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

DOCUMENTO 4 - *MANUALES DE USUARIO Y DE ADMINISTRADOR*

Alumna: Uchuari Vera, Mishel

Director: Casquero Oyarzabal, Oskar

Curso: 2017-2018

Fecha: Bilbao, 23 de febrero del 2018

Contenido

Índice de figuras

I. 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, se encuentra alojado en Github (https://github.com/mishel-uchuari/Modelo-Para-La-Generacion-De-Datos-Enlazados.git), sin embargo, los archivo JAR a los que se hará referencia en este proyecto se encuentran en Drive (https://drive.google.com/open?id=14ylhH1pE1j7w3fPtk9SInqXTlssjPgoe). 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 divididas 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.

1. 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 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 1 se presenta un ejemplo de ejecución con las rutas en las que se encuentran dichos ficheros en el repositorio.

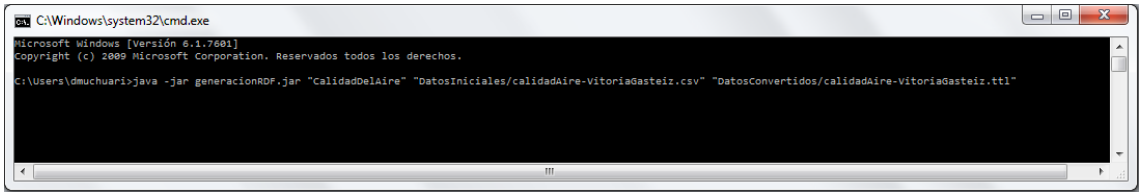

Figura 1: Ejecución JAR "generaciónRDF.jar"

2. 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.

2. 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 2 se muestra un ejemplo de ejecución del JAR con las rutas en las que se encuentran dichos ficheros en el repositorio.

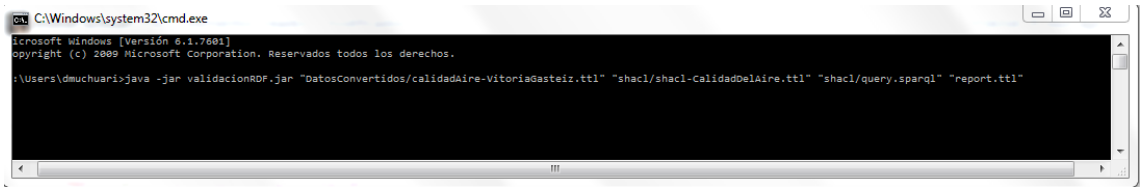

Figura 2: Ejecución JAR "validacionRDF"

3. 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 3 se presenta un ejemplo de ejecución.

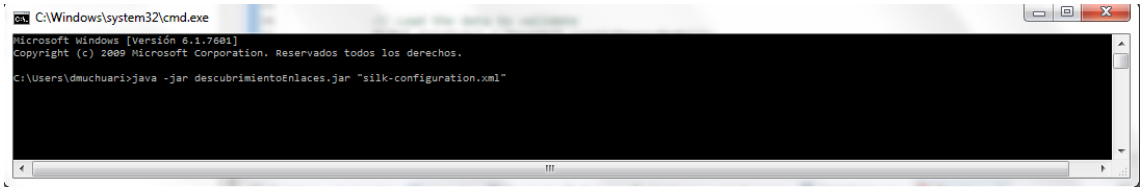

Figura 3: Ejecución JAR "descubrimientoEnlaces.jar"

4. 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 4 se puede observar el resultado de la petición de un recurso desde un navegador web en la que se solicita HTML, en las figuras 5 y 6 el resultado de una petición HTTP con negociación de contenido utilizando INSOMNIA.

Figura 4: Petición servidor Linked Data a través de navegador web

Figura 5: Petición servidor Linked Data utilizando INSOMNIA solicitando la información en HTML

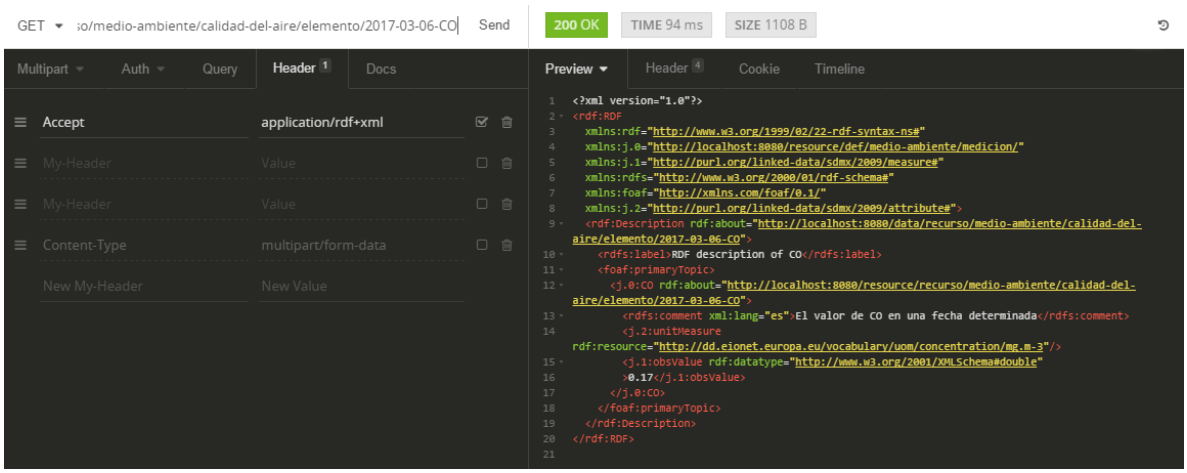

Figura 6: Petición servidor Linked Data utilizando INSOMNIA solicitando la información en RDF/XML

5. 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 7 se muestra la localización del botón.

Figura 7: Localización botón RUN

4. Si se quieren ver los resultados en forma de grafo pulsar Grafo. En la figura 8 se muestra la localización del botón.

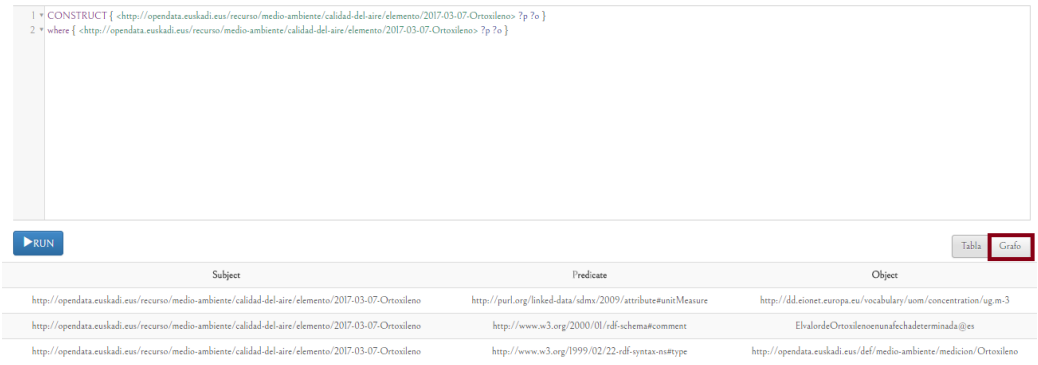

Figura 8: Localización botón para cambiar visualización a Grafo

5. 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 9 se muestra el resultado de esta acción.

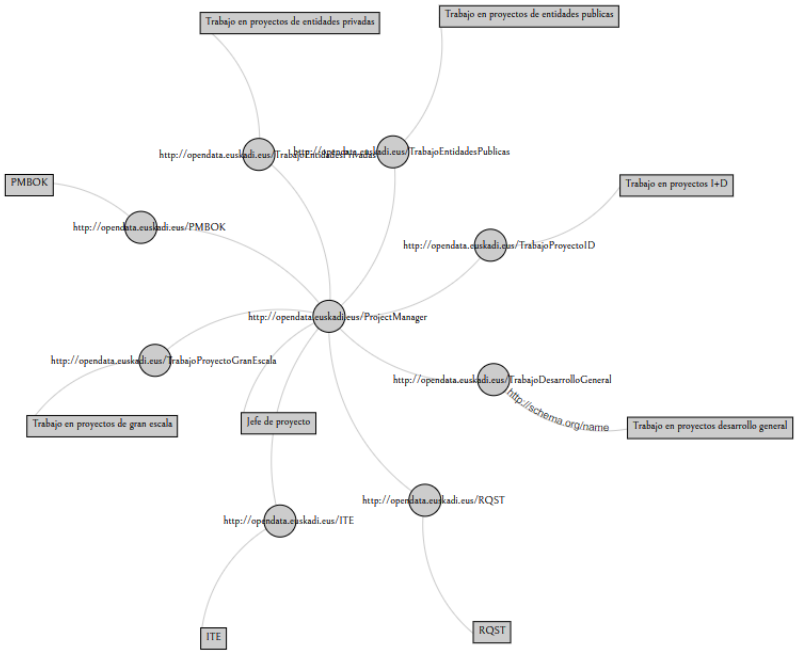

Figura 9: Arista con texto visible

6. Si se desea focalizar la atención del grafo sólo sobre un nodo, es decir un recurso, posar el ratón sobre él. En la figura 10 se pueden ver un resultado similar al que se obtendrá.

Figura 10: Focalización nodo sobre recurso concreto

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

II. 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), los JARS a los que se hace referencia se encuentran en Drive(https://drive.google.com/open?id=14ylhH1pE1j7w3fPtk9SInqXTlssjPgoe). 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.

1. 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:

- Análizar de datos de partida
- Elegir de ontologías a utilizar o creación de ontologías propias
- Crear 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.

2. 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.

3. 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”.

4. 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 11 se puede observar el resultado de la petición de un recurso desde un navegador web en la que se solicita HTML, en la figura 12 el resultado de una petición HTTP con negociación de contenido utilizando INSOMNIA.

Figura 11: Petición servidor Linked Data desde navegador web

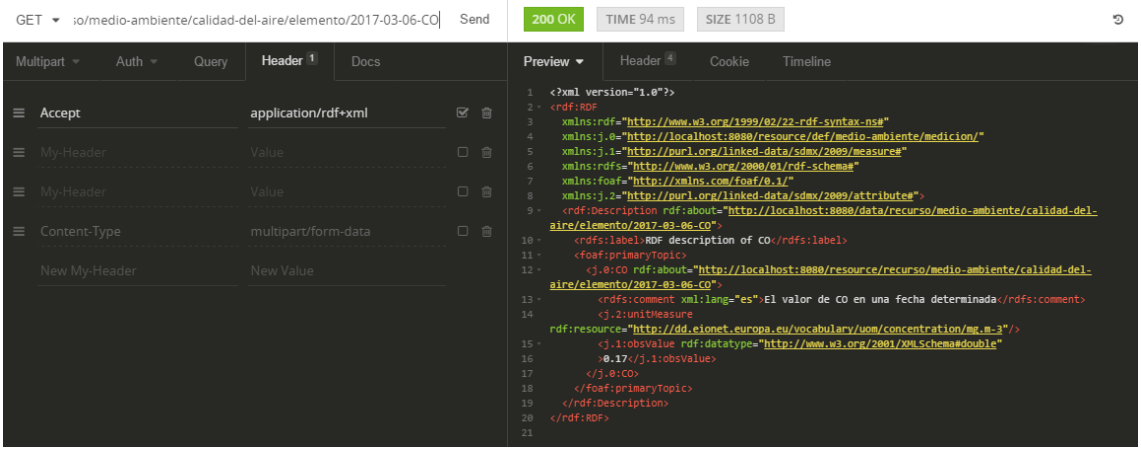

Figura 12: Petición servidor Linked Data utilizando INSOMNIA y solicitando información en RDF/XML

5. 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 13 se muestra la localización del botón.

Figura 13: Localización botón RUN

2. Si se quieren ver los resultados en forma de grafo pulsar Grafo. En la figura 14 se muestra la localización del botón.

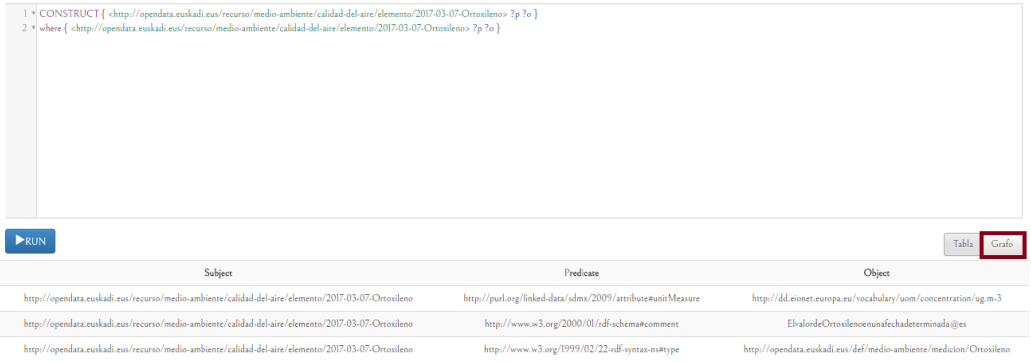

Figura 14: Localización botón para cambiar la visualización a grafo

3. 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 15 se muestra el resultado de esta acción.

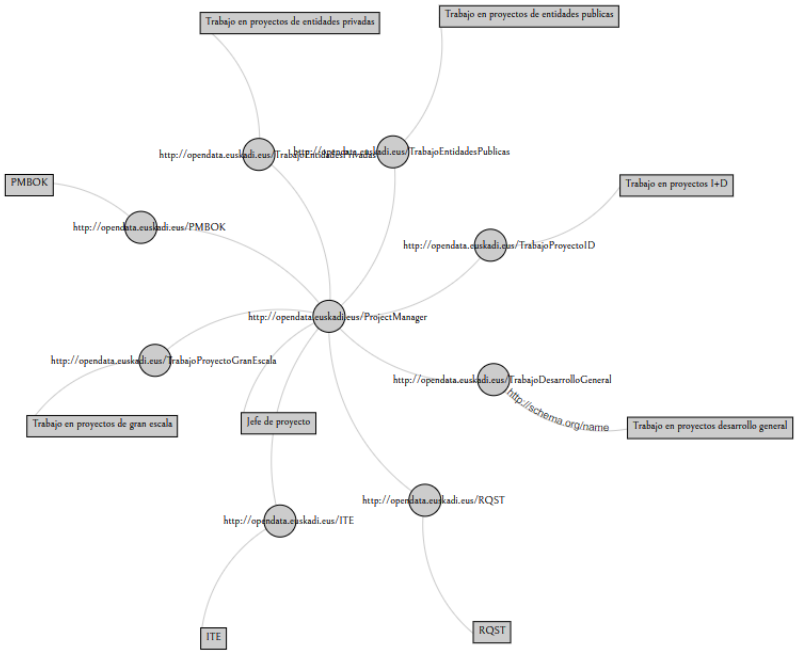

Figura 15: Texto arista visible

4. Si se desea focalizar la atención del grafo sólo sobre un nodo, es decir un recurso, posar el ratón sobre él. En la figura 16 se pueden ver un resultado similar al que se obtendrá.

Figura 16: Focalización de información del grafo sobre un recurso concreto

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