febrero de 2018

## RESUMEN

A diario múltiples organizaciones tanto públicas como privadas publican Datos Abiertos, incrementándose el número de ellas que eligen hacerlo a través de Datos Enlazados. Este nuevo método tiene como objetivo la interconexión, reutilización e integración de la información para crear una gran base de datos interconectada y distribuida que pueda ser consumida tanto por personas como por máquinas.

En el proyecto a exponer se pretenden crear representaciones de datos públicos publicados por Open Data Euskadi en RDF. El RDF generado se convertirá posteriormente en Datos Enlazados a través de la creación de nexos entre estos con la información existente en la web.

La idea central del proyecto surge en el marco de una empresa, Eurohelp Consulting, ante la demanda de un cliente, Open Data Euskadi, cuya necesidad era mejorar los medios en los que publicaba la información de la que disponía. La elección tomada para realizarlo fue utilizar RDF.

Se partió de datasets existentes en el portal de Open Data Euskadi referentes a la calidad del aire, registros de contratos, estaciones meteorológicas y retribuciones nominativas de altos cargos del gobierno. Tomándolos como base se obtuvo RDF de calidad interconectado en medida de lo posible con datos relacionados. Se consiguió, además, presentar la información generada en distintas formas que facilitaban al usuario su compresión y manipulación.

El proyecto realizado constara de las siguientes fases: La generación de RDF, su validación y la interconexión del mismo con el ya existente. Por último, la implementación de distintos medios para consumir la información.

En cuanto a las tecnologías empleadas, se utilizó Grafter para la generación de RDF, Shacl para la validación de la información creada y Silk para el descubrimiento de enlaces. Para la visualización gráfica se usó la librería D3 para la creación de gráficos en forma de grafo dinámicos.

## Laburpena

Egunero hainbat erakunde publiko nahiz pribatu datu irekiak aurkezten dituzte, eta horretarako, hauetariko askok, gero eta gehiago, elkarren menpekoak diren datuak erabiltzeko joera izaten dute.

Metodo berri honek, hiru helburu hauek ditu; informazioaren interkonektibitatea, berrerabilpena eta integrazioa sustatzea. Helburu hauen atzean, elkarren artean konektatutako datu base sarea sortzea dago, edozein erabiltzaile zein makina erabili ahal izateko.

Lan honen bidez, Open Data Euskadiko datu publikoak deskribatu nahi dira, RDF formatuan. Dokumentu hau sortuta dagoela, web orrian agertzen den informazioaren arteko loturak eginez, elkarrekiko lotura duten datu bilakatuko dira, elkarren menpekoak diren datuak, alegia.

Proiektuaren ideia orokorra enpresa baten inguruan dihardu, Euro Help Consulting. Bertan, bezero baten esanetan, Open Data Euskadi, informazioaren inguruko kudeaketa hobeagoa egiteko erremintak eskatzen ditu. Hau aditzera emateko, RDF formatua erabiltzea erabaki da.

Open Data Euskadi portalean zeuden datuak, hala nola; airearen kalitatea, kontratuen erregistroa, estazio meteorologikoak eta ordainsari nominatiboei buruzkoak ziren. Horrela, kalitatezko RDF bat lortu zen arestian aipatutako datu guztiak, hein handi batean, konektatuz. Gainera, eskuratutako informazioa hainbat eratara aurkeztu ahal izan zen eta horrek ahalbidetu zuen erabiltzaileek informazioa argiago izatea eta manipulatzea.

Proiektuak honako fase hauek izango ditu; lehenik, RDF-aren sorkuntza, baliospena eta enpresak duenarekiko aldea ikustea. Azkenik, informazio berri honen kudeaketa egiteko erreminta desberdinak diseinatzeko.

Erabilitako teknologiari dagokionez, RDF-a sortzeko "Grafter" programa erabili zen, sortutako informazioa balioesteko "Shacl" programa eta "Silk" loturen aurkikuntzarako. Bistaratze grafikoa egiteko, "libreria D3" erabili zen grafikoak sortzeko, grafiko dinamiko gisa.

Palabras clave

Linked Data, Datos Enlazados, Open Data, Datos Abiertos, SPARQL, Shacl.

## Glosario/Acrónimos

**W3C:** El consorcio WWW o World Wide Web Consortium es un consorcio internacional que genera recomendaciones y estándares para el crecimiento de la World Wide Web.

**TFG**: Trabajo fin de grado.

**Dataset:** Conjunto de datos.

**XML:** Extensible Markup Language o lenguaje de marcas extensible es un metalenguaje desarrollado por el W3C que se utiliza para almacenar datos de forma legible.

**CSV:** Valores separados por comas o comma separated values, este término hace referencia a un documento de formato libre utilizado para representar datos en forma tabular, siendo las columnas separadas por coma o punto y coma, y las filas por saltos de línea.

**Grafo:** Conjunto de objetos denominados nodos unidos por enlaces llamados aristas a que representan relaciones binarias entre elementos de un conjunto.

**Grafo dirigido:** Tipo de grafo en el cual las aristas tienen un sentido definido.

**EDT:** EDT o estructura de descomposición de trabajo es una representación sencilla y organizada de las tareas a realizar antes de concluir un proyecto.

**API**: La interfaz de programación de aplicaciones o Application Programming es la agrupación de procesos, métodos o procedimientos que ofrece una librería para su utilización por otro software como capa de abstracción.

**Framework:** Un framework o entorno de trabajo es un conjunto estandarizado de prácticas o procedimientos para la realización de un determinado trabajo que sirve como precedente para resolver problemas de la misma naturaleza.

**Servidor:** Software capaz de atender las peticiones de un cliente y devolverle una respuesta acorde a su solicitud.

**Front-end:** En desarrollo web hace referencia a la capa de presentación, a la parte del software con la que el usuario interactúa.

**URI**: Un identificador de recursos uniforme o uniform resource identifier es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

**Triple Store**: Base de datos orientada a grafos RDF.

**RDF:** El Marco de Descripción de Recursos O Resource Description Framework es una familia de especificaciones del W3C para el intercambio de datos en la web.

**RDF/XML:** Sintaxis normativa definida por el W3C para expresar un grafo RDF como XML.

**JSON-LD:** JSON-LD o JavaScript Object Notation for Linked Data, es una sintaxis para expresar RDF usando JSON.

**Turtle:** Turtle o Terse RDF Triple Language, es una sintaxis para representar RDF similar a SPARQL.

**SPARQL:** SPARQL Protocol and RDF Query Language o protocolo SPARQL y lenguaje de consulta sobre RDF, es el lenguaje de consulta RDF.

**SPARQL Endpoint:** Interfaz web con una funcionalidad, ejecutar consultas SPARQL sobre una Triple Store.

**GREL:** General Refine Expresion Language o lenguaje de expresión general de Refine. Es el lenguaje utilizado por Open Refine para realizar transformaciones sobre los valores de las columnas de los CSV que manipula.

**SHACL:** Shapes Constraint Language, es un lenguaje para validar grafos RDF sobre un conjunto de condiciones. Estas condiciones son expresadas a su vez en forma de grafo RDF.

**SQL:** Structured Query Language o lenguaje de consulta estructurada es un lenguaje de consulta sobre bases de datos relacionales.

**Norma NTI**: La Norma Técnica de Interoperabilidad es el estándar del W3C que agrupa una serie de condiciones sobre la selección, identificación, descripción, formato, condiciones de uso y puesta a disposición de los datos públicos.

**Wikipedia:** Wikipedia es una enciclopedia online libre, ​políglota y editada de manera colaborativa por sus usuarios.

**DBpedia**: DBpedia es un proyecto para la extracción de conocimiento de Wikipedia, es decir, una “Wikipedia” semántica.

**Schema.org:** Es una comunidad de colaboración cuyo objetivo es la creación, mantenimiento y la promoción de esquemas para datos estructurados en Internet, páginas web, emails, etc.

**Eionet:** Eionet es una red de asociaciones de la Agencia Europea de Medio Ambiente que apoya la recopilación y organización de datos y el desarrollo y difusión de información sobre el medio ambiente en Europa.

**Jar**: Java Archive o archivo java tipo de archivo que permite ejecutar aplicaciones desarrolladas usando el lenguaje de programación Java.

**War**: Web Application Archive o Archivo de aplicación web es un archivo JAR que permite el despliegue de una aplicación web.

**GeoNames**: Ontología estandarizada por el W3C para la adición de información geoespacial a los datos.

## INDICE

[RESUMEN 2](#_Toc506778590)

[Laburpena 2](#_Toc506778591)

[Palabras clave 3](#_Toc506778592)

[Glosario/Acrónimos 3](#_Toc506778593)

[INDICE 6](#_Toc506778594)

[INDICE DE FIGURAS 8](#_Toc506778595)

[INDICE DE TABLAS 10](#_Toc506778596)

[1. MEMORIA 11](#_Toc506778597)

[Introducción 11](#_Toc506778598)

[Contexto 11](#_Toc506778599)

[Objetivos y alcance del trabajo 12](#_Toc506778600)

[Beneficios del trabajo 12](#_Toc506778601)

[Marco teórico: Introducción al RDF 12](#_Toc506778602)

[Descripción de requerimientos y/o Análisis del estado del arte 14](#_Toc506778603)

[Análisis de alternativas 15](#_Toc506778604)

[Selección/Descripción de solución propuesta 17](#_Toc506778605)

[2. METODOLOGÍA SEGUIDA EN EL DESARROLLO DEL TRABAJO 22](#_Toc506778606)

[Descripción de tareas, fases, equipos o procedimientos 22](#_Toc506778607)

[Diagrama Gantt 28](#_Toc506778608)

[Cálculos, algoritmos 28](#_Toc506778609)

[Descripción de los resultados 43](#_Toc506778610)

[3. CONCLUSIONES 44](#_Toc506778611)

[BIBLIOGRAFIA 45](#_Toc506778612)

[ANEXO I: Pliego de condiciones. Normativa aplicable. Plan de pruebas 46](#_Toc506778613)

[Plan de pruebas 46](#_Toc506778614)

[Anexo II: Planos, esquemas, diseño de detalle o de bajo nivel, código 60](#_Toc506778615)

[Diagrama clases 60](#_Toc506778616)

[ANEXO III: Manuales de usuario y/o de administrador 81](#_Toc506778617)

[Manual de usuario 81](#_Toc506778618)

[Manual administrador 87](#_Toc506778619)

## INDICE DE FIGURAS

Figura 1: Esquema general RDF 13

Figura 2: RDF en formato RDF/XML 13

Figura 3: RDF en formato Turtle 13

Figura 4: Ejemplo query SPARQL 14

Figura 5: Esquema general del sistema construido 18

Figura 6: Esquema de la funcionalidad “Generación de RDF” 19

Figura 7: Esquema de la funcionalidad “Validación RDF” 19

Figura 8: Esquema funcionalidad “Descubrimiento de enlaces” 20

Figura 9: Esquema funcionalidad “Servidor Linked Data” 21

Figura 10: Esquema funcionalidad “SPARQL Endpoint” 22

Figura 11: Diagrama EDT del proyecto 23

Figura 12: Diagrama Gantt del proyecto 28

Figura 13: Modelo dominio dataset “Calidad del Aire” 30

Figura 14: Modelo dominio dataset “Estaciones meteorológicas: lecturas recogidas” 31

Figura 15: Modelo dominio dataset “Relación de puestos de trabajo” 32

Figura 16: Modelo dominio dataset “Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual” 33

Figura 17: Esquema general de la representación en RDF del CSV perteneciente a “Calidad del aire” 35

Figura 18: RDF dataset “Calidad del aire” 36

Figura 19: Ejemplo definición pruebas Shacl en la que se fija como *Violation* no poseer *identificador* 38

Figura 20: Ejemplo archivo configuración Silk para la configuración de las Triple Stores a utilizar 39

Figura 21: Ejemplo archivo configuración Silk para la limitación de los recursos a tener en cuenta 40

Figura 22: Ejemplo archivo configuración Silk: configuración del filtro de recursos 40

Figura 23: Ejemplo archivo configuración Silk para la configuración del medio en el que se almacenarán los enlaces descubiertos 41

Figura 24: Ejemplo de archivo XML de configuración de Pubby 41

Figura 25: Editor queries SPARQL 42

Figura 26: Diagrama clases Java 61

Figura 27: Diagrama clases Clojure 63

Figura 28: Diagrama de secuencia funcionalidad generar RDF 64

Figura 29: Diagrama de secuencia funcionalidad validación RDF I/II 65

Figura 30: Diagrama de secuencia funcionalidad validación RDF II/II 66

Figura 31: Diagrama de secuencia funcionalidad descubrimiento enlaces 67

Figura 32: Modelo de dominio del sistema 68

Figura 33: Diagrama casos de uso sistema 69

Figura 34: Diagrama jerarquía de actores en casos de uso del sistema 70

Figura 35: Caso de uso "Petición servidor Linked Data" 71

Figura 36: Petición a Servidor Linked Data solicitando recurso en formato HTML desde navegador web cualquiera 71

Figura 37: Petición a Servidor Linked Data solicitando recurso en formato HTML desde INSOMNIA 72

Figura 38: Petición a Servidor Linked Data solicitándole la información en RDF, específicamente en formato RDF/XML 72

Figura 39: Caso de uso "Generar RDF" 72

Figura 40: Mensaje de aviso de que el RDF se ha generado 73

Figura 41: Caso de uso "Validación RDF" 73

Figura 42: Mensaje avisando al usuario que el RDF es válido. 74

Figura 43: Report indicando que el RDF es válido 74

Figura 44: Mensaje avisando al usuario que el RDF no es válido. 74

Figura 45: Ejemplo de report indicando las razones por las cuales el RDF no es válido. 74

Figura 46: Caso de uso "Descubrimiento enlaces" 75

Figura 47: Caso de uso "Ejecutar query SPARQL Endpoint" 75

Figura 48: Query en SPARQL Endpoint 76

Figura 49: Resultado query en forma de tabla 76

Figura 50: Caso de uso “Ver resultados en forma tabular” 77

Figura 51: Resultados en forma tabular 77

Figura 52: Botón que debe pulsar el usuario para volver a visualizar los resultados en forma tabular 78

Figura 53: Caso de uso "Ver resultados en forma de grafo" 78

Figura 54: Botón para ver los resultados de forma grafo 79

Figura 55: Resultado query en forma de grafo 79

Figura 56: Información grafo concreto 80

Figura 57: Se focaliza la información del grafo en un nodo concreto y sus adyacentes 80

Figura 58: Se muestra la información de una arista concreta 81

Figura 59: Ejecución jar "generaciónRDF.jar" 82

Figura 60: Ejecución jar "validacionRDF" 83

Figura 61: Ejecución jar "descubrimientoEnlaces.jar" 83

Figura 62: Petición servidor Linked Data a través de navegador web 83

Figura 63: Petición servidor Linked Data utilizando INSOMNIA solicitando la información en HTML 84

Figura 64: Petición servidor Linked Data utilizando INSOMNIA solicitando la información en RDF/XML 84

Figura 65: Localización botón RUN 85

Figura 66: Localización botón para cambiar visualización a Grafo 85

Figura 67: Arista con texto visible 86

Figura 68: Focalización nodo sobre recurso concreto 86

Figura 69: Petición servidor Linked Data desde navegador web 89

Figura 70: Petición servidor Linked Data utilizando INSOMNIA y solicitando información en RDF/XML 90

Figura 71: Localización botón RUN 90

Figura 72: Localización botón para cambiar la visualización a grafo 91

Figura 73: Texto arista visible 91

Figura 74: Focalización de información del grafo sobre un recurso concreto 92

## INDICE DE TABLAS

[Tabla 1: Análisis de objetivos 24](#_Toc506778694)

[Tabla 2: Análisis herramientas necesarias 24](#_Toc506778695)

[Tabla 3: Análisis de ontologías a utilizar 24](#_Toc506778696)

[Tabla 4: Reuniones con el tutor del proyecto I 24](#_Toc506778697)

[Tabla 5: Creación ontologías propias 25](#_Toc506778698)

[Tabla 6: Implementación software creación RDF 25](#_Toc506778699)

[Tabla 7: Definición e implementación de pruebas Shacl 25](#_Toc506778700)

[Tabla 8: Implementación software descubrimiento enlaces 26](#_Toc506778701)

[Tabla 9: Implantación servidor Linked Data 26](#_Toc506778702)

[Tabla 10: Creación SPARQL Endpoint 26](#_Toc506778703)

[Tabla 11: Reuniones con el tutor del proyecto II 26](#_Toc506778704)

[Tabla 12: Pruebas 27](#_Toc506778705)

[Tabla 13: Memoria 27](#_Toc506778706)

[Tabla 14: Revisión memoria por parte del tutor del proyecto 27](#_Toc506778707)

[Tabla 15: Horas invertidas en la realización del proyecto 28](#_Toc506778708)

[Tabla 16: Generación RDF 47](#_Toc506778709)

[Tabla 17: Evaluación RDF 48](#_Toc506778710)

[Tabla 18: Evaluación RDF “Calidad del aire” respecto a las observaciones 49](#_Toc506778711)

[Tabla 19: Evaluación RDF “Calidad del aire” respecto a las mediciones con resultados numéricos 49](#_Toc506778712)

[Tabla 20: Evaluación RDF “Calidad del aire” respecto a las mediciones con resultados literales 50](#_Toc506778713)

[Tabla 21: Evaluación RDF “Estaciones meteorológicas: lecturas recogidas” respecto a las observaciones 51](#_Toc506778714)

[Tabla 22: Evaluación RDF “Estaciones meteorológicas: lecturas recogidas” respecto a las mediciones 51](#_Toc506778715)

[Tabla 23: Evaluación RDF perteneciente a “Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual” 53](#_Toc506778716)

[Tabla 24: Evaluación RDF perteneciente a “Relación de puestos de trabajo” 55](#_Toc506778717)

[Tabla 25: Pruebas sobre el SPARQL Endpoint 56](#_Toc506778718)

[Tabla 26: Pruebas sobre la representación de RDF usando tablas 57](#_Toc506778719)

[Tabla 27: Pruebas sobre la representación de RDF usando gráficos dinámicos en forma de grafo 59](#_Toc506778720)

[Tabla 28: Pruebas sobre la funcionalidad "Descubrimiento Enlaces" 60](#_Toc506778721)

# MEMORIA

## Introducción

Desde que se empezara a publicar información en Internet los medios en los que se ha realizado han ido variando. Cualquier persona independientemente de sus conocimientos puede compartir datos en la web, como resultado nos encontramos con una web con masificación de contenido.

La web actual o “web de los documentos” se presenta en un formato amigable para el ser humano que la consume, como consecuencia es entendible sólo por personas que son capaces de interpretar la información que se les expone a través de una pantalla. En la “web de los datos”, usando Datos Enlazados, la información ofrecida por la web puede ser interpretada tanto por personas como por máquinas.

Entre las ventajas de utilizar Datos Enlazados o Linked Data para la publicación de información aparte de la anteriormente mencionada capacidad de ser consumida por distintos tipos de usuarios, se encuentra la posible reutilización, integración y conexión de la información a publicar con la ya existente en la Web. Debido a esto, son cada vez más las organizaciones que publican datos abiertos que deciden utilizar este medio.

Como resultado de utilizar Linked Data se pueden crear aplicaciones que sean capaces de interpretar la información contenida en la web como si de un solo ente se tratara, es decir, como una única base datos. Explorando la “web de los datos” nos encontramos con información interconectada, o dicho de otra forma, al analizar un recurso se puede extraer conocimiento no sólo de su fuente de publicación, sino también de todas las fuentes de datos con las que tiene relación.

En el siguiente trabajo se realizará para un determinado conjunto de datos publicados por Open Data Donosti su representación en RDF y su posterior interconexión con la información ya existente en otras fuentes de datos, convirtiéndose estos así en Linked Data. Además, se priorizará la calidad de los datos enlazados por lo que la información generada pasará por un proceso de evaluación, obteniendo así RDF de calidad.

## Contexto

Los Datos Enlazados se refieren a una forma de publicar datos que permite vincular datos distribuidos entre sí con el objetivo de construir la Nube de Datos Enlazados que pueda explorarse de forma automática por máquinas, de forma análoga a como la web puede explorarse mediante enlaces por personas. Esta vinculación entre datos distribuidos permite dar mayor significancia a los datos publicados dado que en este formato se puede extraer mucha más información y de mayor utilidad que de ficheros en formato CSV, XML y otras formas de almacenaje.

Esto permite que los datos publicados de esta forma sean más accesibles, reusables e integrables, características que se alinean con la filosofía de Datos Abiertos y, por tanto, permiten pensar en los Datos Enlazados como una solución tecnológica adecuada para dar soporte a los Datos abiertos.

## Objetivos y alcance del trabajo

El objetivo principal del proyecto presentado es la generación de RDF de calidad basado en los datos publicados por Open Data Euskadi. Junto a este se plantean los siguientes objetivos secundarios:

* Creación SPARL Endpoint para consultas sobre los datos
* Creación servidor Linked Data para el consumo de la información tanto por humanos como por máquinas.

Además, se valorarán las siguientes ampliaciones:

* Validación RDF generado
* Ampliación de las funcionalidades del SPARQL Endpoint añadiendo distintas formas gráficas de representación de los datos solicitados.

## Beneficios del trabajo

Se presenta en este proyecto un modelo completo del proceso de generación de RDF. Es decir, partiendo de la necesidad real de un cliente de mejorar la forma en la que publicaba los datos, se ha construido un sistema que aúna varias de las etapas de generación de RDF: manipulación de los datos de partida para darles formato RDF, representación de los datos en RDF, validación y descubrimiento de enlaces relacionados.

Este trabajo, además de servir de guía teórica y práctica para una persona que está empezando a familiarizarse con los conceptos Linked Data o RDF, suplió la demanda del cliente, se generó la representación en RDF de sus datos.

Como se exponía anteriormente, este Trabajo de Fin de Grado bien podría ser utilizado como guía teórica ya que en él se exponen y se desglosa el uso de varias herramientas relacionadas con Linked Data. Una persona cualquiera, podría rehacer el procedimiento seguido por el alumno para llegar a conocer las tecnologías aquí tratadas.

## Marco teórico: Introducción al RDF

A lo largo de este documento a menudo se hará referencia a términos como RDF o SPARQL. En este apartado se presentarán estos términos para crear una base de conocimiento y facilitar su lectura y comprensión.

Como se ha planteado anteriormente, el objetivo principal del proyecto es la creación de Datos Enlazados, esto se consigue enlazando RDF con información relacionada ya existente. A continuación, explicaremos en qué consiste la representación de información utilizando RDF.

RDF es un modelo estándar para el intercambio de información. Utiliza la estructura de enlaces de la web, es decir, usa URIs para designar relaciones entre objetos o recursos. A esta relación se la denomina triple o tripleta. En la figura 1 se puede observar el esquema de tripleta RDF, formada por un recurso, un predicado y un objeto.

Informalmente se puede entender una tripleta como un recurso, el predicado, que corresponde a la acción o estado que acompaña al recurso y el objeto complementándolos en cuanto a que es lo que se dice del recurso, es decir, una tripleta se puede entender como atributos o características de los recursos.

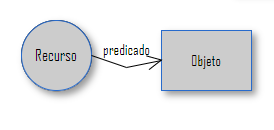


Figura : Esquema general RDF

El primer paso a seguir para crear RDF es la elección de la ontología u ontologías a utilizar para describir la información. Una ontología es una definición formal de tipos o relaciones entre entidades que existen en un dominio de discusión determinado.

Utilizando ontologías externas o propias se crea RDF. El RDF creado siguiendo la estructura de tripleta formara un grafo dirigido con etiquetas donde los nodos de cada enlace corresponderán a recursos que mantienen una relación.

La sintaxis básica de RDF es XML, que da como resultado RDF/XML, pero también se puede representar en otros formatos como Turtle, JSON-LD, N-Quads, etc. En las figuras 2 y 3 se muestra RDF en formato RDF/XML y Turtle respectivamente.

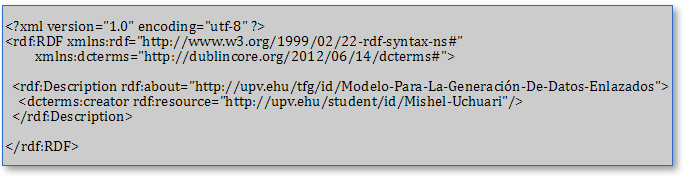


Figura : RDF en formato RDF/XML

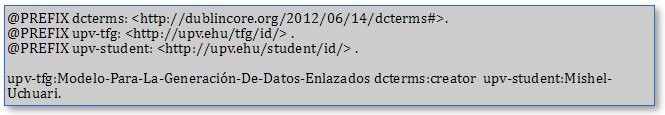


Figura : RDF en formato Turtle

Las consultas sobre RDF se realizan utilizando SPARQL. Al igual que ocurre con SQL, existen múltiples implementaciones de SPARQL, generalmente ligados a entornos de desarrollo y plataforma tecnológicas. En la figura 4 se presenta un ejemplo de consulta SPARQL.

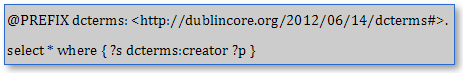


Figura : Ejemplo query SPARQL

## Descripción de requerimientos y/o Análisis del estado del arte

Descripción de requerimientos

En la elección del TFG a construir como en la de las tecnologías a utilizar el parámetro que más influyó fue el beneficio académico que este dejaría al estudiante, dicho de otra forma, qué tanto se podría aprender llevándolo a cabo. Por ello se decidió crear un proyecto relacionado con tecnologías Linked Data ya se trataba de un tema de actualidad y era un área no estudiada en el grado en Ingeniería Informática.

Antes de comenzar con la implementación del proyecto se fijaron unos requerimientos mínimos que el sistema debía conseguir. En algunas de ellas, sin embargo, los requerimientos eran inherentes a la funcionalidad que se pretendía construir.

Eso se dio con la funcionalidad de descubrimiento de enlaces. La funcionalidad implementaba debía conseguir usar Silk de tal forma que configurándolo adecuadamente pudiese encontrar nexos entre dos fuentes de datos.

La funcionalidad de creación de RDF en cambio, debía de estar construida de modo que teniendo como entrada ficheros en formato CSV, concretamente ficheros CSV delimitados por comas, generara RDF de forma automática para los 4 datasets a utilizar pertenecientes a los apartados: “calidad del aire”, “estaciones meteorológicas: lecturas recogidas”, “relación de puestos de trabajo” y “evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual”.

En otras palabras, el sistema debería poder soportar la representación en RDF de un número ilimitado de ficheros en formato CSV correspondientes a estos apartados, por ello debía ser programable y adaptable.

La implementación del software que realizara las pruebas sobre el RDF generado debía estar capacitado para evaluar cualquier tipo de RDF Turtle que le entrara como parámetro siempre y cuando las reglas Shacl que se le proporcionaran estuviesen en formato RDF Turtle.

En cuanto al servidor Linked Data, este debía de estar configurado para soportar cualquier tipo de petición sobre los recursos que estuviesen almacenados en la Triple Store. Siendo estos HTML, RDF y sus distintas representaciones, texto plano, etc.

Por último, el SPARQL Endpoint creado debía permitir representar la información en forma tabular y en forma de grafo. Esta última cuando la query ejecutada en el SPARQL Endpoint fuese un CONSTRUCT. Un CONSTRUCT es un tipo de query SPARQL que devuelve RDF.

Análisis estado del arte

La generación de Datos Enlazados y por consiguiente RDF es una tarea que actualmente necesita supervisión humana y no se puede automatizar por completo. Tanto las fases de análisis previo en el que se crea la representación en RDF de los datos como la posterior de descubrimiento de enlaces necesitan a una persona que las supervise. Por ello es difícil encontrar alguna aplicación o software desarrollado anteriormente que auné todas las características que este TFG ofrece.

Por lo anterior mencionado, para la realización de este proyecto se analizaron distintas herramientas que realizaban funcionalidades aisladas de este. A continuación, se presentarán las distintas referencias que se tomaron:

En el caso de la generación de RDF se evaluó Open Refine. Open Refine es una herramienta que permite al usuario importar sus datasets y realizar alteraciones sobre ellos tanto de forma manual como usando funciones para posteriormente crear una representación en RDF de los datos. A diferencia de la herramienta usada con ese fin en este proyecto, el proceso no se puede automatizar.

Para la creación del servidor Linked Data se examinó Elda. Elda es una implementación en Java de Linked Data API, que provee un medio configurable para acceder a datos RDF usando RESTful URLs que son traducidas a queries RDF que se ejecutan sobre un SPARQL Endpoint. Provee al usuario de una interfaz gráfica en la que la información en RDF se puede presentar en distintos formatos. Elda es un sistema que ofrece multitud de funcionalidades, pero a diferencia de Pubby, el servidor Linked Data elegido, resulta complejo de configurar.

En la realización del SPARQL Endpoint influyo la Triple Store utilizada a lo largo del proyecto para el almacenaje de información, GraphDB. GraphDB es una base de datos orientada a grafos que cuenta con una interfaz gráfica sobre la cual ejecutar queries SPARQL.

## Análisis de alternativas

La base del proyecto es la generación de RDF. Las demás funcionalidades partirán de los datos generados, por lo que el software utilizado en esta etapa sería fundamental para la elección del que se usaría en las posteriores. Se examinaron varias formas de generar RDF, pero se descartaron todas aquellas en las cuáles no existía medios para automatizar el proceso, es decir, se excluyó el software no programable.

Se consideraron herramientas como Grafter, RML.io o simplemente crear programas que manipularan los datos de entrada y generaran RDF. Grafter es una librería de Clojure que permite generar RDF a partir de datos tabulares, es decir ficheros en formato Excel o CSV, RML.io a su vez, permite a través de un archivo RDF en formato Turtle generar RDF.

Crear RDF a través de programas creados para ese fin permitía el uso de varios lenguajes de programación, pero el proceso en sí, sería relativamente sencillo con lo que cual esta alternativa se descartó, ya que, en el caso de ser la herramienta elegida, no se llegaría a ahondar en el aprendizaje de ningún lenguaje nuevo. RML.io también se descartó por la sencillez de su uso.

Finalmente se elegiría Grafter, usar esta herramienta permitía conocer en profundidad esta librería de Clojure, un lenguaje no tratado hasta el momento. Usando Clojure, además, se podía emplear JAVA para la manipulación del RDF generado por la viabilidad de la integración de los dos lenguajes.

Para la validación de RDF se descartó toda alternativa distinta a la API Shacl. La API Shacl es una implementación del lenguaje de restricción de formas Shacl que se utiliza para validar RDF. La decisión fue simple ya que el lenguaje de restricción de formas era el único medio para validar RDF sobre reglas estandarizado por el W3C.

En la elección del software para la realización del descubrimiento de enlaces se tuvo en cuenta Open Refine ya que este posee un módulo de descubrimiento de enlaces, siendo descartado nuevamente por el hecho de que el proceso no se podía automatizar. Finalmente, se elegiría Silk. Silk es un framework para la integración de fuentes de datos y descubrimiento de enlaces relacionados entre ellas.

Llegado el momento de elegir una herramienta para crear el servidor Linked Data, aparte de la herramienta utilizada, Pubby, se consideró Elda. Pubby al igual que Elda es un Linked Data Frontend para SPARQL Endpoints, es decir, permite construir una interfaz gráfica sobre la cual realizar peticiones con negociación de contenido tomando como fuente de datos una Triple Store. Cotejando las dos opciones se optó por la primera ya que aunque Elda ofrece multitud de funcionalidades interesantes estas tienen una compleja configuración y son innecesarias para las necesidades de este TFG.

Partiendo de JAVA como lenguaje para aunar todas las partes del proyecto, se eligió JavaScript, JQuery, HTML y CSS para crear la interfaz web. Se tomaron estas alternativas ya que al examinar librerías para elaborar gráficos dinámicos en forma de grafo, la mayoría de ellas estaban implementadas en JavaScript.

Las alternativas examinadas para la generación del grafo fueron Vis.js y D3. Al final se decidió utilizar D3 por ser una librería de uso muy extendido y considerar la ventaja de su aprendizaje respecto a otras.

Para llevar a cabo peticiones sobre el servidor Linked Data se consideró usar INSOMNIA y CURL. INSOMNIA es un cliente REST para llevar a cabo peticiones HTTP, Curl a su vez se puede usar para realizar este tipo de peticiones. Se eligió INSOMNIA por ser más intuitivo su uso ya que posee una interfaz gráfica a diferencia de Curl que tiene que ser ejecutado por línea de comandos.

La Triple Store elegida para el almacenaje de la información que se generaría a lo largo del proyecto fue GraphDB. GraphDB es una base de datos orientada a grafos. Se tomaron en cuenta otras muchas bases de datos como Virtuoso, Stardog o Blazegraph, pero al ser este un proyecto pequeño las diferencias entre ellas eran apenas relevantes, por ello, por familiaridad, se utilizó GraphDB ya que ya se había trabajado previamente con ella.

En la creación del SPARQL Endpoint se necesitaba crear un editor de queries SPARQL que en cierta medida las validara antes de ejecutarlas. Para llevarlo a cabo se tenían dos opciones: crear un editor propio o utilizar Yasqe. Yasqe es una librería JavaScript para la creación de editores de queries SPARQL, es decir a través de Yasqe se podría monitorizar la sintaxis de las queries introducidas en el SPARQL Endpoint. Ante la dificultad y el tiempo necesario para crear un analizador sintáctico de SPARQL se optó por utilizar Yasqe.

Por último, se le dio formato a la interfaz gráfica usando Bootstrap. Bootstrap es un kit de herramientas de código abierto para desarrollar con HTML, CSS y JS

## Selección/Descripción de solución propuesta

El sistema creado tiene distintos módulos. En la figura 5 se puede ver de forma general y sin incidir mucho en detalles su funcionamiento. Como se puede observar, el proceso inicia con la generación de RDF, partiendo de un CSV, se obtiene su representación en RDF.

Posteriormente el RDF pasa a ser evaluado, en caso de obtener un resultado positivo en esta evaluación se almacena en la Triple Store, GraphDB. A continuación, Silk utilizando una fuente de datos externa en conjunto con la propia las relaciona y descubre nexos entre ellas. Esta nueva información también se publica en la Triple Store.

Una vez toda la información está en la Triple Store puede ser consumida, esto se realizará de dos formas: a través del SPARQL Endpoint o a través del servidor Linked Data. En el caso del servidor Linked Data las peticiones se realizarán con negociación de contenido, a su vez el SPARQL Endpoint ofrecerá distintas representaciones sobre los datos.

En este apartado se presentarán a grandes rasgos el comportamiento de cada fase.

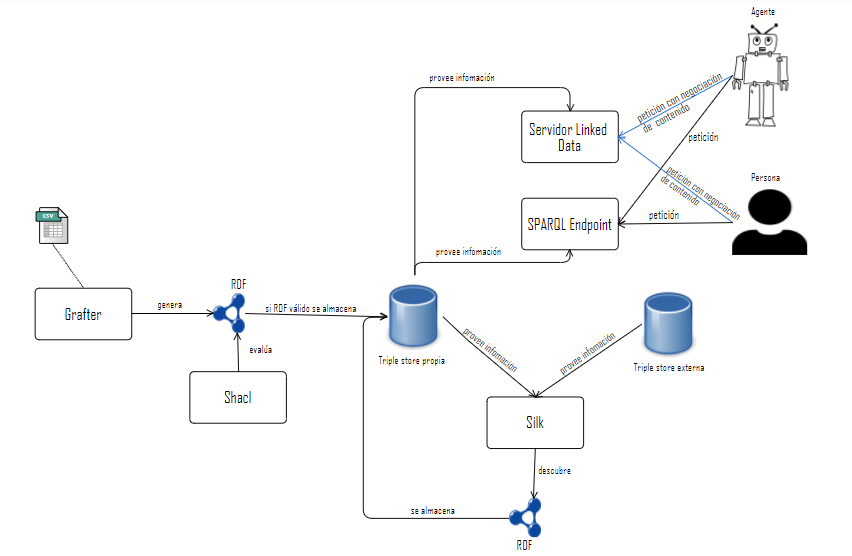


Figura : Esquema general del sistema construido

### Generación de RDF

En la figura 6 se puede observar cómo se desarrolla esta parte del proyecto. Una clase JAVA le pasa un CSV a una clase Clojure que utiliza la librería Grafter. Esta a su vez crea una representación en RDF del CSV y se lo devuelve a la clase JAVA.

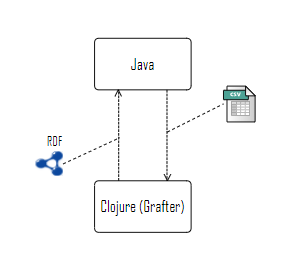


Figura : Esquema de la funcionalidad “Generación de RDF”

### Validación RDF

Para la validación de RDF se utilizó SHACL, el estándar del W3C y la API que implementa el estándar. Como se explicaba con anterioridad el estándar Shacl se utiliza para la validación de RDF.

Las directrices que el RDF debe cumplir se definen en un archivo escrito en RDF, concretamente en Turtle. La API Shacl lee el fichero donde están definidas las reglas que definen las pruebas y comprueba que el RDF a evaluar las cumple.

Tras la evaluación de los datos se genera un archivo en formato Turtle que indicará si el RDF examinado era válido, en caso contrario, indica qué reglas de las impuestas incumple. Si el RDF examinado era válido, se almacenará en la Triple Store. En la figura 7 se expone mediante un gráfico el proceso.

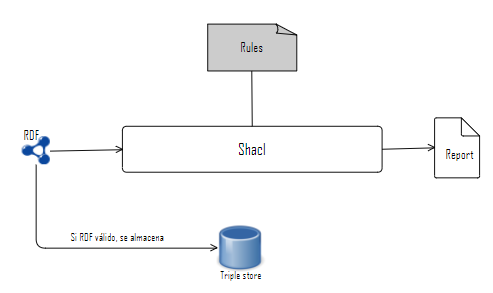


Figura : Esquema de la funcionalidad “Validación RDF”

### Descubrimiento enlaces

En la figura 8 se expone el desarrollo de la funcionalidad de descubrimiento de enlaces. Silk utiliza un fichero XML de configuración, en este, se especifican las distintas fuentes de datos sobre las cuales buscará información relacionada.

Tras la configuración de Silk este intenta encontrar enlaces relacionados entre las distintas Triple Stores, esta tarea puede obtener resultados o no, si los obtiene, estos se almacenarán en la Triple Store propia.

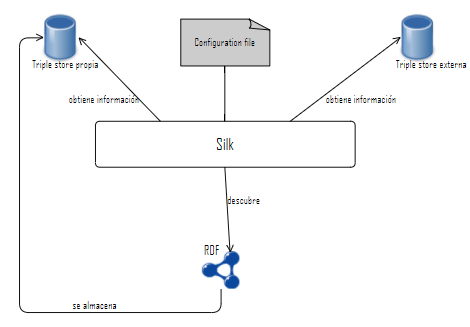


Figura : Esquema funcionalidad “Descubrimiento de enlaces”

### Servidor Linked Data

Como se puede observar en la figura 9, Pubby utiliza un archivo Turtle de configuración donde se especifican los datos de la Triple Store de la cual tomará información. Una vez configurado, un usuario cualquiera, persona o máquina puede realizar peticiones HTTP para solicitar un recurso y elegir mediante negociación de contenido la forma en la que desea que se le presente la información. La información puede ser solicitada tanto en formato HTML como en distintas representaciones RDF como RDF/XML, Turtle, JSON-LD, etc.

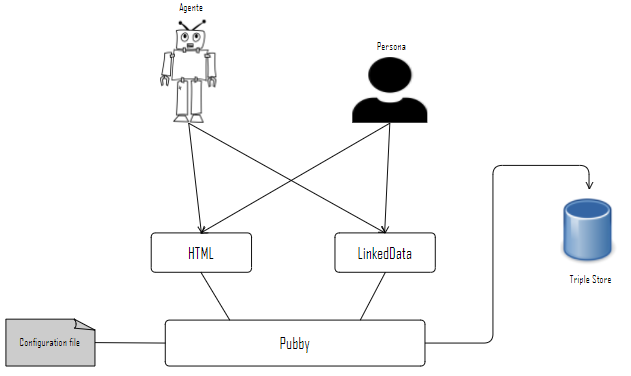


Figura : Esquema funcionalidad “Servidor Linked Data”

### SPARQL Endpoint

El SPARQL Endpoint se utiliza para ejecutar consultas sobre la Triple Store. En la figura 10 se expone como una vez un usuario, persona o máquina ejecuta una query el SPARQL Endpoint toma los datos de la Triple Store acordes a la consulta ejecutada y los representa de distintas formas gráficas: en formato tabla o en forma de grafo.

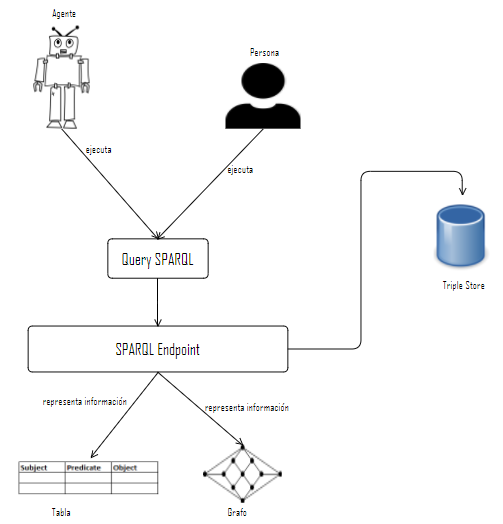


Figura : Esquema funcionalidad “SPARQL Endpoint”

# METODOLOGÍA SEGUIDA EN EL DESARROLLO DEL TRABAJO

## Descripción de tareas, fases, equipos o procedimientos

En este apartado se describirán las tareas a realizar, la forma en la que se organizaron estas a lo largo del tiempo, el equipo de trabajo que lo realizó, etc. En el desarrollo de este trabajo de fin de grado participaron dos personas: el tutor a cargo de él que actuó con el rol de jefe de proyecto, y el estudiante que tuvo el rol de analista y programador.

### Fases del proyecto

Las fases del proyecto se desglosaron en 4 bloques: investigación, desarrollo, pruebas y memoria. A continuación, se expondrá en qué consiste cada uno de ellos:

* **Investigación:** Durante esta fase se analizaron trabajos relacionados con el sistema que se deseaba crear, se cotejaron distintas herramientas para llevarlo a cabo y se estudió qué ontologías se podían utilizar.
* **Desarrollo:** En esta etapa del proyecto se implementaron las distintas funcionalidades planteadas en los objetivos.
* **Pruebas:** En esta fase se evaluó todo lo creado en la etapa anterior. Se definieron distintos tipos de pruebas para llevarlo a cabo.
* **Memoria:** En esta última fase se documentó todo el trabajo realizado.

En el diagrama EDT representado en la figura 11 se presentan las distintas etapas del proyecto. Adicionalmente cada etapa contará con reuniones con el tutor del TFG.

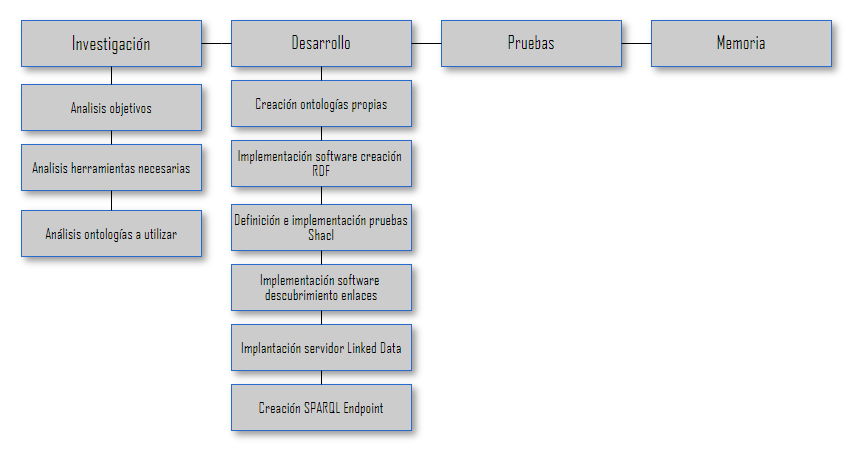


Figura : Diagrama EDT del proyecto

### Descripción de tareas

En este apartado se describirán más detalladamente las tareas especificadas en el diagrama EDT. Las tareas se descompondrán en paquetes de trabajo.

#### Investigación

* **Paquete de trabajo 1: Análisis de objetivos**

|  |  |
| --- | --- |
| **Duración:** | 25 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Durante esta fase se definirán los objetivos a cumplir: lo que se pretende conseguir al finalizar el proyecto |
| **Entradas:** | - |
| **Recursos necesarios:** | word |
| **Salidas:** | Listado de objetivos a alcanzar |

Tabla : Análisis de objetivos

* **Paquete de trabajo 2: Análisis herramientas necesarias**

|  |  |
| --- | --- |
| **Duración:** | 15 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Análisis de distintas herramientas para llevar a cabo las funcionalidades planteadas en los objetivos |
| **Entradas:** | - |
| **Recursos necesarios:** | Internet, word |
| **Salidas:** | Listado de herramientas a utilizar |

Tabla : Análisis herramientas necesarias

* **Paquete de trabajo 3: Análisis de ontologías a utilizar**

|  |  |
| --- | --- |
| **Duración:** | 10 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Análisis de ontologías existentes para elegir las idóneas para la representación en RDF de los datasets |
| **Entradas:** | - |
| **Recursos necesarios:** | Internet, word |
| **Salidas:** | Listado de ontologías a utilizar |

Tabla : Análisis de ontologías a utilizar

* **Paquete de trabajo 4: Reuniones con el tutor del proyecto**

|  |  |
| --- | --- |
| **Duración:** | 5 horas |
| **Encargado:** | Mishel Uchuari, Oskar Casquero |
| **Descripción:** | Reunión para comentar el trabajo realizado. Puesta en común de las decisiones tomadas para aprobación del tutor |
| **Entradas:** | - |
| **Recursos necesarios:** | Word |
| **Salidas:** | - |

Tabla : Reuniones con el tutor del proyecto I

#### Desarrollo

* **Paquete de trabajo 5: Creación ontologías propias**

|  |  |
| --- | --- |
| **Duración:** | 20 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Dado que algunos datos de los ficheros en formato CSV no podían ser expresados con las ontologías exploradas se crearon ontologías propias para representar ciertos datos |
| **Entradas:** | - |
| **Recursos necesarios:** | Internet, editor de textos |
| **Salidas:** | Ontologías propias para la representación en RDF de los datos |

Tabla : Creación ontologías propias

* **Paquete de trabajo 6: Implementación software creación RDF**

|  |  |
| --- | --- |
| **Duración:** | 90 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Definición de todas las clases Clojure y Java para la representación en RDF de los CSV iniciales |
| **Entradas:** | Conjunto de datos iniciales |
| **Recursos necesarios:** | Java, Eclipse, CounterClockWise, Clojure, Grafter |
| **Salidas:** | Representación en RDF de cada dataset |

Tabla : Implementación software creación RDF

* **Paquete de trabajo 7: Definición e implementación de pruebas Shacl**

|  |  |
| --- | --- |
| **Duración:** | 40 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Se definen las pruebas Shacl contra las que se evaluará el RDF generado y se implementa la funcionalidad que ejecutara las pruebas |
| **Entradas:** | Representación en RDF de cada dataset |
| **Recursos necesarios:** | Ficheros RDF creados, Eclipse, Java, API Shacl, Lenguaje Shacl |
| **Salidas:** | Resultados de la evaluación del RDF |

Tabla : Definición e implementación de pruebas Shacl

* **Paquete de trabajo 8: Implementación software descubrimiento enlaces**

|  |  |
| --- | --- |
| **Duración:** | 20 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Estudio de la herramienta Silk y la mejor configuración para conseguir enlaces entre dos fuentes de datos. Implementación de herramienta que ejecuta Silk |
| **Entradas:** | Triple Store propia y externa |
| **Recursos necesarios:** | Silk, Triple Store propia y externa, Java |
| **Salidas:** | Funcionalidad descubrimiento enlaces implementada y enlaces descubiertos |

Tabla : Implementación software descubrimiento enlaces

* **Paquete de trabajo 9: Implantación servidor Linked Data**

|  |  |
| --- | --- |
| **Duración:** | 20 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Estudio herramienta Pubby y configuración para su uso con nuestra base de datos |
| **Entradas:** | - |
| **Recursos necesarios:** | Base de datos propia |
| **Salidas:** | Servidor Linked Data implantado |

Tabla : Implantación servidor Linked Data

* **Paquete de trabajo 10: Creación SPARQL Endpoint**

|  |  |
| --- | --- |
| **Duración:** | 180 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Creación SPARQL Endpoint para permitir al usuario ejecutar sus queries en una interfaz amigable. Creación funcionalidades para la representación de la información de diversas formas |
| **Entradas:** | - |
| **Recursos necesarios:** | Java, D3, Javascript, Jquery, Bootstrap, HTML, CSS, base de datos propia |
| **Salidas:** | SPARQL Endpoint creado |

Tabla : Creación SPARQL Endpoint

* **Paquete de trabajo 11: Reuniones con el tutor del proyecto**

|  |  |
| --- | --- |
| **Duración:** | 15 horas |
| **Encargado:** | Mishel Uchuari, Oskar Casquero |
| **Descripción:** | Reuniones continuas para la discusión de los avances del proyecto |
| **Entradas:** | - |
| **Recursos necesarios:** | - |
| **Salidas:** | - |

Tabla : Reuniones con el tutor del proyecto II

#### Pruebas

* **Paquete de trabajo 12: Pruebas**

|  |  |
| --- | --- |
| **Duración:** | 20 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Definición y ejecución de pruebas sobre el software creado. Corrección fallos |
| **Entradas:** | Software creado |
| **Recursos necesarios:** | Word |
| **Salidas:** | Listado de pruebas con sus resultados |

Tabla : Pruebas

#### Memoria

* **Paquete de trabajo 13: Memoria**

|  |  |
| --- | --- |
| **Duración:** | 50 horas |
| **Encargado:** | Mishel Uchuari |
| **Descripción:** | Definición de la memoria del proyecto creado: descripción de todas las etapas del proyecto, creación de gráficos, etc. |
| **Entradas:** | - |
| **Recursos necesarios:** | Word, Cacoo, Excel |
| **Salidas:** | Memoria proyecto |

Tabla : Memoria

* **Paquete de trabajo 14: Revisión memoria por parte del tutor del proyecto**

|  |  |
| --- | --- |
| **Duración:** | 10 horas |
| **Encargado:** | Oskar Casquero |
| **Descripción:** | Revisión de la memoria del proyecto |
| **Entradas:** | Memoria |
| **Recursos necesarios:** | Word |
| **Salidas:** | Memoria revisada |

Tabla : Revisión memoria por parte del tutor del proyecto

En la siguiente tabla se especifica el total de horas que fueron necesarias para completar el proyecto:

|  |  |
| --- | --- |
| **Tarea** | **Tiempo Invertido** |
| **Investigación** | **55 horas** |
| Análisis de objetivos | 25 horas |
| Análisis herramientas necesarias | 15 horas |
| Análisis de ontologías a utilizar | 10 horas |
| Reuniones con el tutor del proyecto | 5 horas |
| **Desarrollo** | **385 horas** |
| Creación ontologías propias | 20 horas |
| Implementación software para la creación RDF | 90 horas |
| Definición e implementación de pruebas Shacl | 40 horas |
| Implementación software descubrimiento enlaces | 20 horas |
| Implantación servidor Linked Data | 20 horas |
| Creación SPARQL Endpoint | 180 horas |
| Reuniones con el tutor del proyecto | 15 horas |
| **Pruebas** | **20 horas** |
| Pruebas | 20 horas |
| **Memoria** | **70 horas** |
| Memoria | 60 horas |
| Revisión memoria por parte del tutor del proyecto | 10 horas |
| **Total horas invertidas** | **530 horas** |

Tabla : Horas invertidas en la realización del proyecto

## Diagrama Gantt

El total de horas invertidas en el proyecto fue de 530. Teniendo en cuanta que a la semana se realizaba una media de 20 horas de trabajo, el trabajo se terminó en aproximadamente 27 semanas, es decir casi 7 meses. Todo esto trabajando de lunes a domingo, incluyendo festivos, cerca de 3 horas al día.

En el siguiente grafico Gantt se representa la planificación temporal por semanas. El proyecto se empezó en julio del 2017 y se terminó en febrero del 2018.

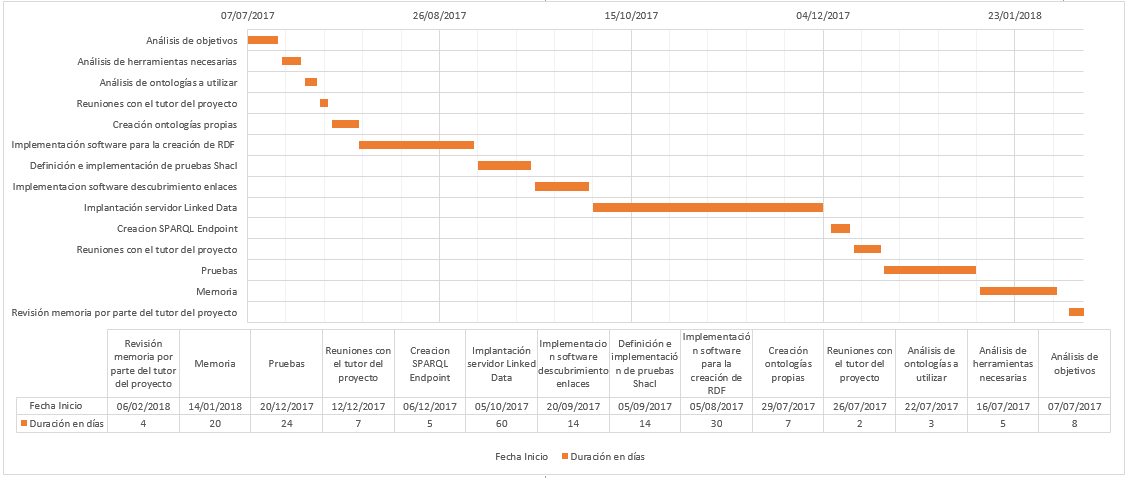


Figura : Diagrama Gantt del proyecto

## Cálculos, algoritmos

En este apartado se ampliará la información ofrecida en la sección “Selección/Descripción de solución propuesta”. Se analizará en detalle las tecnologías utilizadas y la forma en la que se utilizaron. Se empezará por una descripción de los datos empleados a lo largo del proyecto para la generación de RDF y se continuará con las distintas fases ya planteadas en secciones anteriores.

### Los datos

Los datos utilizados para la realización del proyecto son datos públicos publicados por Open Data Euskadi. Open Data Euskadi es una iniciativa del Gobierno Vasco en cuanto a la publicación abierta y libre de los datos de los que disponen y que no tienen restricciones de privacidad, seguridad o propiedad.

Se utilizarán 4 datasets pertenecientes a distintas categorías de Open Data Euskadi: calidad del aire, estaciones meteorológicas: lecturas recogidas, relación de puestos de trabajo y evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual. Dada la importancia del contenido de cada dataset en la elaboración de su representación en RDF se elaboró por cada uno de ellos su modelo de dominio.

#### Calidad del aire

El departamento de Medio Ambiente y Política Territorial del Gobierno Vasco es el encargado de la monitorización de los niveles de contaminación en la Comunidad Autónoma Vasca a través de la Red de Control de Calidad del Aire. Esta red dispone de diversos analizadores y sensores que miden los contaminantes.

En Open Data Euskadi sobre cada sensor se publican datos diarios y horarios. En este caso utilizaremos los datos pertenecientes a la estación de Av. Gasteiz recogidos durante el mes de febrero y marzo del año 2017. En la siguiente figura utilizando un modelo de dominio se representan los distintos datos contenidos en el dataset.

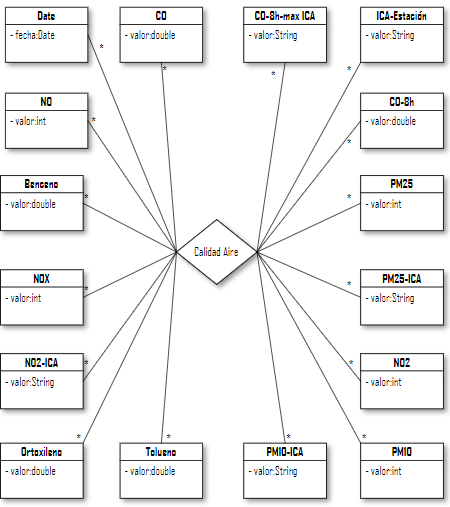


Figura : Modelo dominio dataset “Calidad del Aire”

El CSV podrá tener distintos valores en cada medición y estas se realizan sobre elementos como óxidos de nitrógeno, monóxido de carbono, benceno, partículas en suspensión, etc.

#### Estaciones meteorológicas: lecturas recogidas

Open Data Euskadi publica la información bruta recogida en las estaciones de la Dirección de Meteorología del Gobierno Vasco. La estructura de los datos varía en función de cada estación y se representan en intervalos mensuales con diferencias temporales en cada medición de 10 minutos.

En este caso se han usado los datos pertenecientes a la estación c040 durante el mes de enero del 2017. En la figura 14 se exponen los datos del dataset: mediciones de la dirección del viento, la humedad, la temperatura del aire, etc.

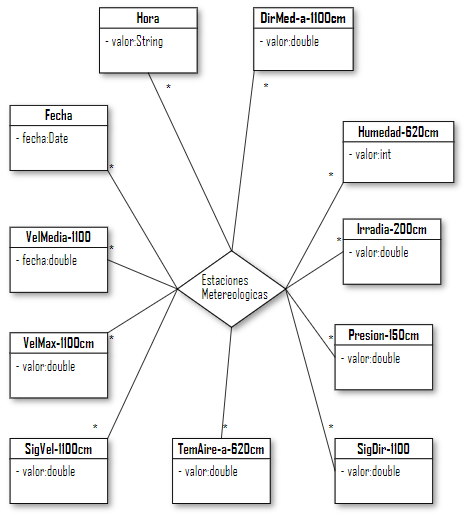


Figura : Modelo dominio dataset “Estaciones meteorológicas: lecturas recogidas”

#### Relación de puestos de trabajo

Se publica también información relativa a los puestos de trabajo de funcionarios/as, personal laboral y eventuales al servicio de la Administración General de la Comunidad Autónoma de Euskadi. Esta información contiene la denominación de los puestos, su nivel de complemento de destino, perfil lingüístico asignado al puesto y su fecha de preceptividad en caso de constar de una, departamento u Organismo Autónomo al que pertenece, centro orgánico y lugar de trabajo, entre otras. En la figura 15 se presenta su modelo de dominio.

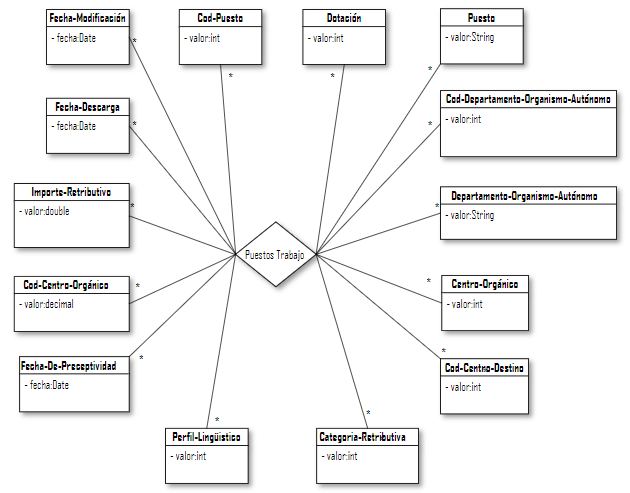


Figura : Modelo dominio dataset “Relación de puestos de trabajo”

#### Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual

El portal de Open Data Euskadi publica datos referentes a las distintas retribuciones brutas que reciben los altos cargos, asesores y el resto del personal eventual del Gobierno Vasco, sus Organismos Autónomos y Entes Públicos dependientes desde el 2009 hasta la actualidad. En la figura 16 se representa la información contenida en el dataset.

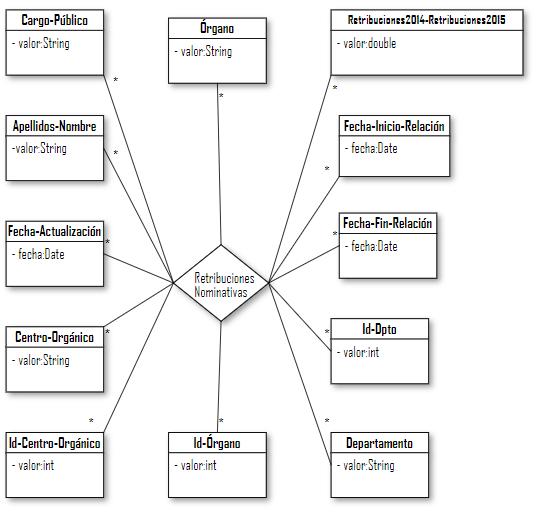


Figura : Modelo dominio dataset “Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual”

### Generación RDF

Para la generación de Datos Enlazados se utilizó Grafter. Grafter permite la creación de RDF a partir de ficheros en formato CSV.

Uno de los primeros pasos a seguir en la creación de RDF es la elección de ontologías. En este caso se utilizaron ontologías reconocidas a nivel mundial como Schema.org, Data cube, Basic Geo, ontologías creadas por Eionet, etc. Además, se definieron ontologías propias para representar ciertos datos.

El esquema seguido es el siguiente:

1. Definición de ontologías a emplear.
2. A los datos contenidos en el CSV se les aplican funciones para transformarlos y darles significancia RDF. A las fechas, por ejemplo, se les da formato de fecha RDF, lo mismo ocurre con los tipos de datos numéricos dependiendo de su tipo.
3. Se crean URIs para identificar los datos del CSV, en adelante recursos.
4. Se genera RDF integrando los predicados elegidos de las ontologías y las URIs de los recursos.

Todo el proceso de conversión a RDF se realiza mediante clases Clojure. Java sólo se utiliza para inicializar el proceso.

A continuación, mediante un ejemplo, utilizando el dataset perteneciente a *calidad del aire*, se mostrará cómo se construyó su representación en RDF.

Las lecturas tomadas en el sensor de Av. Gasteiz tendrán asociada una fecha, unas mediciones y los resultados de estas. Por ello, se decidió que la representación en RDF de estos datos seguiría el esquema representado en la figura 17.

En él se puede observar que se representa en RDF la fecha en la que se realizó la observación, el lugar donde se hizo, un *identificador*, un comentario sobre ella y los distintos tipos de mediciones que se realizaron. Sobre las mediciones se almacena su *identificador,* el *tipo* de medición que es, el valor obtenido como resultado y la unidad de medida en la que se encuentra representado dicho resultado.

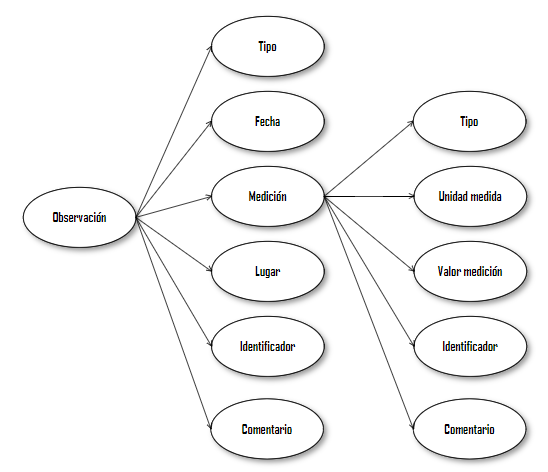


Figura : Esquema general de la representación en RDF del CSV perteneciente a “Calidad del aire”

El RDF generado es muchísimo más extenso que el que se representa en la figura 18, pero para su correcta visualización se restringieron los datos para poder representarlos en forma de gráfico. Como se puede observar se encuentran representados tres recursos principales: uno de tipo observación y dos de tipo medición.

Se eligió representar en este gráfico las mediciones de CO y NO2AirQuality porque son claros ejemplos de los dos tipos de mediciones encontradas en el dataset: mediciones que tienen como resultados valores literales y las que tienes resultados de tipo literal. En la figura 18 se puede apreciar que los resultados literales tienen representación en dos idiomas: euskera y castellano.

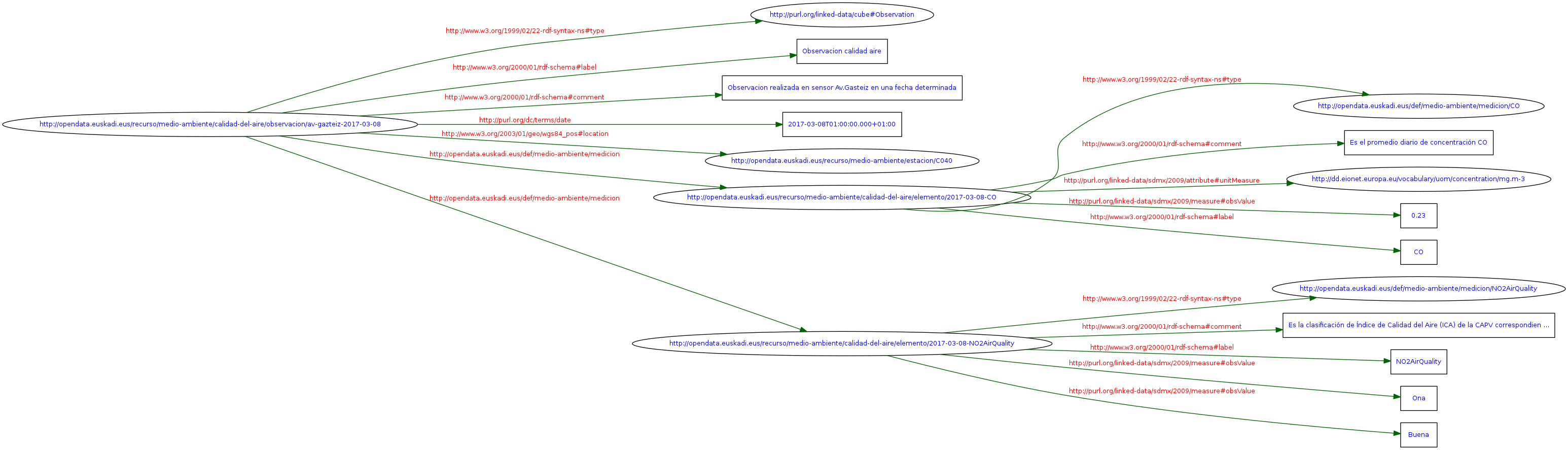


Figura : RDF dataset “Calidad del aire”

### Validación RDF

La validación del RDF generado consiste en la definición de unas reglas Shacl específicas que cada RDF debía cumplir. Para llevarlo a cabo se utilizó la API Shacl que implementaba las reglas Shacl del estándar del W3C.

Generalmente las reglas especificadas para cada RDF eran diferentes. Se fijaron teniendo en cuenta los datos disponibles en cada uno de ellos. Es decir, las reglas se formaron de tal forma que se tuviese control absoluto sobre los datos generados, si se consideraba estrictamente necesario que un RDF tuviese ciertas características en cuanto a los datos representados estas se especificaban.

A pesar de todo, hubo pruebas comunes a todos los RDF. Convencionalmente, todo recurso representado en RDF debe tener un *identificador* además de un recurso asociado que especificara el *tipo* de dato que es, es decir, un *rdfs:label* y un *rdfs:type*. *Rdf:label* y *rdfs:type* son predicados para describir recursos, para asignar a cada recurso un *identificador* y un *tipo* respectivamente.

Se categorizaron las distintas pruebas siguiendo un esquema de relevancia de las posibles faltas que podía cometer el RDF evaluado: *Violation* que eran faltas graves, *Warning* faltas a tener en cuenta e *Info* faltas que servían de información pero que no significaban que el RDF tuviese algún problema. A efectos de este TFG se consideró que cualquier RDF que no tuviese ninguna falta de tipo *Violation* era *válido* y de calidad.

Cabe resaltar que se fijó como prueba común a todos los RDF examinados que tuviesen *identificador* y *tipo* asociado siendo categorizada su falta como *Violation* y dando como resultado que el RDF se tomara como no válido. En la figura 19 se muestra un ejemplo de definición de pruebas Shacl.

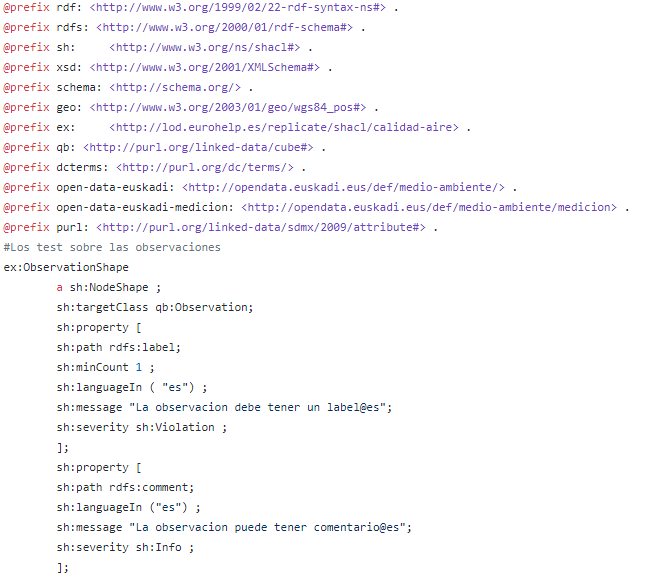


Figura : Ejemplo definición pruebas Shacl en la que se fija como *Violation* no poseer *identificador*

### Descubrimiento Enlaces

La fase de descubrimiento de enlaces consistía en crear o usar un software capaz de explotar la información que poseíamos e intentar enlazarla con información ya existente en la web. Para llevarlo a cabo se utilizó Pubby.

Pubby es una herramienta que partiendo de un archivo de configuración integra dos bases de datos distintas e intenta buscar nexos entre ellas. Esto lo realiza partiendo de un archivo de configuración donde se le especifica dónde y cómo buscar.

De esta manera examinando los datos de los que se disponía se concluyó que el único RDF creado que permitía ser empleado con este fin dada la naturaleza de sus datos era el perteneciente a “Tablas retributivas de los miembros del gobierno, altos cargos y personal eventual”*.* Este dataset contenía datos entre otros, de altos cargos del gobierno.

Algunos de los altos cargos del gobierno tendrían una bibliografía en Wikipedia, y a su vez en DBpedia. Es así como se decidió tomar como fuente de datos externa para Silk, DBpedia, y como fuente de datos propia la Triple Store donde se estaba almacenando todo el RDF generado.

Silk utiliza un archivo de configuración XML, donde además de configurar las distintas fuentes de datos se especifica el “aspecto” que tendrán las URIs que serán consumidas por Silk. Uno de los parámetros más importantes a configurar es el modo en el que se realizarán la exploración de la información contenida en las distintas bases de datos. En la figura 20 se especifica el modo en el que se configura Silk para el uso de distintas bases de datos.



Figura : Ejemplo archivo configuración Silk para la configuración de las Triple Stores a utilizar

En este caso para realizar un filtro inicial sobre toda la información contenida en las bases de datos usadas, se fijó que de DBpedia se debían descartar todos los recursos que no fueran de tipo *dbpedia-ontology:Politician* y en nuestra base datos se debían eliminar todos los que no fuesen de tipo *schema:Person.* Estos tipos hacen referencia a ontologías creadas por la DBpedia y Schema.org.

Adicionalmente se especificó que los recursos encontrados que tuviesen relación con los datos de los que disponíamos, se debían enlazar con estos utilizando el predicado *owl:sameAs*, que les otorgaba igualdad. Es decir, si teníamos un recurso X en nuestro repositorio que se encontraba descrito en otro recurso en la DBpedia, se enlazaba con él con significancia de igualdad. En la figura 21 se ilustra la configuración de Silk en este aspecto:

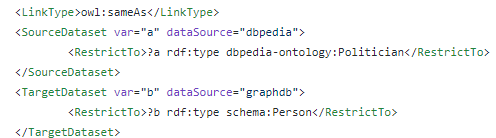


Figura : Ejemplo archivo configuración Silk para la limitación de los recursos a tener en cuenta

Para el filtro de comparación definitivo se compararon los labels de los recursos restantes, en otras palabras, los *identificadores* de los recursos que no habían sido descartados en el filtro anterior. Para realizarlo se utilizó la distancia de Levenshtein. La distancia de Levenshtein es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Es decir, se compararían los *identificadores* de los recursos intentando encontrar semejanzas entre ellos.

La parte del archivo de Silk que hace referencia a esta configuración se muestra en la figura 22. Como se puede observar en ella a los recursos se les realiza una transformación, pasar a minúsculas sus *identificadores* para realizar la comparación de forma más efectiva.

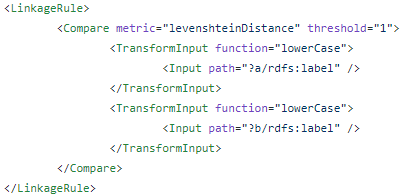


Figura : Ejemplo archivo configuración Silk: configuración del filtro de recursos

Por último, se especifica dónde y de qué forma se desea guardar la información generada. En este caso se usó un fichero de texto en RDF, ya que, aunque Silk permite almacenar directamente los enlaces creados en la Triple Store que se desee, los medios que utiliza para realizarlo están desactualizados y no todas las Triple Stores permiten que se realicen de esa forma. Por ello, los enlaces descubiertos durante esta fase serán almacenados en nuestra Triple Store posteriormente a través de un programa Java.

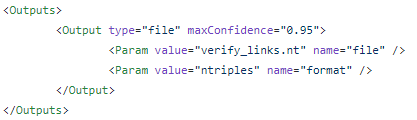


Figura : Ejemplo archivo configuración Silk para la configuración del medio en el que se almacenarán los enlaces descubiertos

### Servidor Linked Data

En la realización del SPARQL Endpoint se utilizó Pubby como ya se mencionara con anterioridad. Pubby utiliza un archivo de configuración donde se especifica dónde están almacenados los datos a los que debe proveer de una interfaz gráfica y el aspecto que tendrán las URIs que identificarán a esos datos.

El archivo de configuración está escrito en RDF Turtle. En la figura 24 se muestra como se configuró Pubby.

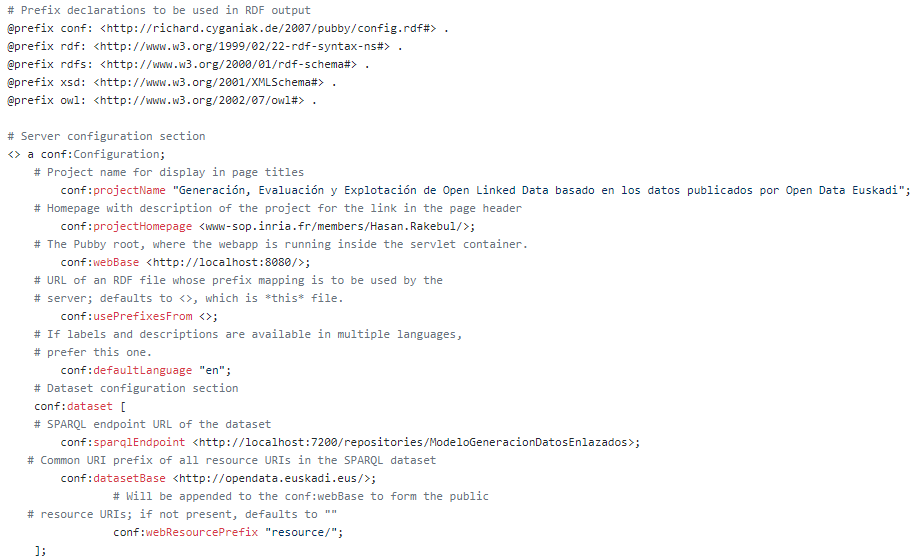


Figura : Ejemplo de archivo XML de configuración de Pubby

### SPARQL Endpoint

La interfaz gráfica que actuará como SPARQL Endpoint en este trabajo consta de dos partes claramente diferenciadas: el área donde el usuario ejecuta su query y en la que se muestran los resultados de la ejecución, para la primera se utilizó Yasqe y en la segunda Bootstrap y D3.

Yasqe es una librería Javascript que permite la creación de un editor de queries SPARQL. Con su uso se añadían funcionalidades muy útiles como que las queries fuesen resaltadas en caso de tener errores sintácticos. En la figura 25 se presenta el resultado de usar Yasqe en este proyecto.

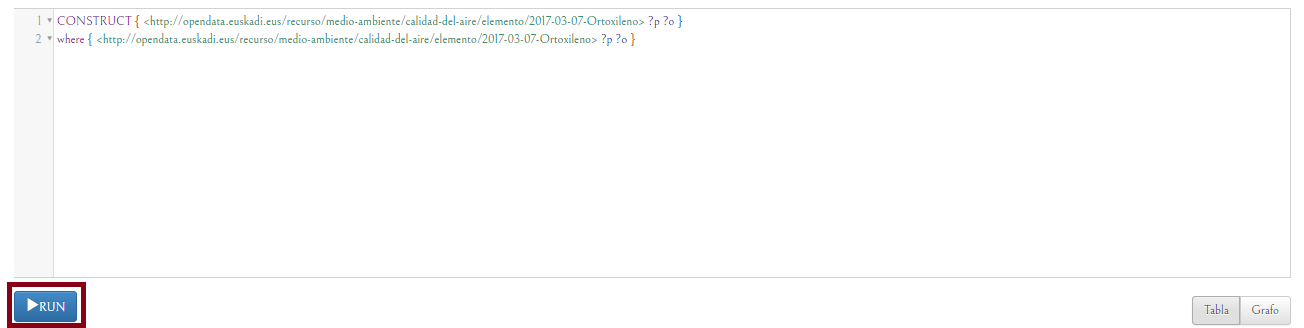


Figura : Editor queries SPARQL

Para la representación de la información resultante de las queries SPARQL se crearon dos modos de visualización. Uno en forma de tabla y el otro en forma de grafo. La representación en forma de tabla se creó para mostrar al usuario los datos en una forma conocida y habitual, similar a como se presenta en las bases de datos relacionales. Sin embargo, la representación en forma de grafo no es tan común ni siquiera en bases de datos orientadas a grafos RDF, aunque ya muchas de ellas la están implementando.

La visualización en forma de grafo fue creada con la intencionalidad de que el usuario visualizara los resultados RDF representados en forma de grafo. Este tipo de visualización es muy útil ya que con él los nexos entre los recursos resultan evidentes. Para hacerlo como se comentaba anteriormente se utilizó D3.

D3 permitía representar resultados en formato JSON en forma de grafo. Se partió de ejemplos encontrados en Internet para crear estos gráficos, pero estos fueron adaptados y se les añadió muchísima más funcionalidad.

Entre las diferencias encontradas entre el gráfico generado para este proyecto y el gran número de ejemplos que pululan por Internet, se encuentra que nuestro grafo está pensando específicamente para representar datos RDF. Por ello se creó un algoritmo que diferenciaba el tipo de dato que estaba recibiendo: si era un recurso, es decir una URI, o un literal, o dicho de otra forma un valor numérico o texto.

Si el dato recibido era un recurso el nodo que se creaba en el grafo para representarlo era un circulo, si se trataba de un literal se representaba mediante un rectángulo. Todo esto siguiendo el estándar de representación de RDF.

Otras funcionalidades que se integraron en el gráfico fue la capacidad de explorar recursos dentro del grafo. Esto proporcionaba un medio para explotar una de las ventajas del uso de RDF que es la capacidad de ir descubriendo más información a través de los recursos publicados.

Por ejemplo, si se solicita información sobre las observaciones correspondientes a calidad del aire en una fecha determinada se obtendrá información sobre las mediciones que se realizaron en esa observación, tal y como se comentaba en la subapartado “Generación de RDF” del apartado “Cálculos, algoritmos”. En el grafo diseñado si se quisiera obtener información más detallada sobre una medición concreta representada en ese grafo de resultados bastaría con hacer doble click sobre él, a partir de ahí, se generaría una nueva tabla de resultados específicos al recurso explorado que adicionalmente, podría ser visualizado nuevamente en forma de grafo.

Otro aspecto interesante es la focalización de información de un grafo. Sucede que al representar un conjunto grande de datos mucha información se solapa, por lo que para este proyecto se implementó que al pasar el ratón sobre un recurso concreto, el resto del grafo que no formara parte de ese recurso y sus adyacentes se desdibujara permitiendo una visualización más clara de ese recurso.

Para solventar también la aglomeración de información al representar el grafo se fijó que los *predicados* que unieran las tripletas que forman el RDF sólo fuesen visibles al pasar el ratón por la arista que a la que perteneciesen. Gráficos donde se ven en acción estas funcionalidades pueden ser encontrados en “ANEXO II: Planos, esquemas, diseño de detalle o de bajo nivel, código” en el apartado “Casos de Uso” concretamente en el caso de uso llamado “Ver resultados en forma de grafo”.

## Descripción de los resultados

Al concluir el proyecto se ha conseguido representar en RDF ficheros en formato CSV pertenecientes a las categorías especificadas al inicio del proyecto. Adicionalmente, en la medida de lo posible se han usado ontologías ya existentes con lo cual se ha provisto de mucha significancia a los datos.

Para los casos en los que no era posible se definieron predicados o prefix siguiendo un estándar del W3C, el NTI, lo que también provee a los datos calidad en cuanto a qué es el estándar para la interoperabilidad de la información del sector público.

Los archivos RDF generados cumplen con las reglas dispuestas para su evaluación siguiendo la norma Shacl del W3C. Con lo cual, los medios utilizados para la realización de este apartado siguen un estándar muy conocido y usado actualmente para la validación de grafos RDF, otorgándole más valor a este proyecto.

Además de representar mediante RDF los datasets publicados por Open Data Euskadi ya mencionados, para el CSV perteneciente a la categoria “evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual” se consiguió encontrar nexos con información ya existente en la web. Esto otorga más valor al trabajo realizado ya que al explorar los datos de retribuciones nominativas se podrá además seguir obteniendo conocimiento de fuentes distintas a las nuestras.

Para concluir cabe mencionar las distintas formas de consumición de los datos que se le otorgo al usuario. A través del Servidor Linked Data y el SPARQL Endpoint se podían explorar los datos a través de diversos medios: por medio de visualizaciones web, a través de tablas y través de gráficos. Esto tuvo como consecuencia que los datos pudiesen ser examinados de forma intuitiva por el usuario.

# CONCLUSIONES

Una vez concluido el proyecto, volviendo la vista atrás y analizando los objetivos planteados en sus inicios se puede afirmar que han sido cumplidos. Pese a las dificultades surgidas a lo largo de él por la poca experiencia en varias de las tecnologías utilizadas, se cumplieron tanto los objetivos principales como los secundarios.

Cabe mencionar que, pese al buen funcionamiento del sistema construido, este tiene ciertas limitaciones, ya que se deben poseer conocimientos de programación para utilizarlo en el caso de querer elaborar más RDF. De cara al futuro se considera la posibilidad de integrar el código existente con una interfaz gráfica que permita a cualquier tipo de usuario utilizarla de forma sencilla sin tener que codificar.

Otra mejora interesante a crear a futuro es la creación un asistente que partiendo de un banco de datos con ontologías ya existentes asesore al usuario en la creación de RDF. El usuario podría definiría la naturaleza de sus datos de partida y su contenido en sí para ayudar a que el asistente le haga sugerencias adecuadas. A partir de ahí, él podría elegir de entre las ontologías planteadas cuáles se adaptarían mejor a su información y empezar con el proceso de generación de RDF. Esta herramienta tendría gran utilidad ya que ahorraría en gran medida el proceso de investigación para la elección de ontologías.

Además, debido a la complejidad y la comúnmente poca experiencia en SPARQL, realizar consultas sobre datos representados en RDF puede resultar complicado y tedioso. Por ello se podría estudiar también la viabilidad de crear un *traductor* de lenguaje humano a SPARQL, definir unas directrices para que el usuario exprese en lenguaje no técnico siguiendo unas estructuras definidas por el programador los datos que quiere obtener y que el sistema de alguna forma transforme esa petición a SPARQL.

Finalmente, se considera que la realización de este proyecto ha supuesto un gran crecimiento profesional y ha aumentado las capacidades de resolución e iniciativa del alumno. No solo se ahondado en tecnologías inexploradas anteriormente, sino que se ha trabajado con varios lenguajes desconocidos hasta el momento por él.

Todas estas novedades en cuanto a tecnologías y lenguajes produjeron en varias ocasiones atascos lo que ocasionó retrasos en la finalización de ciertos apartados, pero como se comentaba anteriormente, esto promovió la resolución del alumno ya que éste no contaba con referentes expertos en ciertas tecnologías a los que consultar, sólo información en Internet o foros de ayuda y al ser los Datos Enlazados un tema novedoso, en ciertos casos, era escasa.

# BIBLIOGRAFIA

[1] http://scielo.isciii.es/scielo.php?script=sci\_arttext&pid=S0212-16112012000200001

[2] http://datos.gob.es

[3] http://www.eionet.europa.eu/

[4] http://silkframework.org/

[5] https://github.com/TopQuadrant/shacl

[6] https://www.w3.org/TR/shacl/

[7] https://grafter.org

[8] http://yasqe.yasgui.org/

[9] https://getbootstrap.com/

[10] http://opendata.euskadi.eus/inicio/

[11] https://es.wikipedia.org

[12] https://dbpedia.org

[13] http://graphdb.ontotext.com/

[14] https://github.com/JoshData/rdfabout/blob/gh-pages/intro-to-rdf.md

[15] Hallo, María Asunción. Martínez González, M. Mercedes; Fuente Redondo, Pablo de la. Las tecnologías de Linked Data y sus aplicaciones en el gobierno electrónico. // Scire. 18:1 (en.-jun. 2012) 49-61. ISSN 1135-3716

[16] Berners-Lee, Tim (2006). Design Note: Linked Data. http://www.w3.org/DesignIssues/LinkedData.html (2011- 07-02)

[17] Cyganiak, Richard; Bizer, Chistian (2011). Pubby. A Linked Data Frontend for SPARQL Endpoints. http://www4. wiwiss.fu-berlin.de/pubby/ (2011-07-02).

[18] Méndez Rodríguez, E. M. (1999). RDF: Un modelo de metadatos flexible para las bibliotecas digitales del próximo milenio. Jornades catalanes de documentació (7. 1999. Barcelona). Les biblioteques i els centres de documentació al segle XXI: peça clau de la societat de la informació. 4, 5 i 6 de novembre de 1999. Palau de Congressos de Barcelona. Barcelona: Col.legi Oficial de Bibliotecaris-Documentalistes de Catalunya, pp. 487-498.

# ANEXO I: Pliego de condiciones. Normativa aplicable. Plan de pruebas

## Plan de pruebas

Al definir la captura de requisitos y los objetivos del proyecto se planteó como uno de ellos la validación del RDF. El objetivo de la validación de RDF consiste en el análisis del RDF generado para corroborar que sigue unas pautas necesarias para considerar que es o no de calidad. Para llevarlo a cabo se crea un documento SHACL adecuado a cada uno con las condiciones que debe cumplir. El funcionamiento de este proceso se explica en la sección “Cálculos, algoritmos” en el subapartado “Validación RDF”.

Las evaluaciones realizadas en cada documento SHACL son muy concretas, por lo que se consideró oportuno especificar en este apartado cuáles eran las que se hacían sobre cada RDF.

Además, se especificarán las distintas pruebas que se realizaron sobre el resto de funcionalidades del sistema. Se dividirán las pruebas en 6 bloques: en el primer bloque se hablará sobre las pruebas relacionadas con la generación de RDF, en el segundo se examinarán las realizadas sobre la evaluación del RDF generado, en el tercero se examinarán las pruebas realizadas sobre el SPARQL Endpoint y sus distintas representaciones gráficas y por último las ejecutadas sobre la funcionalidad de descubrimiento de enlaces.

### PRIMER BLOQUE

#### Generación RDF

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | Número incorrecto de parámetros | Mensaje de error indicando el número de parámetros a introducir | Resultado esperado | **CORRECTO** |
| 2 | Los parámetros no son del tipo especificado | Mensaje de error indicando el tipo de parámetro esperado | Resultado esperado | **CORRECTO** |
| 3 | Número correcto de parámetros y del tipo solicitado | Generación de RDF | Resultado esperado | **CORRECTO** |
| 4 | Generación de RDF para celdas con valores vacíos | Se genera RDF para las celdas que poseen valor | No se genera RDF para ninguno de los elementos de las columnas que posean valores vacíos | **ERROR\*** |

Tabla : Generación RDF

\*Fallo en el identificador 1: El error se corrigió utilizando un método que no tenía en cuenta las celdas con valores vacíos en el CSV.

### SEGUNDO BLOQUE

#### Evaluación RDF

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | Número incorrecto de parámetros | Mensaje de error indicando el número de parámetros a introducir | Resultado esperado | **CORRECTO** |
| 2 | Los parámetros no son del tipo especificado | Mensaje de error indicando el tipo de parámetro esperado | Resultado esperado | **CORRECTO** |
| 3 | Número correcto de parámetros y del tipo solicitado | Validación RDF | Resultado esperado | **CORRECTO** |

Tabla : Evaluación RDF

##### Evaluación sobre RDF perteneciente a “Calidad del aire”

###### Respecto a las observaciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La observación debería tener asociado un label | La observación tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | La observación puede tener asociado un comentario | La observación puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | La observación debe tener asociada una fecha | La observación tiene asociada una fecha | Resultado esperado | **CORRECTO** |
| 4 | La observación debería tener asociada una geo-localización | La observación tiene asociada una geo localización | Resultado esperado | **CORRECTO** |
| 5 | La observación debería tener asociada diferentes mediciones | La observación tiene asociadas diferentes mediciones | Resultado esperado | **CORRECTO** |

Tabla : Evaluación RDF “Calidad del aire” respecto a las observaciones

###### Respecto a las mediciones con resultados numéricos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La medición debería tener asociado un label | La medición tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | La medición puede tener asociado un comentario | La medición puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | La medición debería tener un valor numérico como resultado asociado | La medición tiene un valor numérico como resultado asociado | Resultado esperado | **CORRECTO** |
| 4 | La medición debería tener asociada una unidad de medida | La medición tiene asociada una unidad de medida | Resultado esperado | **CORRECTO** |

Tabla : Evaluación RDF “Calidad del aire” respecto a las mediciones con resultados numéricos

Respecto a las mediciones con resultados literales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La medición debería tener asociado un label | La medición tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | La medición puede tener asociado un comentario | La medición puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | La medición debería tener un valor literal como resultado asociado | La medición tiene un valor literal como resultado asociado | Resultado esperado | **CORRECTO** |

Tabla : Evaluación RDF “Calidad del aire” respecto a las mediciones con resultados literales

##### Evaluación sobre RDF perteneciente a “Estaciones meteorológicas: lecturas recogidas”

###### Respecto a las observaciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La observación debería tener asociado un label | La observación tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | La observación puede tener asociado un comentario | La observación puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | La observación debe tener asociada una fecha | La observación tiene asociada una fecha | Resultado esperado | **CORRECTO** |
| 4 | La observación debería tener asociada una geo-localización | La observación tiene asociada una geo localización | Resultado esperado | **CORRECTO** |
| 5 | La observación debería tener asociada diferentes mediciones | La observación tiene asociadas diferentes mediciones | Resultado esperado | **CORRECTO** |

Tabla : Evaluación RDF “Estaciones meteorológicas: lecturas recogidas” respecto a las observaciones

###### Respecto a las mediciones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La medición debería tener asociado un label | La medición tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | La medición puede tener asociado un comentario | La medición puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | La medición debería tener un valor numérico como resultado asociado | La medición tiene un valor numérico como resultado asociado | Resultado esperado | **CORRECTO** |
| 4 | La medición debería tener asociado una unidad de medida | La medición tiene asociado una unidad de medida | Resultado esperado | **CORRECTO** |

Tabla : Evaluación RDF “Estaciones meteorológicas: lecturas recogidas” respecto a las mediciones

##### Evaluación sobre RDF perteneciente a “Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | El contrato debería tener asociado un label | El contrato tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | El contrato puede tener asociado un comentario | El contrato puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | El contrato debería tener asociado un empleado | El contrato tiene un empleado asociado | Resultado Esperado | **CORRECTO** |
| 4 | El contrato debería tener asociada la fecha en la que se formalizo | El contrato tiene asociada la fecha en la que se formalizo | Resultado Esperado | **CORRECTO** |
| 5 | El contrato debería tener asociada la fecha en la que terminará | El contrato tiene asociada la fecha en la que terminará | Resultado Esperado | **CORRECTO** |
| 6 | El contrato debería tener asociado el código del departamento en el que se encontrará el puesto | El contrato tiene asociado el código del departamento en el que se encontrará el puesto | Resultado Esperado | **CORRECTO** |
| 7 | El contrato debería tener asociados datos respecto a su departamento gerente | El contrato tiene asociados datos respecto a su departamento gerente | Resultado Esperado | **CORRECTO** |
| 8 | El contrato debería tener asociado el nombre del centro orgánico del puesto | El contrato tiene asociado el nombre del centro orgánico del puesto | Resultado Esperado | **CORRECTO** |
| 9 | El contrato debería tener asociado el identificador del centro orgánico del puesto | El contrato tiene asociado el identificador del centro orgánico del puesto | Resultado Esperado | **CORRECTO** |
| 10 | Sólo si el contrato ha sufrido alguna modificación se debería almacenar la fecha en la que se realizo | Si el contrato ha sido modificado se almacena la fecha en la que se realizo | Resultado Esperado | **CORRECTO** |
| 11 | El contrato debería tener asociado un sueldo correspondiente al puesto | El contrato tiene asociado un sueldo correspondiente al sueldo | Resultado Esperado | **CORRECTO** |
| 12 | El contrato debería tener una 'descripción' del puesto al que se refiere | El contrato tiene asociada una 'descripción' del puesto al que se refiere | Resultado Esperado | **CORRECTO** |

Tabla : Evaluación RDF perteneciente a “Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual”

##### Evaluación sobre RDF perteneciente a “Relación de puestos de trabajo”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | El contrato debería tener asociado un label | El contrato tiene asociado un label | Resultado esperado | **CORRECTO** |
| 2 | El contrato puede tener asociado un comentario | El contrato puede tener asociado un comentario | Resultado esperado | **CORRECTO** |
| 3 | Sólo si el contrato ha sufrido alguna modificación se debería almacenar la fecha en la que se realizo | Si el contrato ha sido modificado se almacena la fecha en la que se realizo | Resultado Esperado | **CORRECTO** |
| 4 | El contrato debería tener asociado el código del puesto al que se refiere | El contrato debería tener asociado el código del puesto al que se refiere | Resultado Esperado | **CORRECTO** |
| 5 | El contrato debería tener almacenada la fecha en la que se realizó la descarga de datos | El contrato tiene almacenada la fecha en la que se realizó la descarga de datos | Resultado Esperado | **CORRECTO** |
| 6 | El contrato debería tener asociado una dotación en referencia al puesto al que se refiere | El contrato tiene asociado una dotación en referencia al puesto al que se refiere | Resultado Esperado | **CORRECTO** |
| 7 | El contrato debería tener asociado el código del departamento en el que se encontrará el puesto | El contrato tiene asociado el código del departamento en el que se encontrará el puesto | Resultado Esperado | **CORRECTO** |
| 8 | El contrato debería tener asociado el código del centro de destino del puesto | El contrato tiene asociado el código del centro de destino del puesto | Resultado Esperado | **CORRECTO** |
| 9 | El contrato debería tener asociado el código del centro orgánico del puesto | El contrato tiene asociado el código del centro orgánico del puesto | Resultado Esperado | **CORRECTO** |
| 10 | El contrato debería tener asociado el nombre del departamento gerente del puesto | El contrato tiene asociado el nombre del departamento gerente del puesto | Resultado Esperado | **CORRECTO** |
| 11 | El contrato debería tener asociado el nombre del centro orgánico del puesto | El contrato tiene asociado el nombre del centro orgánico del puesto | Resultado Esperado | **CORRECTO** |
| 12 | El contrato debería tener asociado el nombre del centro destino del puesto | El contrato tiene asociado el nombre del centro destino del puesto | Resultado Esperado | **CORRECTO** |
| 13 | El contrato debería tener asociado el código de la categoría retributiva del puesto | El contrato tiene asociado el código de la categoría retributiva del puesto | Resultado Esperado | **CORRECTO** |
| 14 | El contrato debería tener asociado el código del perfil lingüístico del puesto | El contrato tiene asociado el código del perfil lingüístico del puesto | Resultado Esperado | **CORRECTO** |
| 15 | El contrato debería tener asociado un sueldo | El contrato tiene asociado un sueldo | Resultado Esperado | **CORRECTO** |
| 16 | El empleado debe tener asociado un label | El empleado tiene asociado un label |  |  |

Tabla : Evaluación RDF perteneciente a “Relación de puestos de trabajo”

### TERCER BLOQUE

#### SPARQL Endpoint

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La query no es de tipo CONSTRUCT ni SELECT | Mensaje de error avisando de que sólo se pueden ejecutar queries de tipo CONSTRUCT o SELECT | Resultado esperado | **CORRECTO** |
| 2 | La query es sintácticamente incorrecta | Mensaje de error avisando de que la query tiene un error de sintaxis | Resultado esperado | **CORRECTO** |
| 3 | La query es sintácticamente correcta y de tipo CONSTRUCT o SELECT | Se ejecuta la consulta que devolverá o no resultados | Resultado esperado | **CORRECTO** |
| 4 | La query es sintácticamente correcta y de tipo CONSTRUCT o SELECT pero no devuelve resultados | Mensaje avisando que no se obtuvieron resultados | Resultado esperado | **CORRECTO** |

Tabla : Pruebas sobre el SPARQL Endpoint

#### Tabla

Estas pruebas se realizan sobre la funcionalidad presente en la interfaz web en la que se genera una tabla representando los resultados obtenidos de la query ejecutada en el SPARQL Endpoint cuando esta es de tipo SELECT o CONSTRUCT.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La query no es de tipo CONSTRUCT ni SELECT | No se genera tabla | Resultado esperado | **CORRECTO** |
| 2 | La query es sintácticamente incorrecta | No se genera tabla | Resultado esperado | **CORRECTO** |
| 3 | La query es sintácticamente correcta y de tipo CONSTRUCT o SELECT y devuelve resultados | Se genera tabla | Resultado esperado | **CORRECTO** |
| 4 | La query es sintácticamente correcta y de tipo CONSTRUCT o SELECT pero no devuelve resultados | No se genera tabla | Resultado esperado | **CORRECTO** |

Tabla : Pruebas sobre la representación de RDF usando tablas

#### Grafo

Las pruebas que se presentan a continuación se ejecutaron sobre la funcionalidad existente en la interfaz web en la que se genera un grafo representando los datos cuando la query ejecutada en el SPARQL Endpoint es de tipo CONSTRUCT.

En este caso no se corroborará si se generará el grafo cuando la query no haya obtenido resultados ya que ese error se coteja y se soluciona en el caso de prueba 4 de la tabla ya que la funcionalidad que genera el grafo imposibilita al usuario de solicitar la representación en forma de grafo hasta que los datos hayan sido representados en forma de tabla.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | La consulta no es de tipo CONSTRUCT | Mensaje indicando que los resultados no se pueden presentar en forma de grafo, este no se genera y se mantienen los resultados representados en forma tabular | Resultado esperado | **CORRECTO** |
| 2 | Se ejecuta una query CONSTRUCT que devuelve un resultado con tamaño adecuado | Se genera el grafo para los resultados obtenidos de la query | Resultado Esperado | **CORRECTO** |
| 3 | Se ejecuta una query CONSTRUCT cuyo resultado es demasiado grande para ser representado en forma de grafo | Mensaje de error indicando que las dimensiones de los resultados son excesivamente grandes como para ser representados en forma de grafo, el grafo no se genera y se mantienen los resultados representados en forma tabular | Resultado esperado | **CORRECTO** |
| 4 | Al hacer click en un elemento para obtener información sólo de ese recurso, la query automática que se ejecuta no obtiene resultados | Mensaje de error y se mantiene el grafo representado hasta el momento | Resultado esperado | **CORRECTO** |
| 5 | Al hacer click en un elemento para obtener información sólo de ese recurso, la query automática que se ejecuta obtiene resultados | Se ejecuta query consulta de información sobre ese recurso y se representa en forma tabular | Resultado esperado | **CORRECTO** |
| 6 | Al pasar el ratón sobre un nodo para focalizar el grafo en él y sus adyacentes el identificador del nodo contiene caracteres especiales | Se focaliza la parte del grafo correspondiente al nodo y sus adyacentes | No se produce la focalización correctamente | **ERROR\*** |
| 7 | Al pasar el ratón sobre un nodo para focalizar el grafo en él y sus adyacentes el identificador del nodo no contiene caracteres especiales | Se focaliza la parte del grafo correspondiente al nodo y sus adyacentes | Resultado Esperado | **CORRECTO** |

Tabla : Pruebas sobre la representación de RDF usando gráficos dinámicos en forma de grafo

\*Fallo en identificador 6: Este error se producía porque la función creada que realizaba las focalizaciones estaba diseñada de tal manera que no soportaba realizar búsquedas de elementos HTML que contuviesen identificadores con caracteres especiales. Se corrigió pasando un filtro a los identificadores que eliminase esos caracteres especiales para que la función funcionase correctamente.

### CUARTO BLOQUE

#### Descubrimiento enlaces

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Descripción | Resultado Esperado | Resultado Obtenido | Funcionamiento |
| 1 | Número incorrecto de parámetros | Mensaje de error indicando el número de parámetros a introducir | Resultado esperado | **CORRECTO** |
| 2 | Los parámetros no son del tipo especificado | Mensaje de error indicando el tipo de parámetro esperado | Resultado esperado | **CORRECTO** |
| 3 | Número correcto de parámetros y del tipo solicitado | Comienzo proceso descubrimiento enlaces que devolverá o no resultados | Resultado esperado | **CORRECTO** |

Tabla : Pruebas sobre la funcionalidad "Descubrimiento Enlaces"

# Anexo II: Planos, esquemas, diseño de detalle o de bajo nivel, código

## Diagrama clases

En la figura 26 y 27 se presenta los diagramas de clases del sistema, exclusivamente se han representado las clases de los lenguajes que ofrecían las funcionalidades principales, las clases referentes a la interfaz gráfica, escritas en HTML, CSS, JavaScript y JQuery, se han obviado. Los diagramas se han divido en dos: el diagrama de clases Java y el de clases Clojure. Como se puede observar examinándolos, la mayor parte de la programación se ha realizado en Clojure.

La peculiaridad de las clases Clojure, en las que existen clases solo con atributos, y clases solo con métodos, se debe a la metodología seguida por Grafter para la creación de RDF. En ella se separan las clases que almacenan los prefix o predicados a utilizar por un lado y las clases que ofrecerán funcionalidades de transformación a los datos y generarán el RDF por otro.

### Diagrama clases Java

* **GraphDB**: Clase que lleva el control de la conexión del sistema con la base de datos GraphDB. Las funcionalidades relacionadas con la conexión, consulta o modificación de los datos.
* **ServGeneradorResultados**: Clase que maneja las peticiones de la interfaz gráfica y le devuelve los resultados que esta solicita.
* **PipelineManager**: Clase encargada de ejecutar a través de Java los pipelines Clojure, es decir, ejecuta las clases Clojure de creación de RDF definidas en el sistema.
* **SilkManager**: Clase encargada de ejecutar Silk para el descubrimiento de enlaces.
* **ShaclManager**: Clase encargada de ejecutar Shacl para la evaluación de RDF.
* **ResultAdapter**: Clase encargada de transformar los resultados obtenidos a través de GraphDB para ser representados estos en forma de grafo.

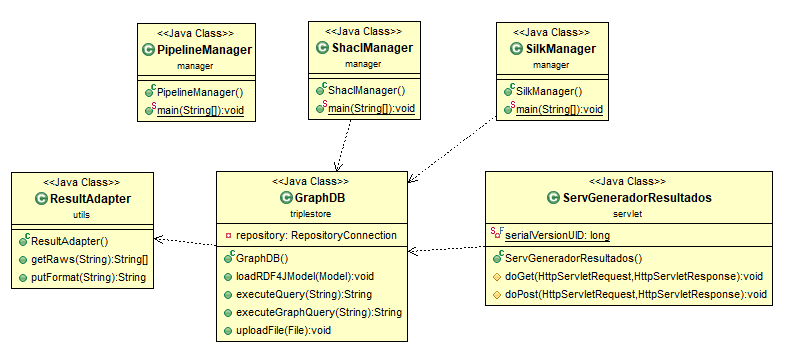


Figura : Diagrama clases Java

### Diagrama clases Clojure

* **CalidadAire:** Pipeline que crea la representación en RDF de los ficheros en formato CSV publicados por Open Data Euskadi en el apartado “Calidad del Aire”.
* **EstacionesMetereologicas**: Pipeline crea la representación en RDF de los ficheros en formato CSV publicados por Open Data Euskadi en el apartado “estaciones meteorológicas: lecturas recogidas”.
* **RelacionesPuestosTrabajo**: Pipeline crea la representación en RDF de los ficheros en formato CSV publicados por Open Data Euskadi en el apartado “Relación de puestos trabajo”.
* **RetribucionesNominativas**: Pipeline crea la representación en RDF de los ficheros en formato CSV publicados por Open Data Euskadi en el apartado a “Evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual”.
* **TransformacionGeneral**: Clase Clojure que posee funciones utilizadas por los distintos pipelines para modificar sus datos.
* **TransformacionesCalidadAire**: Clase Clojure que posee funcionalidades de transformación de datos utilizadas sólo por el pipeline CalidadAire.
* **TransformacionesEstacionesMetereologicas**: Clase Clojure que posee funcionalidades de transformación de datos utilizadas sólo por el pipeline EstacionesMetereologicas.
* **TransformacionesRelacionesPuestosTrabajo**: Clase Clojure que posee funcionalidades de transformación de datos utilizadas sólo por el pipeline RelacionesPuestosTrabajo.
* **TransformacionesRetribucionesNominativas**: Clase Clojure que posee funcionalidades de transformación de datos utilizadas sólo por el pipeline RetribucionesNominativas.
* **Prefix**: Clase Clojure que almacena los prefix y predicados comunes a todos los pipelines.

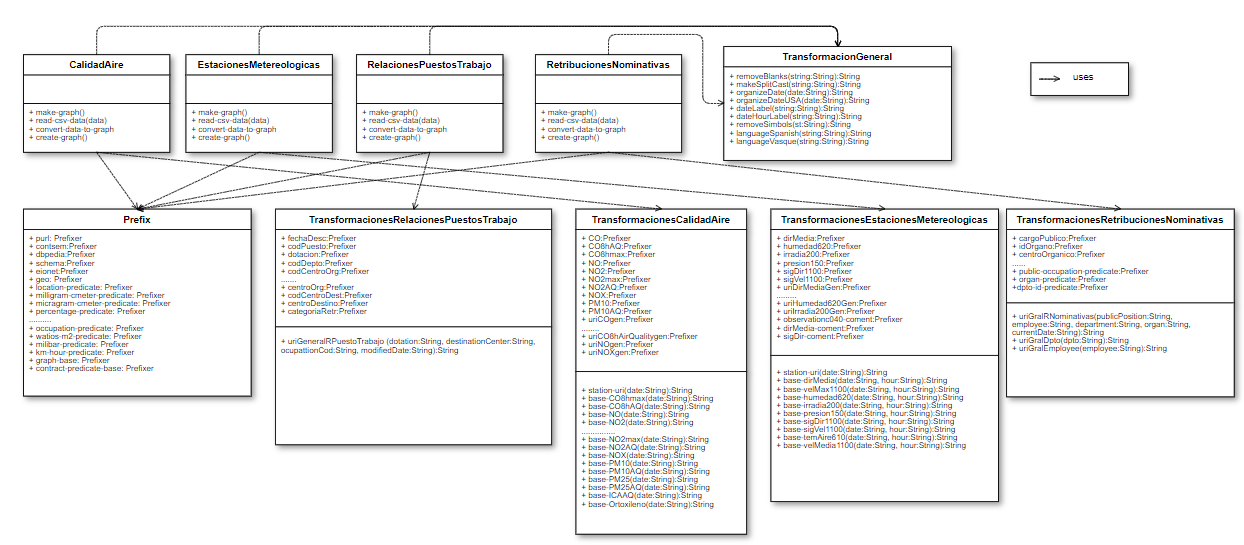


Figura : Diagrama clases Clojure

Diagramas de secuencia

### Diagrama de secuencia funcionalidad generar RDF

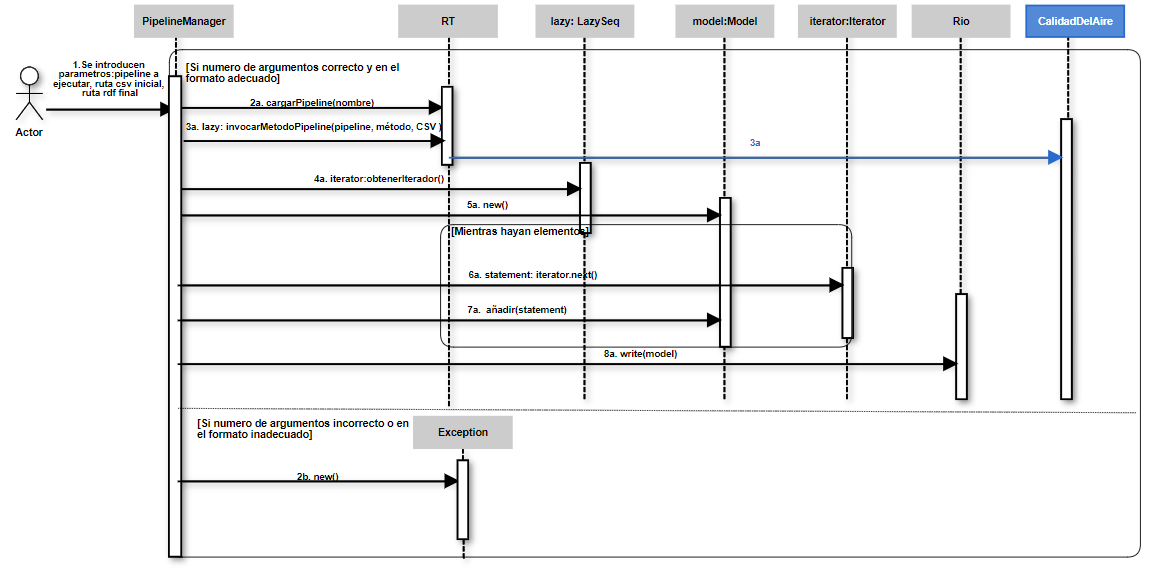


Figura : Diagrama de secuencia funcionalidad generar RDF

### Diagrama de secuencia funcionalidad validación RDF

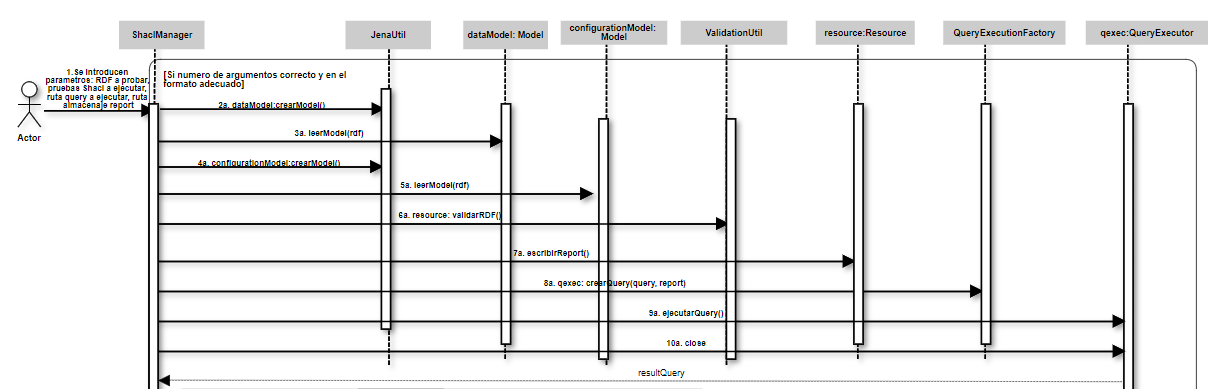


Figura : Diagrama de secuencia funcionalidad validación RDF I/II

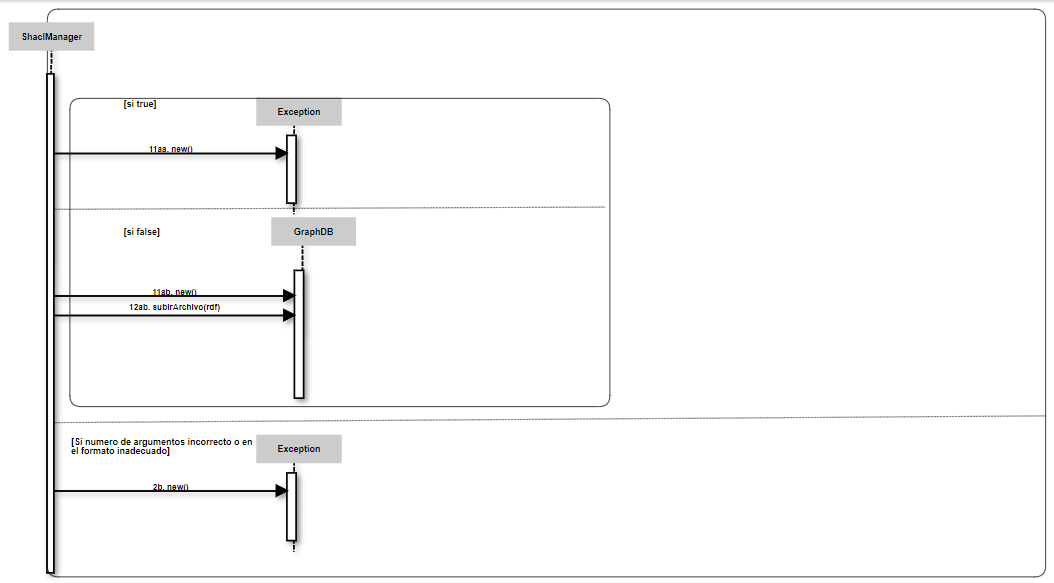


Figura : Diagrama de secuencia funcionalidad validación RDF II/II

### Diagrama de secuencia funcionalidad descubrimiento enlaces

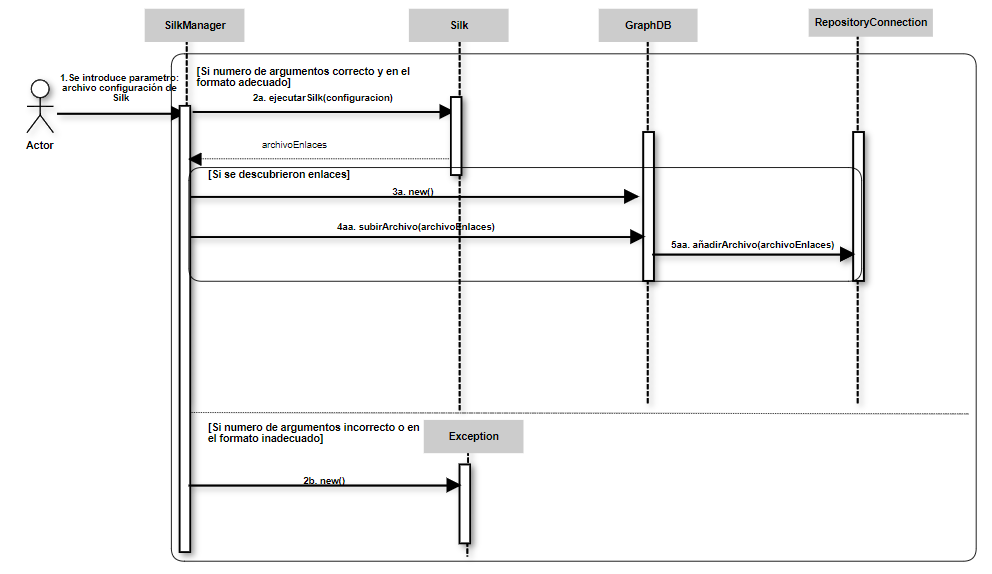


Figura : Diagrama de secuencia funcionalidad descubrimiento enlaces

Modelo de dominio

En la figura 32 se presenta el modelo de dominio del sistema. Como se puede observar contiene una sola relación ternaria, Tripleta. Tripleta hace referencia a la relación formada por el sujeto, objeto y predicado en RDF, formándose las entidades Sujeto, Predicado y Objeto.

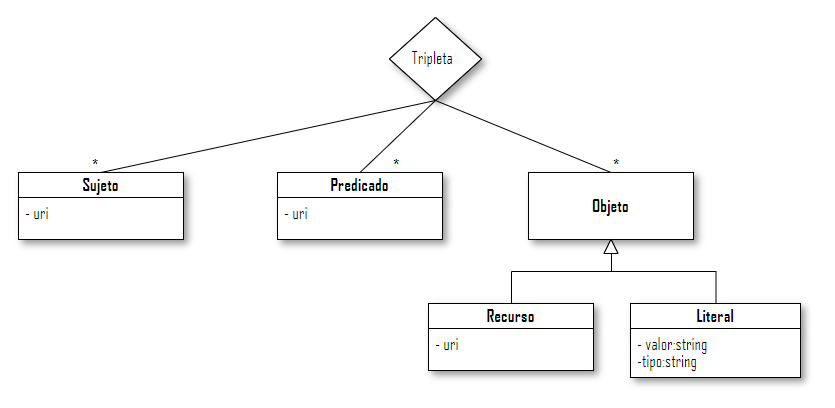


Figura : Modelo de dominio del sistema

Casos de uso

En este apartado se detallarán los casos de uso del sistema. Los casos de uso se separarán en dos grupos: casos y subcasos.

### Esquema general de casos de uso

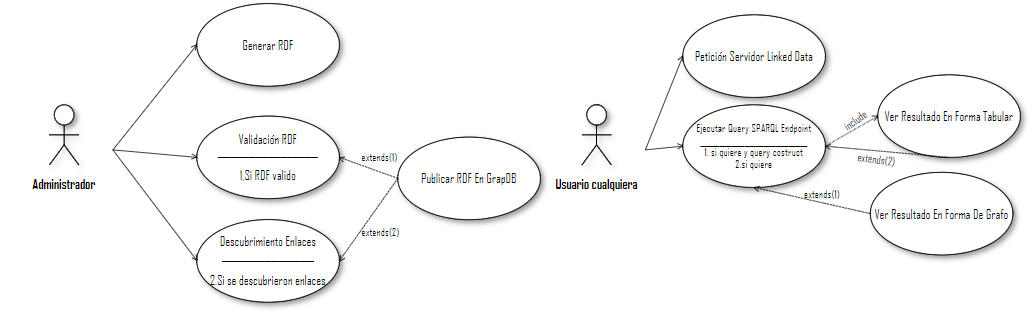


Figura : Diagrama casos de uso sistema

### Jerarquía de actores

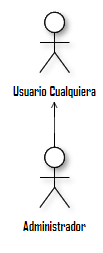


Figura : Diagrama jerarquía de actores en casos de uso del sistema

### Casos de uso generales

* **Petición servidor Linked Data:** Tanto el administrador del sistema como un usuario cualquiera podrán realizar peticiones al servidor Linked Data con negociación de contenido.
* **Generar RDF:** El administrador podrá generar RDF a partir de los modelos creados para los distintos datasets.
* **Validación RDF:** El administrador podrá realizar una validación adecuada a cada RDF creado a partir de los modelos.
* **Descubrimiento enlaces:** El administrador podrá descubrir enlaces relacionados con el RDF creado.
* **Ejecutar query SPARQL Endpoint:** Tanto el administrador del sistema como un usuario cualquiera que tenga acceso al sistema podrá realizar queries SELECT o CONSTRUCT sobre el SPARQL Endpoint creado.

|  |  |
| --- | --- |
| Figura : Caso de uso "Petición servidor Linked Data" | |
| **Nombre:** | Petición servidor Linked Data |
| **Descripción:** Tanto el administrador del sistema como un usuario cualquiera podrán realizar peticiones al servidor Linked Data con negociación de contenido. | |
| **Actores**: Usuario cualquiera, administrador. | |
| **Precondiciones:** - | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario solicita una URI al Servidor Linked Data pudiendo hacerlo de diversas formas de acuerdo al formato que desea recibir.   [Si desea formato HTML]  1.1 Puede introducir la URI directamente en cualquier navegador web o puede hacer uso de otras herramientas que permitan negociación de contenido como INSOMNIA, o Curl desde línea de comandos especificando en las cabeceras que el formato deseado es HTML. (Figura 36 y 37)  [Si desea un formato RDF]  1.2 Puede realizar la petición desde una herramienta que le permita enviar cabeceras como Curl o INSOMNIA especificando el formato RDF en el que desea los resultados. (Figura 38) | |
| **Postcondiciones:** Se le mostrará al usuario la información solicitada en el formato adecuado. | |
| **Interfaz gráfica:**    Figura : Petición a Servidor Linked Data solicitando recurso en formato HTML desde navegador web cualquiera    Figura : Petición a Servidor Linked Data solicitando recurso en formato HTML desde INSOMNIA    Figura : Petición a Servidor Linked Data solicitándole la información en RDF, específicamente en formato RDF/XML | |

|  |  |
| --- | --- |
| Figura : Caso de uso "Generar RDF" | |
| **Nombre:** | Generar RDF |
| **Descripción:** El administrador podrá generar RDF a partir de los modelos creados para los distintos datasets. | |
| **Actores**: Administrador | |
| **Precondiciones:** - | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario ejecuta jar de generación de RDF pasándolo como parámetros el CSV a convertir, el nombre del pipeline que se usará para la conversión y la ruta donde se almacenará el RDF creado. 2. Se muestra mensaje indicando que el RDF se ha generado. (Figura 40) | |
| **Postcondiciones:** Se habrá creado una representación en RDF del CSV. | |
| **Interfaz gráfica:**    Figura : Mensaje de aviso de que el RDF se ha generado | |

|  |  |
| --- | --- |
| Figura : Caso de uso "Validación RDF" | |
| **Nombre:** | Validación RDF |
| **Descripción:** El administrador podrá realizar una validación adecuada a cada RDF creado a partir de los modelos. | |
| **Actores**: Administrador | |
| **Precondiciones:** - | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario ejecuta jar de validación de RDF pasándole como parámetro el RDF a validar, las reglas SHACL que debe cumplir y la ruta donde se almacenará el report.   [Si el RDF es válido]  2.1 Se le mostrará al usuario un mensaje indicado que el RDF es válido además de generarse un report en el que se indica que el RDF es válido. (Figura 42 y 43)  2.2 Se sube el RDF generado a GraphDB  [Si el RDF no es válido]  2.3 Se le mostrará al usuario un mensaje indicando que el RDF no es válido además de generarse un report en el que se indican las causas por las cuales el RDF no es válido. (Figura 44 y 45) | |
| **Postcondiciones:** Se habrá realizado un testeo para comprobar la validez del RDF. | |
| **Interfaz gráfica:**    Figura : Mensaje avisando al usuario que el RDF es válido.    Figura : Report indicando que el RDF es válido    Figura : Mensaje avisando al usuario que el RDF no es válido.    Figura : Ejemplo de report indicando las razones por las cuales el RDF no es válido. | |

|  |  |
| --- | --- |
| Figura : Caso de uso "Descubrimiento enlaces" | |
| **Nombre:** | Descubrimiento enlaces |
| **Descripción:** El administrador podrá descubrir enlaces relacionado con el RDF generado | |
| **Actores**: Administrador | |
| **Precondiciones:** - | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario ejecutar jar de descubrimiento de enlaces pasándole como parámetro el archivo de configuración de Silk. 2. Se genera archivo que contendrá RDF relacionado con la información existente si ha sido correctamente configurado.   [Si se han encontrado enlaces]  3.1 Se suben los enlaces encontrados a GraphDB | |
| **Postcondiciones:** Si habían enlaces a descubrir de acuerdo a las configuraciones realizadas en el archivo de configuración de Silk estos enlaces se almacenarán en el archivo mencionado con anterioridad y se almacenarán en la Triple Store. | |
| **Interfaz gráfica:** - | |

|  |  |
| --- | --- |
| Figura : Caso de uso "Ejecutar query SPARQL Endpoint" | |
| **Nombre:** | Ejecutar query SPARQL Endpoint |
| **Descripción:** Tanto el administrador del sistema como un usuario cualquiera que tenga acceso podrá realizar queries SELECT o CONSTRUCT sobre el SPARQL Endpoint creado. | |
| **Actores**: Usuario cualquiera, administrador. | |
| **Precondiciones:** - | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario escribe su query en el campo de texto y pulsa RUN. (Figura 48) 2. Se le muestra al usuario los resultados de la query en formato tabular. (Figura 49) | |
| **Postcondiciones:** Se visualizan los resultados de la query en forma tabular | |
| **Interfaz gráfica:**    Figura : Query en SPARQL Endpoint    Figura : Resultado query en forma de tabla | |

### Subcasos de uso

* **Ver resultados en forma tabular**: Una vez el usuario (tanto el administrador como un usuario cualquiera) haya realizado una consulta y esta sea de tipo SELECT o CONSTRUCT se verán los resultados de forma tabular. Además, si el usuario ha cambiado la forma de visualizar los datos a forma gráfica y desea volver a verlos de forma tabular, podrá hacerlo sin problema.
* **Ver resultados en forma de grafo**: Una vez el usuario (tanto el administrador como un usuario cualquiera) haya realizado una consulta y esta sea de tipo CONSTRUCT si así lo desea podrá ver los resultados representados en forma de grafo.

|  |  |
| --- | --- |
| Figura : Caso de uso “Ver resultados en forma tabular” | |
| **Nombre:** | Ver resultados en forma tabular |
| **Descripción:** Una vez el usuario (tanto el administrador como un usuario cualquiera) haya realizado una consulta y esta sea de tipo SELECT o CONSTRUCT se verán los resultados de forma tabular. Además si el usuario ha cambiado la forma de visualizar los datos a forma gráfica y desea volver a verlos de forma tabular, podrá hacerlo sin problema. | |
| **Actores**: Usuario cualquiera, administrador. | |
| **Precondiciones:** La query debe ser de tipo SELECT o CONSTRUCT | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario escribe su query en el campo de texto y pulsa RUN 2. Se muestran los resultados en forma tabular (Figura 51)   [Si el usuario ha seleccionado previamente la opción de ver los resultados en forma gráfica y quiere volver a verlos en forma tabular]  2.1 El usuario pulsa “Tabla”( Figura 52) | |
| **Postcondiciones:** Se muestran los resultados de la query en forma tabular | |
| **Interfaz gráfica:**    Figura : Resultados en forma tabular    Figura : Botón que debe pulsar el usuario para volver a visualizar los resultados en forma tabular | |

|  |  |
| --- | --- |
| Figura : Caso de uso "Ver resultados en forma de grafo" | |
| **Nombre:** | Ver resultados en forma de grafo |
| **Descripción:** Una vez ejecutada la query el usuario podrá ver los resultados representados en forma de grafo. | |
| **Actores**: Usuario cualquiera, administrador. | |
| **Precondiciones:** La query se ha ejecutado y es de tipo CONSTRUCT. | |
| **Requisitos no funcionales:** - | |
| **Flujo de eventos:**   1. El usuario pulsa en “Grafo”. (Figura 54) 2. Se muestran los resultados en forma de grafo. (Figura 55)   [Si pincha sobre un nodo concreto]  2.1 Se genera un nuevo grafo con información concreta del nodo en el que se ha pinchado. (Figura 56)  [Si posa el rato sobre un nodo concreto]  2.2 Se abstrae la información del grafo y se focaliza la atención del grafo sobre el nodo sobre el que se ha posado el ratón y sus adyacentes. (Figura 57)  [Si pasa el ratón sobre las aristas del grafo]  2.4 Se muestra el texto de las aristas del grafo. (Figura 58) | |
| **Postcondiciones:** Se ha generado el grafo adecuado a la query | |
| **Interfaz gráfica:**    Figura : Botón para ver los resultados de forma grafo  **C:\Users\Mishel\git\web-rrhh\img\grafo.PNG**  Figura : Resultado query en forma de grafo  **C:\Users\Mishel\git\web-rrhh\img\grafoRecursoConcreto.png**  Figura : Información grafo concreto  **C:\Users\Mishel\git\web-rrhh\img\grafoFocalizado.png**  Figura : Se focaliza la información del grafo en un nodo concreto y sus adyacentes  **C:\Users\Mishel\git\web-rrhh\img\grafoPredicados.png**  Figura : Se muestra la información de una arista concreta | |

# ANEXO III: Manuales de usuario y/o de administrador

## Manual de usuario

En este manual se detallará como ejecutar el sistema creado, es decir, las herramientas y las instrucciones necesarias para hacerlo funcionar. Si se desea ampliar o alterar alguna de las funcionalidades ofrecidas consultar el manual de usuario extendido.

El sistema ha sido implementado usando Eclipse como entorno de desarrollo y Tomcat como contenedor web. Para su uso es necesario tener instalado GraphDB y adicionalmente se recomienda utilizar CounterClockWise como entorno de desarrollo para Clojure, se puede instalar desde Eclipse Market Place.

El código del proyecto además de los datasets utilizados en él, se encuentran alojados en [Github](https://github.com/mishel-uchuari/Modelo-Para-La-Generacion-De-Datos-Enlazados.git). Las directrices a seguir aquí especificadas parten de que se cuenta con el proyecto clonado y se tiene acceso a los datos que contiene y sus jars.

Las funcionalidades ofrecidas por el sistema pueden ser dividas en cinco bloques: generación de RDF, validación RDF, descubrimiento de enlaces, creación de servidor Linked Data y creación SPARQL Endpoint. A continuación, se explicará cómo ejecutar cada una de sus partes.

### Generación Datos Enlazados

Se han definido estructuras de creación de RDF para archivos en formato CSV publicados por Open Data Euskadi que estén dentro de los apartados de “calidad del aire”, “estaciones meteorológicas: lecturas recogidas”, “evolución de las tablas retributivas de los miembros del gobierno, altos cargos y personal eventual” y ”relación de puestos de trabajo”. Hay que tener en cuenta que en caso de querer utilizar otro CSV, este debe contar la misma estructura que los ficheros en formato CSV utilizados en este proyecto. Su estructura se puede ver en la sección “Cálculos, algoritmos” en el subapartado “Los datos” de la memoria del proyecto o en la carpeta [DatosIniciales](https://github.com/mishel-uchuari/Modelo-Para-La-Generacion-De-Datos-Enlazados/tree/feature-nuevas-funcionalidades/DatosIniciales) en el repositorio Github.

Para ejecutar funcionalidad:

1. Ejecutar jar "generacionRDF.jar" introduciendo tres parámetros: el nombre del pipeline que se quiere ejecutar, la ruta donde está localizado el CSV del que se tomarán los datos y por último la ruta donde queremos que se almacene el RDF generado. En la figura 59 se presenta un ejemplo de ejecución con las rutas en las que se encuentran dichos ficheros en el repositorio.

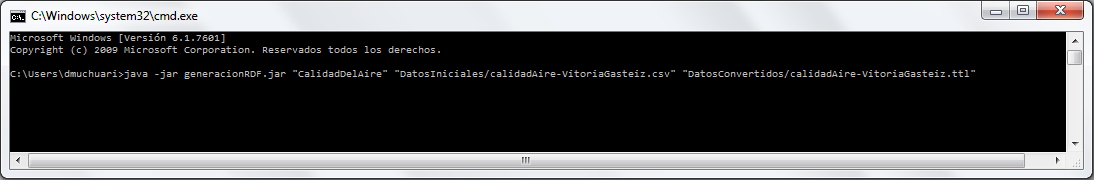


Figura : Ejecución jar "generaciónRDF.jar"

### Validación RDF

En caso de querer usar la funcionalidad implementada en este proyecto para la validación de RDF con otros distintos a los creados en el proyecto, se recomiendo leer la documentación del estándar [Shacl](https://www.w3.org/TR/shacl/) para construir el documento Turtle que contiene las reglas sobre las que se probará.

1. Ejecutar jar "validacionRDF.jar" introduciendo cuatro parámetros: el primer parámetro corresponde al RDF a evaluar, el segundo parámetro serán las reglas Shacl contra las que el RDF será evaluado, el tercero la ruta donde se define la query SPARQL para conocer el resultado de las pruebas, por último, la ruta donde se almacenará el report de las pruebas que contendrá los resultados de la prueba, si ha sido exitosa o en caso contrario el porqué de su fracaso. En la figura 60 se muestra un ejemplo de ejecución del jar con las rutas en las que se encuentran dichos ficheros en el repositorio.

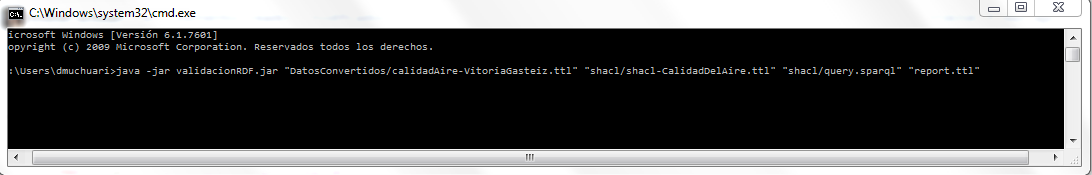


Figura : Ejecución jar "validacionRDF"

### Descubrimiento de enlaces

Para su uso:

1. Ejecutar jar "descubrimientoEnlaces.jar" introduciendo por parámetro el archivo XML que contiene la configuración de Silk, este se puede encontrar como “silk-configuration.xml” en el repositorio. En la figura 61 se presenta un ejemplo de ejecución.

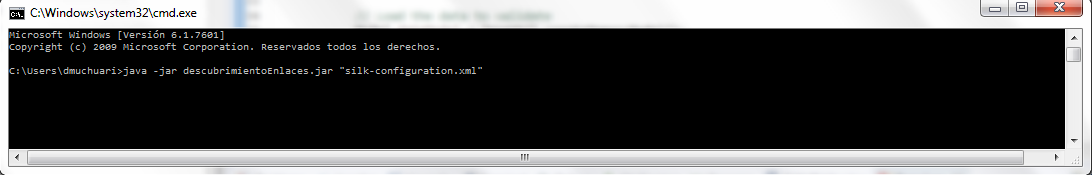


Figura : Ejecución jar "descubrimientoEnlaces.jar"

### Servidor Linked Data

1. Copiar la carpeta ROOT al directorio webapps de Tomcat.
2. Desplegar Tomcat
3. Solicitar recurso utilizando petición HTTP desde un navegador web o una herramienta con esa finalidad como INSOMNIA.

En la figura 62 se puede observar el resultado de la petición de un recurso desde un navegador web en la que se solicita HTML, en las figuras 63 y 64 el resultado de una petición HTTP con negociación de contenido utilizando INSOMNIA.

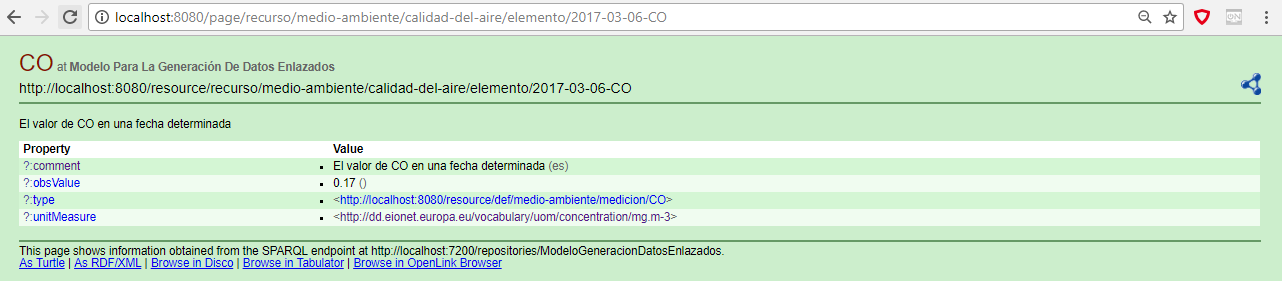


Figura : Petición servidor Linked Data a través de navegador web



Figura : Petición servidor Linked Data utilizando INSOMNIA solicitando la información en HTML

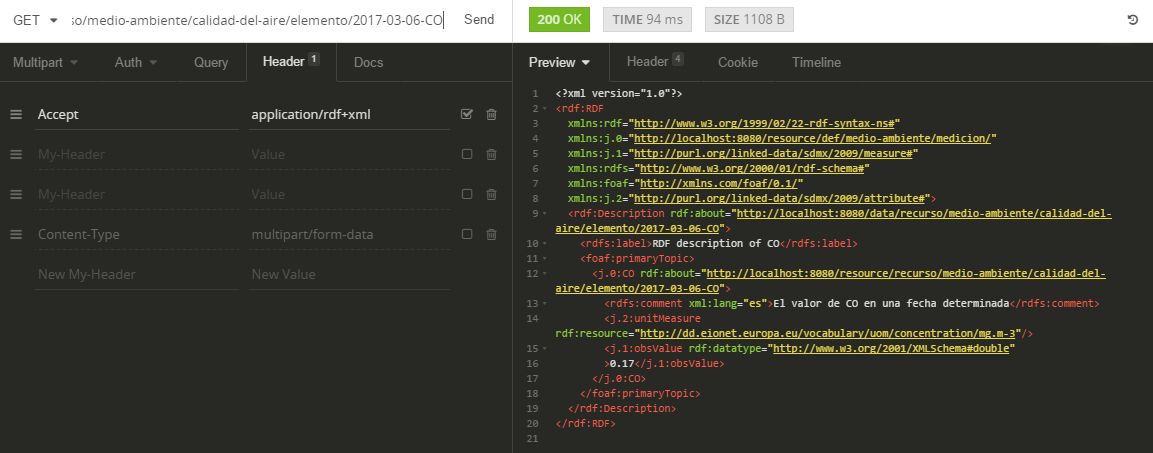


Figura : Petición servidor Linked Data utilizando INSOMNIA solicitando la información en RDF/XML

### SPARQL Endpoint

1. Cambiar el path donde está alojado GraphDB en la clase GraphDB contenida en el paquete “src/main/java/triplestore”.
2. Desplegar la aplicación en Tomcat
3. Insertar query en el campo de texto y pulsar RUN. En la figura 65 se muestra la localización del botón.

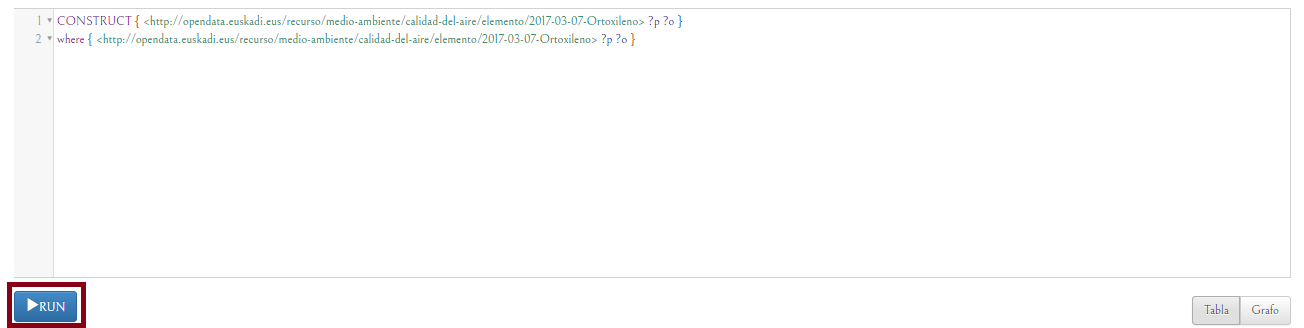


Figura : Localización botón RUN

1. Si se quieren ver los resultados en forma de grafo pulsar Grafo. En la figura 66 se muestra la localización del botón.

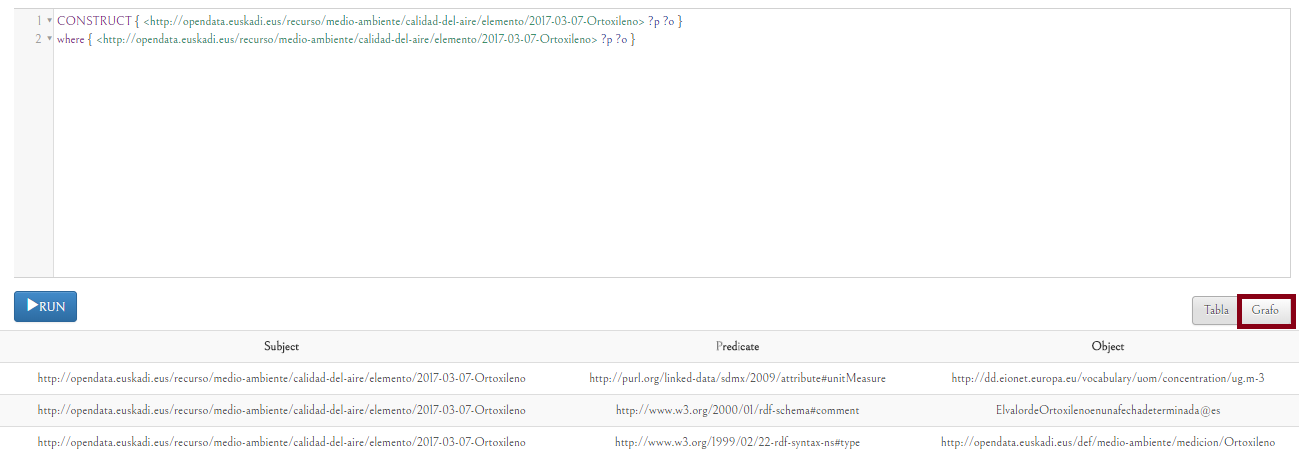


Figura : Localización botón para cambiar visualización a Grafo

1. Si se quiere ver el valor de la arista del grafo, es decir el valor del predicado de la tripleta, posar el ratón sobre la arista deseada, en la figura 67 se muestra el resultado de esta acción.

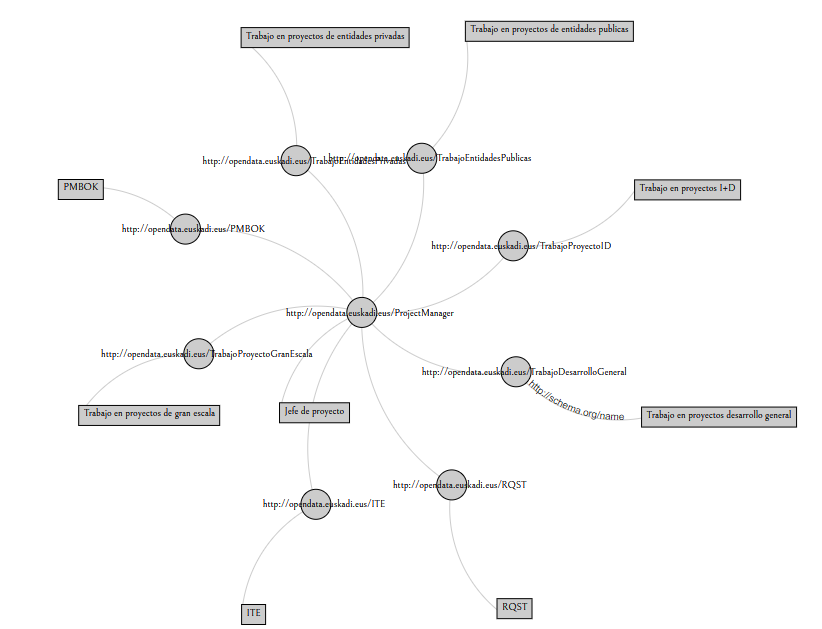
****

Figura : Arista con texto visible

1. Si se desea focalizar la atención del grafo sólo sobre un nodo, es decir un recurso, posar el ratón sobre él. En la figura 68 se pueden ver un resultado similar al que se obtendrá.

****

Figura : Focalización nodo sobre recurso concreto

1. Si se desea obtener toda la información disponible sobre un recurso determinado en la Triple Store pinchar dos veces sobre él.

## Manual administrador

En este manual se presentará el trabajo creado en profundidad, de manera que cualquier persona que así lo deseara pueda replicarlo, ampliarlo o alterarlo. Para la implementación del sistema se utilizó como entorno de desarrollo Eclipse añadiéndole el plugin CounterClockWise para añadir un entorno de desarrollo adicional para el lenguaje Clojure. Adicionalmente se utilizó Tomcat como contenedor web.

La totalidad del código generado así como los datasets utilizados se encuentran alojados en [Github](https://github.com/mishel-uchuari/Modelo-Para-La-Generacion-De-Datos-Enlazados.git). Todas las referencias a carpetas como archivos que se realizarán a lo largo de este documento se referirán a las que se pueden encontrar en el citado repositorio.

Las funcionalidades creadas se van a detallar separándolas en cuatro: la generación de RDF, la validación de RDF, el descubrimiento de enlaces, generación de servidor Linked Data y creación de SPARQL Endpoint.

### Generación de RDF

Para la creación de RDF se utilizó [Grafter](http://grafter.org/). Grafter es una librería de Clojure para la creación de RDF.

Para ejemplificar el proceso en cuanto a lo implementado en este proyecto se utilizará el dataset “DatosIniciales/calidadAire-VitoriaGasteiz.csv” referente a la calidad del aire en Vitoria Gasteiz. Para utilizar Grafter hay que tener nociones de RDF, por lo que a partir de ahora se continuará con la explicación de cómo usar Grafter partiendo de esa premisa.

Para la generación de RDF se utilizó un esquema simple:

* Análisis de datos de partida
* Elección de ontologías a utilizar o creación de ontologías propias
* Creación de RDF

Para llevarlo a cabo tras el estudio de los datos iniciales se construyeron las clases “src/main/clojure/transformaciones/ TransformacionesCalidadDelAire.clj” y “src/main/clojure/transformaciones/TransformacionGeneral.clj” que contendrían las funciones a aplicar sobre los datos del CSV, la primera cuenta con funciones específicamente creadas para aplicar a los datasets de calidad del aire y la segunda con operaciones de transformación comunes a todos los datasets. Todas están clases estarán implementadas en lenguaje Clojure.

La definición de los prefix y predicados a utilizar se realizó en “src/main/clojure/transformaciones/Predicados.clj” y la clase que llevara a cabo el proceso de conversión del CSV a RDF se encuentra en “src/main/clojure/pipelines/CalidadDelAire.clj”. Por último, la clase JAVA que pondrá en marcha todo el proceso se encuentra en “src/main/java/manager/PipelineManager.java”.

Si se desea replicar el proceso seguido para la construcción de otro pipeline de conversión a RDF se puede tomar como base el código contenido en dichas clases. Por otra parte, para la ejecución del proceso bastara con ir a la clase “src/main/java/manager/PipelineManager.java” desde Eclipse y ejecutarla pasándole como parámetros los siguientes: el nombre del pipeline a ejecutar, en este caso “CalidadDelAire”, en segundo lugar, el CSV inicial, en este caso “DatosIniciales/calidadAire-VitoriaGasteiz.csv” y por último la ruta donde se quiere que se almacene el RDF creado.

Si se quisiera ejecutar simplemente el jar creado para esta funcionalidad bastara con ejecutar el jar "validacionRDF.jar" pasándole los anteriores parámetros mencionados. El RDF creado se encontrará en la ruta que se le haya pasado por parámetro representado en Turtle.

### Validación RDF

Para la validación de RDF se usará el lenguaje [Shacl](https://www.w3.org/TR/shacl/) y la [API Shacl](https://github.com/TopQuadrant/shacl.git) que lo implementa. Se recomienda la lectura del estándar Shacl del W3C en caso de querer usar esta funcionalidad para evaluar archivos RDF distintos a los planteados en este proyecto para definir un nuevo archivo de pruebas.

En este caso se partirá del RDF generado en la sección de “Generación de RDF” de este mismo manual, éste se puede encontrar en “DatosConvertidos/calidadAire-VitoriaGasteiz.ttl”. Las pruebas Shacl contra las que se ejecutaran las pruebas se pueden encontrar en “shacl/shacl-calidaddelaire.ttl”.

Los resultados de las pruebas se almacenarán en un fichero Turtle, a partir de ahora llamado report. Contra ese report se ejecutará una query SPARQL con la que se determinará si el RDF es válido o no. Un ejemplo de la query utilizada a lo largo de este proyecto se encuentra en “shacl/query.sparql”.

La funcionalidad se puede ejecutar desplazándose en Eclipse a la clase “src/main/java/manager/ShaclManager.java” y pasándole tres parámetros: el archivo RDF a probar, las pruebas Shacl que se ejecutaran sobre él, la query SPARQL que se ejecutará sobre los resultados del report y la ruta donde se almacenará el report que contendrá los resultados de las pruebas.

Otra opción de ejecución es utilizar el jar creado para ese fin. Su nombre es “validacionRDF.jar” e introduciendo los parámetros ya comentados.

### Descubrimiento Enlaces

Para crear la funcionalidad de descubrimiento de enlaces se utilizó [Silk](http://silkframework.org/). Silk es una herramienta para el descubrimiento de nexos entre recursos publicados en distintas fuentes de datos. Para el uso de Silk se empleó como fuente de datos la Triple Store utilizada durante todo el proyecto donde se han ido almacenando los archivos RDF creados, GraphDb y como fuente de datos externos, DBpedia.

En caso de querer utilizar la implementación de Silk realizada en este proyecto para el descubrimiento de enlaces utilizando una configuración distinta a la establecida, se recomienda leer los manuales de uso de Silk. El archivo de configuración de Silk usado se puede encontrar en el repositorio Github con el nombre de “silk-configuration.xml” para tomarlo como referente.

Para ejecutar el software creado para el descubrimiento de enlaces basta con desplazarse desde Eclipse a la clase “src/main/java/manager/SilkManager.java” y pasarle como parámetro la localización del archivo de configuración. Si se desea ejecutarlo desde el jar creado con ese fin, este se puede encontrar bajo el nombre “descubrimientoEnlaces.jar”.

### Servidor Linked Data

Para la creación del Servidor Linked Data se utilizó [Pubby](http://wifo5-03.informatik.uni-mannheim.de/pubby/). Pubby permite construir una interfaz gráfica sobre la cual realizar peticiones con negociación de contenido tomando como fuente de datos una Triple Store. Utiliza un archivo de configuración donde se especifica la ruta de la Triple Store a usar y la forma que tendrán las URIs de los recursos a consumir.

La Triple Store hace referencia a la utilizada durante todo el proyecto, GraphDB. El archivo de configuración de Pubby utilizada en este proyecto se puede encontrar en “ROOT/WEB-INF/config.ttl”.

Se puede desplegar copiando la carpeta “ROOT” dentro del directorio webapps del Tomcat que se esté utilizando. Una vez desplegado se puede solicitar un recurso utilizando una petición HTTP desde un navegador web o una herramienta con esa finalidad como INSOMNIA.

En la figura 69 se puede observar el resultado de la petición de un recurso desde un navegador web en la que se solicita HTML, en la figura 70 el resultado de una petición HTTP con negociación de contenido utilizando INSOMNIA.

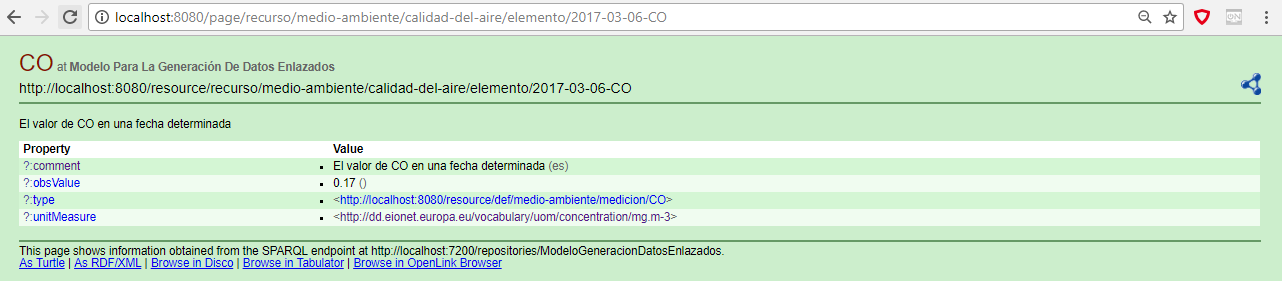


Figura : Petición servidor Linked Data desde navegador web

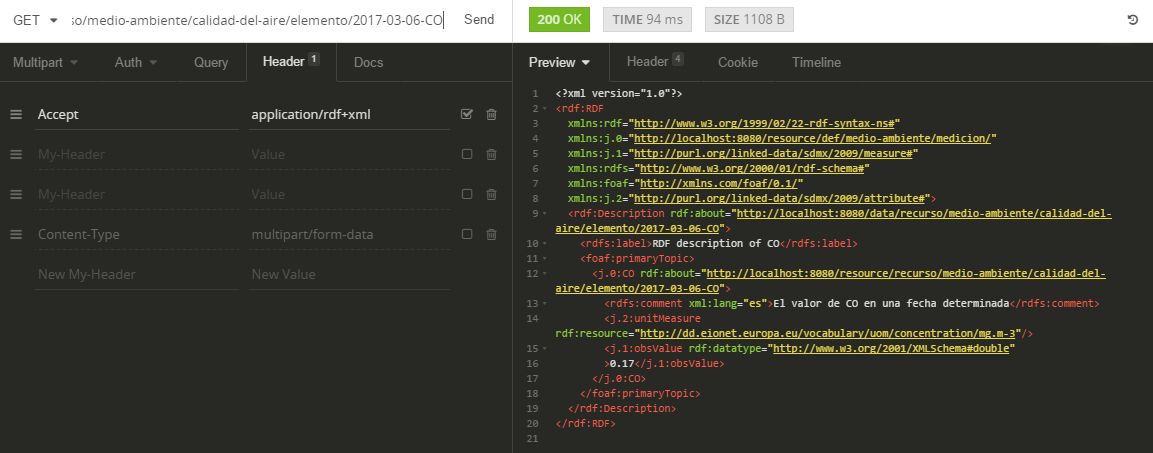


Figura : Petición servidor Linked Data utilizando INSOMNIA y solicitando información en RDF/XML

### SPARQL Endpoint

Para la creación del SPARQL Endpoint se utilizó la librería D3 y la Triple Store donde se ha estado almacenando toda la información generada. El SPARQL Endpoint será usado como una interfaz gráfica donde los usuarios pueden realizar consultas sobre la Triple Store y visualizar los resultados de distintas formas gráficas.

La clase donde se configura el path de la Triple Store es “src/main/java/triplestore/GraphDB.java”. Para su uso bastará con la configuración de GraphDB.

El proyecto podrá ser desplegado desde Eclipse o desplegando el war “sparqlEndpoint.war”. Una vez desplegado, para usarlo:

1. Insertar query en el campo de texto y pulsar RUN. En la figura 71 se muestra la localización del botón.

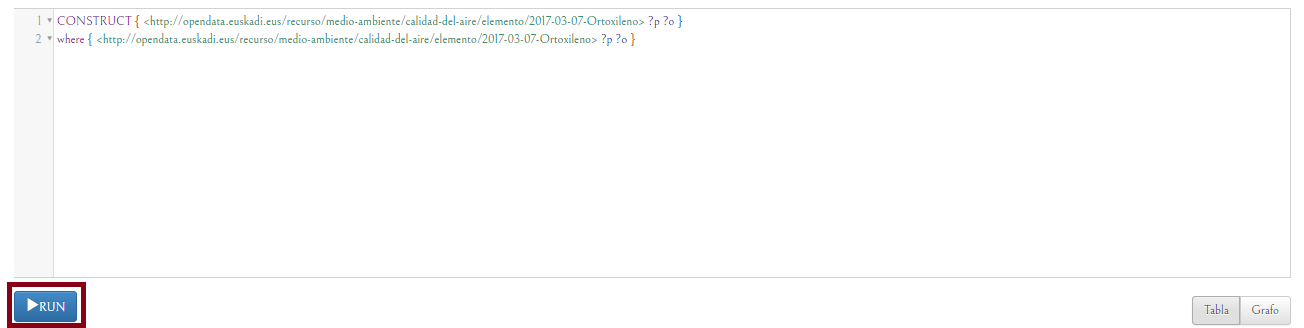


Figura : Localización botón RUN

1. Si se quieren ver los resultados en forma de grafo pulsar Grafo. En la figura 72 se muestra la localización del botón.

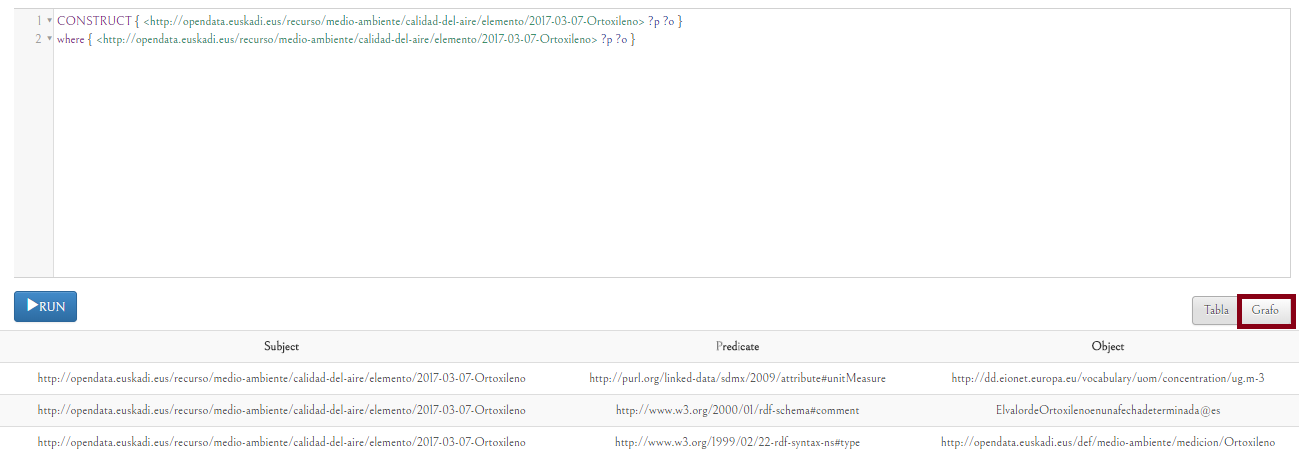


Figura : Localización botón para cambiar la visualización a grafo

1. Si se quiere ver el valor de la arista del grafo, es decir el valor del predicado de la tripleta posar el ratón sobre la arista deseada. En la figura 73 se muestra el resultado de esta acción.

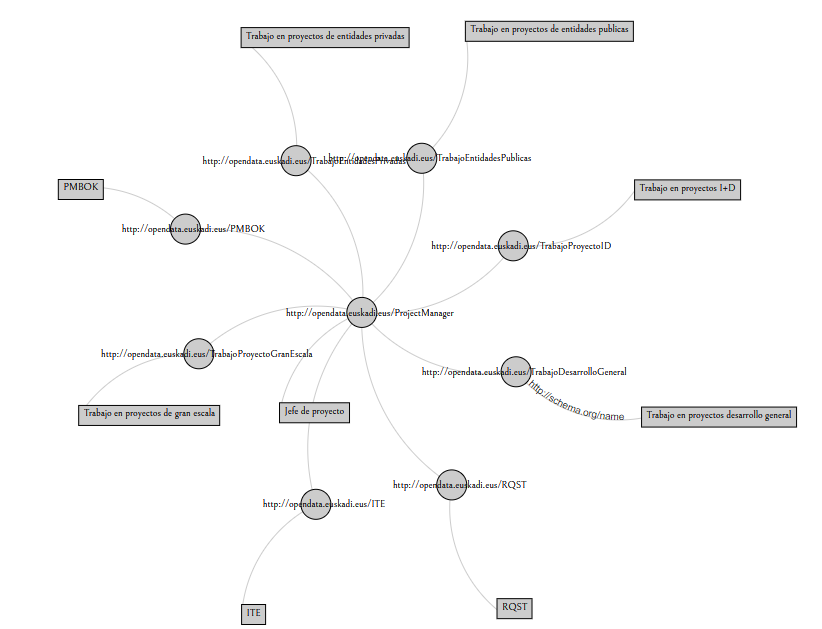
****

Figura : Texto arista visible

1. Si se desea focalizar la atención del grafo sólo sobre un nodo, es decir un recurso, posar el ratón sobre él. En la figura 74 se pueden ver un resultado similar al que se obtendrá.

****

Figura : Focalización de información del grafo sobre un recurso concreto

1. Si se desea obtener toda la información disponible sobre un recurso determinado en la Triple Store pinchar dos veces sobre él.