Obtención de datos CRTM

Autor: Ignacio Arias Barra

CONTENIDO

1. INTRODUCCIÓN

En la presente práctica de la asignatura de Obtención de datos, se han combinado técnicas de scrapping web [7], creación y tratamiento de ficheros .csv [6] y estructuración y representación de datos para el desarrollo de la misma.

En la dinámica seguida se ha comenzado por obtener datos concretos de diferentes páginas webs. Después, se ha llevado a cabo la creación de un fichero con extensión *.csv* [6] que contenía la combinación de todos los datos obtenidos. Por último, se han extraídos los datos de una forma predeterminada para finalmente conseguir una estructuración y representación de las relaciones existentes entre los mismos.

2. OBJETIVO

Esta práctica tiene como objetivo principal la presentación estructurada de los datos obtenidos de diferentes sitios web, para ayudar en la búsqueda de rutas accesibles entre en los transportes públicos de Madrid (centrados en Metro, Metro Ligero y Cercanías).

3. OBTENCIÓN DE LOS DATOS

El primer paso a realizar es la obtención de los datos que serán analizados posteriormente. Como se ha mencionado en el objetivo principal, se han tomado datos de diferentes sitios webs relacionados con el transporte público de la Comunidad de Madrid, en concreto de los sistemas de Metro, Metro Ligero y Cercanías.

3.1. Fuentes de datos

Las principales fuentes de obtención de datos que se tendrán en cuenta en el desarrollo de la práctica serán las siguientes:

- *Datos abiertos del Consorcio Regional de Transporte de Madrid (CRTM)* [1], con sitio web: http://datos.crtm.es/

- *Información de los medios de transporte en el CRTM* [3], con sitio web: http://www.crtm.es/tu-transporte-publico.aspx

3.2. Datos de interés

Antes de dar paso a la explicación de la extracción de los datos, explicaremos qué datos nos son de interés.

La primera fuente [1] hace referencia a conjuntos de open data, proporcionados por el *Consorcio Regional de Transporte de Madrid,* acerca de las redes de transporte de *Metro*, *Autobuses Urbanos del Municipio de Madrid, Autobuses Urbanos de la Comunidad de Madrid, Autobuses Interurbanos de la Comunidad de Madrid, Metro Ligero/Tranvía y Cercanías.* Los transportes tenidos en cuenta son Metro, Metro Ligero/Tranvía y Cercanías.

En la página de cada medio de transporte, se han descargado los archivos GTFS [4] de cada uno de ellos en formato zip. Al descomprimir estos zips, se pueden encontrar una gran cantidad de ficheros con diferente información acerca de los medios de transporte.

El archivo que es de interés es el *stops.txt* [2]. En este archivo se encuentra la siguiente información de las paradas de cada medio de transporte:

- *stop_id,* código singular para cada parada.
- *stop_code,* código relacionado con el *stop_id.*
- *stop_name,* nombre de la parada.
- *stop_desc,* dirección de la parada.
- *stop_lat,* latitud de la parada.
- *stop_lon,* longitud de la parada.
- *zone_id,* zona tarifaria de servicio [5].
- *stop_url,* url de la parada.
- *location_type,* tipo de parada.
- *parent_station,* estación a la hace referencia.
- *stop_timezone,* zona horaria.
- *wheelchair_boarding,* acceso para silla de ruedas.

Estos datos formarán parte de las columnas del *.csv* [6] que se formará con la unión de toda la información.

El resto de columnas, se han obtenido de la segunda fuente [3]. En este caso, los datos se extraen directamente de la web de cada transporte sin que haga falta la descarga de ningún fichero. La información que se extrae es la siguiente:

- *line_number,* línea a la que pertenece la parada.
- *order_number,* posición de la parada en el itinerario de la línea, unido con la línea a la que pertenece.
- *metro_linked,* líneas de metro con las que está conectada la parada.
- *ml_linked,* líneas de metro ligero con las que está conectada la parada.
- *cr_linked,* líneas de cercanías con las que está conectada la parada.
- *accessible,* si es accesible con silla de ruedas.
- *parking,* si tiene parking.

Además, añadiremos una columna al fichero llamada *transportmean_name,* que indicará si es una parada de tipo Metro, CR (Cercanías) o ML (Metro Ligero/Tranvía)

3.3. Extracción de los datos

Los datos han sido extraídos de la web. La técnica empleada ha sido el *web scrapping* [7]. Con esta técnica se consigue, mediante scripts programables, recorrer los elementos de una página web y recoger la información que es de interés. Para eso es necesario realizar un análisis previo de la estructura que presenta el código fuente de la página web. En ambos casos, dicho código es HTML.

En el caso de la descarga de los ficheros GFTS [4], el script primero ha tenido que ser programado para recorrer la página principal de los datos [1] y obtener el link de descarga del .zip de cada medio de transporte. Una vez obtenido, han sido descargados y realizado una combinación en un mismo fichero *.csv* [6] para su fácil manejo posterior. El fichero es llamado *stops.csv.*

En el caso de la obtención del resto de datos, para cada medio de transporte de interés, en la url principal [3] ha obtenido la url de las líneas de cada uno de ellos.

Después, para cada link, se ha ido introduciendo en ellos, encontrándose una página con todas las líneas de cada medio. El script ha recorrido todas las líneas, obteniendo el link de cada una, accediendo a él y obteniendo los datos de cada una de las paradas disponibles.

4. ESTRUCTURACIÓN DE LOS DATOS

En este apartado se detallará la estructuración de los datos obtenidos en cada paso hasta su final representación.

4.1. Fichero Transports*.csv*

Una vez obtenidos los datos mencionados en el apartado 3.2, realizamos una combinación de todos ellos en un fichero . *csv* [6] llamado *Transports.csv.*

Las cabeceras de cada columna, tendrán la etiqueta del dato asociado y mencionado con anterioridad en la misma sección 3.2.

Cada línea corresponderá a una estación, parada o acceso de un medio de transporte. A continuación, se detallará el valor que puede adoptar cada columna, explicando con una mayor claridad estos conceptos mencionados.

- *transportmean_name,* METRO, ML o CR en función del tipo de transporte.
- *stop_id,* código del tipo <3 letras> + <código para diferenciar paradas con mismo stop_code> + _ + <stop_code> :
- *3 letras,* “est” para el concepto de estación, “par” para el concepto de parada y “acc” para el concepto de acceso a una estación. Para una misma estación, puede

haber diferentes paradas que corresponden a diferentes líneas y además varios accesos a cada una de las paradas. Este fichero se encuentra incompleto en la web, ya que no se encuentran todas las estaciones como concepto “est” (ver caso de *Noviciados* y *Acacias* en el Metro) y la no existencia de la estación de cercanía de *Valdebebas*.

En el fichero transports.csv sólo aparecen los valores con concepto “par”, ya que ha sido la forma más eficaz de compararlo con los datos obtenidos del segundo scrapping [7]. Teniendo en cuenta las paradas, se evitaba el problema de la desaparición del concepto “est” para Noviciados y Acacias. Esto también tenía como objetivo conservar cada parada de cada estación que esté unida a otras líneas. Todo ello obliga a realizar al menos este previo filtrado de información. En el fichero *stops.csv,* sí aparecen los tres tipos de dato.

- *Código para diferenciar paradas con mismo stop_code,*
    - Metro:
        - General: 4
        - *stop_code* repetido: 90
    - Metro Ligero
        - General: 10
    - Cercanías:
        - General: 5
        - *stop_code* repetido: 90

  En el archivo *stops.csv,* para el concepto “acc” aparece un código intermedio entre el valor anterior y el *stop_code.* Este valor ha sido eliminado junto el concepto “acc” en el fichero *Transports.csv*.

- Stop_code, detallado a continuación.
- *stop_code,* un número entero de una a tres cifras, unívoco para cada parada o concepto “par”. Este ha sido otro motivo por el que se ha elegido el concepto “par” para la combinación de datos. Toda estación o concepto “est” que tenía un código de *stop_id* con un 90 (es decir, stop_code coincidente con otra parada diferente), tenía un concepto “par” con un *stop_code* univoco*.* De esta forma, con la extracción del stop_code del segundo scrapping [7], se ha podido realizar una combinación muy simple y muy eficaz de todos los datos de ambos scrappings [7].
- *stop_name,* nombre de la parada.
- *stop_desc,* dirección de la parada.
- *stop_lat,* valor de la atitud de la parada.
- *stop_lon,* valor de la ongitud de la parada.
- *zone_id,* zona tarifaria de servicio [5].
- *stop_url,* url de la parada. Este valor se encontraba con el enlace a esta página del consorcio, http://www.crtm.es, para todas las paradas. Por ello, se ha decidido modificar este campo con la verdadera url de cada parada, con vistas a usarlo como URI [8] de cada parada en la estructuración final de los datos.

- *location_type,* tipo de parada.
    - *Valor 0,* concepto “par”
    - *Valor 1,* concepto “est”
    - *Valor 2,* concepto “acc”

En el fichero *Transports.csv* sólo aparece el valor 0.

- *parent_station,* estación a la hace referencia. Este valor lo poseen los conceptos “acc y “par”, no así los “est”.
- *stop_timezone,* zona horaria.
- *wheelchair_boarding,* acceso para silla de ruedas.
- *line_number,* número de la línea a la que pertenece la parada.
- *order_number,* número de la posición de la parada en el itinerario de la línea, unido con la línea a la que pertenece.
- *metro_linked,* líneas de metro con las que está conectada la parada.
- *ml_linked,* líneas de metro ligero con las que está conectada la parada.
- *cr_linked,* líneas de cercanías con las que está conectada la parada.
- *accessible,* si es accesible con silla de ruedas (Yes/No).
- *parking,* si tiene parking (Yes/No).

4.2. Diagrama UML de clases

Una vez realizada la convergencia de los datos en el fichero *Transports.csv,* hay que establecer relaciones entre los datos de las diferentes columnas para cada una de las filas. Una vez decididas las relaciones, se generará una estructuración RDF [10] de dichos datos dando lugar a un fichero RDF/XML. Posteriormente, dicho fichero generado podrá ser interpretado por un navegador.

Para representar las relaciones que se han establecido entre los diferentes datos, se ha creado un diagrama UML [9], presente en la figura 1.

Dicho diagrama ha sido creado en un creador online [11].

En dicho esquema se han de definir:

- Clases o conceptos: concepto del dato. En diagrama: “localización”, “coordenadas”, “infraestructura”, “correspondencia”, “información general.
- Propiedades: de un individuo. Vienen de ontologías de la web semántica [13] o las podemos crear, una vez creada una ontología particular para nuestro caso de estudio. En el diagrama, todos los enunciados que se encuentran en las flechas de unión.
- Individuos: todos los rectángulos del esquema, además de la estación.

En la figura 1 se observan elipses y rectángulos. Esto es porque los primeros estarán representados por una URI para cada estación y los segundos son datos literales. Esto se explicará en las siguientes secciones.

Por último, se ha de mencionar las cifras observadas al principio y final de cada relación. Esto indica el número de elementos que componen o pueden componer dicha relación:

- 1: uno y sólo uno
- 0..*: cero o varios
- 5: cinco y sólo cinco

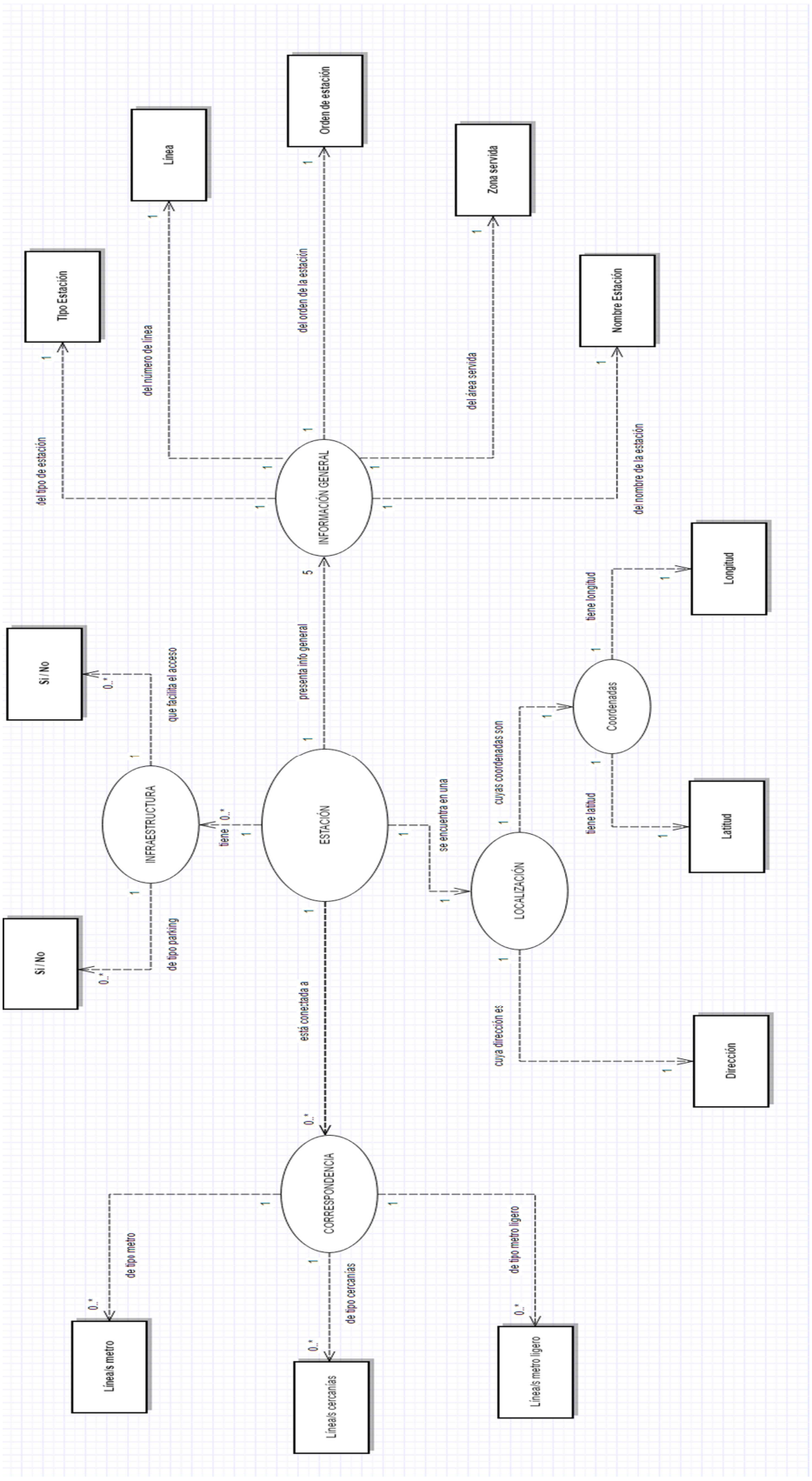

Figura 1. Diagrama UML [9]

4.3. Fichero RDF/XML

El fichero RDF/XML es un fichero en formato XML, llamado *estaciones*, que muestra las relaciones de datos establecidas para cada estación de forma estructurada. Dicho fichero tiene la particularidad de que en la cabecera vienen definidas las propiedades de las ontologías [13] usadas (ya sean predefinidas o creadas).

El comiendo del fichero es el siguiente (línea de un xml):

```
<?xml version="1.0" encoding="UTF-8"?>
```

En la cabecera del rdf, se definen las ontologías como namespaces [14], para poder definir propiedades:

```
<rdf:RDF
   xmlns:ns1="http://xmlns.com/foaf/0.1/"
   xmlns:ns2="http://www.w3.org/2003/01/geo/wgs84_pos/"
   xmlns:ns3="http://dublincore.org/"
   xmlns:ns4="http://schema.org/"
   xmlns:ns5="http://mytransportONT.org/"
   xmlns:ns6="http://ontologi.es/rail/vocab/"
   xmlns:ns7="http://www.daml.org/2003/05/subway/subway-ont/"
   xmlns:ns8="http://e-tourism.deri.at/ont/e-tourism.owl/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
```

El resto del fichero se compone de relaciones ordenadas de los datos, de la siguiente manera:

```
<rdf:Description    rdf:about="http://www.crtm.es/tu-transporte-
publico/metro/estaciones/4_205.aspx">
    <ns5:has_generalInfo
rdf:resource="http://mytransportONT.org/general_info/205ginfo"/>
    <ns5:haslocation
rdf:resource="http://mytransportONT.org/location/205loc"/>
    <ns5:linked_to
rdf:resource="http://mytransportONT.org/linked_to/205link"/>
    <ns5:hasinfrastructureInfo
rdf:resource="http://mytransportONT.org/infrastructure/205infra"
/>
  </rdf:Description>
```

Para cada tipo de relación entre se establecen estos nodos donde se pueden observar las propiedades provenientes de cada ontología, referenciado con la abreviatura del namespace [14].

Este fichero ha sido creado a partir de un script Python, con la ayuda de la librería *rdflib* [15]. En la parte de explicación del código se entrará en más detalle.

Una vez generado puede ser interpretado por un navegador como si de un XML normal se tratase.

Pero el siguiente paso en el proceso es usar un validador [12] de RDF/XML para crear el grafo con las relaciones de cada estación.

Antes de eso, cabe mencionar que este validador [12] empleado también genera las tripletas de las relaciones. Una tripleta consiste en un sujeto, un predicado y un objeto. Los sujetos y predicados son definidos con URIs [8] mientras que los objetos pueden ser literales.

Un ejemplo de tripleta podría ser el siguiente, extraído directamente del validador [12].

Subject	Predicate	Object
http://www.crtm.es/tu-transporte-publico/metro/estaciones/4_205.aspx	http://mytransportONT.org/has_generalInfo	http://mytransportONT.org/general_info/205ginfo

4.4. Representación gráfica

La siguiente figura muestra la escenificación del diagrama UML una vez aplicado el validador [12] a partir del fichero RDF/XML.

Como se ha mencionado con anterioridad, las elipses están representadas por URIs mientras que los rectángulos son literales de los datos obtenidos previamente.

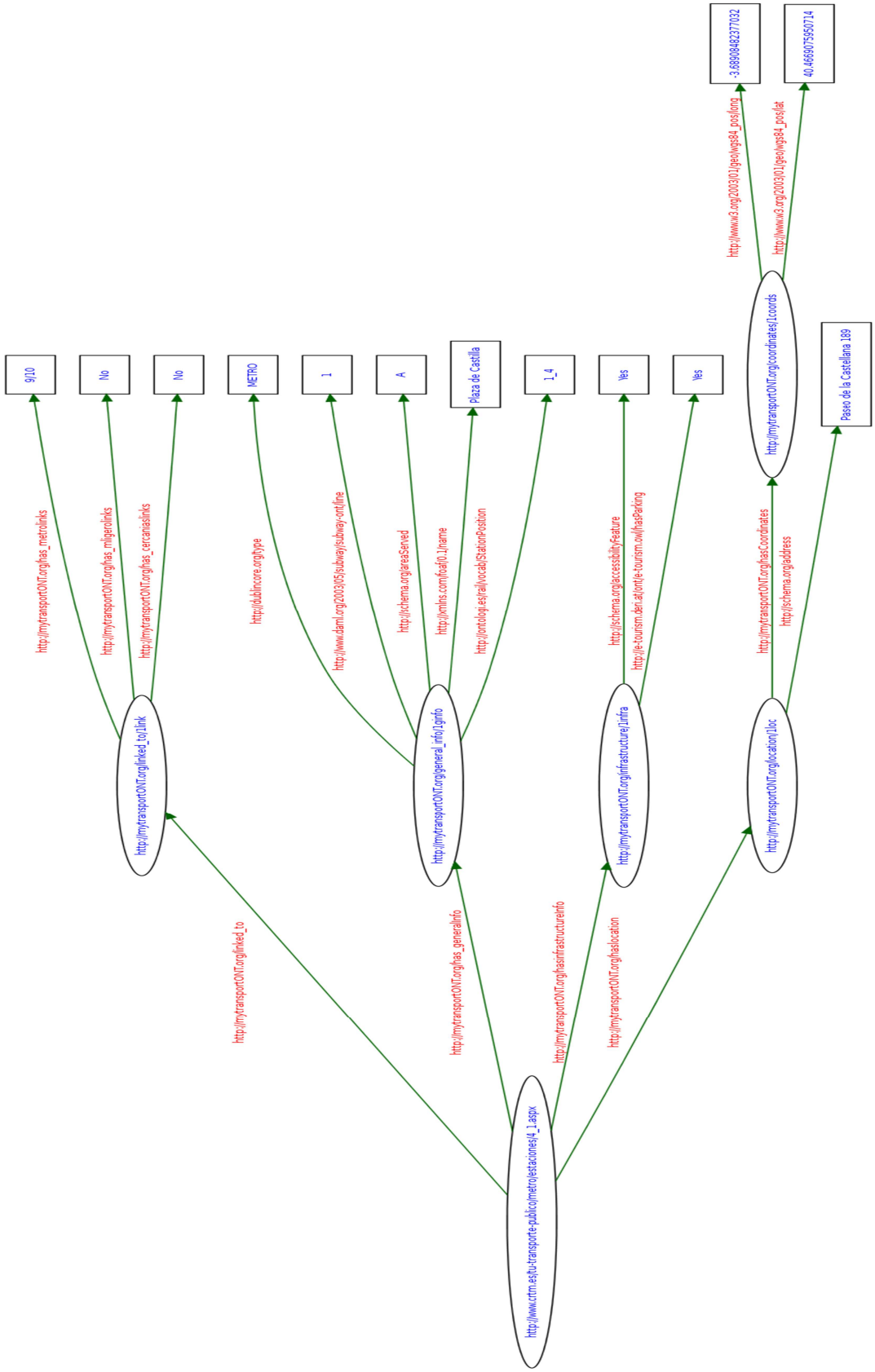

Figura 2. Grafo generado.

5. ONTOLOGÍAS ANALIZADAS

En el presente apartado se indicará en qué consisten las principales ontologías analizadas. Cabe decir que no todas las analizadas han sido utilizadas para nuestro propósito.

Para buscar posibles ontologías válidas para la práctica, se ha utilizado un buscador de ontologías [16], así como el buscador de Google.

Ontología	**Namespace**	**URL Homepage**	**Descripción principal**
Socially Interconnected Online Communities (SIOC)	*http://rdfs.org/sioc/ns#*	http://sioc-project.org/	Relación de datos acerca de personas en comunidades WEB y como están conectadas.
The Dublin Core (DC)	http://purl.org/dc/terms/	http://dublincore.org/	Contiene propiedades y vocabulario generales sobre metadata.
The Friend Of A Friend	http://xmlns.com/foaf/0.1/	http://www.foaf-project.org/	Relación de datos acerca de personas y como están conectadas.
DARPA Agent Markup Language	http://www.daml.org/2003/05/subway/subway-ont/	http://www.daml.org/	Relación de datos de líneas de metro y sus estaciones.
Good Relations	http://purl.org/goodrelations/v1	http://purl.org/goodrelations/	Relación de datos sobre e-commerce.
Schema.org	http://schema.org/	http://schema.org/	Es una comunidad que proporciona una gran cantidad de propiedades de diversos temas, listo para usar en la web semántica.
E-Tourism Working Group	http://e-tourism.deri.at/ont/e-tourism.owl	http://e-tourism.deri.at/	Relación de datos con la intención de revolucionar el eTourism a través de la web semántica.
W3C	*http://www.w3.org/2003/01/geo/wgs84_pos/*	https://www.w3.org/2003/01/geo/	Relación de datos de localización espacial de las cosas.
ontologies	*http://ontologi.es/rail/vocab*	http://ontologi.es/rail/	Relación de datos para redes de metro de UK

6. ONTOLOGÍAS CONSIDERADAS

A continuación, se detallarán las ontologías escogidas y las propiedades utilizadas. Pero antes, se explicará la ontología que se ha creado y sus propiedades.

http://mytransportONT.org/

Esta ontología ha sido creada debido a la necesidad de completar las características proporcionadas en las ontologías escogidas y con la finalidad de dotar de una estructura mejor organizada de los datos.

Propiedades y descripción:

Las tres primeras características hacen referencia a los enlaces de metro, metro ligero y cercanías que puede tener cada parada de cada medio de transporte.

1. has_metrolinks
2. has_mligerolinks
3. has_cercaniaslinks
4. hasCoordinates

   Esta característica nos sirve para agrupar en un elemento las coordenadas latitud y longitud de una parada.

5. linked_to

   Esta propiedad junta en un elemento las características de los enlaces de cada parada, mencionados con anterioridad.

6. has_generalinfo

   Proporciona un nodo padre que recogerá la información general de una estación como el nombre, la línea a la que pertenece, el tipo de medio de transporte de la parada, el orden en la línea según el itinerario y la zona tarifaria.

7. has_infrastructureinfo

   Relaciona la estación con la información sobre la infraestructura de la estación, si es accesible o si tiene parking.

8. has_location

   Ha sido necesaria para unir las propiedades de dirección y coordenadas de una estación.

http://dublincore.org

Propiedades y descripción:

1. type

Determina el tipo de transporte público al que pertenece la parada.

http://www.daml.org/2003/05/subway/subway-ont/

Propiedades y descripción:

1. line

   Indica la línea de transporte a la que pertenece la parada.

http://schema.org/

Propiedades y descripción:

1. address

   Indica la dirección (calle y número de portal) de la parada.

2. areaServed

   Indica la zona tarifaria donde ofrece su servicio la parada.

3. accessibilityFeature

   Indica la propiedad de si una estación es accesible o no para sillas de ruedas.

http://e-tourism.deri.at/ont/e-tourism.owl

Propiedades y descripción:

1. hasParking

   Indica si tiene parking la parada o no.

http://www.w3.org/2003/01/geo/wgs84_pos/

Propiedades y descripción:

-lat

-long

http://xmlns.com/foaf/0.1/

Propiedades y descripción:

1. name

   Indica el nombre de la estación

http://ontologi.es/rail/vocab

Propiedades y descripción:

1. StationPosition

   Indica la posición de la estación según el itinerario.

7. CÓDIGO

En esta sección se describirá el comportamiento de cada código generado para la presente práctica.

7.1. SCRAPPER 1, *GFTSscrapper.py*

El primer programa que se ha generado, tiene como objetivo la descarga de los ficheros zip GFTS [4], extracción de los ficheros stops.txt de cada medio de transporte y la combinación de su información en un *.csv* [6], llamado *stops.csv*, que facilitará su uso y análisis posterior. El lenguaje de este código es Python [17].

Primero, con ayuda de la librería Python *requests* [18], se descarga el HTML de la página principal del consorcio [1]. Gracias a la librería *bs4* [19], se genera un objeto para el HTML que proporcionará una forma más fácil de acceder a los datos. Analizando el principal HTML, se ha visto que en este no se encuentran los enlaces de cada medio de transporte deseado. Este primer HTML corresponde a una máscara que hay en la página por lo que, haciendo scrapping [7], se extrae la verdadera página donde se encuentran los medios de transporte. Como para la página principal, se descarga el HTML de este nuevo enlace y se obtienen los enlaces de cada transporte.

El siguiente paso corresponde a la descarga de los ficheros zip GFTS. Dichos enlaces no se encuentran en el HTML ya que se ejecutan scripts java que los llaman cada vez que se pincha en con el ratón en el botón de descarga de la página. Debido a esto, se analizó el tráfico generado al pinchar en el botón de descarga y se pudo obtener el enlace al que hay que realizar la petición HTTP para conseguir los ficheros. Esto se realizó con el programa Wireshark [20].

A partir de aquí y como en los pasos anteriores, se descargan los ficheros realizando peticiones GET con la librería requests [18]. Dichos ficheros se descargan en zips, por lo que con ayuda de la librería zip de Python [17], se procede a descomprimirlos en una carpeta para cada transporte.

Por último, se combina la información de los ficheros *stops.txt* [2] en un *.csv* [6] llamado *stops.csv.*

7.2. SCRAPPER 2, *consorcioScrapper.py*

El segundo programa que se ha generado, tiene como objetivo conseguir los datos faltantes en los ficheros *stops.txt* [2] de cada transporte y la combinación de estos con los del ya generado *stops.csv* en el fichero *Transports.csv.* El lenguaje de este código es Python [17].

Antes de comenzar la explicación, se ha de mencionar que se usará la librería *pandas* [21], ya que el uso de la clase DataFrame [21] facilitará el manejo de la información en tablas y su posterior combinación con el fichero *stops.csv y* volcado al fichero *Transports.csv.*

El primer paso es la descarga del HTML de la página principal [3] así como la creación del objeto bs4 para el fácil manejo y acceso de la información del mismo. En éste se accederá a los enlaces de cada transporte deseado. A recordar, *Metro, Metro Ligero/Tranvía, Cercanías.*

Una vez obtenidos, se realizan de nuevo peticiones GET para acceder a dichas páginas. En la página de cada medio, de nuevo haciendo scrapping [7] se extraerá el enlace correspondiente a todas las líneas.

Ahora, para cada transporte se accederá a cada enlace de cada línea obtenida en el paso anterior. En cada línea se accederá a la información de cada parada, presentada en orden según el itinerario y como en casos anteriores, con el uso del scrapping [7] del HTML. La información que complementa la adquirida del *stops.csv* y que se obtiene en este scrapping [7], por parada, es la siguiente:

- Nombre de la parada.
- Orden.
- Correspondencias con otras líneas de metros.
- Correspondencias con otras líneas de metro ligero.
- Correspondencias con otras líneas de cercanías.
- Si es accesible o no.
- Si tiene parking.

Tanto la información de la línea a la que pertenece la parada como el tipo de transporte al que pertenece la parada, serán añadidos aprovechando la lógica del programa.

Como se ha mencionado con anterioridad en esta misma sección, se usará un DataFrame para el almacenamiento de los datos. El proceso pasa por crear una lista Python con todos los valores obtenidos por parada y añadir dicha lista al DataFrame de todos los datos de manera recursiva. Este DataFrame es llamado *Infotoadd*.

Los siguientes pasos consisten en realizar ajustes en la información obtenida, como el reemplazo de la línea llamada como 13 de Metro, por la letra R de Ramal, que es su verdadera identidad o el relleno de blancos en los datos de las correspondencias con otras líneas de otros transportes con el valor “No”.

Acto seguido accedemos, con ayuda de la librería pandas [21], al fichero previamente creado en el primer scrapper, *stops.csv.*

Corregimos algunos errores en algunos nombres de columnas de este fichero debido a la codificación. Dichos cambios son:

- Renombrar columna *ufeffstop_id* con *stop_id*
- Renombrar columna *wheelchair_boarding**n* con *wheelchair_boarding*

La información del fichero ha sido guardada en un DataFrame llamado *StopsDF.* Será dividido en tres diferentes, cada uno correspondiente a la información de un medio de transporte. Esto facilitará el manejo y análisis de la información de cada uno de ellos.

Se realiza el mismo paso de dividir en tres DataFrames diferentes el creado con anterioridad, Infotoadd. Para el caso de cercanías, existen líneas duplicadas en el DataFrame por lo que son eliminadas.

Volviendo a los DataFrames generado a partir del *stops.csv* (stopsDF_metro, stopsDF_ml, stopsDF_cr), filtramos por la columna *location_type* con valor igual a 0, correspondientes al concepto "par" explicado con anterioridad.

Se cambia el tipo de los datos de la columna *location_type* del DataFrame del Metro Ligero/Tranvía a string, ya que surgían problemas en el análisis de su información.

El objetivo es realizar una combinación de la información de los DataFrame, con la columna *stop_code,* como eje pivotante sobre el que realizar dicha combinación. Para ello, se eliminan columnas repetidas entre los DataFrames que provienen de diferentes DataFrame padres (stopsDF e Infotoadd)

Antes de llevar a cabo dicha combinación, se ha de añadir la parada de Valdebebas al DataFrame de cercanías que proviene del stopsDF.

Por último, combinamos los DataFrame. Primero en función de cada medio de transporte con la función merge de *pandas* [21] y posteriormente todos con la función *concat.*

Para la exportación de los datos a un fichero *.csv* [6], se establece el orden de columnas que se indica en el enunciado de la práctica. Acto seguido y con la función *to_csv* de *pandas* [21], exportamos la información al fichero *Transports.csv.*

7.3. Código creación fichero RDF/XML, *rdf_xml.py*

El tercer programa que se ha generado, tiene como objetivo la lectura del fichero *Transports.csv* y con ayuda de la librería RDFlib [15], se establecerán las relaciones entre los datos de cada parada y se generará un fichero en formato RDF/XML. El lenguaje de este código es Python [17].

El primer paso, es la lectura del fichero *.csv* [6] en un DataFrame.

Acto seguido, generamos el objeto *grafo,* proveniente de la librería RDFlib [15]. Este objeto será usado para añadirle tripletas que establezcamos con las relaciones entre los datos.

Pero antes de realizarse la introducción de las tripletas en el *grafo* y que se puedan generar los nodos del fichero XML*,* se crea la ontología http://mytransportONT.org/ con la función *Namespace.* También se generan las propiedades anteriormente mencionadas con la función *URIRef.* Ésta última generara URIs para estos recursos.

Además de las propiedades provenientes de la ontología creada, se generan también las que vienen de las ontologías ya analizadas en secciones anteriores.

A partir de aquí, se entra en un bucle que, para cada línea del fichero *Transports.csv,* genera recursos provenientes de la ontología creada, que necesitan ser únicos por cada parada. Con la función *add,* se añaden las tripletas con la forma:

Grafo.add((sujeto, propiedad, objeto))

Una vez establecidas las relaciones para cada parada, se exporta la información a un fichero XML, con la siguiente instrucción:

grafo.serialize(destination="estaciones.xml", format="xml")

Como se aprecia, el fichero creado es el adjunto en el fichero zip, *estaciones.xml.*

8. CONCLUSIÓN

Conforme a las especificaciones del enunciado de la práctica, se ha llevado a cabo un scrapping [7] de dos páginas web [1] [3] del Consorcio de transportes de la Comunidad de Madrid y se han combinado los datos requeridos en un fichero en formato *.csv* [6]. Una vez comprimidos todos los datos en una sola estructura, se han establecido relaciones entre los datos de cada estación, y se han representado tanto gráficamente como en un fichero XML, de forma estructurada, teniendo como base las premisas de la web semántica y empleando las ontologías descritas.

La finalidad de esta estructuración es ayudar a la búsqueda de rutas accesibles a través de los medios de transporte de Madrid.

9. REFERENCIAS

[1] “Datos abiertos del Consorcio Regional de Transporte de Madrid (CRTM)” url: http://datos.crtm.es/

[2] “Google Transit Feed Specification (GTFS)” url: https://developers.google.com/transit/gtfs/reference?hl=es-419

[3] “Información de los medios de transporte en el CRTM.” url: http://www.crtm.es/tu-transporte-publico.aspx

[4] “Qué es la GFTS ” url: https://developers.google.com/transit/gtfs/?hl=es-419

[5] “Zonas tarifarias del transporte público de Madrid.” url: http://www.crtm.es/billetes-y-tarifas/zonas-tarifarias.aspx

[6] “Qué es un CSV - Wikipedia”url: https://es.wikipedia.org/wiki/Valores_separados_por_comas

[7] “Qué es el web Scrapping, Wikipedia.” url: https://es.wikipedia.org/wiki/Web_scraping

[8] “Qué es una URI.” url: https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme

[9] “Qué es UML – Unified Modelig Language” url: http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=688:ique-es-y-para-que-sirve-uml-versiones-de-uml-lenguaje-unificado-de-modelado-tipos-de-diagramas-uml&catid=46:lenguajes-y-entornos&Itemid=163

[10] “Qué es RDF.” url: http://www.seofreelance.es/que-es-rdf-introduccion-a-rdf/

[11] “Creación de diagramas UML.” url: https://www.gliffy.com/

[12] “Validador RDF online.” url:https://www.w3.org/RDF/Validator/

[13] “Ontología web semántica.” url: https://www.w3.org/2007/09/OWL-Overview-es.html

[14] “Namespaces of a XML.” url: http://www.w3schools.com/xml/xml_namespaces.asp

[15] “RDFlib.” url: https://pypi.python.org/pypi/rdflib

[16] ”Buscador de ontologías.” url:http://swoogle.umbc.edu/

[17] “Python.“ url:http://www.es.python.org/

[18] “Requests Python library.” url: http://docs.python-requests.org/en/master/

[19] “Beautiful Soup Python library” url: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[20] “Wireshark.” url: https://www.wireshark.org/

[21] “pandas Python library.” url: http://pandas.pydata.org/