ONTOLOGIA DE TWITTER:
El primer pas per inferir informació dels tweets

Marta Cantero Rebollo
Enginyeria Tècnica en Informàtica de Gestió

Consultor: Joan Anton Pérez Braña

FITXA DEL TREBALL FINAL

Títol del treball:	ONTOLOGIA DE TWITTER: El primer pas per inferir informació dels tweets
Nom de l'autor:	Marta Cantero Rebollo
Nom del consultor:	Joan Anton Pérez Braña
Data de lliurament:	06/2014
Àrea del Treball Final:	XML i web semàntica
Titulació:	Enginyeria Tècnica en Informàtica de Gestió

Resum del Treball:

Amb la popularització de les xarxes socials en els darrers anys es presenta un nou camp d'investigació i recerca. Les dades que circulen per les xarxes socials defineixen nous estils relacionals, allunyats de les clàssiques estructures jeràrquiques i és per això que el seu estudi desperta gran interès. En aquest treball es fa una primera aproximació a l'ús d'una ontologia per inferir informació sobre els tweets. Amb l'ontologia s'intenta conceptualitzar la informació que s'obté de Twitter i crear una vista simplificada de les dades per tal de representar-les fàcilment.

L'ontologia que s'ha dissenyat contempla els conceptes bàsics de Twitter, les relacions entre ells i les restriccions que cal respectar. L'ontologia s'ha dissenyat amb el programa Protégé i està disponible en format OWL.

S'ha desenvolupat una aplicació per poblar l'ontologia amb els tweets que s'obtenen a partir d'una cerca a Twitter. L'accés a Twitter es fa via l'API que ofereix per accedir a les dades des d'aplicacions de tercers. El resultat de l'execució de l'aplicació és un fitxer RDF/XML amb les tripletes corresponents a les instàncies dels objectes en l'ontologia.

També es mostra les possibilitats que existeixen per inferir informació dels tweets. Així s'han realitzat consultes en llenguatge SPARQL i s'esmenta una alternativa més visual per l'explotació de les dades.

El volum creixent de dades i la diversitat de formats que es van incorporant cada dia a la xarxa genera la necessitat d'avançar en les tecnologies que facilitin tant la seva gestió com la seva conservació. Els avenços en el camp de les ontologies i la web semàntica permetran estructurar el coneixement, dotar-lo de semàntica i així donar solució a la conceptualització de la informació.

Abstract (in English):

With the popularity of social networks in recent years it appears a new research field and focus. The data flowing through social networks defined new styles of interaction, far from traditional hierarchical structures and that is why their study arouses a great interest. This work is a first approach to the use of an ontology to infer information about the tweets. With the ontology I attempt to conceptualize the information obtained from Twitter and I wante to create a simplified view of the data in order to represent them easily.

The ontology is designed covers the basics of Twitter, their relationships and constraints that must be respected. The ontology is designed with the program and is available in Protégé OWL format.

I have developed an application to populate the ontology with the tweets obtained from a search on Twitter. The access to Twitter is through API. The API provides access to data from third party applications. The result of the execution of the application is a file in RDF / XML triples related to the instances of objects in the ontology.

It also shows the possibilities that exist to infer information from tweets. So were performed SPARQL query language and mentioned an alternative vision for the exploitation of the data.

The increasing volume and diversity of data formats that are being generated daily in the network need to advance technologies that facilitate both the management and conservation. Advances in the ontology and in the field of semantic allow structuring knowledge, giving it semantics and thus provide a solution to the conceptualization of the information.

Paraules clau:

Ontologia, Twitter, Web semàntica, OWL, RDF

Índex

1 Introducció 4
1.1 Context i justificació del Treball 4
1.1.1 Raó i oportunitat 4
1.1.2 Situació actual 4
1.2 Objectius del Treball 4
1.2.1 Objectius generals 4
1.2.2 Objectius específics 5
1.3 Enfocament i mètode seguit 5
1.4 Breu sumari de productes obtinguts 6
1.5 Breu descripció dels altres capítols de la memòria 6
2 Web semàntica i Ontologies 7
2.1 Web semàntica 7
2.2 Ontologies 7
3 RDF i OWL 9
3.1 RDF i RDF Schema 9
3.2 OWL 9
4 Disseny de l'Ontologia de Twitter 10
4.1 Qüestions de disseny 10
4.2 Jerarquia de classes 11
4.3 Objectes de Twitter 13
4.3.1 Tweets 13
4.3.2 Coordinates 14
4.3.3 User 15
4.3.4 Entities 16
4.3.5 Hashtags 17
4.3.6 Media 18
4.3.7 URL 19
4.3.8 User Mention 19
4.3.9 Places 20
4.3.10 Bounding box 20
4.4 Propietats 21
5 Poblar l'ontologia. Implementació 26
5.1 Anàlisi i disseny 26
5.2 Implementació 27
5.2.1 Python 27

5.2.2 API de Twitter 28
5.2.3 Tweepy 28
5.2.4 RDFLib 28
5.2.5 twitter.py 29
5.3 Requeriments de programari necessari per el funcionament de l'aplicació ... 34
5.4 Resultat de l'execució. Instàncies creades per l'aplicació 35
6 Extreure informació del tweets 40
6.1 Consultes 40
6.2 Altres formes d'inferir informació 41
7 Conclusions 44
8 Glossari 45
9 Bibliografia 46
10 Annexos 48
10.1 Annex 1: Entorn de desenvolupament i d'execució 48
10.2 Annex 2: Codi font de l'aplicació per poblar l'ontologia 49

Llista de figures

Fig. 1 Jerarquia de classes 12
Fig. 2 Jerarquia de classes en Protégé 12
Fig. 3 Classe Twitter en Protégé 14
Fig. 4 Classe Coordinate en Protégé 15
Fig. 5 Classe User en Protégé 16
Fig. 6 Classe Entity en Protégé 17
Fig. 7 Propietats de Entity 17
Fig. 8 Classe Hashtag en Protégé 17
Fig. 9 Classe Media en Protégé 18
Fig. 10 Propietats de Size 18
Fig. 11 Classe Size en Protégé 19
Fig. 12 Classe URL en Protégé 19
Fig. 13 Classe Place en Protégé 20
Fig. 14 Classe BoundingBox en Protégé 21
Fig. 15 Propietats d'objecte 21
Fig. 16 Diagrama de propietats reduït 22
Fig. 17 Diagrama de propietats ampliat 24
Fig. 18 Propietats classe Entity 25
Fig. 19 Diagrama de classes UML 27

1 Introducció

1.1 Context i justificació del Treball

1.1.1 Raó i oportunitat

Actualment una part important de la població fa un gran ús de les xarxes socials, portant-les a convertir-se en un nou mitjà de comunicació i d'espai de relacions interpersonals. De manera que tenen un gran impacte, no només en les relacions interpersonals, sinó que també són fonts de consulta en la presa de decisions sobre temes tant dispars com pot ser el consum o les opinions polítiques, passant per campanyes de boicot o recolzament sobre temàtiques ben diverses.

És per això que hi ha un gran nombre d'entitats, empreses i científics interessat en poder obtenir i analitzar el gran volum de dades generades i les relacions entre elles. En aquest sentit Twitter, per les seves característiques, és una de les més interessants d'analitzar. Els seus missatges són curts plens de referències a usuaris, altres tweets, altres xarxes socials i a continguts externs.

En aquest context, disposar d'una eina que permeti analitzar, estudiar i, en definitiva, comprendre millor tota la informació que és generada i difosa mitjançant Twitter interessa per diferents raons a molts col·lectius i entitats. Així doncs, l'objectiu principal d'aquest treball és proporciona aquesta eina, si més no, un primer pas cap a la possibilitat d'inferir informació partint dels tweets generats a diari pels usuaris de Twitter.

1.1.2 Situació actual

En aquest moment hi ha un nombre gens menyspreable de projectes en aquesta direcció, si bé molt d'ells tenen objectius més específics com per exemple inferir informació de Twiter sobre els interessos dels usuaris[1], inferir informació sobre les relacions entre usuaris de diferents xarxes socials[2]... Tot i això aquest treball es centrarà únicament en Twitter. Es crea així, una eina que actua sobre Twitter, i permet inferir informació sobre el seu ús.

1.2 Objectius del Treball

1.2.1 Objectius generals

Alguns dels objectius generals d'aquest treball estan especificats en pla de treball de l'assignatura i són els següents:

- Estudiar els conceptes bàsics de la web semàntica
- Conèixer l'estructura i representació d'alguns llenguatges de representació d'informació a la Web

[1] User interests ontology: https://sites.google.com/site/twitterresearch09/articles/userinterestsontology
[2] The Friend of a Friend (FOAF) project: http://www.foaf-project.org/

- Conèixer l'organització i estructura de les BD:XML natives, així com l'estat actual d'utilització d'alguns sistemes concrets
- Analitzar els llenguatges de consulta actuals que incorporen els sistemes de BD:XML natives
- Aplicar l'ús de les ontologies en la representació del coneixement emprant wikis semàntiques

Caldria afegir alguns objectius generals intrínsecs a l'assignatura de treball de fi de carrera:

- Sintetitzar els coneixements adquirits als estudis en un projecte concret
- Fer treball de recerca sobre la temàtica del projecte
- Redactar una memòria de projecte

1.2.2 Objectius específics

Els objectius específics indicats a l'enunciat del projecte són els següents:

- Crear una ontologia per emmagatzemar els tweets (també es podrà reutilitzar ontologies existents si es troba adient)
- Crear un programa que permeti poblar l'ontologia a partir dels tweets
- Crear un programa o un conjunt de regles sobre l'ontologia que permeti extreure informació dels tweets de l'ontologia. (objectiu optatiu)
- Analitzar les instàncies creades a l'ontologia a partir dels tweets de Twitter

Altres objectius específics derivats dels generals són:

- Conèixer l'estat actual així com els treballs en curs i tendències de a web semàntica
- Estudiar i aprendre el llenguatge RDF i XML com a mitjans per a representar informació
- Estudiar el llenguatge OWL, la seva relació amb RDF i XML i la seva aplicació concreta en la creació d'ontologies
- Estudiar les aplicacions per a la creació d'ontologies, com ara Protégé
- Estudiar l'estructura de les parts de Twitter: missatges, usuaris, relacions...
- Estudiar com utilitzar la API de Twitter, guardar la informació rellevant dels tweets i omplir l'ontologia
- Analitzar la possibilitat de reutilitzar recursos existents com ara la DBpedia i altres ontologies

1.3 Enfocament i mètode seguit

A la primera fase el projecte s'ha fet una recerca exhaustiva per una banda sobre els conceptes i per una altra sobre l'estat actual de la tecnologia, projectes i productes reutilitzables. En base als resultats d'aquesta recerca, als coneixements adquirits i a l'estudi realitzat dels diferents llenguatges,

tecnologies i programaris s'han pres les decisions necessàries per la resolució del treball.

D'una banda s'ha dissenyat una ontologia de Twitter nova, i s'ha intentat representar des d'una visió més basada en els conceptes de Twitter que en el disseny clàssic orientat a objectes. S'ha dissenyat i desenvolupat una aplicació que pobla de dades aquesta ontologia i genera un rdf amb aquestes dades per la seva posterior explotació. S'ha pres la decisió de utilitzar el llenguatge Python com a resultat de la avaluació que s'ha realitzat de les diferents llibreries i eines disponibles pels diferents llenguatges.

Amb les dades recollides es realitzen una sèrie de consultes en SPARQL com a petita mostra de les possibilitats que ofereix l'explotació de les dades.

1.4 Breu sumari de productes obtinguts

S'espera, en finalitzar aquest projecte, disposar d'un producte que representi la informació rellevant de Twitter i que permeti inferir informació sobre els tweets. El producte es composarà d'una ontologia, un programari que ompli l'ontologia a partir dels tweets i, opcionalment, un programari que permeti fer consultes a l'ontologia.

1.5 Breu descripció dels altres capítols de la memòria

A la esta da capítols de la memòria es tracten els diferents temes relacionats amb l'objecte del projecte. Inicialment s'introdueix els temes i conceptes generals, web semàntica, ontologies, RDF i OWL. A continuació es descriu detalladament el disseny de l'ontologia, aprofundint en el disseny realitzat i en l'ontologia obtinguda. Tot seguit es tracta el tema del poblament de l'ontologia i l'aplicació desenvolupada a tal efecte. Per finalitzar es descriu l'extracció d'informació de les dades recollides a l'ontologia així com les conclusions de tot el projecte.

2 Web semàntica i Ontologies

2.1 Web semàntica

El coneixement i la seva gestió esdevé cada dia més important, el seu volum creixent i la diversitat de formats que es van incorporant i es mantenen genera la necessitat d'avançar en les tecnologies que facilitin tant la seva gestió com la seva conservació. En aquest sentit la web semàntica intenta avançar en aquests aspectes, i la manera és estructurant el coneixement i donant solució a la conceptualització de la informació i el coneixement, així, proposa organitzar-lo per conceptes i semàntica.

D'aquesta manera s'intenta afegir a les tecnologies actuals un nivell semàntic als serveis i continguts, per permetre que puguin ser analitzats, raonats i manipulats, inclús creats, de manera automàtica per agents de programari, i no només per agents humans.

Aquestes tecnologies es deriven dels avenços fets per el camp de la Intel·ligència Artificial, però sense arribar a ser una àrea d'aquesta. La Intel·ligència Artificial es basa sobre tot en el raonament humà i intenta reproduir-lo. Per contra, les tecnologies de la web semàntica es centren en donar solucions parcials, de manera que es pot inferir informació i arribar a conclusions parcials però encara molt allunyades del raonament humà. Així, intenten introduir descripcions semàntiques sobre els recursos per permetre que les màquines comprenguin prou el contingut de la web com per poder ocupar-se d'una part de les tasques que actualment fa l'usuari del web manualment i així assistir-lo. S'avança d'aquesta manera en la evolució de la web actual.

2.2 Ontologies

Un element essencial i principal en la web semàntica és l'ontologia, encara que s'apliquen a altres àrees. En el camp de la informàtica es pot prendre com a definició d'ontologia, com apunten Antoniou i Harmelen (2008), la definició de T.R. Gruber, redefinida més tard per R.Studer; *Una ontologia és una especificació explícita i formal d'una conceptualització*.[3]

En conseqüència, per definir les ontologies s'utilitza un llenguatge formal que pot ser entès per les màquines, o concretament per programari. Aquestes descriuen un domini de discurs d'una àrea concreta, especificant els conceptes, que generalment estan jerarquitzats, i les relacions entre ells. Mitjançant les ontologies es pot compartir el coneixement, és a dir, la terminologia emprada sobre un domini està consensuada i és compartida. S'eliminen així, les diferències terminològiques. Es formen, per tant, xarxes semàntiques d'informacions interrelacionades.

[3] G. Antoniou; F van Harmelen (2008) *A Semantic Web Primer*. Ed. Cambridge (MA): The MIT Press

Les ontologies són especialment útils per l'organització de webs així com per millorar les cerques a Internet. Possibiliten fer un pas més enllà als buscadors i poder fer cerques a nivell conceptual, pel contingut del web, evitant les paraules clau que comporten ambigüitat i poca precisió.

L'ontologia de Twitter que ens ocupa pretén ser un primer pas per inferir informació del tweets que cada dia circulen entre els usuaris de Twitter. Com a primer pas es defineix l'estructura dels elements bàsics de Twitter, l'estructura dels tweets, del usuaris i de la resta d'elements necessaris per arribar a possibilitat la inferència posteriorment.

3 RDF i OWL

3.1 RDF i RDF Schema

RDF és un model de dades per descriure objectes, les seves relacions i la descripció semàntica, a més d'un estàndard de W3C.

La seva sintaxis està basada en XML, però soluciona el problema del XML que no té mecanismes per expressar la semàntica de les dades. I, com l'XML, està pensat per facilitar el intercanvi de dades a la web.

L'estructura de dades bàsica del RDF són les sentències que consisteixen en triples: subjecte-predicat-objecte. El subjecte és el recurs sobre el qual es parla, el predicat és una propietat que descriu la relació entre el subjecte i l'objecte, i l'objecte pot ser un valor o un altre recurs.

L'RDF és independent del domini, de manera que s'ha de definir també el domini mitjançant un llenguatge de descripció de vocabularis, l'RDF Schema (RDFS). Mitjançant l'RDFS es defineixen les classes d'objectes i les seves propietats o relacions i permet establir una jerarquia de les classes i també de les propietats.

3.2 OWL

Ontology Web Language (OWL) és un llenguatge de descripció d'ontologies, utilitzat per a la descripció de classes i propietats. Les ontologies que descriu estan basades en la web. S'acostuma a considerar que és una extensió de RDFS ja que a més d'incorporar tota la capacitat expressiva del RDF afegeix la possibilitat d'utilitzar expressions lògiques i amplia les possibilitats de definir restriccions a les classes i les propietats.

És un estendard de W3C i aquest consorci el té especificat en varis documents públics.

Existeixen varis subllenguatges del OWL :

- OWL Lite
- OWL DL
- OWL Full

Per desenvolupar aquest projecte s'ha triat el OWL DL. S'ha arribat a aquesta decisió donat que el OWL Full, tot i ser completament compatible amb RDF i tenir l'expressibilitat més alta, comporta certs problemes de computabilitat i el OWL Lite és més limitat. Així doncs s'ha optat pel OWL DL, i encara que es perd una part de la compatibilitat completa amb RDF, és eficient computacionalment.

4 Disseny de l'Ontologia de Twitter

Per el disseny de l'ontologia objecte d'aquest projecte s'ha pres com a referència l'estructura dels diferents objectes que utilitza Twitter. Aquesta informació es proporciona a: A field guide to Twitter Platform objects.

Com a base hi ha 4 tipus d'objectes: `Tweets`, `Users`, `Entities` i `Places`. Cada un d'ells està especificat quins atributs i objectes el composen. De tota la informació que inclouen s'ha fet una tria d'aquella que es considera rellevant i/o interessant per a la creació de l'ontologia, la seva posterior alimentació i explotació de les seves dades, per tant, hi ha alguns del camps que no s'han inclòs.

4.1 Qüestions de disseny

S'ha pres la decisió de crear una ontologia nova ja que les existents sobre xarxes socials són molt generals i no s'ha considerat pertinent reutilitzar-les com a base. Tot i això s'ha importat 3 ontologies externes. Dos d'elles de la DBpedia: `Language` i `Country`. La primera s'utilitza per a la codificació del llenguatge dels tweets i la segona pel País que es relaciona amb la classe `Place`. La tercera ontologia importada és SpatialThing de W3C per utilitzar la classe `Point` com a punt geogràfic, conté les coordenades expressades en longitud i latitud i es relaciona amb les classes `Coordinate` i `BoundingBox`.

El disseny s'ha realitzat amb el programa protégé v 3.5, i els diagrames amb els plugins OWLViz i Ontograf. S'ha generat l'ontologia en OWL, en la seva versió DL.

L'ontologia s'ha intentat representar des d'una visió més basada en els conceptes de Twitter que en el disseny clàssic orientat a objectes. Com a resultes d'això hi ha alguns atributs dels objectes de Twitter que en l'ontologia són relacions i no propietats de dades, a més els id's que identifiques els objectes de Twitter seran les URI dels objectes i per tant en l'estructura de l'ontologia no es veuen reflectits.

Cal tenir en compte que una ontologia sempre està en "estat beta", això vol dir que a mida que es treballi amb ella es van detectant possibles modificacions i/o ampliacions per tal de fer-la més eficient per als objectius plantejats. Això, a més, és una característica intrínseca al fet de que les ontologies són compartides i reutilitzables per altres ontologies i usuaris que les completen i adapten als seus projectes i objectius.

4.2 Jerarquia de classes

El disseny general es basa en els objectes de Twitter, com s'ha exposat anteriorment, i d'això han resultat 4 classes principals: `Tweet`, `User`, `Places` i `Entity`.

A més s'ha creat la classe `Geo` que en deriven dos filles: `Coordinate`, que és la coordenada geogràfica d'un tweet i la `BoundigBox` que amb 4 punt geogràfics delimita un àrea geogràfica d'un `Place`. La classe `Geo` té definida la propietat d'objecte `hasPoint` que la relaciona amb la classe `Point`. Les seves filles hereten aquesta propietat i l'especialitzen afegint una restricció de cardinalitat. Per la classe `Coordinate` la cardinalitat és 1, és a dir només pot relacionar-se amb un únic punt i per la classe `BoundingBox` la cardinalitat és 4, que són els 4 punts que defineixen un àrea que és la boundingbox.

Una altra especialització de classes es dona amb les classes relacionades amb les mides que estan definides als tweets per els objectes `media`. S'ha definit una classe `Size` que té 4 filles `Thumb`, `Small`, `Medium` i `Large`. Cada objecte `media` d'un tweet pot tenir aquests 4 objectes que indicaran per cada mida quin són els valors. Aquestes 4 classes són disjuntes entre elles.
També s'ha creat una altra especialització de classes mitjançant l'herència entre les diferents tipus d'entities que pot contenir un tweet. Així hi ha la classe `Entity` que té 3 filles `Hastag`, `URL` i `Media`, reproduint d'aquesta manera l'estructura d'objectes de Twitter. Tot i així a Twitter les filles d' `Entity` són 4, inclou les anteriors més `Mention`, però per la naturalesa d'aquesta última s'ha pres la decissió de modelar-la com a propietat d'objectes i s'ha creat les propietats `mentions` i `inMentionedBy` que relacionen les classes `Tweet` i `User`, que modelen les possibles mencions que es fan d'un usuari dins d'un tweet.

En el següent diagrama es pot veure l'estructura de la jerarquia creada:

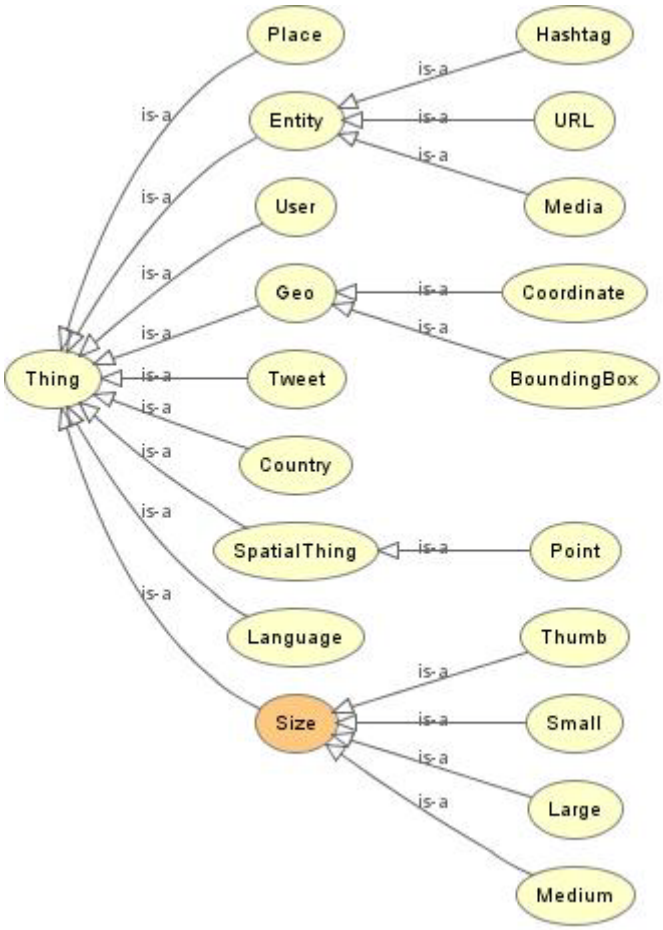


Fig. 1 Jerarquia de classes

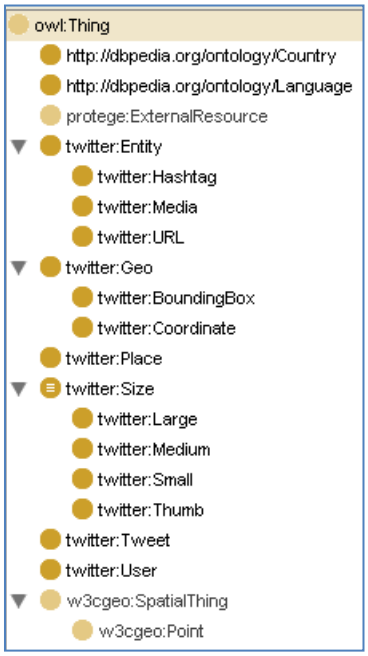


Fig. 2 Jerarquia de classes en Protégé

4.3 Objectes de Twitter

Els objectes de Twitter que s'han modelat a l'ontologia són els següents:

- Tweet: Els tweets són la peça bàsica de Twitter: els usuaris tuitejen tweets.
- Coordinates: són coordenades que indiquen l'ubicació d'un tweet.
- User: usuari de Twitter. Crea els tweets
- Entities: proporcionen metadades i informació contextual associada al contingut del tweet. Inclou Hastags, Media, URL i User_mentions
- Places: Llocs amb nom i informació geogràfica. Poden anar annexades a un tweet.
- Boundig box, annexat a places, indica les 4 coordenades que delimiten el lloc geogràfic.

A continuació es fa una breu descripció de la informació que inclou cada objecte i que s'ha considerat que ha de estar recollida a l'ontologia.

4.3.1 Tweets

Els tweets són la peça bàsica de Twitter: els usuaris tuitejen tweets.

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
text *(text)*	String	Text UTF-8 del tweet.	DataProperty
retweet_count *(retweetCount)*	Int	Nombre de vegades que aquest tweet s'ha retuitejat.	DataProperty
in_reply_to_status_id_str *(isReplyToTweet)*	String	Pot ser nul. Si el tweet és una resposta, aquest camp conté l'ID del tweet original.	ObjectProperty / Tweet
lang *(hasLanguage)*	String	Pot ser nul. Identificador BCP 47 de l'idioma del tweet que s'ha detectat, o "und" si no s'ha pogut identificar.	ObjectProperty /Language
coordinates *(hasCoordinate)*	Coordinates	Ubicació geogràfica del tweet reportada per l'aplicació client.	ObjectProperty / Coordinates
entities *(hasEntities)*	Entities	Entitats extretes del text del tweet.	ObjectProperty / Entities
place *(hasPlace)*	Places	Pot ser nul. Indica que el tweet està associat amb un lloc (Place).	ObjectProperty / Place
retweeted_status *(isRetweetOf)*	Tweet	Conté una representació del tweet original en cas de retweet.	ObjectProperty/ Tweet
user *(isTweetedBy)*	Users	L'usuari que ha enviat el tweet.	OjectProperty/ User

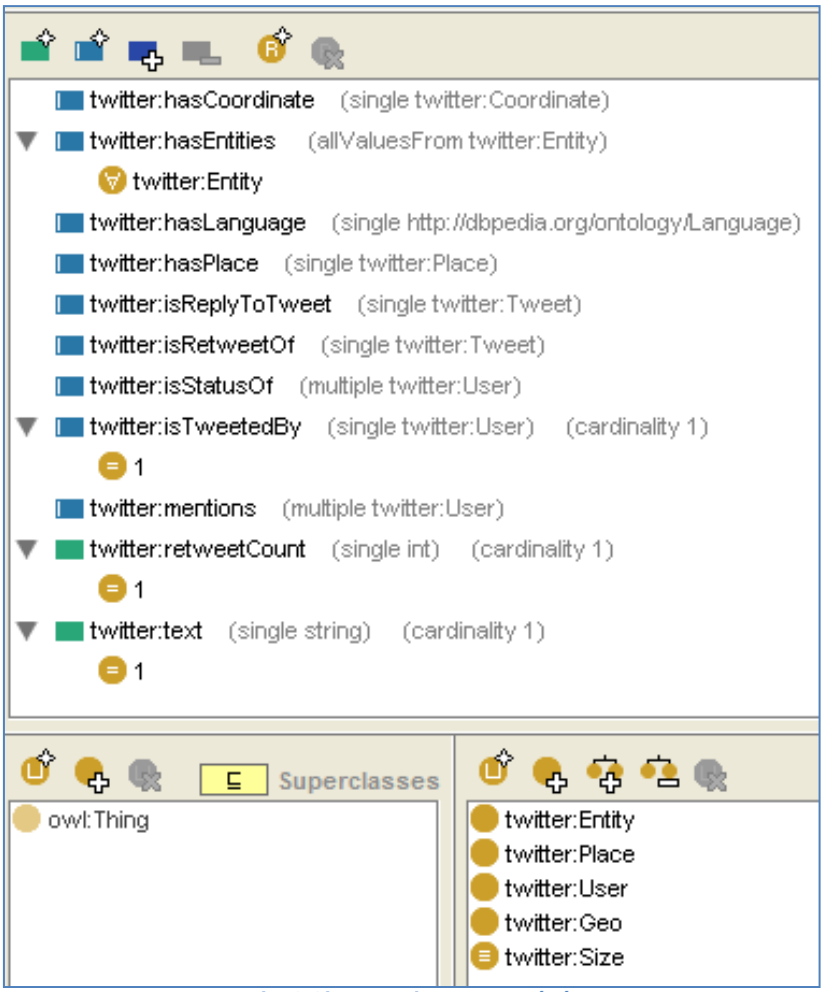


Fig. 3 Classe Twitter en Protégé

Altres propietats:

Object Property Mentions: Modela l'Entity Mention, quan un tweet conté una Entitty Mention a un altre usuari. No s'ha creat l'objecte `Mention`, es modela amb aquesta propietat i la seva inversa: `isMentiodedBy`

S'utilitza el camp `id_str` del tweet de twitter per crear la URI de l'objecte.

4.3.2 Coordinates

Són coordenades que indiquen l'ubicació d'un tweet

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
coordinates *(hasPoint)*	Collection of Float	Longitud i latitud del tweet	ObjectProperty / Point

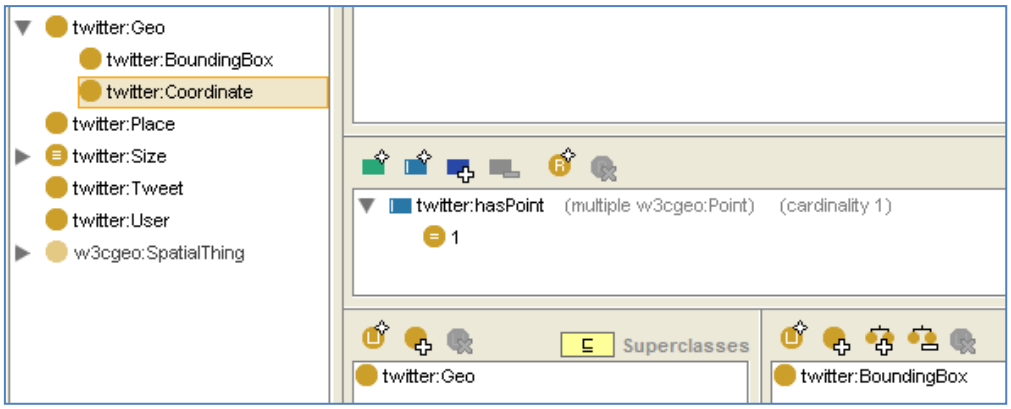


Fig. 4 Classe Coordinate en Protégé

4.3.3 User

Usuari de Twitter. Crea els tweets

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
description *(description)*	String	Pot ser nul. Text UTF-8 creat per l'usuari i que descriu el seu compte.	DataProperty
name *(name)*	String	El nom de l'usuari tal i com l'ha definit ell mateix.	DataProperty
screen_name *(screenName)*	String	El nom en pantalla o àlies amb què s'identifica l'usuari.	DataProperty
url *(url)*	String	Pot ser nul. URL afegida per l'usuari al seu perfil.	DataProperty
status *(hasStatus)*	Tweets	Pot ser nul. Si és possible, és el tweet o retweet més recent de l'usuari.	ObjectProperty / Tweet

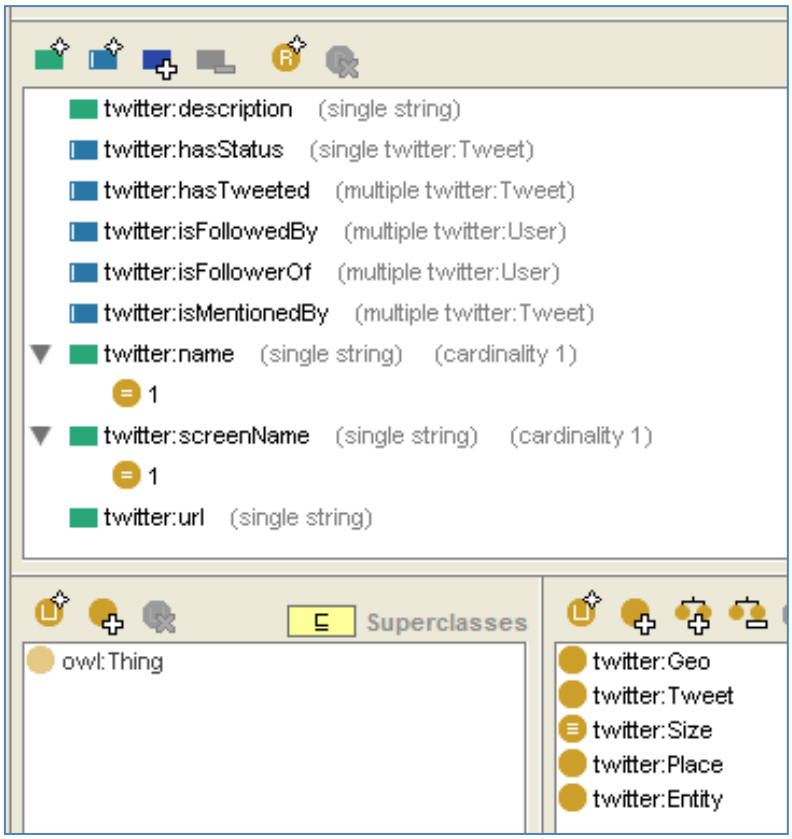


Fig. 5 Classe User en Protégé

S'utilitza el camp `id_str` de l'usuari de Twitter per crear la URI de l'objecte Usuari.

Altres propietats:

Estableixen relacions entre objectes, les quals s'han considerat interessants de modelar

- `hasTweeted` – indica que l'usuari ha fet un Tweet
- `isFolowedBy` – indica uns relació entre dos usuaris, un és seguidor de l'altre
- `isFollowerOf` – és la inversa de l'anterior
- `isMentionesBy` – indica que l'usuari ha estat mencionat a un tweet.

4.3.4 Entities

Proporcionen metadades i informació contextual associada al contingut del tweet. Inclou Hastags, Media, URL i User_mentions

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
hashtags *(Classe hashtag)*	Array of Object	Hashtags extrets del text del tweet.	ObjectProperty/Hashtag
media *(Classe Media)*	Array of Object	Elements multimèdia que acompanyen el tweet.	ObjectProperty/Media
urls *(Classe URL)*	Array of Object	URLs que apareixen en el text del tweet o bé en camps textuals de l'objecte usuari.	ObjectProperty/URL

user_mentions *(Classe Mentions)*	Array of Object	Representa altres usuaris de Twitter mencionats en el text del tweet.	Sense classe/ Modelat amb 2 ObjectProperties inverses entre elles -mentions Tweet-User -is MentionedBy User-Tweet

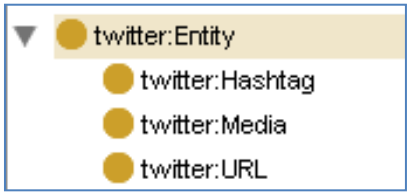


Fig. 6 Classe Entity en Protégé

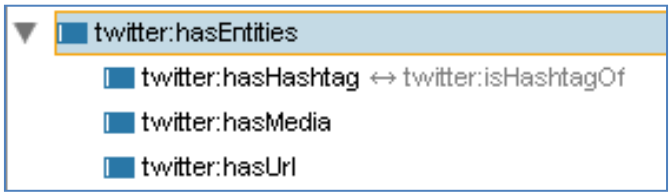


Fig. 7 Propietats de Entity

4.3.5 Hashtags

Text precedit una # que s'utlitza per indicar el tema sobre el que versa el tweet.

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
text *(text)*	String	Nom del hashtag sense el '#' inicial.	DataProperty

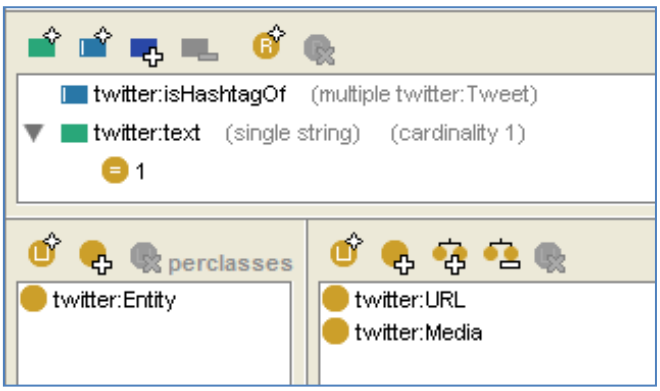


Fig. 8 Classe Hashtag en Protégé

4.3.6 Media

És un recurs media, pot ser una imatge o altre tipus de contingut media, que està referenciat a un tweet.

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
type *(mediaType)*	String	Tipus de dada multimèdia.	DataProperty
url *(url)*	String	URL que enllaça al contingut multimèdia tal i com apareix en el text del tweet.	DataProperty
sizes *(hasSize, hasThumbSize, hasSmallSize, hasMediumSize, hasLargeSize)*	Object	Objecte amb les possibles mides de fitxer disponibles.	ObjectPopert y/Large, Large Medium, Small, Thumb Size

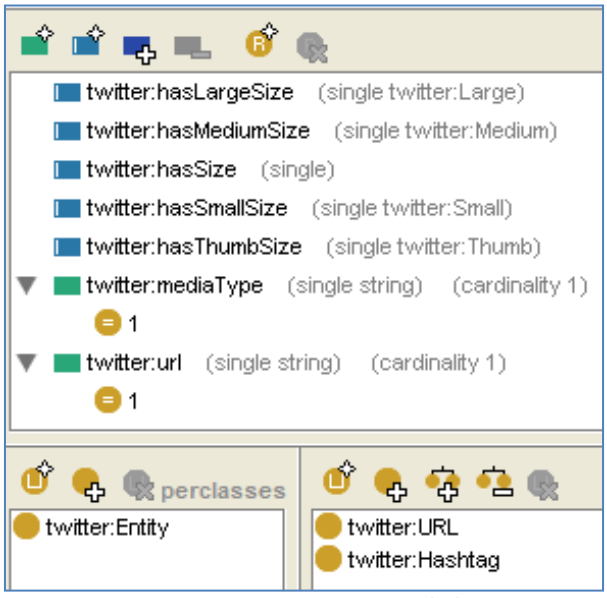


Fig. 9 Classe Media en Protégé

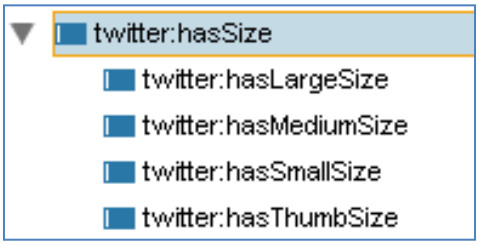


Fig. 10 Propietats de Size

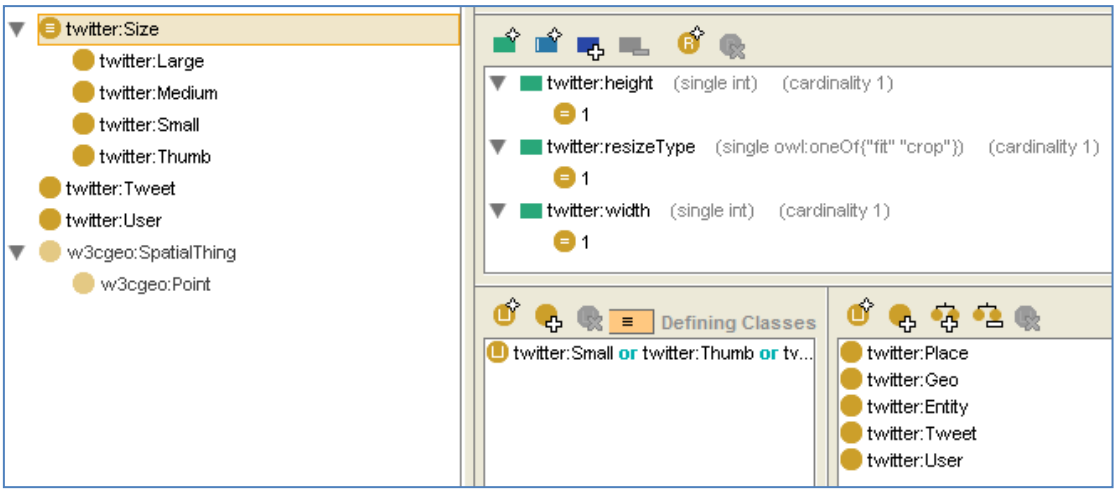


Fig. 11 Classe Size en Protégé

4.3.7 URL

És una adreça URL que està continguda a un tweet.

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
url *(url)*	String	URL tal i com apareix en el text del tweet.	DataProperty

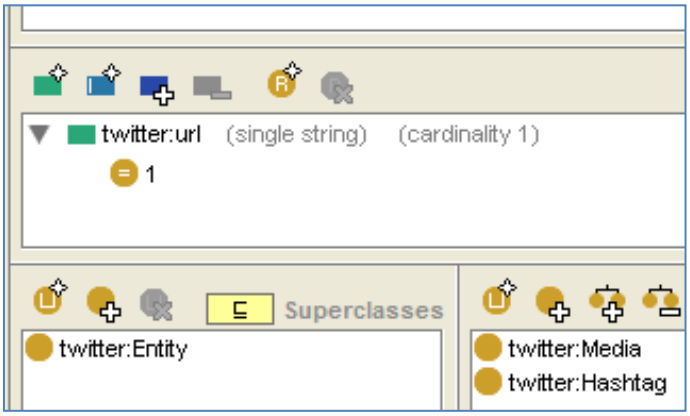


Fig. 12 Classe URL en Protégé

4.3.8 User Mention

És una menció d'un usuari concret a un tweet.

No està modelat com a classe, s'ha modelat amb les Object Properties:

- `mentions` amb domini: `Tweet` i rang: `User`
- `isMentionedBy` amb domini: `User` i rang: `Tweet`

que són inverses entre elles

4.3.9 Places

Llocs amb nom i informació geogràfica. Poden anar annexades a un tweet.

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
name *(name)*	String	Nom curt del lloc.	DataProperty
place_type *(placeType)*	String	Tipus d'ubicació representada per aquest lloc.	DataProperty
url *(url)*	String	URL amb informació addicional i metadades sobre el lloc	DataProperty
bounding_box *(hasBoundingBox)*	Object	Una bounding box (o perímetre) que encercla o determina el lloc.	ObjectProperty/BoundinBox
country_code *(hasCountry)*	String	Codi del país on és el lloc.	ObjectProperty/Country

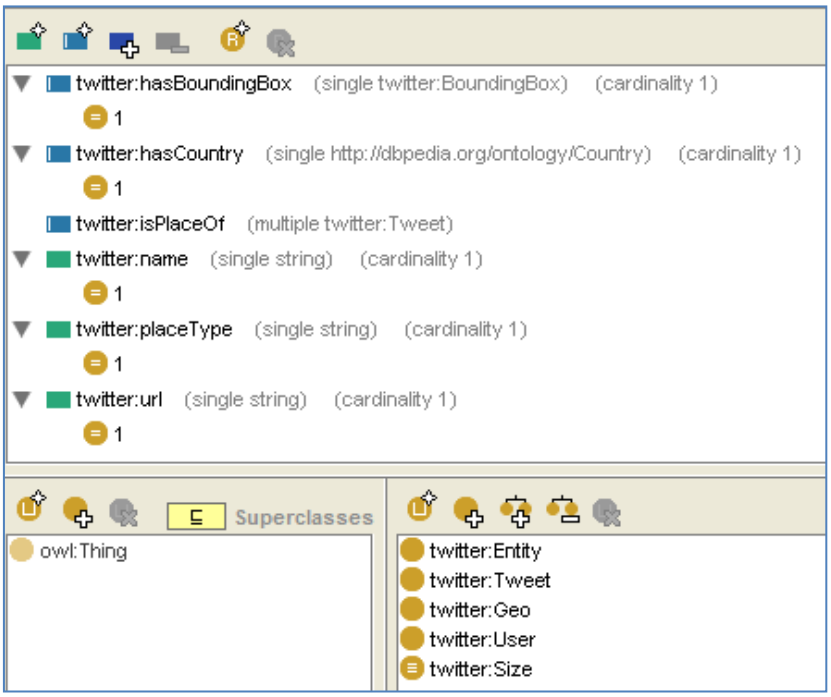


Fig. 13 Classe Place en Protégé

4.3.10 Bounding box

Annexat a places, indica les 4 coordenades que delimiten el lloc geogràfic.

Camp *(propietat)*	**Tipus**	**Descripció**	**Modelat /classe**
coordinates *(hasPoint)*	Array of Array of Array of Float	Coordenades del polígon que encercla un lloc.	ObjectProperty / Point

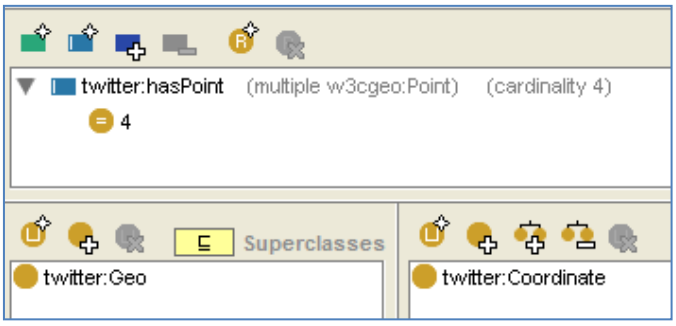


Fig. 14 Classe BoundingBox en Protégé

4.4 Propietats

S'han creat tant propietats d'objecte per definir les relacions entre classes com propietats de dades per relacionar valors amb les classes.
Les propietats d'objecte resultants són les següents:

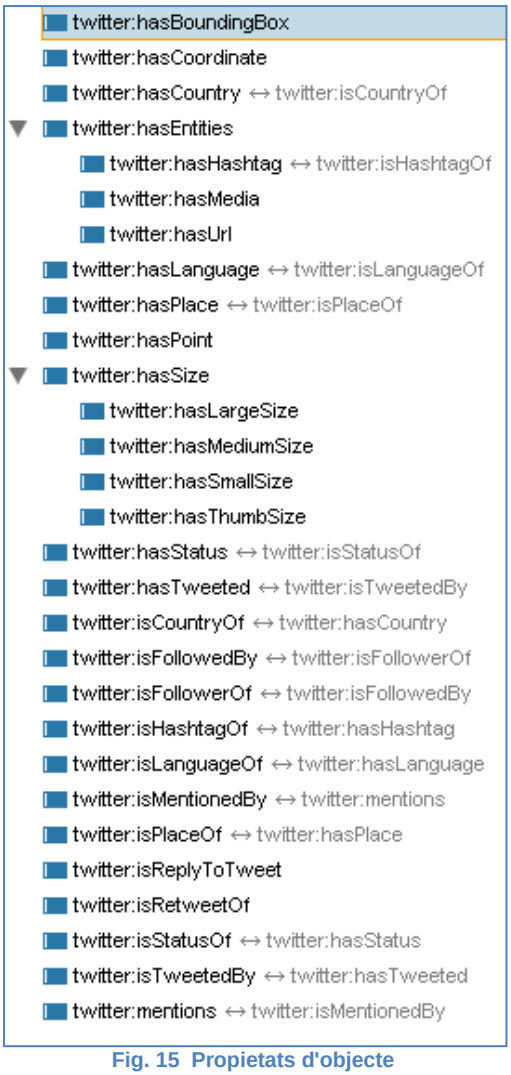


Fig. 15 Propietats d'objecte

Les relacions més destacables queden reflectides en el següent diagrama reduït, no apareixen totes les classes i relacions només aquelles més rellevants. Es pot observar com l'eix central és la classe `Tweet`.

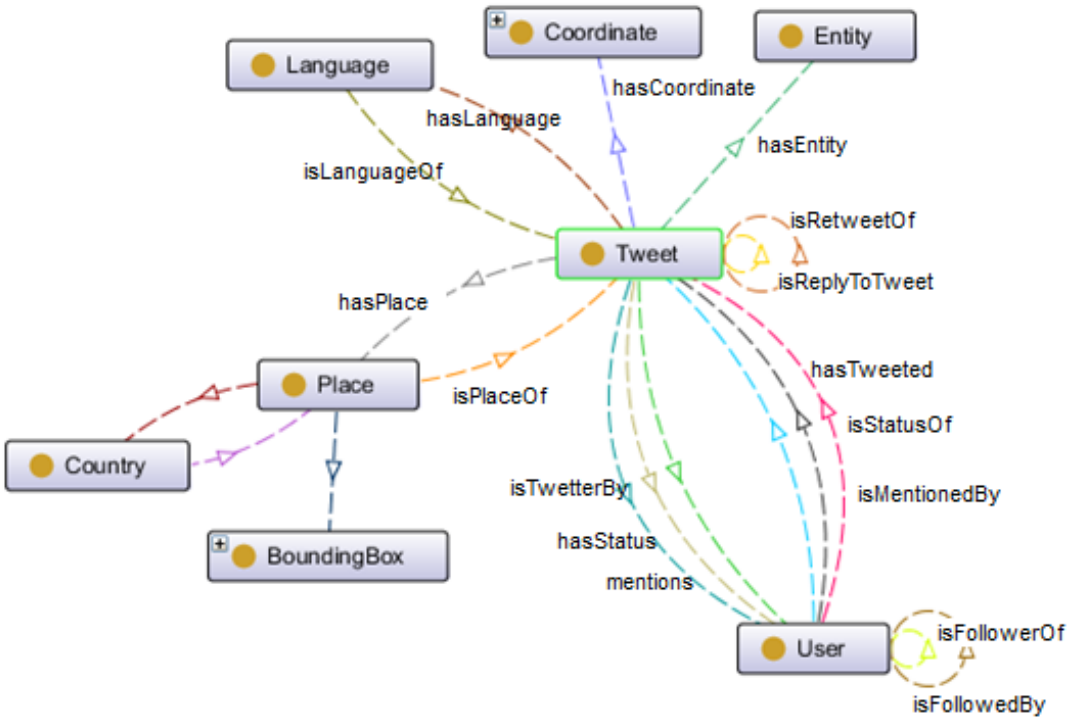


Fig. 16 Diagrama de propietats reduït

Entre la classe `Tweet` i la classe `User` existeixen varies propietats que les relacionen:

- `isTwettedBy` – indica que hi ha un determinar usuari que ha creat un tweet concret, aquesta propietat estableix la relació entre l'instància d'aquest usuari i l'intancia del seu tweet. És funcional, ja que un tweet unicament pot tenir un sol usuari.
- `hasTweeted` – és la inversa de l'anterior, proporciona la relació quan l'usuari té un tweet, i és inversa funcional.
- `isStatusOf` – indica que un determinat tweet és l'estatus d'un usuari, és a dir, l'últim tweet que aquell usuari ha tuitejat.
- `hasStatus` – la inversa de l'anterior, diu que d'instància de l'usuari té un tweet d'estatus, és funcional

- `mentions` – indica que el tweet conté una menció a un usuari concret
- `isMentionedBy` – la inversa de l'anterior, indica que l'usuari ha estat mencionat en un tweet

Hi ha dos propietats que relacionen la classe `Tweet` amb ella mateixa i són:

- `isRetweetOf` – indica que un tweet és un retweet d'un altre tweet, i és funcional. És ha dir, un tweet que ha estat reenviat, normalment per un altre ussuari.
- `isReplyToTweet` – indica que un tweet és reposta d'un altre tweet, també és funcional.

També hi ha dos propietats que relacionen la classe `User` amb ella mateixa i són `isFollowerOf` i la seva inversa `isFollowedBy`, relacionen instàncies de la classe Usuari amb altres instàncies de la mateixa classe. Semànticament que un usuari és seguidor d'un altre usuari i a la inversa, així proporcionen la informació sobre els seguidors d'un usuari en concret i la inversa a quin usuaris segueix un usuari en concret. No són funcionals ja que són relacions de 1 a molts i de molts a 1.

Un `tweet` pot contenir informació geogràfica d'un lloc determinat, així un tweet es pot relacionar amb una instància de la classe `Place`. Es relacionen mitjançant la propietat `hasPlace` i la seva inversa `isPlaceOf`. D'aquesta manera es pot obtenir els tweets que fan referència a lloc geogràfic, que serà una instància de la classe `Place.`

Una altra informació geogràfica que pot contenir un tweet són les coordenades en forma de longitud i latitud. Una instància d'un tweet es relaciona amb les seves coordenades mitjançant la propietat `hasCordinate` que relaciona la classe `Tweet` amb `Coordinates`.

Mitjançant `hasLanguage` i `isLanguageOf` s'estableix la relació amb el idioma del ttweet. La classe `Language` és importada, tal i com s'ha explicat anteriorment i porta incorporat la ISO639-1 que és la que fa servir Twitter per codificar l'idioma dels tweets.

A continuació es presenta el diagrama complet de totes les classes i relacions existents a l'ontologia per mostrar una visió més ampla i de conjunt.

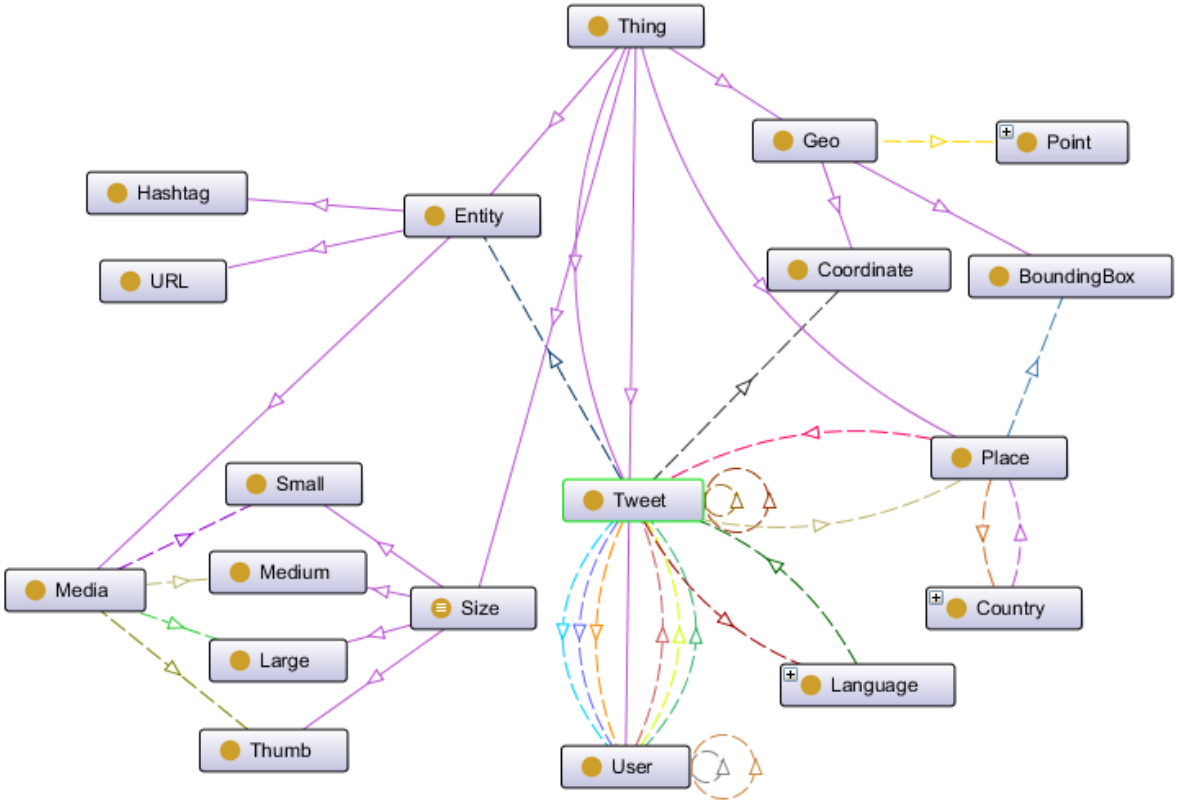


Fig. 17 Diagrama de propietats ampliat

Llegenda:

Les fletxes continues corresponen a relacions d'herència entre classes, són totes del mateix color:
Les línees discontinues corresponen a les relacions Object Properties entre les classes.

Sobre les classes és convenient remarcar alguns aspectes:

- Les subclasses d'`Entity`: `URL`, `Hashtag`, i `Media` estan definides com a classes disjuntes ja que cap objecte pot pertànyer a més d'una d'elles.
- Igualment s'han declarat com a disjuntes les classes: `Tweet`, `Geo`, `Place`, `Entity`, `Size` i `User`.

Un tweet por tenir varies instàncies de la casse `Entities`, que poden ser `Hashtags`, `URL`, `Media` i `Mention`. Un tweet també pot tenir varies Entities de cada tipus o cap.

`Mention` s'ha modelat de diferent manera que les altres Entities, no té una classe definida, s'han creat dos propietats: `mentions` i `isMentiodedBy` inverses que relacionen els objectes de la classe `Ususari` i `Tweet`, de manera que es disposa de la informació que dona la `Entity` de Twitter `Mention`. Les altre Entities tenen assignada la seva corresponent classe i es relacionen amb la classe `Tweet` mitjançant les propietats:

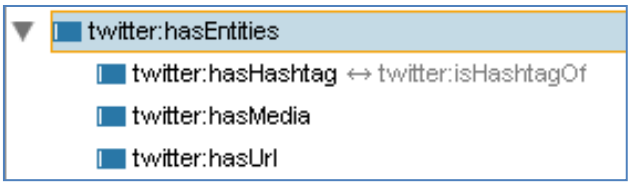


Fig. 18 Propietats classe Entity

5 Poblar l'ontologia. Implementació

5.1 Anàlisi i disseny

S'ha decidit poblar l'ontologia a partir del resultat d'una cerca de paraules clau en Twitter.

Per a cada tweet resultat de la cerca s'obté i es processa:

- Atributs corresponents a les data properties
- Usuari autor del tweet
- Si és una resposta, el tweet que es respon
- Si és un retweet, el tweet original
- Place
- Coordinates
- Entities: hashtag, media, url i mentions

Quan el procés implica un usuari (autor o mencions), per a cada usuari s'obté i es processa:

- Atributs corresponents a les data properties
- Tweet que correspon a l'status de l'usuari

D'aquesta manera, a partir de les mencions, status d'usuari, retweets i respostes a tweets, s'obtenen converses senceres entre usuaris i s'incorporen tweets que no han de ser necessàriament resultat de la cerca de paraules clau. Si la cerca retorna molts resultats existeix el risc de construir un graf d'RDF excessivament gran, i per això ha calgut definir algunes limitacions en el procés dels resultats obtinguts:

- Es limita a 10 el nombre de tweets resultat de la cerca inicial
- Les relacions entre usuaris (friends i followers) es calculen només entre tots els usuaris que han resultat del procés dels resultats, evitant així haver d'obtenir la informació d'usuaris amb milers de friends o followers
- La cerca es fa dels tweets que continguin "flkr", però es pot canviar el terme de la cerca en la línia 44 (q='flickr') del codi font.

Un cop construït el graf amb totes les instàncies de l'ontologia creades s'exporta a un fitxer en format RDF/XML que es pot tractar amb Protégé, Sesame o qualsevol altre programari capaç de llegir documents RDF.

El programa que executa la cerca i omple l'ontologia té com a objectiu, a més, mostrar com es fan servir les llibreries disponibles per tractar l'API de Twitter i els grafs RDF. Per això s'ha prioritzat la simplicitat i la llegibilitat del codi emprant només una classe principal i algunes d'auxiliars.

El diagrama de classes és el següent:

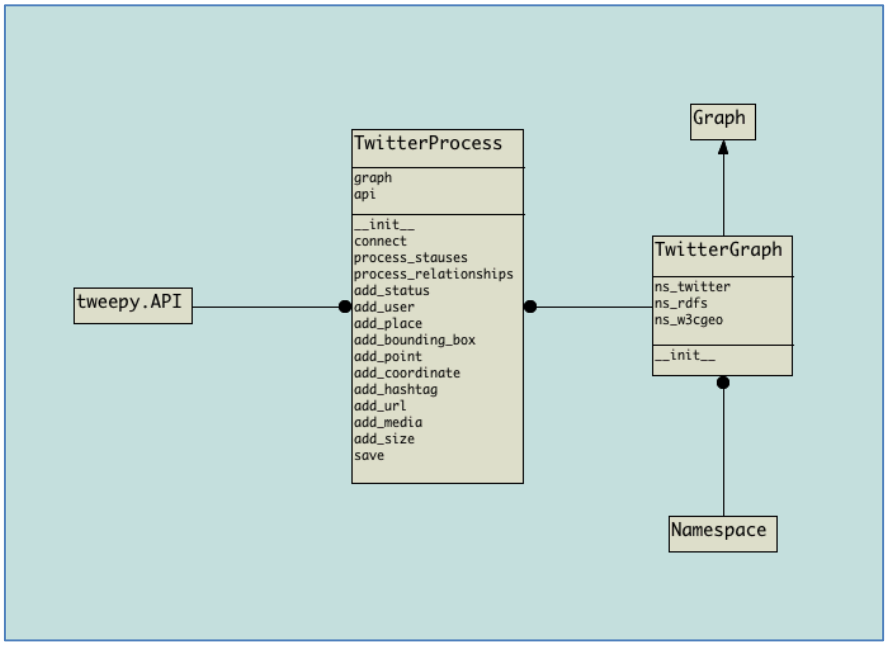


Fig. 19 Diagrama de classes UML

La classe *TwitterGraph* correspon al graf RDF i hereta de la classe *Graph* de la *RDFLib*. Fa servir la classe NameSpace de la RDFLib per crear tres atributs que corresponen als namespaces del graf.

La classe *TwitterProcess* conté un objecte *graph* de tipus *TwitterGraph*, que guardarà el graf d'RDF, i un objecte *api* de tipus *tweepy.API*, que donarà accés a l'API de Twitter. Els mètodes de la classe *TwitterProcess* executen la connexió a Twitter, la cerca i el procés dels resultats. Les crides a aquests mètodes aniran omplint els nodes del graf RDF. Finalment, el mètode *save* serialitzarà i guardarà el graf en format RDF/XML.

5.2 Implementació

La implementació s'ha dut a terme amb Python. S'ha desenvolupat un programa en aquest llenguatge contingut en un únic fitxer twitter.py. Per a l'ús d'RDF al programa s'ha fet servir la llibreria RDFLib i per a l'accés a l'API de Twitter la llibreria Tweepy, ambdues per a Python.

5.2.1 Python

S'ha treballat amb Python versió 2.7.6 per a MS Windows de 32 bits. S'ha fet servir Python 2 i no el més recent Python 3 perquè moltes de les llibreries que s'han avaluat en la recerca encara no estan portades a la versió 3.

Es pot descarregar Python 2.7.6 des del web oficial de Python: http://www.python.org

També s'ha instal·lat el gestor de paquets de Python PIP per facilitar la instal·lació de les llibreries. PIP, a més d'instal·lar el paquet sol·licitat, també instal·la les dependències. Es pot trobar a http://www.pip-installer.org. Per a instal·lar-lo cal descarregar l'script https://raw.github.com/pypa/pip/master/contrib/get-pip.py i fer:

```
python get-pip.py
```

5.2.2 API de Twitter

Per tenir accés a l'API de Twitter cal crear un usuari de Twitter i després una aplicació en Twitter des de l'adreça https://apps.twitter.com. La creació de l'aplicació proporciona les claus necessàries per a autenticar-se en l'API i poder operar amb ella. Les claus són:

- API key
- API secret
- Access token
- Access token secret

5.2.3 Tweepy

La llibreria `Tweepy` (https://github.com/tweepy/tweepy) proporciona una interfície a l'API de Twitter per a Python. A més de gestionar l'autenticació i l'accés a les funcionalitats de l'API, disposa de classes que faciliten treballar amb els resultats. Així, per exemple, els resultats d'una cerca de tweets són objectes del tipus tweepy.Status, de manera que es pot accedir a camps com ara Status.text o bé Status.User. D'igual manera es pot accedir a camps dels objectes User, Place o BoundingBox.

Tweepy també incorpora cursors que implementen la paginació dels resultats.

Es pot instal·lar fent:

```
pip install tweepy
```

5.2.4 RDFLib

Per treballar amb RDF s'ha triat la llibreria `RDFLib` (https://github.com/RDFLib/rdflib). Aquesta llibreria per a Python proporciona com a base un objecte Graph que és una col·lecció de tripletes (Subjecte, Predicat, Objecte), els seus components són URIs o Literals. RDFLib proporciona també Namespaces per agrupar les URIs, mètodes per cercar en l'objecte Graph, exportació a RDF/XML entre d'altres formats, i una implementació d'SPARQL per poder fer consultes.

Es pot instal·lar fent:

```
pip install rdflib
```

5.2.5 twitter.py

El programa en Python té un cos principal que crida a tres mètodes de la classe `TwitterProcess`:

```
if __name__ == '__main__':

    # Instancia la classe que farÃ  el procÃ©s
    twitter_process = TwitterProcess()
    # Processa els tweets
    twitter_process.process_statuses()
    # Processa les relacions entre usuaris
    twitter_process.process_relationships()
    # Desa el graph
    twitter_process.save()
```

Primer s'instancia l'objecte `twitter_process` a partir de la classe `TwitterProcess` i després es criden els mètodes:

`process_statuses()`: fa la cerca i processa els tweets
`process_relationships()`: processa les relacions entre els usuaris resultat del procés anterior
`save()`: exporta el graf d'RDF a un fitxer twitter.rdf en format RDF/XML.

Quan s'instancia un objecte de la classe `TwitterProcess` es crida automàticament el seu mètode `__init__`:

```
class TwitterProcess:
    """Processa el resultat de la cerca a Twitter i els converteix a RDF"""
    def __init__(self):
        """Connecta amb l'API de Twitter i crea un graph buit"""
        self.connect()
        self.graph = TwitterGraph()

    def connect(self):
        """Connecta amb l'API de Twitter"""
        auth = tweepy.OAuthHandler('api_key', 'api_secret')
        auth.set_access_token('access_token', 'access_token_secret')
        self.api = tweepy.API(auth)
```

En aquest cas es crea la connexió amb l'API de Twitter des del mètode `connect()` emprant la llibreria Tweepy i guardant aquesta connexió en la variable de classe `self.api`, i s'instancia la variable de classe `self.graph` que és de tipus `TwitterGraph`.

La classe `TwitterGraph` és descendent de la classe `Graph` d'RDFLib i en el seu mètode `__init__()` es defineixen els `Namespace` i `Prefix` que tindrà l'RDF i que després es faran servir en el procés de les dades:

```
class TwitterGraph(Graph):
    """Graph de RDF"""
    def __init__(self):
        """Inicialitza TwitterGraph amb els Namespaces i els prefix de la ontologi
a"""
        super(TwitterGraph, self).__init__()
        self.ns_twitter = Namespace('http://www.owl-
ontologies.com/Ontology1396114080.owl#')
```

```
        self.ns_rdfs = Namespace('http://www.w3.org/2000/01/rdf-schema#')
        self.ns_w3cgeo = Namespace('http://www.w3.org/2003/01/geo/wgs84_pos#')
        self.bind('twitter', self.ns_twitter)
        self.bind('rdfs', self.ns_rdfs)
        self.bind('w3cgeo', self.ns_w3cgeo)
```

RDFLib proporciona la classe `Namespace` que permet instanciar les variables `ns_twitter`, `ns_rdfs` i `ns_w3cgeo`. D'aquesta manera es pot referir a, per exemple, la propietat `isFollowerOf` del `namespace Twitter` com a `ns_twitter.isFollowerOf`.

Un cop s'ha inicialitzat l'objecte de la classe `TwitterProcess` ja està establerta la connexió amb l'API de Twitter i l'objecte `graph` on guardar les dades RDF. Ara cal fer la cerca i processar els resultats. La cerca, com s'ha indicat anteriorment, es fa des del mètode `process_statuses()` de la classe `TwitterProcess`:

```
def process_statuses(self):
    """Recorre els resultats d'una cerca a Twitter i els processa"""
    for status in tweepy.Cursor(self.api.search, q='flickr').items(10):
        self.add_status(status)
```

El mètode fa la cerca cridant el mètode `search` de `Tweepy` (self.api.search), passant els resultats a un cursor i limitant-los a 10 entrades. Per a cada tweet obtingut es crida al mètode `add_status` i es passa el tweet.

El mètode `add_status` processa tots allò referent a `twitter:Tweet` definit en l'ontologia. Primer de tot es crea una variable `subject`, que serà l'objecte tweet, i es defineix la seva URI a partir del camp `id_str` de l'status obtingut en la cerca. Després s'afegeix al graf una tripleta que especifica que l'objecte subject té com a predicat el valor type del namespace RDF i com a objecte el valor Tweet del `namespace ns_twitter`, és a dir, que subject és de tipus `twitter:Tweet`:

```
def add_status(self, status, user=None):
    """Afegeix un status al graph

    Arguments:
        status: objecte de tipus tweepy.Status
        user: usuari que ha generat l'status en cas que l'status no l'incorpori
    Retorna:
        URI de l'status afegit
    """

    # Creem twitter:Tweet amb URI 'tweet_' + status.id_str
    subject = self.graph.ns_twitter['tweet_' + status.id_str]
    self.graph.add((subject, RDF.type, self.graph.ns_twitter.Tweet))
    ...
```

Ara només cal anar afegint tripletes en el graf associades al tweet contingut en subject, amb les dades que hem obtingut a la cerca. Per exemple, les tripletes que descriuen el text i el nombre de retweets d'un tweet es generen així:

```
# twitter:text
self.graph.add((subject, self.graph.ns_twitter.text, Literal(status.text)))

# twitter:retweetCount
self.graph.add((subject, self.graph.ns_twitter.retweetCount, Literal(status.retwee
t_count)))
```

Quan un tweet és retweet, és ha dir s'ha fet un reenviament d'un altre tweet, d'un altre cal crear el tweet original (`add_status`) i associar l'actual amb aquest mitjançant l'object property `isRetweetOf`:

`# twitter:isRetweetOf`

```
retweeted_status = getattr(status, 'retweeted_status', None)
if retweeted_status:
    retweeted = self.add_status(retweeted_status, user)
    self.graph.add((subject, self.graph.ns_twitter.isRetweetOf, retweeted))
```

De la mateixa manera cal crear (`add_user`) l'usuari autor del tweet i associar el tweet i l'usuari amb `hasTweeted` i `isTweetedBy`:

```
# twitter:User
# Si no ens ha arribat usuari cal crear-lo
if not user:
    user = self.add_user(status.user.id_str)

# twitter:isTweetedBy i twitter:hasTweeted
self.graph.add((subject, self.graph.ns_twitter.isTweetedBy, user))
self.graph.add((user, self.graph.ns_twitter.hasTweeted, subject))
```

La classe `TwitterProcess` proporciona els següents mètodes per crear entrades al graf:

- `add_status`
- `add_user`
- `add_place`
- `add_bounding_box`
- `add_point`
- `add_coordinate`
- `add_hashtag`
- `add_url`
- `add_media`
- `add_size`

El codi va recorrent totes les dades i associacions i cridant aquests mètodes per omplir el graf d'RDF.

La majoria de recursos es poden identificar amb una URI però en alguns casos no existeix un identificador i no té sentit crear-lo artificialment. És el cas de, per exemple, `twitter:Point`, que només té latitud i longitud. Quan no es disposa d'URI per definir un recurs es pot crear a partir d'un `BNode` de la llibreria `RDFLib`. A aquest `BNode` se li assigna un identificador únic de manera automàtica però cal tenir en compte que és local a cada RDF que es genera. És a dir, dos recursos creats a partir de `BNode` en diferents grafs podrien tenir el mateix identificador encara que no siguin, necessàriament, el mateix recurs:

```
def add_point(self, point):
    """Afegeix un punt al graph

    Arguments:
        point: llista [long, lat]
    Retorna:
        URI del punt afegit
    """
    # twitter:point
    # Fem un node en blanc de tipus w3cgeo:Point
    subject = BNode()
    self.graph.add((subject, RDF.type, self.graph.ns_w3cgeo.Point))
    self.graph.add((subject, self.graph.ns_w3cgeo.long, Literal(point[0])))
    self.graph.add((subject, self.graph.ns_w3cgeo.lat, Literal(point[1])))

    return subject
```

A vegades cal comprovar si un objecte ja existeix en el graf per evitar generar múltiples entrades. Una manera de fer-ho és obtenir la URI del recurs que es busca i mirar si en el graf hi ha una tripleta associada, com es fa en el mètode `add_user()` per veure si l'usuari existia, on es busca la tripleta (`usuari, RDF.type, twitter:User`):

```
def add_user(self, user_id_str):
    """Afegeix un usuari al graph

    Arguments:
        user_id_str: id de l'usuari a afegir
    Retorna:
        URI de l'usuari afegit
    """
    # URI de twitter:User
    subject = self.graph.ns_twitter['user_' + user_id_str]

    # Si no existeix en el graph el creem
    if not (subject, RDF.type, self.graph.ns_twitter.User) in self.graph:
        # Obtenim l'usuari de l'API
        user = self.api.get_user(user_id_str)
        # twitter:User
        subject = self.graph.ns_twitter['user_' + user.id_str]
        self.graph.add((subject, RDF.type, self.graph.ns_twitter.User))
        # twitter:name
        self.graph.add((subject, self.graph.ns_twitter.name, Literal(user.name)))

    ...
```

Quan no es pot buscar només una tripleta es pot fer una consulta amb SPARQL. Per veure si, per exemple, un hashtag ja està en el graf no es pot fer mitjançant la URI perquè els hashtag no en tenen (estan creats a partir d'un `BNode`). El què sí es pot fer és buscar un recurs de tipus `twitter:Hashtag` i que tingui la propietat `twitter:text` amb el valor del `hashtag` que es busca:

```
def add_hashtag(self, hashtag):
    """Afegeix un hashtag al graph

    Arguments:
        hashtag: text del hashtag
    Retorna:
        URI del hashtag
    """
    # twitter:hashTag
    # Mirem si existeix amb una query d'SPARQL
    subject = None
    query_str = 'select ?h where {?h rdf:type twitter:Hashtag . ?h twitter:text "'
 + hashtag + '"}'
    qres = self.graph.query(query_str)
    for row in qres:
        subject = BNode(row.h)

    # Si no hem trobat res creem el hashtag
    if not subject:
        # Node en blanc
        subject = BNode()
        self.graph.add((subject, RDF.type, self.graph.ns_twitter.Hashtag))
        self.graph.add((subject, self.graph.ns_twitter.text, Literal(hashtag)))

    return subject
```

Després de processar tots els resultats s'exporta el graf a RDF/XML cridant al mètode `save()` de `TwitterProcess`:

```
def save(self):
    """Desa el graph en format xml"""
    data = self.graph.serialize(format='xml')
    file = open('twitter.rdf', 'w')
    file.write(data)
    file.close()
```

Un exemple d'un `tweet` en RDF/XML seria:

```
<rdf:Description rdf:about="http://www.owl-
ontologies.com/Ontology1396114080.owl#tweet_465102968698974208">
  <twitter:hasUrl rdf:nodeID="Nbdd32234edf6403ea0bc80bbffc5c66d"/>
  <twitter:hasHashtag rdf:nodeID="N41b11898e8d74a7680d134cb8ec1580e"/>
  <twitter:text>RT @19sixty3 : My Home on the Lake http://t.co/sVCoD9nWZa #photogr
aphy #photo #foto #flickr</twitter:text>
  <twitter:mentions rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#user_30615272"/>
  <twitter:isTweetedBy rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#user_288738911"/>
  <twitter:hasHashtag rdf:nodeID="N35eeea658e9f4fe5beaf6b0798c6eeea"/>
  <twitter:hasHashtag rdf:nodeID="Nefed70bee4f048c0acdd55874b45c242"/>
  <twitter:isStatusOf rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#user_288738911"/>
  <twitter:retweetCount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0<
/twitter:retweetCount>
  <twitter:hasHashtag rdf:nodeID="N040839b3c16642709c36829de8600d85"/>
  <rdf:type rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#Tweet"/>
</rdf:Description>
```

Un `usuari`:

```
<rdf:Description rdf:about="http://www.owl-
ontologies.com/Ontology1396114080.owl#user_288738911">
  <twitter:hasStatus rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#tweet_465102968698974208"/>
  <twitter:hasTweeted rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#tweet_465102968698974208"/>
  <rdf:type rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#User"/>
  <twitter:url>None</twitter:url>
  <twitter:description></twitter:description>
  <twitter:screenName>fotoberliner</twitter:screenName>
  <twitter:name>Foto Berlin</twitter:name>
</rdf:Description>
```

Un `hashtag`:

```
<rdf:Description rdf:nodeID="Nefed70bee4f048c0acdd55874b45c242">
  <twitter:text>photography</twitter:text>
  <twitter:isHashtagOf rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#tweet_465102968698974208"/>
  <rdf:type rdf:resource="http://www.owl-
ontologies.com/Ontology1396114080.owl#Hashtag"/>
</rdf:Description>
```

5.3 Requeriments de programari necessari per el funcionament de l'aplicació

Programari i llibreries necessàries per l'execució de l'aplicació:

- Python 2.7.x: http://www.python.org
- PIP (si no s'instal·la per defecte):
  - Descarregar https://raw.github.com/pypa/pip/master/contrib/get-pip.py
  - Executar: python get-pip.py
- Tweepy: pip install tweepy
- RDFLib: pip install rdflib

Execució:

- python twitter.py

Paràmetres:

- Es pot canviar el terme de la cerca en la línia 40 (q='flickr')

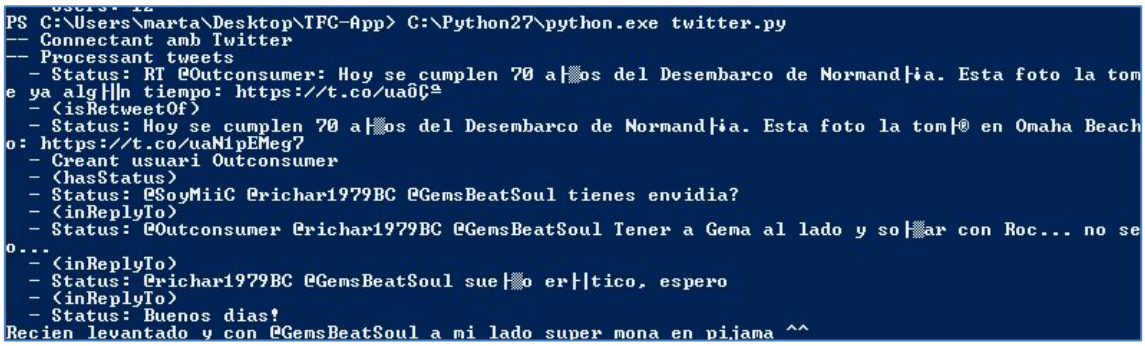
Sortida de resultats de l'execució:

- Fitxer twitter.rdf (RDF/XML)

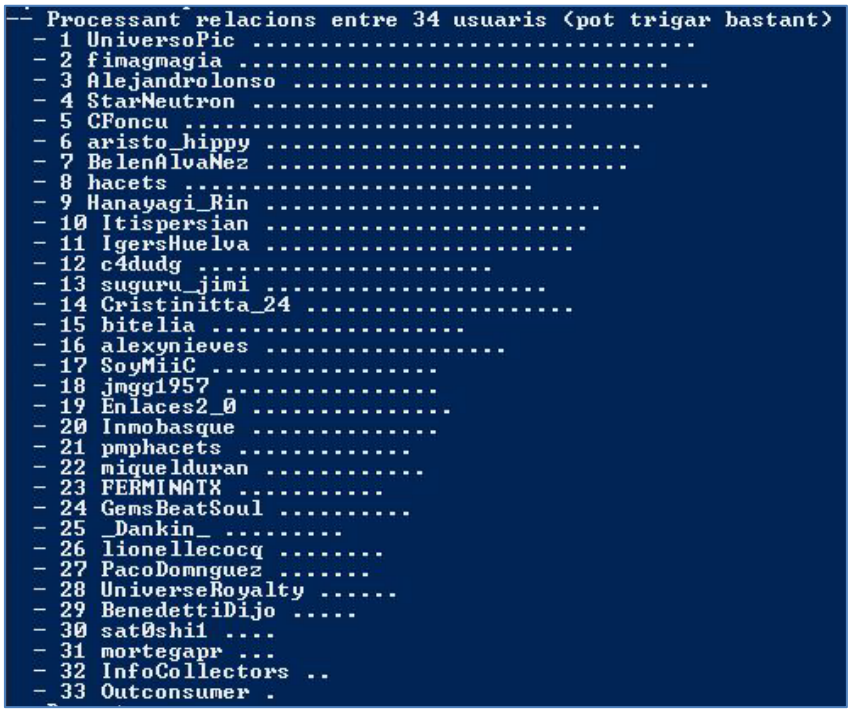
5.4 Resultat de l'execució. Instàncies creades per l'aplicació

Quan s'executa l'aplicació la sortida la pantalla va indicant què s'està fent i el text del què es processa. A continuació es mostren unes pantalles de l'execució de l'aplicació fetes d'exemple i com a resultat es va generar el fitxer twitter.rdf també inclòs a l'entrega.

Inicialment es connecta amb Twitter i comença a processar els tweets. Per a cada tweet, sigui un status, el tweet original d'un retweet o bé un tweet al que es respon, es mostra el text. S'indica també quina propietat de l'RDF es tracta en cada moment (isRetweetOf, hasStatus, etc). Si s'està tractant un usuari, s'indica el nom de l'usuari:



Després dels tweets i els usuaris processa les relacions entre els usuaris. L'algorisme recorre la llista d'usuaris des del principi i per a cadascun d'ells consulta a l'API de Twitter si és friend o follower de tots els següents. Aquesta consulta a l'API està limitada per Twitter: es poden fer 180 consultes inicials i després 15 cada 15 minuts. El programa entra en pausa quan s'esgoten les consultes fins a deixar passar suficient temps per poder tornar a consultar. Així, en funció del nombre d'usuaris, el còmput de les relacions entre ells pot ser bastant llarg, com s'indica en la sortida del programa:



Quan finalitza grava el resultat al fitxer twitter.rdf i fa un resum de nombre de tweets i usuaris creats:

```
-- Desant
-- Resum
  - Tweets: 47
  - Users: 34
PS C:\Users\marta\Desktop\TFC-App>
```

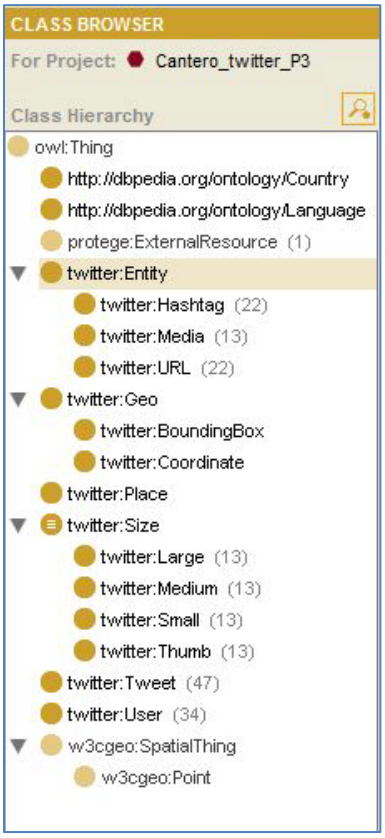
Per millorar la robustesa del programa caldria fer un tractament més extensius dels errors que podem trobar deguts als límits que imposa l'API de Twitter i a casos poc habituals.

Amb l'objectiu de verificar el funcionament de l'aplicació i la seva posterior utilitat a l'hora d'extreure informació de les dades recollides de Twitter s'ha fet una importació del fitxer resultat a Protégé.

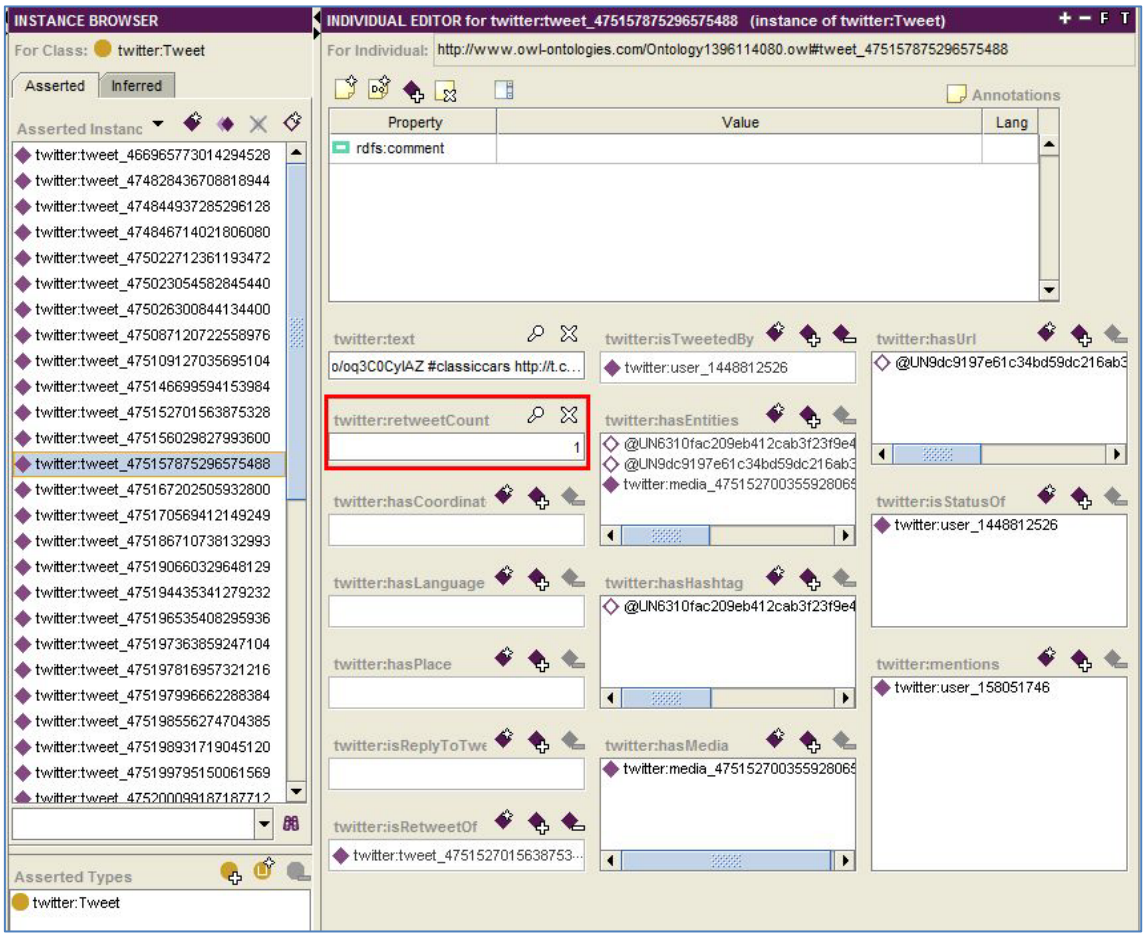
Amb les limitacions imposades, per tal d'evitar grafs excessivament grans que no són l'objectiu d'aquest treball i que dificultarien les tasques de comprensió i verificació dels resultats obtinguts, es genera un fitxer RDF amb 47 tweets, i 34 usuaris (a l'exemple utilitzat per capturar les pantalles, amb la cerca "flickr") línees de instàncies. Així doncs, tot i que d'entrada la restricció de 10 tweets pot semblar un nombre molt reduït de tweets en realitat pot arribar a generar prou informació com per poder explotar-la.

A continuació es mostren algunes pantalles de Protégé amb les instàncies creades per a les classes més destacades de l'ontologia.

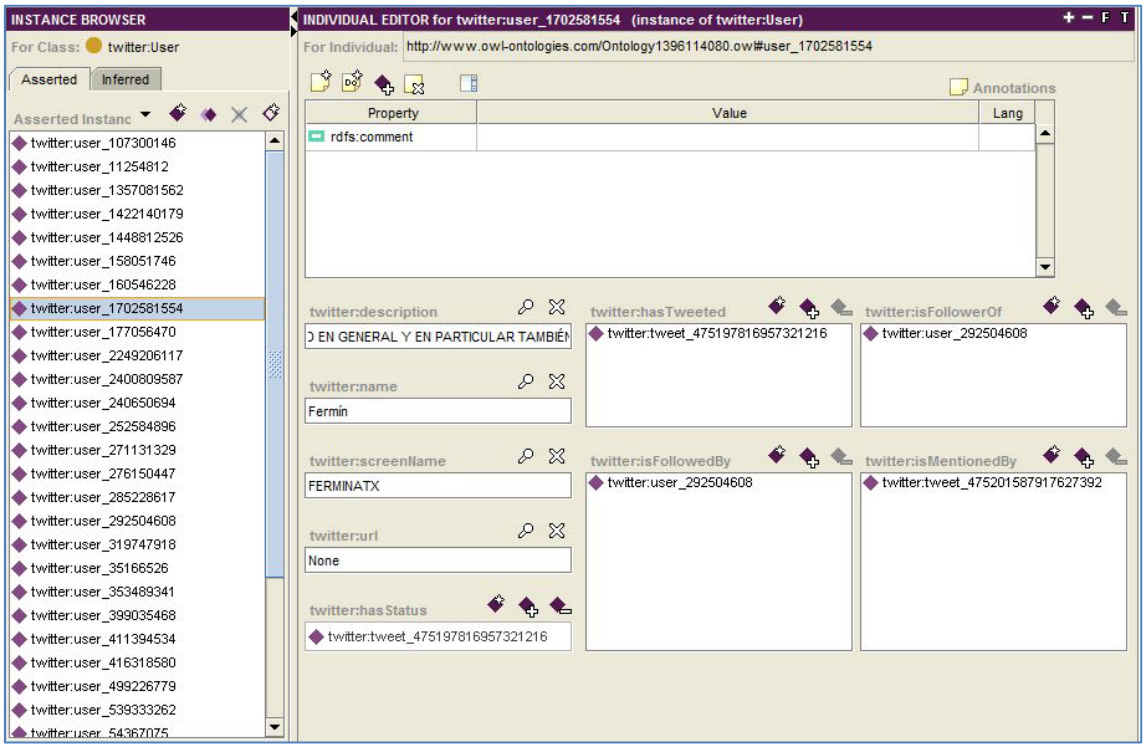
Podem veure el nombre d'instàncies creades per cada classe. I comprovar que tweets (47) i usuaris (34) coincideix amb el resum que ha donat l'aplicació:



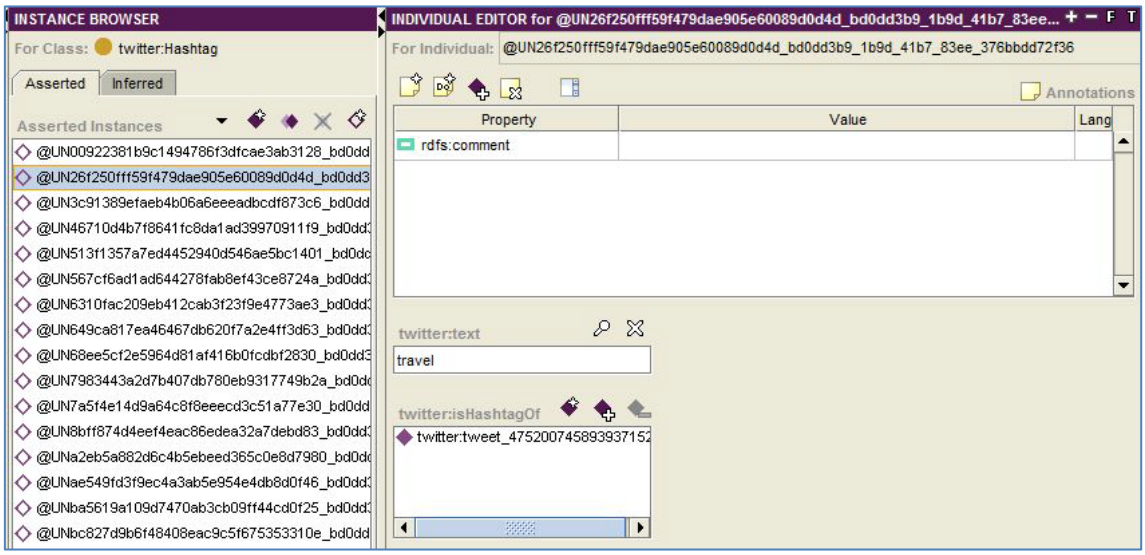
La classe `Tweet`:



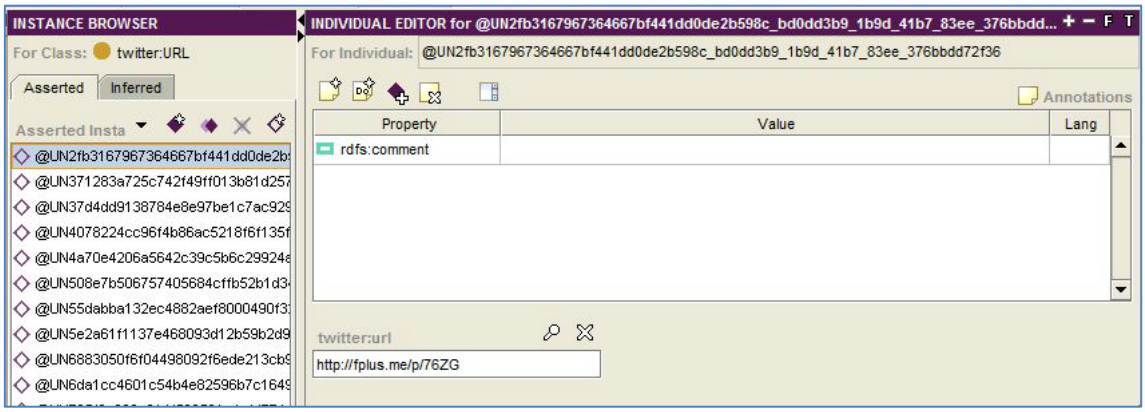
La classe `User`:



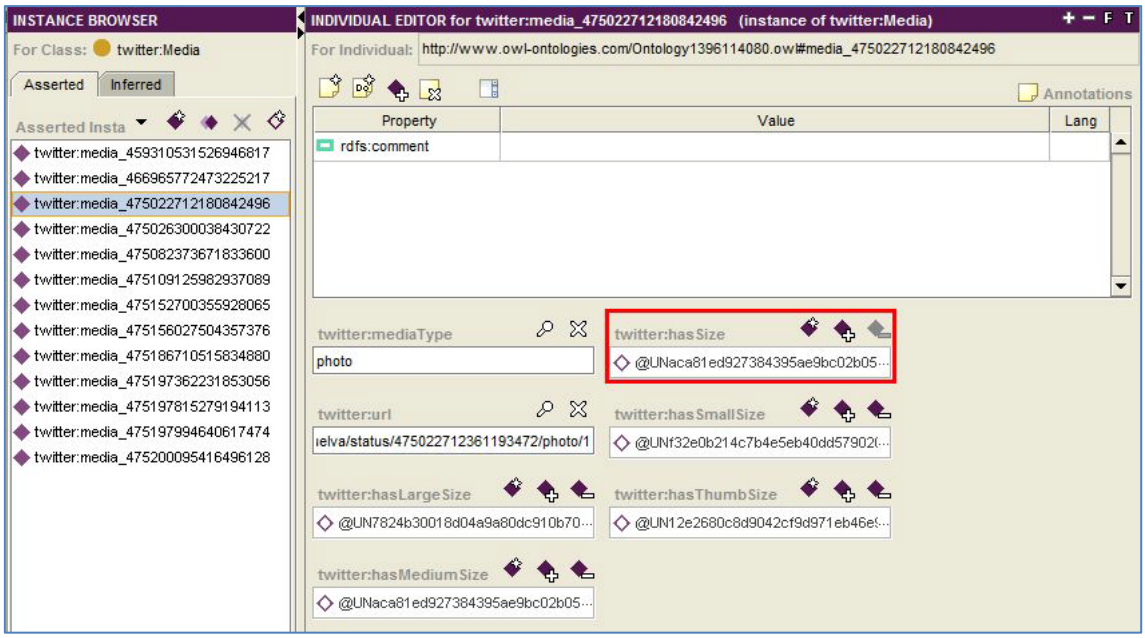
La classe `Hashtag`:



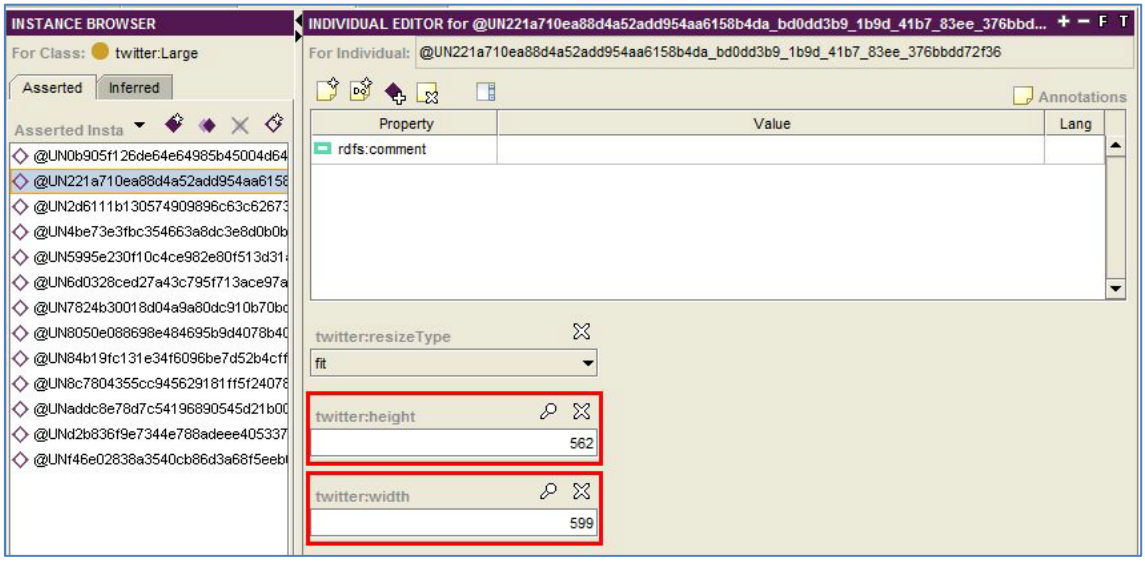
La classe `URL` :



La classe `Media`:



Les classes `Large, Medium, Small i Thumb.` Com que totes són molt similars es mostra una d'elles (Large) a mode d'xemple:



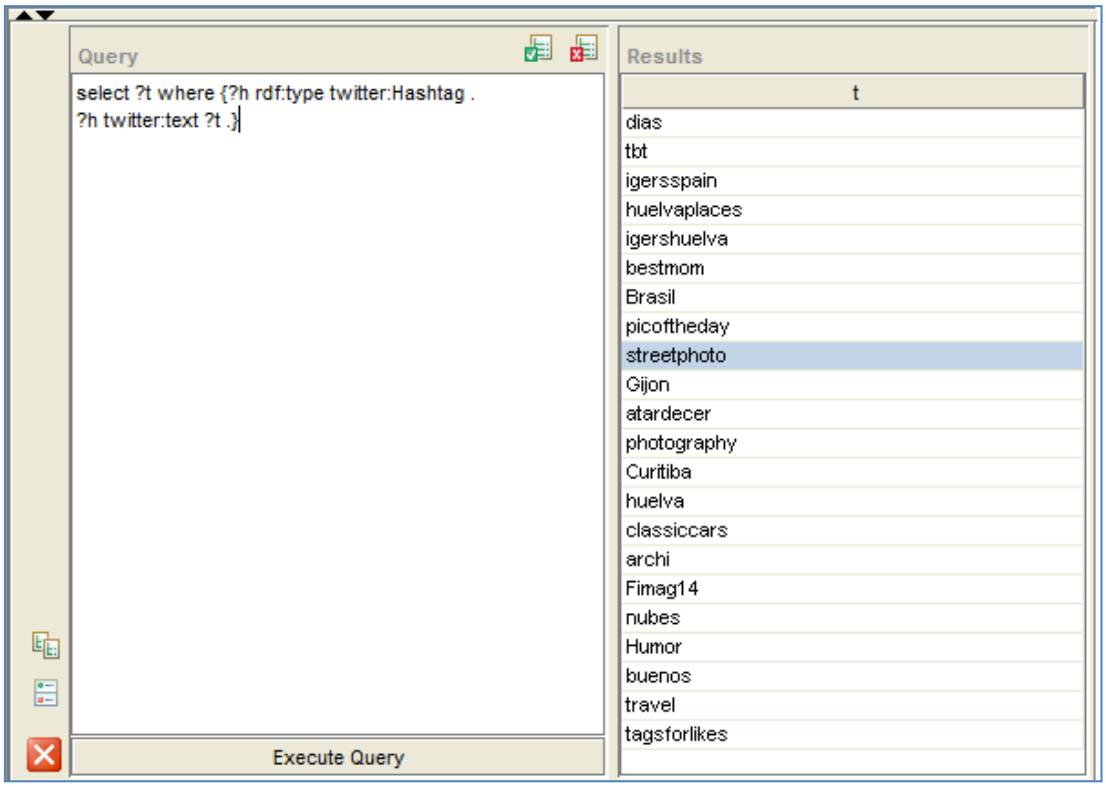
6 Extreure informació del tweets

6.1 Consultes

Un del objectius d'aquest treball, de l'ontologia i l'aplicació és mostrar les possibilitats que donen per inferir informació dels tweets. És per això que s'han realitzat algunes consultes en llenguatge SPARQL en Protégé, com a mostra de les possibilitats d'explotació de les dades recollides de Twitter.

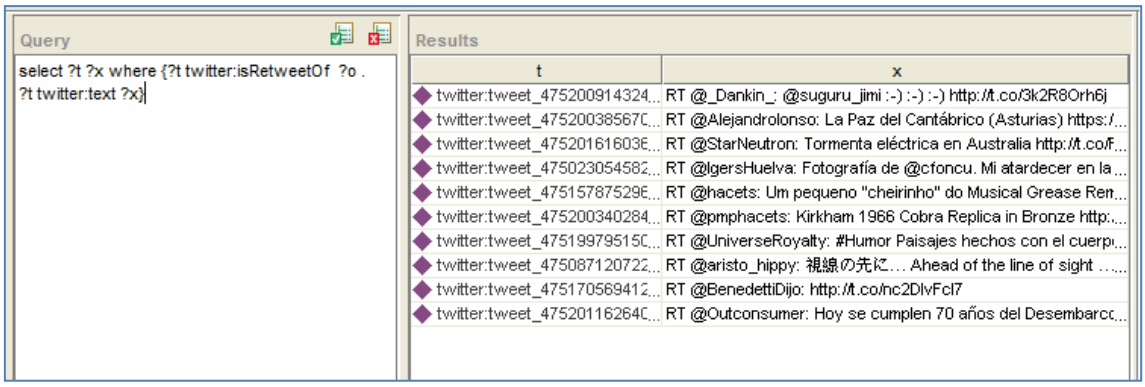
Per exemple, per obtenir tots els textos dels hastags que estaven inclosos als tweets recollits es pot fer una consulta que busqui tots els objectes de tipus `twitter:Hashtag` que tenen algun valor a la propietat `text`:

```
select ?t
where {?h rdf:type twitter:Hashtag .
       ?h twitter:text ?t .}
```



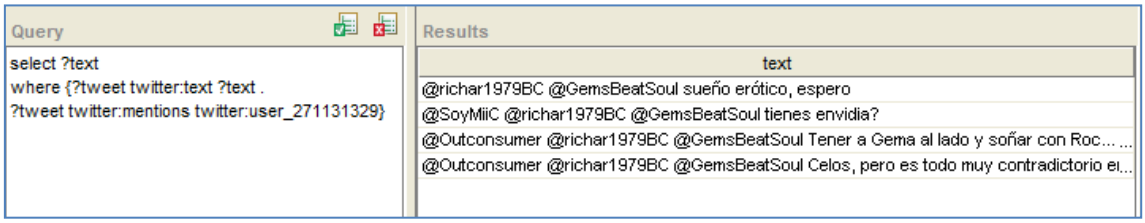
Amb aquesta altra consulta s'obté un llistat dels tweets que són retweet d'un altre i retornarà la URI del tweet i el seu text:

```
select ?t ?x
where {?t twitter:isRetweetOf  ?o .
       ?t twitter:text ?x}
```



La última mostra és una consulta que retorna els textos dels tweets que mencionen un usuari en concret:

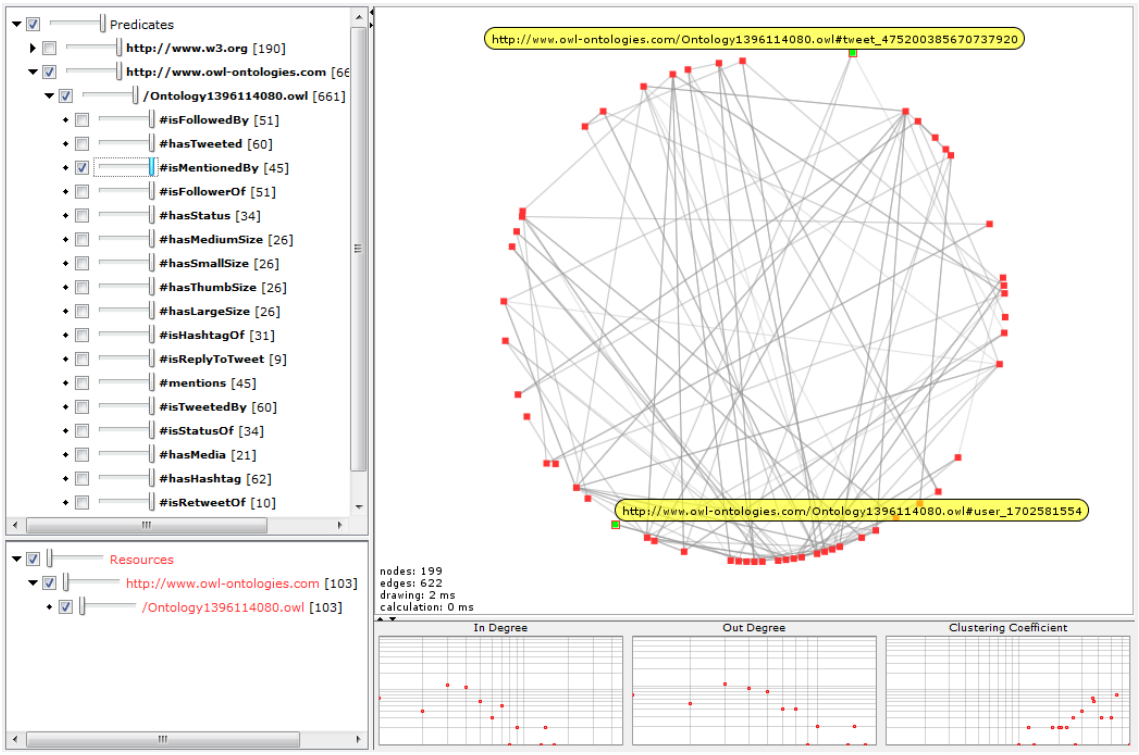
```
select ?text
where {?tweet twitter:text ?text .
?tweet twitter:mentions twitter:user_271131329}
```



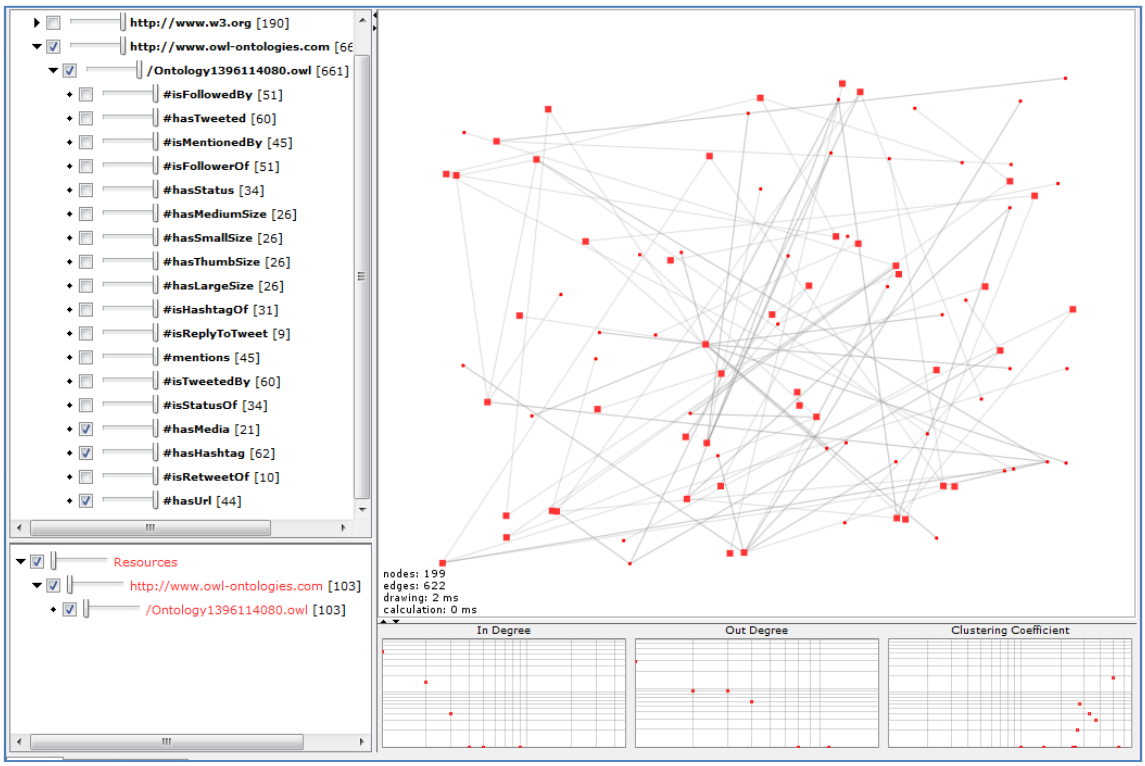
6.2 Altres formes d'inferir informació

Existeixen altres eines que proporcionen altre maneres d'inferir informació a través de les ontologies i les seves instàncies. Un exemple es pot trobar en l'aplicació Welking (http://simile.mit.edu/welkin/) que és un visualitzador d'RDFs. Proporciona informació visual i gràfica sobre les relacions entre instàncies, és a dir les propietats de l'ontologia. Aquesta informació pot ser també de gran interés.

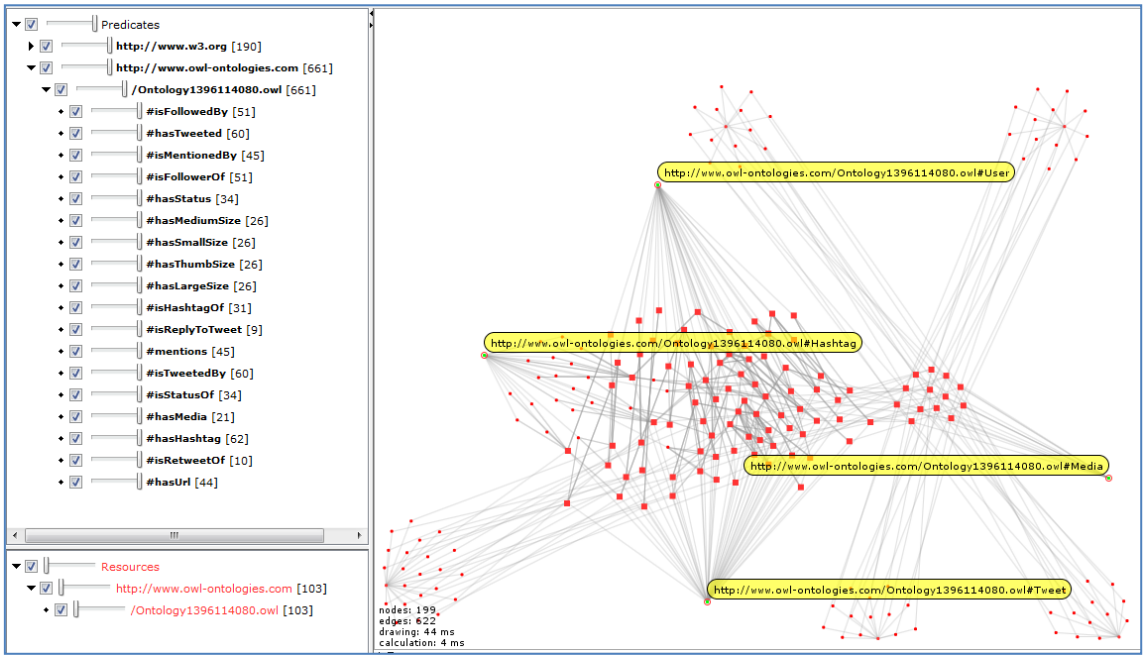
A la pantalla s'ha seleccionat la propietat `isMentionedBy` i per tant al gràfic es mostren els usuaris que han estat mencionats a algun tweet i la relació amb aquest.



Un altre exemple són els tweets que tenen alguna Entity, és ha dir tenen hastags o media o urls:



Aquesta eina també permet generar gràfics dinàmics on els nodes s'agrupen en funció de les relacions entre ells i la quantitat d'aquestes. Això fa que sigui fàcil visualitzar grups d'instàncies similars o que tenen característiques comunes envers d'altres que són diferents. En la següent imatge es veu com els tipus User i Tweet apunten a un gran nombre de nodes:



7 Conclusions

L'assoliment dels objectius generals del projecte ha estat satisfactori. S'ha aprofundit en l'estudi dels conceptes de la web semàntica, així com de les seves estructures i llenguatges de representació, OWL, RDF, SRDF, etc. També s'ha fet una aproximació als llenguatges de consulta de les BD:XML.

Relacionat amb els objectius específics indicats a l'enunciat s'ha creat l'ontologia per emmagatzemar tweets, s'ha creat un programa que permet poblar l'ontologia de dades procedents dels tweets que circulen per Twitter i s'ha fer una aproximació a l'explotació d'aquestes dades mitjançant el llenguatge de consultes SPARQL.

A més s'ha realitzat un treball de recerca per tal d'establir l'estat actual dels projectes vigents i en desenvolupament relacionats amb el tema del projecte. S'ha estudiat els diferents llenguatges de definició d'ontologies i dades OWL, RDF, XML. Així com l'aplicació Protégé, que s'ha utilitzat per definir l'ontologia en OWL, les casses, relacions entre elles i restriccions en funció de l'estructura d'objectes i dades de Twitter. També s'ha estudiat l'estructura de Twitter i l'API que dóna accés a la seva informació.

Referent a l'aplicació que recull les dades i crea les instàncies de l'ontologia s'ha hagut de fer primerament una prospecció dels diferents llenguatges de programació i les seves eines tant a nivell de RDF com de connexió a Twitter. Aquesta prospecció va portar a prendre la decisió de programar en llenguatge Python, del quan no es tenien coneixement i per tant ha calgut un estudi d'aquest llenguatge així com de les llibreries per tractar els grafs de RDF i les connexions amb l'API de Twitter.

Es va considerar reutilitzar varies classes d'ontologies ja existents. Amb èxit per la classe Point, importada de W3C i que s'utilitza l'estructura però que, per la seva naturalesa, es generen les instàncies amb les dades provinents dels tweets. Contràriament es va intentar reutilitzar les classes Language i Country de la DBpedia així com les seves instàncies ja existents, però donat el grau de complicació per realitzar-lo correctament així com el temps disponible per finalitzar el treball s'ha considerat deixar per un futur la implementació d'aquestes dues reutilitzacions. Així les classes estan importades i les relacions amb les altres classes creades però no se'n fa us de les instàncies que ja existeixen a la xarxa.

Una possible línea de futur és la possibilitat que deixa oberta el projecte per enllaçar amb recursos externs algunes de les dades de l'ontologia i així ampliar les possibilitats d'inferència de les dades recollides dels tweets.

8 Glossari

API - Interfície de programació d'aplicacions.

Data Property – defineix la relació entre una classe de l'ontologia i una dada d'un tipus reconegut.

Object Property – defineix una relació entre classes de l'ontologia.

OWL – Web Ontology Language,

Protégé – programari per definir ontologies

Python – llenguatge de programació

RDF – Resource Description Framework,

RDFS - Resource Description Framework Schema

TFC - Treball fi de carrera

Twitter – xarxa social i microblogging, on els usuaris intercanvien missatges amb una longitud màxima de 140 caraters.

9 Bibliografia

Castells, P. (2003). La web semántica. Madrid:
Escuela Politécnica Superior, Universidad Autónoma de Madrid.
Extraídodel World Wide Web:http://arantxa.ii.uam.es/~castells/publications/castells-uclm03.pdf

G. Antoniou; F van Harmelen (2008) *A Semantic Web Primer*. Ed. Cambridge (MA): The MIT Press.

W3C (Consorcio World Wide Web): http://www.w3.org/ (març i abril 2014)

W3C España: http://www.w3c.es (març i abril 2014)

Organización Web Semántica: http://www.semanticweb.org/ (març i abril 2014)

Organización XML: http://www.xml.org/xml/resources_cover.shtml (març i abril 2014)

Sitio web de Protégé: protege.stanford.edu/ (març i abril 2014)

Getting Started with Protégé (2003). Extraído del World Wide Web: http://protege.stanford.edu/doc/tutorial/index.html (març i abril 2014)

http://www.slideshare.net/DocThreeC/diseo-de-ontologas-protg-owl-ejemplo-de-las-pizzas (març i abril 2014)

http://wiki.dbpedia.org/Datasets (març i abril 2014)

http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library (març i abril 2014)

http://semanticweb.com/introduction-to-ontologies_b18705 (març i abril 2014)

http://ivan-herman.name/2010/03/24/twitter-search-results-in-rdf/ (abril 2014)

http://go-to-hellman.blogspot.com.es/2009/04/rdf-and-twitter-compare-and-contrast.html (març i abril 2014)

http://laurensgoessemantic.wordpress.com/2010/11/21/from-rdfxml-to-dynamic-sparql-endpoint/ (maig 2014)

https://dev.twitter.com/docs/twitter-libraries (març i abril 2014)

https://rdflib.readthedocs.org/en/latest/intro_to_parsing.html

https://github.com/RDFLib/rdflib (març i abril 2014)

http://www.liacs.nl/cs/dlt/pickups/Sjoerd/tutorial1.pdf (abril i maig 2014)

http://www.slideshare.net/DocThreeC/diseo-de-ontologas-protg-owl-ejemplo-de-las-pizzas (març i abril 2014)

http://130.88.198.11/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf (març i abril 2014)

http://dbpedia.org/About (març i abril 2014)

http://www.webdatacommons.org (abril i maig 2014)

http://cms.unige.ch/lettres/linguistique/nebhi/wp-content/papercite-data/pdf/beamerColing12.pdf (abril i maig 2014)

http://semanticweb.com/introduction-to-ontologies_b18705 (març i abril 2014)

http://code.google.com/p/ohd-ontology/ (abril i maig 2014)

http://www.w3.org/wiki/Good_Ontologies (març i abril 2014)

http://www.w3.org/TR/owl-guide/ (març i abril 2014)

http://svn.aksw.org/papers/2013/SWJ_DBpedia/public.pdf (abril i maig 2014)

http://blog.3kbo.com/2008/08/11/dbpedia-examples-using-linked-data-and-sparql/comment-page-1/ (maig 2014)

http://stackoverflow.com/questions/17724983/how-can-i-recognize-object-properties-vs-datatype-properties (abril i maig 2014)

10 Annexos

10.1 Annex 1: Entorn de desenvolupament i d'execució

S'ha desenvolupat l'aplicació en un entorn MS Windows Vista de 32 bits. El programari i les llibreries necessàries per al desenvolupament de l'aplicació han estat:

- Python 2.7.x
- Tweepy
- RDFLib

El programa consta d'un únic fitxer i s'ha editat amb Notepad++ (http://notepad-plus-plus.org/), que ofereix resaltat de sintaxi per a Python. Des de línia de comandes s'executa el programa amb l'intèrpret de Python:

- python twitter.py

El resultat de l'execució es guarda en el fitxer twitter.rdf (RDF/XML) que es carrega a un repositori de Sesame per verificar el funcionament correcte del programa. La lectura del fitxer RDF/XML es podria fer amb qualsevol altre programari que accepti fitxer RDF/XML, com ara Protégé.

Tot i que no s'ha provat el programa hauria de poder-se executar en qualsevol entorn que disposi de Python 2.7.x i les llibreries necessàries instal·lades.

El programa controla els límits en el nombre de consultes permeses a l'API de Twitter quan fa la relació entre usuaris. Si el nombre d'usuaris resultat de l'obtenció dels tweets és gran pot ser que calgui molt de temps en processar-los degut a la limitació de 180 consultes inicials i 15 cada 15 minuts en la consulta a l'API referent a les relacions entre usuaris (https://dev.twitter.com/docs/api/1.1/get/friendships/show). També es controla el límit quan es fan cerques d'usuaris en el moment de recuperar els tweets.

Per millorar la robustesa del programa caldria fer un tractament més extensius dels errors que podem trobar deguts als límits que imposa l'API de Twitter i a casos poc habituals.

10.2 Annex 2: Codi font de l'aplicació per poblar l'ontologia

```
"""Executa una cerca en Twitter i desa el resultat en format RDF/XML"""
# -*- coding: utf-8 -*-

import sys
import string
import time
import tweepy
from rdflib.namespace import Namespace
from rdflib import Graph, Literal, RDF, URIRef, BNode

__author__ = "Marta Cantero Rebollo"
__status__ = "Development"

class TwitterGraph(Graph):
    """Graph de RDF"""
    def __init__(self):
        """Inicialitza TwitterGraph amb els Namespaces i els prefix de la ontologia"""
        super(TwitterGraph, self).__init__()
        self.ns_twitter = Namespace('http://www.owl-ontologies.com/Ontology1396114080.owl#')
        self.ns_rdfs = Namespace('http://www.w3.org/2000/01/rdf-schema#')
        self.ns_w3cgeo = Namespace('http://www.w3.org/2003/01/geo/wgs84_pos#')
        self.bind('twitter', self.ns_twitter)
        self.bind('rdfs', self.ns_rdfs)
        self.bind('w3cgeo', self.ns_w3cgeo)


class TwitterProcess:
    """Processa el resultat de la cerca a Twitter i els converteix a RDF"""
    def __init__(self):
        """Connecta amb l'API de Twitter i crea un graph buit"""
        self.connect()
        self.graph = TwitterGraph()

    def connect(self):
        """Connecta amb l'API de Twitter"""
        print "-- Connectant amb Twitter"
        auth = tweepy.OAuthHandler('OKcQhQVcMDfKUwbYxC1QxvoEc', 'wyEXfFTZ9bg8o9gzSnF3hm0mhb75sue50Pe7mkWVb6fCn2rMJK')
        auth.set_access_token('2466251904-mnvxFmJJgyLYeYVy3B6OSHFgMuOESrh4MCoqQmc', 'AzeNOOx5k6Fxs0nD7rc2sArMgzJ2G5kckZZUmyV5DVYq2')
        self.api = tweepy.API(auth)

    def process_statuses(self):
        """Recorre els resultats d'una cerca a Twitter i els processa"""
        print '-- Processant tweets'
        for status in tweepy.Cursor(self.api.search, q='flickr', lang='es').items(10):

            self.add_status(status)

    def process_relationships(self):
        """Relaciona les entrades de tipus twitter:User entre elles"""
        # Obtenim la llista d'users del graph
        users = [user for user in self.graph.subjects(RDF.type, self.graph.ns_twitter.User)]
        n = len(users)
        current_user = 0
        if n > 1:
            # Per a cada usuari de la llista excepte el darrer
            print '--
 Processant relacions entre ' + str(n) + ' usuaris (pot trigar bastant)'
            for user in users[:n-1]:
                current_user += 1
                user_screen_name = self.graph.value(user, self.graph.ns_twitter.screenName)
                print '  - ' + str(current_user) + ' ' + user_screen_name + " ",
                # Recorrem la resta de la llista
```

```
61.                 for other_user in users[users.index(user) + 1:]:
62.                     other_user_screen_name = self.graph.value(other_user, self.graph.ns_
    twitter.screenName)
63.                     sys.stdout.write('.')
64.                     sys.stdout.flush()
65.
66.                     # Aturem per no saturar l'API
67.                     while True:
68.                         try:
69.                             # Consultem a l'API la seva relaciÃ³
70.                             friendship = self.api.show_friendship(source_screen_name = u
    ser_screen_name, target_screen_name = other_user_screen_name)
71.                             break
72.                         except:
73.                             time.sleep(60)
74.
75.                     # Following
76.                     if friendship[0].following:
77.                         # isFollowerOf i isFollowedBy
78.                         self.graph.add((user, self.graph.ns_twitter.isFollowerOf, other_
    user))
79.                         self.graph.add((other_user, self.graph.ns_twitter.isFollowedBy,
    user))
80.                     # Follower
81.                     if friendship[0].followed_by:
82.                         # isFollowerOf i isFollowedBy
83.                         self.graph.add((other_user, self.graph.ns_twitter.isFollowerOf,
    user))
84.                         self.graph.add((user, self.graph.ns_twitter.isFollowedBy, other_
    user))
85.                 print
86.
87.     def add_status(self, status, user=None):
88.         """Afegeix un status al graph
89.
90.         Arguments:
91.             status: objecte de tipus tweepy.Status
92.             user: usuari que ha generat l'status en cas que l'status no l'incorpori
93.         Retorna:
94.             URI de l'status afegit
95.         """
96.         print "  - Status: " + status.text.encode('utf-8')
97.
98.         # Creem twitter:Tweet amb URI 'tweet_' + status.id_str
99.         subject = self.graph.ns_twitter['tweet_' + status.id_str]
100.        self.graph.add((subject, RDF.type, self.graph.ns_twitter.Tweet))
101.
102.        # twitter:text
103.        self.graph.add((subject, self.graph.ns_twitter.text, Literal(status.text)))
104.
105.        # twitter:retweetCount
106.        self.graph.add((subject, self.graph.ns_twitter.retweetCount, Literal(status.retw
    eet_count)))
107.
108.        # twitter:isReplyToTweet
109.        in_reply_to_status_id_str = getattr(status, 'in_reply_to_status_id_str', None)
110.        if in_reply_to_status_id_str:
111.            # Mirem si el tweet al que es fa reply ï¿½s pï¿½blic
112.            # Si ï¿½s privat apuntem a un BNode (en blanc)
113.            try:
114.                replied_status = self.api.get_status(in_reply_to_status_id_str)
115.                print '  - (inReplyTo)'
116.                replied = self.add_status(replied_status, user)
117.            except:
118.                replied = BNode()
119.            self.graph.add((subject, self.graph.ns_twitter.isReplyToTweet, replied))
120.
121.        # twitter:isRetweetOf
122.        retweeted_status = getattr(status, 'retweeted_status', None)
123.        if retweeted_status:
124.            print '  - (isRetweetOf)'
125.            retweeted = self.add_status(retweeted_status, user)
126.            self.graph.add((subject, self.graph.ns_twitter.isRetweetOf, retweeted))
```

```
127.
128.         # twitter:hasPlace i twitter:isPlaceOf
129.         place_str = getattr(status, 'place', None)
130.         if place_str:
131.             place = self.add_place(place_str)
132.             self.graph.add((subject, self.graph.ns_twitter.hasPlace, place))
133.             self.graph.add((place, self.graph.ns_twitter.isPlaceOf, subject))
134.
135.         # twitter:hasCoordinates
136.         coordinates_str = getattr(status, 'coordinates', None)
137.         if coordinates_str:
138.             coordinate = self.add_coordinate(coordinates_str['coordinates'])
139.             self.graph.add((subject, self.graph.ns_twitter.hasCoordinate, coordinate))
140.
141.         # twitter:User
142.         # Si no ens ha arribat usuari cal crear-lo
143.         if not user:
144.             user = self.add_user(status.user.id_str)
145.
146.         # twitter:isTweetedBy i twitter:hasTweeted
147.         self.graph.add((subject, self.graph.ns_twitter.isTweetedBy, user))
148.         self.graph.add((user, self.graph.ns_twitter.hasTweeted, subject))
149.
150.         # Entities
151.         # Obtenim totes les entities de l'status
152.         entities = status.entities
153.
154.         # Entities: twitter:hashtag
155.         if 'hashtags' in entities.keys():
156.             # Per a cada hashtag fem twitter:hasHashtag i twitter:isHashtagOf
157.             for hashtag_str in entities['hashtags']:
158.                 hashtag = self.add_hashtag(hashtag_str['text'])
159.                 self.graph.add((subject, self.graph.ns_twitter.hasHashtag, hashtag))
160.                 self.graph.add((hashtag, self.graph.ns_twitter.isHashtagOf, subject))
161.
162.         # Entities: twitter:url
163.         # Mirem primer si el tweet ja tï¿½ hasUrl d'abans
164.         # Com que les Url no tenen URI evitem afegir mï¿½s hasUrl quan es tracta d'un st
    atus d'usuari
165.         if not (subject, self.graph.ns_twitter.hasUrl, None) in self.graph:
166.             if 'urls' in entities.keys():
167.                 # twitter:hasUrl
168.                 for url_str in entities['urls']:
169.                     url = self.add_url(url_str['expanded_url'])
170.                     self.graph.add((subject, self.graph.ns_twitter.hasUrl, url))
171.
172.         # Entities: mentions
173.         if 'user_mentions' in entities.keys():
174.             # Per a cada menciÃ³ afegim twitter:User, twitter:mentions i twitter:isMenti
    onedBy
175.             for mention in entities['user_mentions']:
176.                 user = self.add_user(mention['id_str'])
177.                 self.graph.add((subject, self.graph.ns_twitter.mentions, user))
178.                 self.graph.add((user, self.graph.ns_twitter.isMentionedBy, subject))
179.
180.         # Entities: media
181.         # Mirem si el tweet ja tï¿½ hasMedia i si en tï¿½ no ho tornem a fer
182.         if not (subject, self.graph.ns_twitter.hasMedia, None) in self.graph:
183.             if 'media' in entities.keys():
184.                 # Afegim cada twitter:Media
185.                 for media_str in entities['media']:
186.                     media = self.add_media(media_str)
187.                     self.graph.add((subject, self.graph.ns_twitter.hasMedia, media))
188.
189.         return subject
190.
191.     def add_user(self, user_id_str):
192.         """Afegeix un usuari al graph
193.
194.         Arguments:
195.             user_id_str: id de l'usuari a afegir
196.         Retorna:
197.             URI de l'usuari afegit
```

```
198.         """
199.         # URI de twitter:User
200.         subject = self.graph.ns_twitter['user_' + user_id_str]
201.
202.         # Si no existeix en el graph el creem
203.         if not (subject, RDF.type, self.graph.ns_twitter.User) in self.graph:
204.             # Obtenim l'usuari i controlem que no saturem l'API
205.             while True:
206.                 try:
207.                     user = self.api.get_user(user_id_str)
208.                     break
209.                 except:
210.                     time.sleep(60)
211.
212.             print '  - Creant usuari ' + user.name.encode("utf-8")
213.             # twitter:User
214.             subject = self.graph.ns_twitter['user_' + user.id_str]
215.             self.graph.add((subject, RDF.type, self.graph.ns_twitter.User))
216.             # twitter:name
217.             self.graph.add((subject, self.graph.ns_twitter.name, Literal(user.name)))
218.             # twitter:screenName
219.             self.graph.add((subject, self.graph.ns_twitter.screenName, Literal(user.scre
    en_name)))
220.             # twitter:description
221.             self.graph.add((subject, self.graph.ns_twitter.description, Literal(user.des
    cription)))
222.             # twitter:url
223.             self.graph.add((subject, self.graph.ns_twitter.url, Literal(user.url)))
224.
225.             # Status de l'usuari
226.             status_str = getattr(user, 'status', None)
227.             if status_str:
228.                 # Alta d'status a partir de l'objecte User.
229.                 # Tweepy no inclou l'user en l'status. El passem.
230.                 print '  - (hasStatus)'
231.                 status = self.add_status(status_str, subject)
232.                 # twitter:isStatusOf i twitter:hasStatus
233.                 self.graph.add((status, self.graph.ns_twitter.isStatusOf, subject))
234.                 self.graph.add((subject, self.graph.ns_twitter.hasStatus, status))
235.
236.         return subject
237.
238.     def add_place(self, place):
239.         """Afegeix una place al graph
240.
241.         Arguments:
242.             place: objecte de tipus tweepy.Place
243.         Retorna:
244.             URI de la place afegida
245.         """
246.         # twitter:Place
247.         subject = self.graph.ns_twitter['place_' + place.id]
248.         self.graph.add((subject, RDF.type, self.graph.ns_twitter.Place))
249.         # twitter:name
250.         self.graph.add((subject, self.graph.ns_twitter.name, Literal(place.name)))
251.         # twitter:placeType
252.         self.graph.add((subject, self.graph.ns_twitter.placeType, Literal(place.place_ty
    pe)))
253.         # twitter:url
254.         self.graph.add((subject, self.graph.ns_twitter.url, Literal(place.url)))
255.         # twitter:hasBoundingBox
256.         # Fem la URI de BoundingBox amb l'id de place
257.         # Mirem primer si existeix
258.         bounding_box = self.graph.ns_twitter['boundingbox_' + place.id]
259.         # Si no, la creem
260.         if not (bounding_box, RDF.type, self.graph.ns_twitter.BoundingBox) in self.graph
    :
261.             bounding_box = self.add_bounding_box(place.bounding_box, place.id)
262.         # twitter:hasBoundingBox
263.         self.graph.add((subject, self.graph.ns_twitter.hasBoundingBox, bounding_box))
264.
265.         return subject
266.
```

```
267.     def add_bounding_box(self, bounding_box, place_id):
268.         """Afegeix una boundingbox al graph
269. 
270.         Arguments:
271.             bounding_box: objecte de tipus tweepy.BoundingBox
272.         Retorna:
273.             URI de la boundingbox afegida
274.         """
275.         # Fem la URI amb l'id de place
276.         subject = self.graph.ns_twitter['boundingbox_' + place_id]
277.         self.graph.add((subject, RDF.type, self.graph.ns_twitter.BoundingBox))
278.         # Afegim cada punt de la boundingbox
279.         for p in bounding_box.coordinates[0]:
280.             point = self.add_point(p)
281.             # twitter:hasPoint
282.             self.graph.add((subject, self.graph.ns_twitter.hasPoint, point))
283. 
284.         return subject
285. 
286.     def add_point(self, point):
287.         """Afegeix un punt al graph
288. 
289.         Arguments:
290.             point: llista [long, lat]
291.         Retorna:
292.             URI del punt afegit
293.         """
294.         # twitter:point
295.         # Fem un node en blanc de tipus w3cgeo:Point
296.         subject = BNode()
297.         self.graph.add((subject, RDF.type, self.graph.ns_w3cgeo.Point))
298.         self.graph.add((subject, self.graph.ns_w3cgeo.long, Literal(point[0])))
299.         self.graph.add((subject, self.graph.ns_w3cgeo.lat, Literal(point[1])))
300. 
301.         return subject
302. 
303.     def add_coordinate(self, coordinate):
304.         """Afegeix una coordenada al graph
305. 
306.         Arguments:
307.             coordinate: camp coordinates de l'status
308.         Retorna:
309.             URI de la coordenada
310.         """
311.         # twitter:coordinate
312.         # Fem un node en blanc
313.         subject = BNode()
314.         self.graph.add((subject, RDF.type, self.graph.ns_twitter.Coordinate))
315.         # Creem el point associat
316.         point = self.add_point(coordinate)
317.         # twitter:hasPoint
318.         self.graph.add((subject, self.graph.ns_twitter.hasPoint, point))
319. 
320.         return subject
321. 
322.     def add_hashtag(self, hashtag):
323.         """Afegeix un hashtag al graph
324. 
325.         Arguments:
326.             hashtag: text del hashtag
327.         Retorna:
328.             URI del hashtag
329.         """
330.         # twitter:hashTag
331.         # Mirem si existeix amb una query d'SPARQL
332.         subject = None
333.         query_str = 'select ?h where {?h rdf:type twitter:Hashtag . ?h twitter:text "' +
    hashtag + '"}'
334.         qres = self.graph.query(query_str)
335.         for row in qres:
336.             subject = BNode(row.h)
337. 
338.         # Si no hem trobat res creem el hashtag
```

```
339.         if not subject:
340.             # Node en blanc
341.             subject = BNode()
342.             self.graph.add((subject, RDF.type, self.graph.ns_twitter.Hashtag))
343.             self.graph.add((subject, self.graph.ns_twitter.text, Literal(hashtag)))
344.
345.         return subject
346.
347.     def add_url(self, url):
348.         """Afegeix una URL al graph
349.
350.         Arguments:
351.             url: string amb la URL
352.         Retorna:
353.             URI de la URL
354.         """
355.         # twitter:Url
356.         # Node en blanc
357.         subject = BNode()
358.         self.graph.add((subject, RDF.type, self.graph.ns_twitter.URL))
359.         self.graph.add((subject, self.graph.ns_twitter.url, Literal(url)))
360.
361.         return subject
362.
363.     def add_media(self, media):
364.         """Afegeix un media al graph
365.
366.         Arguments:
367.             media: camp media de l'status
368.         Retorna:
369.             URI del media
370.         """
371.         # Media
372.         # Mirem si existeix
373.         subject = self.graph.ns_twitter['media_' + str(media['id'])]
374.         # Si no existeix afegim twitter:Media
375.         if not (subject, RDF.type, self.graph.ns_twitter.Media) in self.graph:
376.             self.graph.add((subject, RDF.type, self.graph.ns_twitter.Media))
377.             self.graph.add((subject, self.graph.ns_twitter.mediaType, Literal(media['typ
    e'])))
378.             self.graph.add((subject, self.graph.ns_twitter.url, Literal(media['expanded_
    url'])))
379.             # Afegim cada size que apareix en twitter:Media
380.             for size_str in media['sizes'].keys():
381.                 size = self.add_size(size_str, media['sizes'][size_str])
382.                 self.graph.add((subject, self.graph.ns_twitter['has' + string.capitalize
    (size_str) + 'Size'], size))
383.
384.         return subject
385.
386.     def add_size(self, size_type, size):
387.         """Afegeix un size al graph
388.
389.         Arguments:
390.             size_type: tipus de size ('Large', 'Medium', 'Small', 'Thumb')
391.             size: diccionari amb els detalls del size
392.         Retorna:
393.             URI del size
394.         """
395.         # Afegim un node en blanc
396.         subject = BNode()
397.         self.graph.add((subject, RDF.type, self.graph.ns_twitter[string.capitalize(size_
    type)]))
398.         self.graph.add((subject, self.graph.ns_twitter.height, Literal(size['h'])))
399.         self.graph.add((subject, self.graph.ns_twitter.width, Literal(size['w'])))
400.         self.graph.add((subject, self.graph.ns_twitter.resizeType, Literal(size['resize'
    ])))
401.
402.         return subject
403.
404.     def save(self):
405.         """Desa el graph en format xml"""
406.         print '-- Desant'
```

```
407.         data = self.graph.serialize(format='xml')
408.         file = open('twitter.rdf', 'w')
409.         file.write(data)
410.         file.close()
411.         print '-- Resum'
412.         query_str = 'select ?h where {?h rdf:type twitter:Tweet}'
413.         qres = self.graph.query(query_str)
414.         print '  - Tweets: ' + str(len(qres))
415.         query_str = 'select ?h where {?h rdf:type twitter:User}'
416.         qres = self.graph.query(query_str)
417.         print '  - Users: ' + str(len(qres))
418.
419.
420.if __name__ == '__main__':
421.
422.    # Instancia la classe que farÃ  el procÃ©s
423.    twitter_process = TwitterProcess()
424.    # Processa els tweets
425.    twitter_process.process_statuses()
426.    # Processa les relacions entre usuaris
427.    twitter_process.process_relationships()
428.    # Desa el graph
429.    twitter_process.save()
```