ONTOLOGIA DE TWITTER: EL PRIMER PAS PER INFERIR INFORMACIÓ DELS TWEETS

Esaú González Maclina
Enginyeria en Informàtica de Gestió

Consultor:
Joan Anton Pérez Braña

10/06/2014

FITXA DEL TREBALL FINAL

Títol del treball:	ONTOLOGIA DE TWITTER: EL PRIMER PAS PER INFERIR INFORMACIÓ DELS TWEETS
Nom de l'autor:	Esaú González Maclina
Nom del consultor:	Joan Anton Pérez Braña
Data de lliurament (mm/aaaa):	06/2014
Àrea del Treball Final:	XML i Web Semàntica
Titulació:	Eng. Informàtica de Gestió

Resum del Treball (màxim 250 paraules):

En aquest treball es realitza un estudi sobre l'"estat de l'art" de la web semàntica i els seus estàndards actuals, més concretament sobre ontologies.

Descriu també el proces pràctic emprat pel disseny i la implementació d'una ontologia en el domini concret de *Twitter*, en format *OWL,* fent servir l'aplicació *Protégé* per la seva creació.

Finalment explica la creació (captura de requeriments, disseny i implementació) d'una aplicació capaç d'obtenir dades reals de *Twitter*, processar-les per extreure'n la informació rellevant i emmagatzemar-la a la ontologia creada. Aquesta informació podrà ser consultada i explotada.

Aquest treball es el pas previ a la inferència de dades, no es contempla la creació de regles d'inferència sobre la ontologia ni de cap programa que ho faci.

Abstract (in English, 250 words or less):

This paper is a study on the "state of art" of the Semantic Web and its standards, specifically on ontologies.

It also describes the process used for the design and implementation of an ontology, specifically in Twitter domain, using OWL, and built with the Protégé application.

Finally, explains the creation (requirements capture, design and implementation) of an application able to get real data from Twitter, process it extracting relevant information and store it in the ontology created. This information can be accessed and exploited.

This work is only the previous step to inference data, isn't contemplated in it the creation of inference rules on the ontology nor any program able to do it.

Paraules clau (entre 4 i 8):

XML, OWL, RDF, Twitter, ontologia

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

Avui dia l'ús d'internet i la *World Wide Web* és molt estès, tant a la feina com en el nostre oci i vida privada (tant que cada cop és més pública). S'ha tornat inevitablement en una eina imprescindible en el nostre dia a dia.
La *Web* ha evolucionat molt des que Tim Berners-Lee [1] va fer la primera definició i després, amb l'ajuda de Robert Cailliau [2], va combinar Internet amb el llenguatge HTML (*HiperText Markup Language*) per facilitar la comunicació i la distribució d'informació de forma estructurada a distància. Abans de Berners-Lee, la informació disponible a Internet era poc més que una amalgama d'imatges i text accessibles només amb uns navegadors primitius que tan sols mostraven text. L'aparició del HTML va permetre la estructuració de la informació a nivell de presentació, fent el contingut de les pàgines visualment atractiu pels usuaris i provocant la substitució dels antics navegadors textuals pels nous navegadors gràfics, punt d'arrencada de la web com la coneixem ara.

Aquesta primera versió de la Web era estàtica, només el creador de la pàgina podia aportar informació, la resta d'usuaris només podia consultar-la. Limitació, aquesta, que es va fer evident degut al gran creixement del número d'usuaris, va impulsar el següent salt evolutiu: la dinamització. La informació mostrada es mostra de forma dinàmica fent servir contingut emmagatzemat en bases de dades amb una mínima interacció amb l'usuari, que podia aportar continguts fent servir formularis.

El següent salt evolutiu te més a veure amb l'àmbit d'ús de la Web que amb especificacions tecnològiques: neix la Web social (o Web 2.0). Evolució que permet que els usuaris siguin alhora consumidors i productors de continguts, fomenta un entorn de col·laboració i compartició d'informació en diferents formats (text, imatge, àudio, vídeo...) i en diferents entorns (xarxes socials, blocs...) [3].

Amb l'objectiu de poder processar de forma automàtica la informació continguda a la Web, clarament dirigida a consumidors humans, s'està donant el següent pas endavant en l'evolució d'aquest mitjà: la web semàntica. Aquest canvi no implica cap canvi tecnològic, simplement es tracta de marcar la informació rellevant de la Web amb etiquetes que la identifiquen com a rellevant i que un agent (un programa) sap entendre [4]. La Web Semàntica, doncs, aporta estructura al contingut significatiu de la Web, creant un entorn on agents de programari navegant de pàgina a pàgina seran capaços de fer tasques sofisticades pels usuaris.

Com hem comentat abans, l'ús de la Web es tan extens que avui dia no es concep una empresa, professional, persona en general, artista, ens... sense presència a la Web. Com tampoc no es poden deixar de costat les opinions i comentaris dels usuaris de la Web sobre qualsevol projecte de qualsevol tipus; sigui una empresa per exemple, no podem obviar comentaris negatius de clients insatisfets dins les xarxes socials, cada cop pesen més les opinions d'altres usuaris sobre la qualitat d'un producte o servei a l'hora de contractar-lo o consumir-lo. Les empreses, cada cop més, coneixen aquesta realitat i l'usen pel seu profit.

L'altre cara d'aquesta arma de doble fil que es la gestió de la reputació en línia es que degut al enorme volum d'opinions i de llocs on es poden expressar es molt difícil i costós mantenir una persona o grup de persones que s'encarreguin d'aquesta tasca manualment. Per atansar aquest problema sense caure en l'intent, necessitem que les màquines, els agents de programari que comentàvem abans, siguin capaços de reconèixer, i classificar acuradament en positives o negatives, aquestes opinions vessades a la xarxa.

Twitter [5], un servei de *microblogging* (permet als seus usuaris publicar missatges breus) amb més de 200 milions d'usuaris, més de 65 milions de publicacions (piulades) i més de 800.000 peticions, al dia, es un gran exemple de la influencia que opinions vessades a la xarxa poden afectar de forma positiva o negativa a la reputació en línia. En definitiva, si l'usuari en qüestió te molts seguidors que llegeixen els seus comentaris pot ser molt capaç de fer perdre molts punts de reputació (que significa perdre clients si parlem d'empreses, admiradors si parlem d'artistes, vots si parlem de polítics, etc.) o, per contra, por fer pujar com l'escuma si el comentari es positiu. S'arriba inclús a pagar (o premiar d'alguna forma) per part d'algunes empreses a certs usuaris de *Twitter* amb una gran influencia a la Web, perquè en parlin be.

Coneixent aixo, es molt clara la necessitat de ser capaços de recopilar aquesta informació tan valuosa de forma automàtica, degut al seu volum monstruós, i processada després per poder extreure el coneixement clau al domini que ens ocupi i, si es necessari, actuar en conseqüència.

1.2 Objectius del Treball

- Conèixer l'"estat de l'art" (treballs en curs, tecnologies més punteres, tendències, ...) de la web semàntica, concretament de les ontologies.
- Aprendre a fer servir RDF i XML per la representació de la informació.
- Aprendre i aplicar el llenguatge OWL, juntament amb RDF i XML, a la creació d'ontologies.
- Aprendre a utilitzar aplicacions per crear ontologies, com *Protégé*.
- Estudiar el model de la API de *Twitter*, les seves entitats i relacions rellevants.
- Analitzar la possible reutilització d'ontologies existents.
- Crear, si cal, una ontologia per emmagatzemar la informació rellevant continguda a les piulades, com les entitats que s'hi interrelacionen.
- Crear un programa que s'alimenti de piulades usant les diferents API de *Twitter* i permeti poblar la ontologia d'instàncies de les classes i les relacions definides.
- Analitzar aquestes instancies de la ontologia amb *Protégé*.

1.3 Enfocament i mètode seguit

Per complir aquests objectius es pot partir de diferents enfocaments: crear un producte nou, adaptar algun producte existent, etc. La experiència ens ha ensenyat que la millor opció es la reutilització de components existents, combinada amb parts noves, per obtenir un producte completament nou i que s'adapti millor les necessitats de cada situació.

En quant als llenguatges disponibles per crear ontologies, el mateix enunciat d'aquest treball ja indica que s'ha de fer servir OWL, per tant el pas següent es investigar l'existència d'alguna ontologia sobre *Twitter* publicada prèviament. Després de fer una primera cerca no s'ha trobat cap que s'adapti a les nostres necessitats. Una segona cerca en profunditat per si se'ns havia escapat alguna cosa ha donat el mateix resultat, per tant hem decidit crear una ontologia adient als objectius des de zero.

Sobre el llenguatge de programació utilitzat per crear l'aplicació no hi ha hagut cap dubte, s'ha escollit *java*. En part per ser el principal llenguatge que l'autor d'aquest treball fa servir en el seu dia a dia, i en part també per ser orientat a objectes, que implica una relació bastant directa amb els conceptes de classe, instància i relacions que trobem en una ontologia.

Aquest llenguatge, *java*, acostuma a reutilitzar components ja creats i de lliure us (llibreries), que han sigut provats per la comunitat *online* i, per tant, suficientment fiables, i amb exemples publicats sobre el seu ús i suport tècnic a la mateixa comunitat. Per aquest motiu, per connectar l'aplicació amb la *API* de *Twitter* i per interactuar amb la ontologia

implementada en *OWL*, s'ha decidit fer servir dues llibreries: *twitter4j* [30] i *owlapi* de *sourceforge* [31] respectivament.

1.5 Breu sumari de productes obtinguts

Com a resultat d'aquest treball s'obtenen 2 productes: una ontologia de *Twitter*, creada en llenguatge *OWL* dins un projecte de l'aplicació *Protégé*, i una aplicació que realitza una cerca a *Twitter* sobre algun tema definit per l'usuari i guarda la informació dins aquesta ontologia.

En un principi ens hem plantejat crear una aplicació amb interfície d'usuari en format web, però hem canviat d'idea, en part per la poca disponibilitat de temps per la seva realització, però sobretot per que pensem que serà molt més útil per futures aplicacions pràctiques crear l'aplicació en forma de llibreria, fàcilment reutilitzable i extensible i, de pas, retornar a la comunitat una mica del valor afegit que n'hem tret.

1.6 Breu descripció dels altres capítols de la memòria

En els següents capítols d'aquesta memòria farem un breu però crític, estudi sobre l'estat actual de la web semàntica i les ontologies.

Més endavant descriurem el disseny i la implementació d'una ontologia pel domini de *Twitter,* que fent servir una aplicació omplirem de dades reals, extretes de la mateixa *API* de *Twitter*.

Finalment parlarem sobre els requeriments, el disseny i la implementació d'aquesta aplicació, comprovant que funciona correctament i que es compleix l'objectiu fixat en l'enunciat d'aquest treball.

2. Estudi de l'"estat de l'art"

Un dels principals objectius d'aquest treball es dissenyar i construir una ontologia sobre Twitter i per aconseguir-ho necessitem fer una recerca exhaustiva sobre l'"estat de l'art" dels estàndards i les tecnologies relatives a la web semàntica i, mes concretament, a les ontologies.

En aquest apartat farem una reflexió de l'estat actual de la web semàntica i sobretot de les ontologies, després d'una recerca sobre la matèria.

2.1 – Concepte de web semàntica

El primer pas necessari per entrar al mon de la web semàntica es la pàgina web del W3C [6] (*World Wide Web Consortium),* que es una comunitat internacional que, unint els esforços dels propis treballadors amb el de les organitzacions membre i el del públic, treballa per desenvolupar estàndards web amb l'objectiu de portar la Web a aprofitar el seu màxim potencial.

Pel W3C, i el seu director Berners-Lee, la web semàntica es una web de **dades enllaçades** *(linked data)* que poden ser processades directa i indirectament per maquines. Per a aconseguir aquesta web de dades enllaçades hauríem de tenir totes les dades disponibles en un format estàndard, que es pugui arribar a elles i que sigui manejable per les eines de web semàntica. Al mateix moment hauríem de tenir disponibles les relacions entre aquestes dades.

El propòsit de la web semàntica es que els agents automàtics siguin capaços d'interpretar la informació publicada a la Web, per portar a terme aquesta pesada feina de buscar informació, combinar-la adequadament i finalment actuar en base a ella. Per a que aquestes màquines siguin capaces de entendre i respondre peticions complexes basades en el significat de les dades necessitaran que la informació estigui ben estructurada semànticament, interconnectada i accessible.

En llenguatge pla, es tracta d'afegir, a les pàgines web ja existents i fetes pel consum humà, metadades sobre el seu contingut que puguin entendre les màquines (o agents automàtics). No es tractaria doncs d'un salt evolutiu gran on es canvien les tecnologies ja existents per unes altres més noves, si no més combinar les tecnologies que ja fem servir per facilitar l'accés a la informació de forma automatitzada, per integrar diferents continguts, aplicacions i sistemes d'informació.

Les solucions propostes pel W3C passen per afegir al HTML (llenguatge d'etiquetatge pensat per descriure documents i relacions entre ells, de cara al consum humà) els llenguatges RDF, OWL i XML pensat per

definir recursos, dotar-los de significat i estructurar-los de forma que siguin consumits per agents automàtics.

El propi Berners-Lee ha batejat aquesta nova xarxa de dades enllaçades com *Giant Global Graph* [13] (o Graf Global Gegant) que, en contraposició al concepte *World Wide Web* basat en HTML i pensat per compartir documents, permetrà compartir dades.

Aquest concepte es basa en tres punts principals:
1. Una URL apunta cap on son les dades.
2. Si accedeixes a aquesta URL, t'ha de servir aquestes dades.
3. Les relacions entre dades es descriuen amb més URLs que apunten a més dades.

2.2 – Arquitectura de la web semàntica

Molt sovint es fa servir el terme web semàntica [12] per parlar dels formats i les tecnologies que la fan possible. Aquestes tecnologies permeten descriure formalment els conceptes i les seves relacions dins un domini de coneixement concret.

El conjunt de tecnologies de la web semàntica ens proporciona un entorn on una aplicació es capaç de fer consultes sobre aquestes dades i, extreure'n conclusions, tot sense participació humana. El W3C proposa els estàndards per aquestes tecnologies:
- *Resource Description Framework (RDF)*
- *RDF Schema*
- *Simple Knowledge Organization System (SKOS)*
- *SPARQL*
- *Notation3 (N3)*
- *N-Triples*
- *Turtle (Terse RDF Tripe Language)*
- *Web Ontology Language (OWL)*
- *Rule Interchange Format (RIF)*

Més endavant en aquest document en parlarem d'aquests estàndards.

L'arquitectura de la web semàntica ve definida per la *Semantic Web Stack* [15] (Pila de la Web Semàntica). A la figura 2.1.2.1, creada pel mateix Berner-Lee, podem veure la il·lustració per capes de la jerarquia dels llenguatges descrits anteriorment, on cada capa explota les capacitats de la capa immediatament inferior.

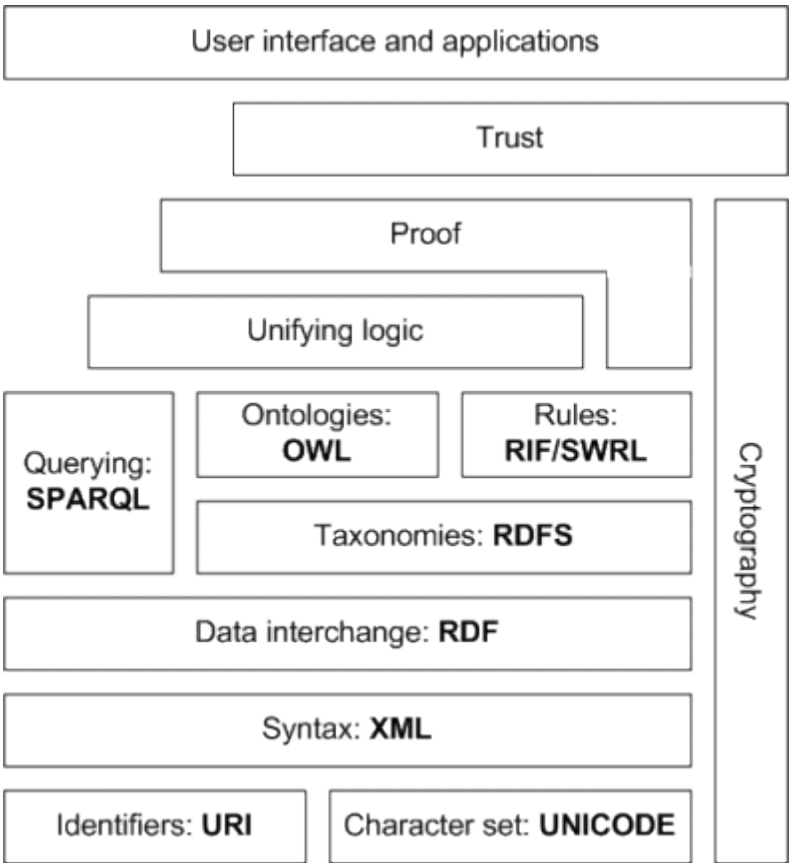


Fig. 2.1.2.1 – Semantic Web Stack

Encara que aquesta arquitectura no para d'evolucionar [14] mentre les capes es van concretant, ens pot fer servir de guia per entendre el concepte.

Les capes més inferiors de la pila son tecnologies ben conegudes al mon de la Web. Els recursos semàntics son identificats unívocament per URI (*Uniform Resource Identifier)* i representats per la codificació de text UNICODE. XML es un llenguatge de marques que permet creació de documents amb les dades estructurades. Aquests estàndards defineixen les dades a les que després hem de donar significat i compartir-les.

Les capes més superiors o exteriors contenen tecnologies que encara no s'han estandarditzat o que simplement son idees. La criptografia ens ha d'ajudar a assegurar i verificar que les declaracions semàntiques provenen d'un origen segur i confiable. La capa *Trust* ens ha de donar la confiança en les deduccions extretes verificant que aquestes provenen d'un origen confiable (fent servir la capa anterior) i s'han deduït fent servir lògica formal de capes inferiors. La interfície d'usuari es la capa final que permet que els humans facin ús de les aplicacions de web semàntica.

Les capes restants son les tecnologies que ens dona el W3C per poder construir aplicacions de web semàntica que veurem amb més detall a continuació.

2.2.1 – RDF [7]

Resource Description Framework, dissenyat per descriure dades i metadades (dades sobre les dades) sobre subjectes específics, per descriure estructures de grups de dades i relacions entre aquestes.

Es un model estàndard per intercanvi de dades, desenvolupat pel W3C, que estén la estructura enllaçada de la Web fent servir URIs (*Universal Resource Identifier*) per definir relacions entre entitats en els dos sentits de l'enllaç (anomenat triple enllaç).

Les característiques del model permeten l'agrupació de diferents dades encara que l'esquema darrera de cada dada sigui diferent i eviten haver de canviar les eines de consulta en evolucionar el model.

Una sentencia RDF permet descriure la relació que te un objecte A amb un objecte B i la relació inversa de B amb A. A l'objecte A se'l anomena **subjecte**, l'objecte B és l'**objecte** i a la relació entre tots dos **predicat**.
El model es un conjunt d'aquestes relacions que es poden veure com un graf dirigit i etiquetat [8].

2.2.2 – RDF Schema

RDF(S) es un llenguatge per descriure vocabularis per *RDF* que defineix classes i jerarquies de classes, propietats i sots propietats i el seu domini (a quina classe pertany la propietat) i abast (de quina classe pren els seus valors). En definitiva, proveeix dels elements bàsics per la definició de ontologies, encara que hi ha certes expressions que no es capaç de representar i que *OWL* sí ho és.

2.2.3 – SKOS

Simple Knowledge Organization System, aplicació de *RDF* que proporciona un model per representar la estructura bàsica i el contingut d'esquemes conceptuals (tesaurus).

2.2.4 – SPARQL[10]

Llenguatge de consultes (molt semblant en la forma al *SQL*) que es pot fer servir per per fer consultes sobre diferents orígens de dades, sempre que sigui accessible en format RDF.

2.2.5 – N3

RDF Notation 3, és un tipus de notació no *XML* per descriure els triplets *RDF*, més simple i fàcil d'entendre que aquest i que ofereix moltes de les característiques per la serialització de *RDF(S)*.

2.2.6 – Turtle

Terse RDF Triple language, és un subconjunt breu de *N3* que només descriu característiques de RDF amb sintaxis molt semblant a *N3*.

2.2.6 – N-Triples [19]

Serialització en text pla, en format de línies, per grafs RDF. És un subconjunt de *Turtle* dissenyat per ser més simple que aquest i ser analitzat i generat per software més fàcilment, però que pot ser molt tediós i difícil de llegir al treballar-hi manualment.

2.2.8 – OWL[11]

Web Ontology Language o llenguatge d'ontologies web, és una extensió d'*RDF*, que afegeix més vocabulari per descriure propietats, classes i relacions que RDF o RDF(S) no son capaços de representar. Permet crear relacions específiques i concretes del domini que s'intenta representar.

Permet, també, que els processos automàtics puguin fer deduccions (crear nous triplets, noves relacions, a partir dels ja existents). Es una eina que serveix per formalitzar vocabularis (ontologies) i realitzar-hi raonaments basats en lògica descriptiva.

Hi ha 3 subllenguatges differents dintre de OWL: *OWL Full*, que permet la maxima expressivitat però no garantitza el poder computar una resposta inferida; *OWL DL*, que es el màxim subgrup del *Full* que permet garantitzar una resposta; i *OWL Lite,* el més petit i feble dels tres que permet classificacions jeràrquiques i restriccions simples de cardinalitat.

2.2.9 – Microformats [16]

Afegeixen semàntica a les pàgines *(X)HTML* (i qualsevol altra aplicació del *XML*) fent servir les seves etiquetes per definir el significat de la informació que hi contenen (com els atributs *class* o *rel*), o fent servir microformats específics com poden ser:

- *hCard:* indica que el contingut del tag es informació de contacte
- *hCalendar:* informació sobre events de calendari.
- *hProduct:* informació sobre un producte.

Podem trobar d'altres solucions com:

- *RDFa* [9] (*RDF in Attributes)*: especificacions de atributs per etiquetes de XML, que estructuren el contingut facilitant el posterior proces i generació de triplets RDF.
- GRDDL [18]: permet transformar el contingut en triplets RDF fent servir unes marques que inclouen la URI de la definició d'aquesta transformació. Els agents GRDDL (o consumidors GRDDL) fan servir aquesta definició per a transformar les dades.

2.3 – Estat actual dels estàndards

Les tecnologies descrites anteriorment ja son estàndards ben establerts, però les capes de *Unifiying Logic* i *Proof* encara estan per definir.

La capa *Unifiying Logic* comprèn el raonament lògic, es a dir, inferir o deduir nous fets, a partir dels triplets RDF, les consultes SPARQL o les ontologies, i comprovar la seva consistència. Y la capa *Proof* ha de demostrar, explicant els passos del raonament lògic, que aquesta informació i els nous raonaments inferits son fiables, precisos i dignes de confiança.

Però, hi ha certes corrents dins la comunitat de la web semàntica que pensen que les capes altes de la pila semàntica no son necessàries [20] perquè no es el que s'està demanant fer als agents automàtics. Des d'aquest punt de vista, la web semàntica es una forma de publicar informació amb significat i no agregada per ser consumida per programes i recau en els desenvolupadors o els usuaris la elecció de quina informació es considera fiable. Aquest es el punt de vista de *linked data*, que diu que amb les tecnologies actuals en tenim prou per construir aplicacions útils i que, fins i tot, alguns pensen que els estàndards com *OWL* no son realment necessaris per construir aquestes aplicacions de *linked data*.

Segons el *W3C*, la intenció de la web semàntica és millorar la usabilitat i utilitat de la Web i els seus recursos interconnectats a través de:

- Servidors que publiquen sistemes de dades ja existents fent servir els estàndards RDF i SPARQL
- Documents marcats amb informació semàntica, estenent les etiquetes <meta> que ja existeixen al HTML. Aquestes marques han de ser informació que pugui entendre una màquina sobre el significat de la informació del document.
- Vocabularis de metadades comuns (ontologies) i mapes entre ells per que els creadors de continguts sàpiguen com marcar els seus documents per tal de que els agents automàtics l'entenguin.
- Agents automàtics que fent servir aquestes marques i ontologies facin tasques pels usuaris de la web semàntica.
- Serveis web (o agents automàtics propis) que donin informació específicament per aquests agents

2.4 – Concepte d'ontologia

Tal com descriu [3] una ontologia es un model d'un domini concret, la *"representació de la conceptualització explícita i compartida d'aquest domini en particular"* (Tomas Gruber, 1993). Extraiem 3 propietats d'aquesta definició:

- Representació explícita: ha d'estar escrita en un llenguatge formal i, es clar, en suport digital per que sigui llegida i interpretada per programes informàtics.

- Conceptualització compartida: ha de ser la visió que una comunitat de persones te sobre un domini.
- Domini concret: ha de representar el domini d'algun problema concret.

Les ontologies permeten especificació comuna i compartida d'informació de domini entre persones i màquines. Son capaces de utilitzar dades de diferent naturalesa, característiques i formats per extreure'n coneixement nou, contràriament a les bases de dades clàssiques.

Però les ontologies també tenen la seva contrapartida, a causa del volum de dades que representen s'augmenta la complexitat en el procés de creació i en la manipulació dels arxius que la contenen. Son de difícil escalabilitat i la integració de diferents ontologies pot ser tant difícil com integrar els mateixos recursos que descriuen.

L'aplicació d'ontologies al camp web també contempla problemes per resoldre, com crear les anotacions pel tractament automàtic (en vers de l'humà), la necessitat de validar el nivell de qualitat dels continguts, la manca de mecanismes per comprendre, visualitzar i recollir les ontologies ja desenvolupades i la falta de maduresa d'alguna de les capes de la pila de la web semàntica.

2.5 – Conclusions

Després d'aquesta investigació de la situació actual dels estàndards de la web semàntica podem dir que es una idea molt ambiciosa, potser massa, per unificar el caos que hi ha actualment a la *Web* i automatitzar algunes de les tasques més repetitives que hem de fer els usuaris per obtenir la informació que necessitem d'ella.

Tot i que molts estàndards ja existeixen i, poc a poc, s'estan fent servir encara queda per definir, concretar i estandarditzar algunes parts de l'esquema de la pila de la web semàntica. Per altra banda algunes corrents dins de la comunitat acadèmica discrepen sobre aquest esquema, al·legant que no són necessàries per l'objectiu que s'està intentant assolir (agents automàtics capaços d'obtenir la informació que necessitem de la web i actuar sense intervenció humana) i que son més pròpies del camp de la intel·ligència artificial.

Queda molta feina a fer per que la web semàntica, o al menys la idea de *linked data,* sigui una realitat, però tots els actors implicats en aquest projecte estan d'acord en que es un pas necessari que ja l'estem donant.

3 Disseny i implementació de la ontologia

No existeix una única forma correcta per desenvolupar ontologies, poden existir varies formes viables de fer-ho i la millor solució sempre dependrà de la aplicació que se li vol donar a la ontologia.
A més, el disseny i la implementació ha de ser necessàriament un proces iteratiu, prenent decisions durant tot el camí fins arribar a una ontologia que acompleixi els objectius pels que s'ha creat.
Els conceptes de la ontologia han de ser molt propers als objectes i relacions dins del domini estudiat: noms (objectes) i verbs (relacions) dins les frases que defineixen aquest domini.
Malgrat tot aixo, existeix una metodologia simple i iterativa [28] que pot ser de gran utilitat en la creació d'una ontologia com la que ens ocupa, descrita en els següents passos:

1. Determinar el domini i l'abast de la ontologia.
2. Considerar la reutilització.
3. Enumerar les entitats importants de la ontologia.
4. Definir les classes i la seva jerarquia.
5. Definir les propietats de les classes.
6. Crear instancies.

3.1 – Determinar el domini i l'abast de la ontologia

Es important definir l'abast i el domini de la ontologia al principi del proces, i es pot fer responent a les següents preguntes:

- Quin domini cobrira la ontologia?
- Per que volem fer-la servir?
- Quin tipus de preguntes ha de respondre?
- Qui la farà servir i la mantindrà?

Aquestes preguntes estan respostes a l'enunciat d'aquest treball: es demana crear una ontologia que tingui en compte la estructura de *Twitter*, ha d'emmagatzemar la informació continguda als *Tweets* i ser capaç de respondre preguntes per saber qui es l'autor del *Tweet*, el seu contingut, si conté *hashtags*, si es un *retweet* i qui es l'autor original, si es menciona algun altre usuari, etc.

En ser un treball d'investigació, a més de pràctic, es pot deduir que la utilitat d'aquesta ontologia pot ser molt diversa i podrà ser utilitzada per d'altres estudis o aplicacions pràctiques, però això no es pot saber a priori.

3.2 – Considerar la reutilització

Val la pena sempre considerar la feina que ha fet algú altre, comprovant si es pot adaptar i estendre per la feina a fer. Es pràctic sobretot si es vol interactuar amb d'altres aplicacions que fan servir ontologies o vocabularis.

En aquest cas, després d'una cerca profunda i infructuosa, no s'ha trobat cap ontologia que defineixi la estructura de *Twitter* o que es pugui adaptar per encabir els seus conceptes. Per tant s'haurà de crear una ontologia de zero que cobreixi les nostres necessitats.

3.3 – Enumerar les entitats importants a la ontologia

Fent un estudi dels objectes més habituals a *Twitter* [21] trobem que els conceptes més comuns son 4:

- *Tweets*
- *Users*
- *Entities*
- *Places*

Hi ha d'altres objectes que podem trobar a la plataforma de Twitter però que a priori no aporten més informació d'utilitat a la nostra ontologia.

Trobem d'altres entitats secundaries com *Coordinates,* que poden ser contingudes per les entitats *Tweet* i *viceversa*. Al mateix temps una entitat de tipus *Places* conté un conjunt d'entitats *Coordinates* que encerclen el lloc exacte que indica la entitat.

L'entitat *Entities* es divideix en 4 tipus:

- *Hashtag*,
- *Media*,
- *Url*,
- *UserMention,*

i tot element d'aquest tipus ha de ser element d'un dels 4 subtipus sense excepció.

D'altres entitats destacades son les relacions que podem trobar entre les entitats principals, com per exemple tot *Tweet* ha de tenir un autor (un *User*) i per tant un *User* pot ser autor d'un número *n* de *Tweets* (*n* pot ser 0 o més).

Tot *User* pot ser seguit per 0 o més *Users* i, inversament, un *User* pot seguir a 0 o més *Users.* També un *User* pot contenir *Url* al seu perfil (per tant inversament una *Url* pot pertànyer al perfil d'un *User).*

El següent diagrama representa les entitats principals que trobem a *Twitter* i les seves relacions:

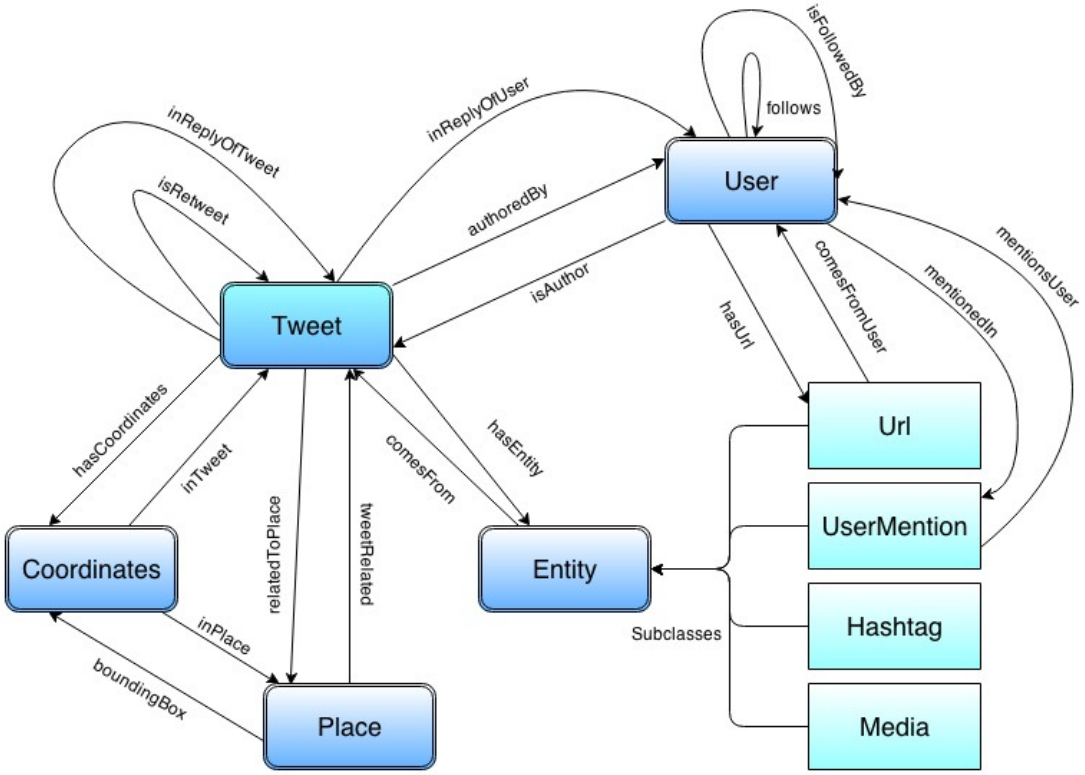


Fig 3.3.1 Entitats i relacions a Twitter

3.4 – Simplificacions

Per simplificar l'esquema s'ha decidit eliminar relacions com *Contributors*, que a més de ser una característica en format beta encara i no gaire estesa, no aporta informació rellevant pel nostre domini. Considerarem, doncs, que un *Tweet* te un, i només un, autor.

Totes les entitats a *Twitter* tenen dos camps per l'identificador únic, un en format numèric i un altre en format text. Vist que alguns llenguatges de programació i formats tenen problemes al processar valors numèrics de més de 53 bits [22] , i els identificadors únics de *Twitter* son de 64 bits, i veient que la entitat *Places* ja ha prescindit de l'identificador numèric, escollirem només el de format text.

S'han deixat de banda els camps que tenen significat des de la perspectiva d'usuari que ha ingressat a l'aplicació (ha fet *login*, posant el nom d'usuari i la contrasenya), com els que defineixen si l'usuari actual ha marcat com a favorit o ha retuitejat un determinat *Tweet*, perquè la nostra ontologia i l'aplicació que en farà us no treballa des d'aquesta perspectiva.

No inclourem els atributs que relacionen a una altra entitat de més d'una forma, com per exemple en el cas del *Tweet*, els camps *in_reply_to_screen_name* i *in_reply_to_user_id_str,* que relacionen

aquest *Tweet amb el User* a qui s'esta responent. En aquest cas farem servir el segon com a relació directa entre *Tweet* i *User* al que s'ha respost. Actuarem de forma similar en d'altres casos.

3.5 – Definir les classes i la seva jerarquia

De les entitats enumerades a l'apartat anterior podem extreure les classes que formaran la nostra ontologia:

- *Tweet*
- *User*
- *Entity*, amb subclasses *Hashtag, Media, Url* i *UserMention*
- *Place*
- *Coordinates*

Per posar nom a les classes s'ha escollit el singular (en contra de com s'anomena a l'API de *Twitter*) excepte en el cas de *Coordinates* ja que el concepte engloba sempre 2 valors (latitud i longitud) i anomenar-la *Coordinate* hauria desvirtuat el concepte i no tindria sentit a la realitat.

A la següent figura podem veure les classes i la seva jerarquia modelades a Protégé:

Fig. 3.5.1 Classes i jerarquia a Protégé

Com que qualsevol instancia de qualsevol d'aquestes classes només pot ser instància d'una d'elles, s'han creat totes les classes germanes disjuntes (incloent les subclasses de *Entity*).

3.6 – Definir les propietats de les classes

A partir dels atributs de cada objecte definits a la documentació oficial de *Twitter* [21], definim les propietats de les classes que ja hem definit al apartat anterior.

3.6.1 – Tweet

La classe principal d'aquesta ontologia es la classe *Tweet*, que hem definit amb les propietats que observem a la figura 3.6.1.1.

Segons la documentació oficial de Twitter [23] , el *Tweet* es la unitat atòmica bàsica de construcció a Twitter. Els usuaris (*User*, veure 3.6.2) escriuen *Tweets* (també anomenats actualització d'estat, *status update*) que poden veure els seguidors (*followers*) que "escolten" el que va publicant.

Un *Tweet* es pot trobar sol, dins del conjunt que ha publicat un usuari o dins dels *timelines* (col.lecció de *Tweets* ordenada cronològicament dels usuaris que segueis un usuari concret). També pot estar incrustat (*embedded*), es pot respondre, marcar/desmarcar com a favorit, retuitejat o esborrat.

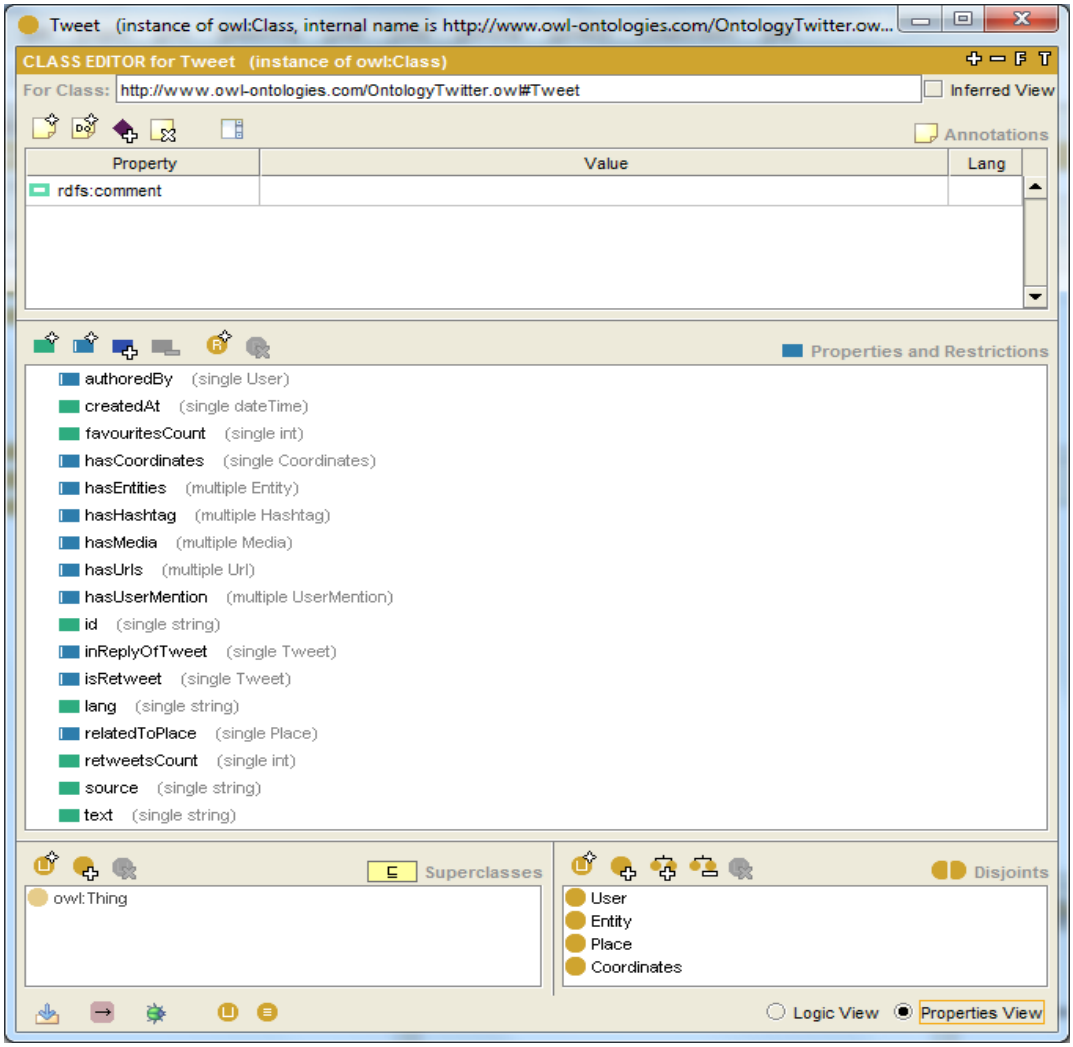


Fig. 3.6.1.1 – Propietats de la classe Tweet

- *authoredBy:* Es de tipus *Users* (veure 3.6.2) i relaciona el *Tweet* amb l'usuari que l'ha creat. Tot *Tweet* té sempre un usuari, no te sentit un *Tweet* sense *User*.
- *createdAt:* propietat de tipus *dateTime* que conté la data de publicació (creació) del *Tweet.*
- *favouritesCount:* indica quantes vegades ha sigut marcat com favorit aquest *Tweet.* Te format *int i pot ser Null.*
- *hasCoordinates:* Pot ser Null o relacionar el *Tweet* amb un objecte de tipus *Coordinates* (veure 3.6.5) que indica la localització geogràfica des d'on s'ha publicat el *Tweet.*
- *hasEntities: Entities* (veure 3.6.3) extretes del text del *Tweet.*
- *hasHashtag:* subpropietat de *hasEntities* especialitzada només per *Entities* del tipus *Hashtag*. Relaciona el *Tweet* amb els *Hashtags* extrets del seu contingut.
- *hasMedia*: igual que l'anterior però per *Media*.
- *hasUrls*: igual que l'anterior però per *Url.* El nom es plural per diferenciar d'una propietat equivalent que te la classe *User*.
- *hasUserMention*: igual que l'anterior però per *UserMention.*
- *id: Representació en cadena de caràcters identificador únic d'aquest Tweet.*
- *inReplyOfTweet: S*i el *Tweet* es en resposta a un altre, *aquesta propietat el relaciona amb el Tweet* respost.
- *isRetweet*: si aquest Tweet es un "retuit", aquest camp el relaciona amb el *Tweet* original en cas de ser un *retweet* i, per tant, es de tipus *Tweet.*
- *lang:* Idioma detectat automàticament del *Tweet*, pot ser Null o *"und"* si no s'ha pogut detectar automàticament cap idioma.
- *relatedToPlace:* pot ser Null o contenir un objecte de tipus *Places* (veure 3.6.4) que representa un lloc on aquest *Tweet* esta relacionat, encara que no vol dir que en sigui l'origen.
- *retweetsCount:* Indica quantes vegades s'ha fet *retweet* (publicar un *Tweet* d'un altre usuari per que el puguin veure els teus seguidors) d'aquest *Tweet.* Te format *int* i pot ser *Null.*
- *source*: nom de l'aplicació des de la que s'ha publicat el *Tweet* (*web,* si s'ha creat des d'el site de *Twitter*), de tipus string.
- *text*: text del *Tweet* en format text UTF-8.

3.6.2 – User

Els concepte *User* (usuari) defineixen un compte a *Twitter* [24], pot ser qualsevol persona, entitat o cosa. Un usuari pot escriure *Tweets*, pot

seguir a d'altres usuaris, crear llistes d'usuaris i marcar/desmarcar *Tweets* com a favorits.

L'usuari també te un *home timeline* (col·lecció dels *Tweets* més recents de l'usuari i dels usuaris que segueix), pot ser mencionat en *Tweets* escrits per d'altres usuaris (fins i tot en els propis *Tweets)* i pot ser buscat.

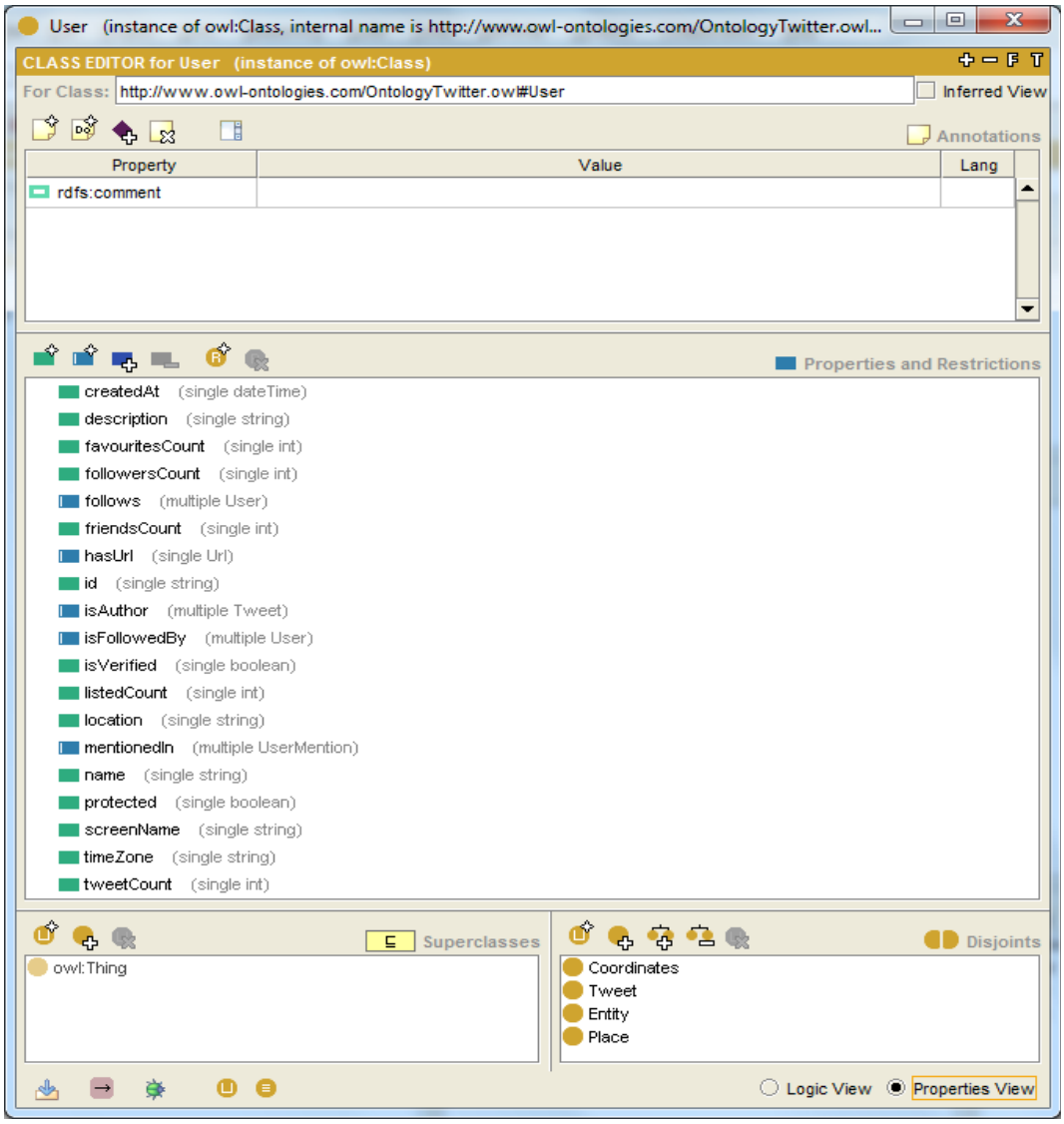


Fig. 3.6.2.1 – Propietats de la classe User

Propietats de la classe *User:*

- *createdAt,* que indica la data de creació del compte d'usuari, de tipus *dateTime*.
- *description:* Cadena de text que descriu el compte, introduït per l'usuari en el moment de crear el perfil del compte.
- *favouritesCount*: Número de *Tweets* que l'usuari ha marcat com a favorit, en format Int.
- *followersCount:* indica el número de seguidors que té, de tipus *int*.

- *follows:* propietat de tipus User, que el relaciona amb els usuaris que segueix.
- *friendsCount:* Número d'usuaris que aquest usuari segueix, format *int.*
- *hasUrl:* propietat de tipus Url (veure 3.6.3.3) que el relaciona amb l'entitat Url que conté al seu perfil, si es que en té.
- *id:* Identificador numèric únic d'aquest usuari, en format S*tring*.
- *isAuthor:* camp de tipus Tweet, que relaciona l'usuari amb els *tweets* que ha escrit. Pot ser buit si no ha escrit res o tenir multiples valors.
- *IsFollowedBy*: de tipus *User*, el relaciona amb els usuaris que el segueixen. Pot estar buit si no el segueix ningú o pot prendre multiples valors.
- *isVerified:* atribut de tipus *boolean* que indica si l'usuari es una personalitat/empresa/ens real i verificat.
- *listedCount:* número de llistes públiques on l'usuari es membre, en format Int.
- *location:* Localització de l'usuari (definida per ell mateix al seu perfil) en format *String.* Pot ser *Null*, pot no ser una localització real i pot no ser extreta.
- *mentionedIn*: propietat de tipus *Tweet* i cardinalitat múltiple. Relaciona l'usuari amb els *tweets* on se'l menciona (fent servir el seu pseudònim amb l'arrova davant, veure *UserMention* a 3.6.3.4)
- *name:* Nom real de l'usuari tal i com ell mateix s'ha definit al perfil (per tant pot ser inventat), en format *String*.
- *protected:* Booleà que indica si l'usuari ha protegit els seus *Tweets* (si es *true* només els usuaris que tenen permís expres poden llegir-los)
- *screenName:* identificador o pseudònim en format text que ha fet servir l'usuari per identificar-se a si mateix. Es guarda sense l'arrova (@).
- *timeZone:* conté la zona horària declarada per l'usuari al seu perfil, de tipus *string*.
- *tweetCount:* de tipus *int,* número de *tweets* que ha publicat en total.

3.6.3 – Entity

Les *Entities* [25] (entitats) son metadades (dades sobre les dades) amb informació addicional sobre el contingut del *Tweet*, a més de ser instruments per resoldre *URLs.* Trobem 4 tipus diferents d'*Entities*: *hashtags*, *media*, *url* i *user_mentions.*

També podem trobar *Entities* dins d'un *User*, però només del tipus *url*, que descriuen les *urls* que l'usuari ha posat al camp url de la descripció del seu compte.

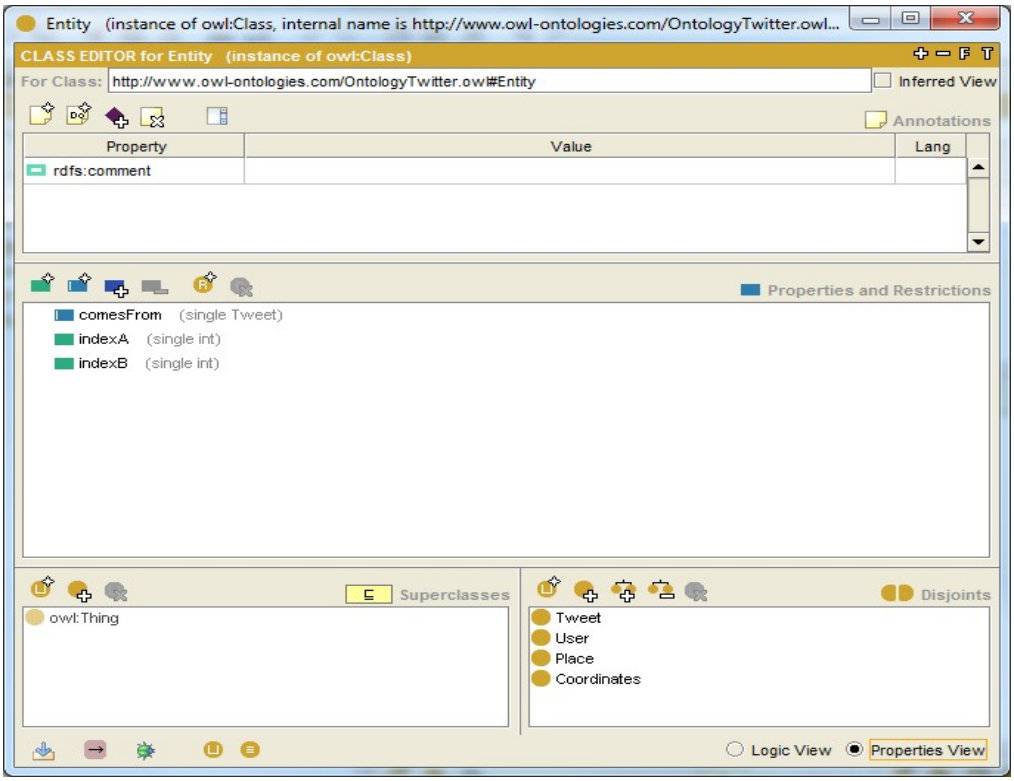


Fig. 3.6.3.1 - Propietats de la classe Entity

La classe abstracta *Entity* (fig. 3.6.3.1) té les propietats:

- *comesFrom:* indica en quin *Tweet* s'ha trobat aquesta *Entity,* és, per tant, de tipus Tweet.
- *indexA:* posició del primer caracter de la *Entity* dins el contingut del *Tweet.* De tipus *int*.
- *indexB:* posició de l'ultim caracter de la *Entity*, també de tipus *int*. Restant-li la propietat anterior podem obtenir la longitud del text de la *Entity*.

Es pot pensar que una mateixa *Entity* pot apareixer a molts *tweets*, llavors podriem considerar la cardinalitat de la propietat *comesFrom* com a múltiple, però com una *Entity* també conté els index que la posicionen dins el contingut del *Tweet*, encara que aparegui a varis *tweets* no han d'estar obligatoriament a la mateixa posició exacta, per tant la definim com a simple. Una altra raó per aquesta decisió es que definint-la com a múltiple afegiria un grau de complexitat a la ontologia que no necessitem pel nostre projecte.

Aquestes 3 propietat les hereten totes les seves subclasses, que son:

3.6.3.1 – Hashtag

Un *hashtag* es un tros de text precedit d'un coixinet (#, en anglès *hash)* que es fan servir a *Twitter* per etiquetar el contingut, principalment. Aquesta etiqueta es pot fer servir per cerques de *Tweets* relacionats amb algun tema.

La subclasse de *Entity*, *Hashtag* te, a més de les propietats heretades de la classe pare:

- hashtagComesFrom: propietat que hereta de *comesFrom* i relaciona aquest *Hashtag* amb el *Tweet* d'on s'ha extret.
- *text:* de tipus *string* conté el text concret de la etiqueta, sense el #.

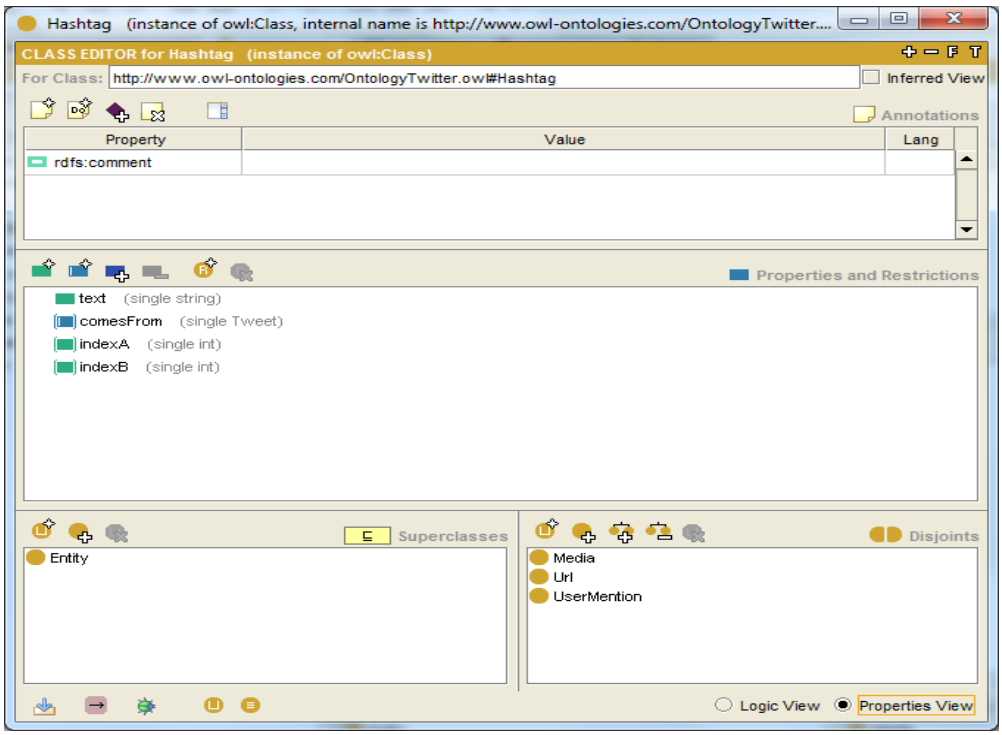


Fig. 3.6.3.1.1 – Propietats de la classe Hashtag

3.6.3.2 – Media

Entitats que representen imatges o vídeos incrustats en el contingut del *Tweet.* Els camps rellevants per la nostra ontologia son:

- *displayUrl:* de tipus *string*, conté la url tal i com es mostra al text del *Tweet.*
- *id:* identificador únic del recurs en format de text.
- *mediaUrl:* una url *http://* que apunta directament al recurs adjuntat al *Tweet*. Aquesta url identifica unívocament el recurs representat des de qualsevol punt de la Web i, per tant, es pot fer servir per incrustar-la a qualsevol lloc web.
- *type:* tipus del recurs pujat (foto, vídeo, ...), de tipus *string*.

Hem decidit eliminar del disseny de la ontologia els diferents camps de *url* i ens quedarem només amb els que defineixen únicament el

recurs a la Web i el que descriu la url tal i com apareix al Tweet. Les diferents mides del recurs tampoc no ens seran de molta utilitat al nostre model i podrien afegir complexitat al disseny.

Com que *Media* te identificadors únics (*id* i *mediaUrl*) es pot pensar que dos objectes de tipus *Media* son el mateix si tenen els mateixos identificadors, però, com hem comentat abans, aquesta classe conté els index que la posicionen dins el contingut del *Tweet* i aquests no han de ser iguals sempre.

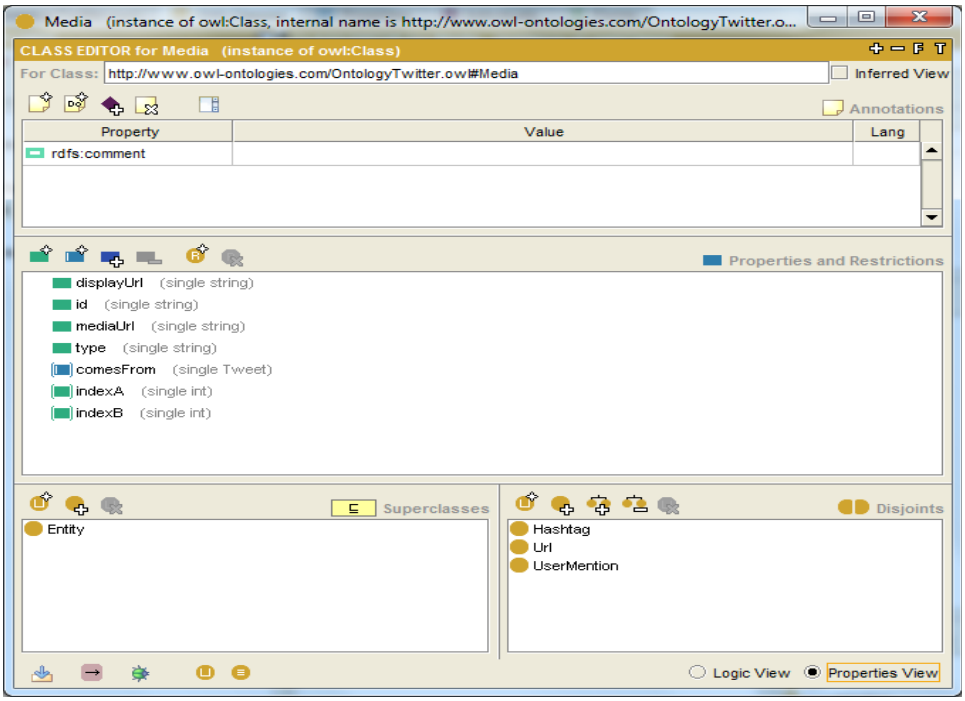


Fig. 3.6.3.2.1 – Propietats de la classe Media

3.6.3.3 – Url

Una *URL* (*Uniform Resources Locator*, o Localitzador Uniforme de Recursos) serveix per localitzar qualsevol recurs disponible a la Web. *Twitter* ens permet copiar *urls* dins el contingut del text i de la seva API extreiem aquestes propietats:

- *comesFromUser*: de tipus *User*, relaciona la *Url* amb un *User* que l'hagi definit al seu perfil.
- *displayUrl*: veure 3.6.3.2.
- *mediaUrl*: veure 3.6.3.2.

En aquest cas, dels camps definits a la API de *Twitter* [25], també ens quedem només amb la url única del recurs a més de la url tal i com es mostra al camp de text (ja sigui d'un *Tweet* o d'un *User)*.

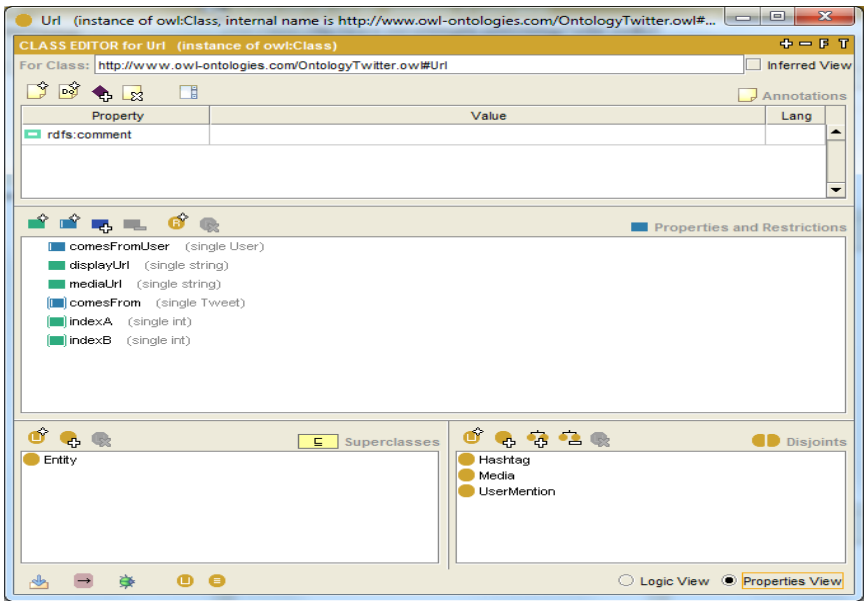


Fig. 3.6.3.3.1 – Propietats de la classe Url

3.6.3.4 – UserMention

Twitter permet comunicar-se entre usuaris mencionant-los en el contingut del *Tweet* fent servir el simbol @ (*arroba) encapçalant* el seu *screenName* (veure 3.6.2). Aixo permet que l'altre usuari rebi el *Tweet* al seu *time line* i el pugui llegir.

De l'*API* de *Twitter* s'han extret els camps:

- *mentionsUser*: de tipus *User*, relaciona aquesta entitat amb l'usuari que es menciona.
- *screenName*: de tipus *string*, indica el pseudònim de l'usuari, sense l'arrova.

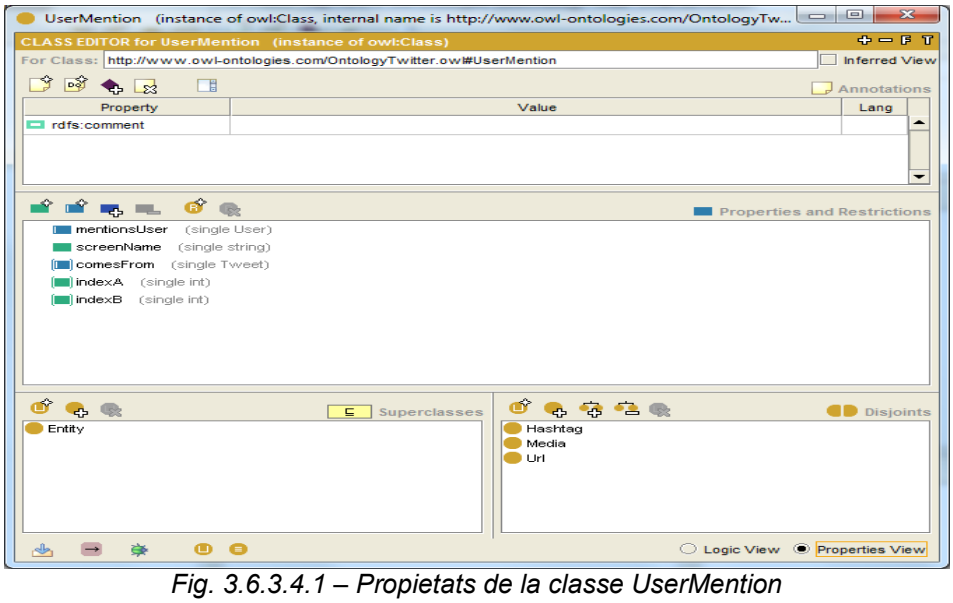


Fig. 3.6.3.4.1 – Propietats de la classe UserMention

3.6.4 – Place

Twitter permet associar llocs del món real als *Tweets* [26]. No vol dir que aquest provingui necessàriament del lloc associat, però el contingut del *Tweet* hi pot estar relacionat.

Podem buscar aquests llocs a *Twitter* o podem trobar els *tweets* associats a un lloc concret.

Places conté els següents atributs:

- *attributes:* conté informació addicional sobre el lloc, com pot ser l'adreça. Es de tipus *string*.
- *boundingBox:* de tipus *Coordinates* i cardinalitat múltiple, relaciona *Place* amb un número concret de *Coordinates* que l'encerclen.
- *countryCode:* codi del país on està situat el *Place.*
- *countryName:* nom del país al que pertany el lloc.
- *fullName:* Nom complet del lloc, llegible per humans .
- *id:* identificador únic del lloc, en format text.
- *tweetRelated*: de tipus *Tweet*, que relaciona *Place* amb el *Tweet* (poden ser més d'un) associat a aquest lloc.
- *type:* de tipus *string*,indica quin tipus de *Place* es (ciutat, barri, etc.).
- *url:* url que apunta a meta dades addicionals sobre aquest lloc. De tipus *string*.

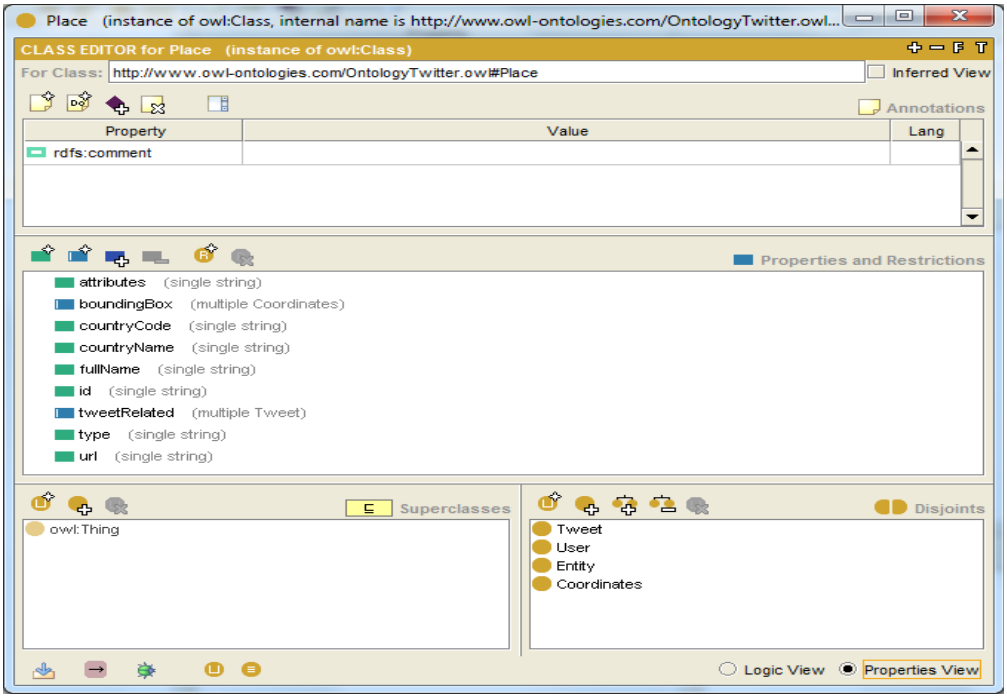


Fig. 3.6.4.1 – Propietats de la classe Place

3.6.5 – Coordinates

Per la classe *Coordinates* definim les propietats de la figura 3.6.5.1. Les propietats *latitude* i *longitude* no necessiten de més explicació, son de tipus float. La propietat type acostumara a ser “point”, depenent del que trobem al obtenir *Tweets* reals decidirem si podem prescindir d'ella, si sempre conté el mateix valor. Les propietats relacionals indiquen l'habitat de la classe *Coordinates*, com *inPlace* o *inTweet,* que indica que forma part d'una instància de *Place* o de *Tweet* respectivament.

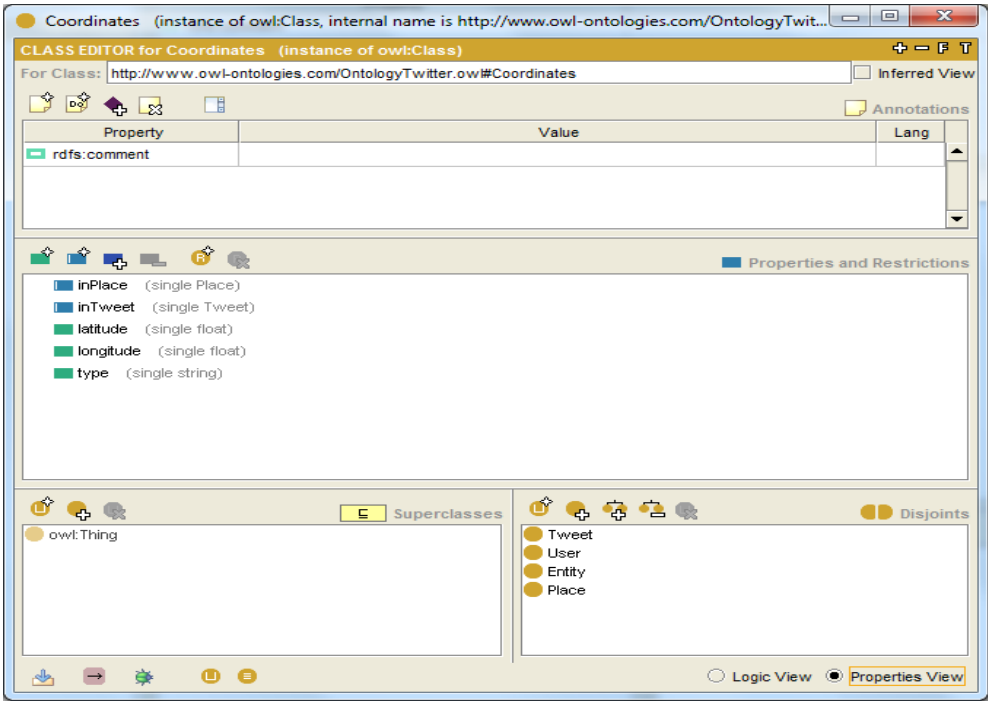


Fig. 2.3.6.5.1 - Propietats de la classe Coordinates

3.7 – Crear instancies

Un dels principals objectius d'aquest treball es implementar una aplicació que sigui capaç de llegir informació de *Twitter* i omplir la ontologia que acabem de definir, per tant aquest ultim pas de la creació d'ontologies el veurem al capítol següent.
Un cop la aplicació estigui dissenyada, implementada i provada mostrarem les instàncies que ha emmagatzemat a la ontologia.

4. Implementació de l'aplicació

4.1 Anàlisi de requeriments

Els requeriments de l'aplicació a implementar son molt oberts, simplement es tracta de construir una aplicació que sigui capaç d'omplir la ontologia, descrita en l'apartat anterior, amb dades reals extretes de *Twitter,* fent servir alguna de les diferents opcions de la seva API.

Després de pensar-ho molt be ens hem decantat per fer servir *Twitter Search API* que es la forma més senzilla i deslligada d'obtenir informació ja que no s'ha de mantenir l'aplicació funcionant tot el dia (i estar pendent d'un *timeline* d'un usuari concret). La primera idea va ser implementar una aplicació que donat un tema de cerca definit per l'usuari, pregunti a *Twitter* quina informació (*tweets*) te sobre aquest tema de cerca, processi la informació obtinguda i la guardi a la ontologia. A partir d'aquesta idea s'ha desenvolupat l'aplicació.

4.1.1 – Decisions preses

Durant el disseny d'aquesta aplicació s'han hagut de prendre unes decisions sobre l'abast de la mateixa:

- **Reply de *Tweet* però no de *User*:** Des del moment del disseny de la ontologia es va decidir representar el *reply* (resposta) a un *Tweet*, però no el *reply* a un usuari, sobretot per simplificar la ontologia, però també perquè te més sentit respondre a un *Tweet*, i com que aquest ja implica un autor, la relació *replyToUser* es pot inferir fàcilment.

  En l'aplicació tractem els *reply* a *User* que rebem de *Twitter* com a *UserMention*, i per tant anem a buscar les dades de l'usuari mencionat a *Twitter*.

- **Als *retweets o replies* obtenim *Tweet* i *User* original:** però només en un nivell de recursivitat. Es a dir, si per exemple el *Tweet* original es també un *reply* a un altre, no anem a cercar-lo. Aquesta decisió s'ha pres per simplificar però sobretot per evitar bucles recursius massa profunds o fins i tot infinits (la quantitat d'informació que conté *Twitter* es immensa i depenent de qui l'ha escrit, un *tweet* pot arribar a estar connectat amb milers i milions d'altres *tweets* i *users).*

- ***Retweet* i citació els tractem igual:** com un *retweet*. Una citació es molt semblant a un *retweet*, amb la diferencia que en la citació, l'autor que cita copia el contingut complet del *tweet* citat (potser editat una mica) i l'hi afegeix alguna cosa pròpia. En la llibreria *twitter4j*, si un *Status* es un *retweet* s'indica amb dos atributs: *isRetweeted=true* i *retweetedStatus* conté el *id* del *tweet* original. En el cas d'una citació, el flag *isRetweeted=false* però *retweetedStatus*

encara conté el *id* del *tweet* citat. L'aplicació tracta els dos casos de igual forma.

- **Per cada *User* no busquem els seus *followers* i *friends*:** per que el volum de dades a buscar es molt gran i donada la limitació de peticions per període de temps [27] que te la *API* de *Twitter* (t'obliga a esperar un temps determinat quan es supera el límit per fer una altra petició) l'aplicació podria estar hores només per cercar els ids dels *followers* i *friends* d'un usuari concret.
- **Per cada *User* no busquem tota la llista de *tweets que ha creat:*** perquè la llista pot ser molt gran (hi ha gent molt prolífica a *Twitter)* i gastaríem tota la energia en aquesta tasca. No busquem si ja tenim algun *tweet* seu a la nostra ontologia perquè tots els *tweets* que hi guardem ja els relacionem amb el seu autor.
- **Per cada *User* no busquem tota la llista de *UserMention*s que te:** per la mateixa raó d'abans. El que si fem es mirar si algun dels *tweets* que ja tenim guardats el menciona i establim la relació *mentionedIn* i la seva inversa *mentionsUser*.
- **La query definida per l'usuari té un limit de resultats:** també definit per l'usuari. S'ha pres aquesta decisió per dos motius: un, d'aquesta manera es limita l'execució de l'aplicació i sempre te un final (si no, donat el volum de dades de *Twitter* podríem estar sempre esperant el final d'una query); i dos, el límit de peticions per període de temps que te l'API de *Twitter* podria tenir-nos esperant molt de temps fins poder veure els resultats de la query.

4.2 Disseny de l'aplicació

Pel disseny d'aquesta aplicació s'ha seguit un paradigma de programació molt utilitzat actualment pels grans en el camp de proces de volums de dades molt grans: el paradigma de Productor – Consumidor.

Aquest paradigma es basa principalment en tres components, un primer component, el Productor, que, com el seu nom indica, es dedica només a produir les dades que es volen processar i emmagatzemar. No es preocupa de que passarà amb les dades que produeix un cop les ha enviat al seu destí, només captura dades i les entrega.

Per l'altra banda, el Consumidor es dedica a rebre aquesta informació en forma de dades amb un format concret, la processa i la guarda segons les especificacions amb les que ha estat dissenyat.

El tercer component en discòrdia acostuma a ser un sistema de cues, un simple recipient on el Productor bolca la informació que produeix i el Consumidor va agafant aquesta informació i la va processant.

En el cas que ens ocupa, el Productor s'encarrega de realitzar una cerca a *Twitter* mitjançant l'*API* escollida, processa la informació i la envia al

sistema de cues. Per l'altra cara de la moneda, el Consumidor, esta llegint de la mateixa cua aquesta informació provinent del Productor, la va processant segons arriba i l'emmagatzema a la ontologia OWL fent servir una API per OWL. Podem veure aquest esquema representat a la figura 3.2.1.

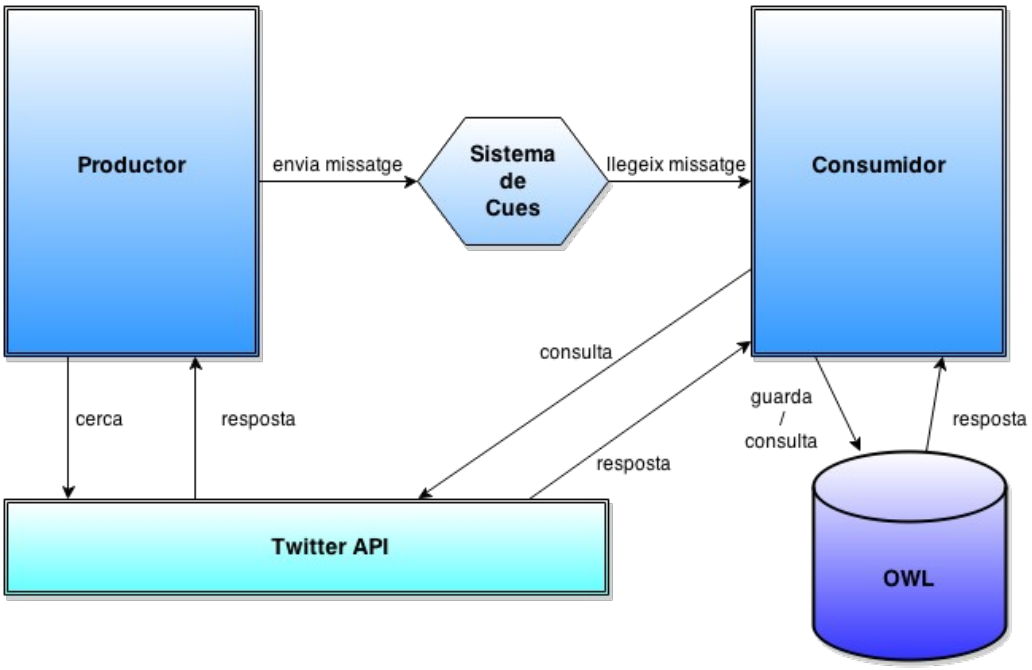


Fig. 4.2.1 - Esquema general, paradigma Productor-Consumidor

A continuació descriurem amb més detall cada un dels components de l'esquema anterior. No entrarem en detalls de codi ja que es pot consultar al codi font, però descriurem els components en alt nivell.

4.2.1 – Productor

El productor el està definit pel component *QueryHandler*, que es el punt d'entrada de la cerca definida per l'usuari. Aquest component esta format per 3 components més: *QueryResultProducer*, *QueryResultConsumer* i *WorkQueue.* Aquests tres components interns del productor també compleixen el paradigma de Productor-Consumidor, la millor opció per deslligar la obtenció de dades de gran volum amb el seu processament, com hem comentat abans.

El productor intern, QueryResultProducer fa servir una interfície *Querier*, que sap com executar una cerca a *Twitter Search API*. Executa la cerca definida per l'usuari (fent servir la API de *Twitter,* veure 3.2.4) i guarda els resultats a la *WorkQueue*.

La resposta a la cerca està dividida en grups que contenen un número concret de resultats (configurable abans de realitzar la cerca), i també conté metadades sobre aquest resultat que ens diuen

si hi ha més grups de resultats i si, per tant, hem de demanar al *Querier* més resultats.

La interfície *Querier* està implementada pel component *TwitterQueryImpl*, que es qui realment sap com fer la cerca a *Twitter.* La raó de fer servir una interfície dins el productor intern i no directament la classe que la implementa es que aixi deslliguem la implementació interna del component que fa les cerques amb la del productor que el fa servir. D'aquesta manera si volem canviar la implementació (per que trobem una forma millor de fer-la, algun component d'una llibreria existent que funciona millor, etc.) només hem de modificar un component, mentre la implementació compleixi el contracte definit a la interfície, tot continuarà funcionant.

Per la banda del consumidor intern, *QueryResultConsumer*, també es fa servir una interfície que es diu *MessageSender*, que sap com enviar un missatge a la cua externa. El consumidor l'únic que fa es llegir un missatge de la cua interna, extreure els *Tweets* (en format *Status* [29], que es la implementació del *Tweet* a la *twitter4j API*) i enviar-los un per un a la cua externa.

La implementació del *MessageSender* es descriurà al punt 3.2.3, on es parla de la implementació del sistema de cues que hem dissenyat pel cas.

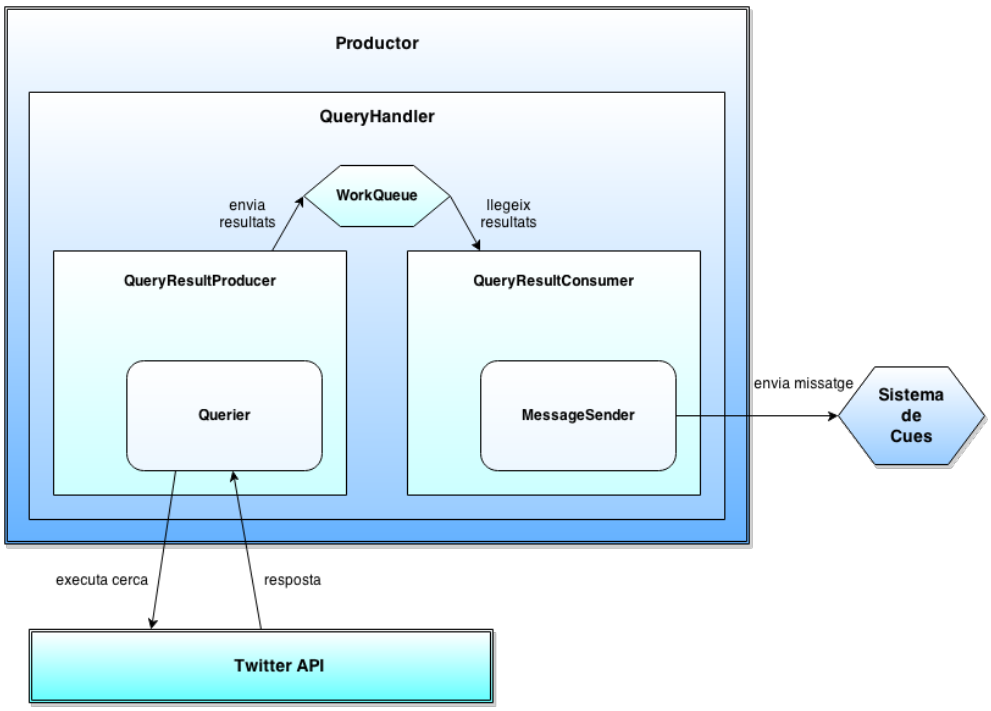


Fig. 4.2.1.1 – Esquema del Productor

4.2.2 – Consumidor

El Consumidor es la part més complexa de tota l'aplicació ja que es la que fa la feina bruta de processar les dades que ha generat el Productor, busca la informació relacionada amb els *tweets* que no contenen els *Status* i, finalment, l'emmagatzema a la ontologia OWL. El component que implementa el Consumidor es diu

MessageHandler i gestiona els 3 subcomponents dels que està format: *MessageReader, TwitterManager i OwlService.*

Igual que al Productor, *MessageReader* es una interfície de la que el Consumidor només l'interessa que sap com llegir missatges del sistema de cues (més detalls a 3.2.3). Es, per tant, el punt d'entrada de les dades a processar.

El flux intern del Consumidor es ben simple: llegeix un missatge fent servir *MessageReader,* el processa i omple els buits de dades fent servir *TwitterManager* i finalment guarda la informació a la ontologia fent servir *OWLService*. Va repetint aquest proces mentre quedin missatges per llegir a la cua.

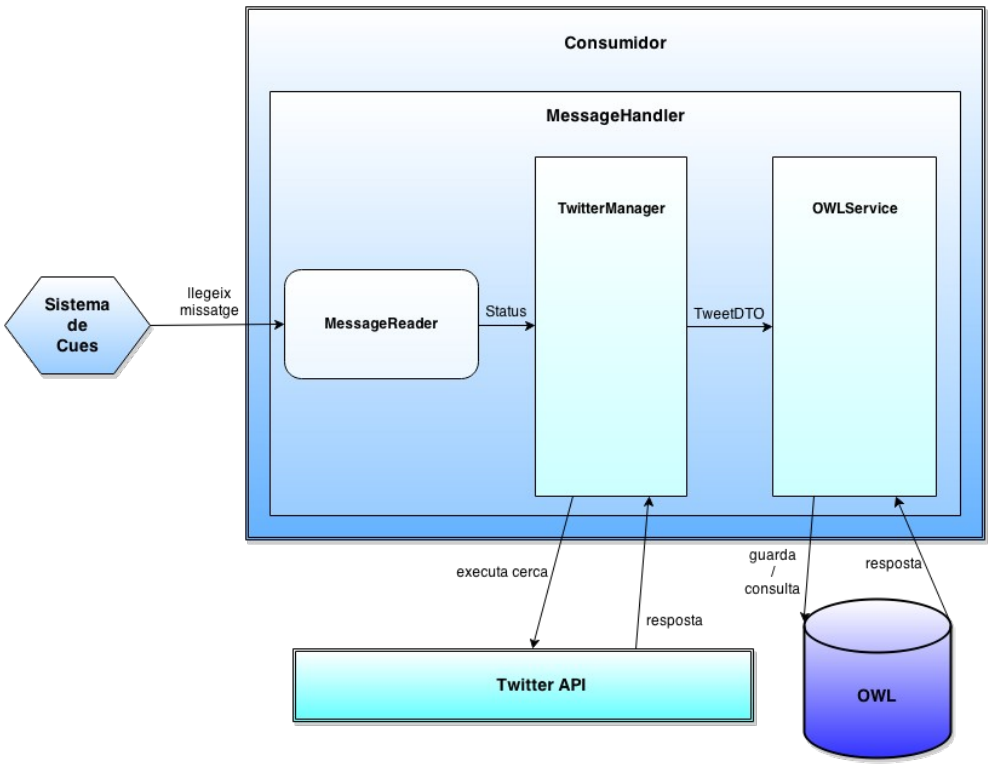


Fig. 4.2.2.1 – Esquema del Consumidor

Parlem ara dels dos components principals del Consumidor amb detall:

4.2.2.1 – TwitterManager

Com hem comentat abans, el component que processa les dades que porta el MessageReader (en format de *Status*, la versió lleugera del *Tweet* amb la que treballa la *twitter4j API*), omple els buits i construeix els DTO (acrònim de *Data Transfer Object*) que formaran el *TweetDTO que finalment farà servir el component* OWLService.

Al principi pensàvem fer servir classes que representessin les entitats en joc (*Tweet*, *User, Place, Entity*...), però vam decidir optar per fer servir només els *DTO*s. Aquestes son unes classes molt semblant a les que representen les entitats del món real, però orientades a la transferència de dades, molt més lleugeres i

normalment amb els atributs en format String i tipus de dades primaris de java. Vam prendre aquesta decisió basant-nos en la feina que comportaria processar els *Status* per passar-los a *DTO*s, tornar a processar aquests per construir les classes entitats i tornar a construir els *DTO*s per que el component *OWLService* els fes servir per emmagatzemar les dades a OWL.

Com dèiem, el *TwitterManager* processa les dades dels *Status* (internament en format *JSON*) per construir els *DTO*s i fa servir 3 components *DAO* (*Data Access Object*) per demanar la informació que falta a *Twitter* mitjançant la llibreria *twitter4j* [30].

En un principi vam pensar en crear un *DAO, també,* per a cada entitat que trobem a *Twitter* però mentre anàvem implementant aquesta part de l'aplicació, i coneixent més en profunditat la informació continguda als *Status*, vam veure que moltes de les entitats que es poden trobar en un *Tweet* ja venien incloses en aquest. L'única informació extra que ens faltava era sobre els *User* (necessitàvem trobar els *followers* i els *friends*, a més de la informació sobre la *Url* que pot contenir al seu perfil), sobre les *Place* (l'*Status* només contenia el *id* de la *Place*) i dels mateixos *Tweet* (en els casos de ser un *reply* o un *retweet* només teníem el *id* del *Tweet* original). Per tant només hem deixat aquests tres *DAO*s: *TweetDAO, UserDAO i PlaceDAO.*

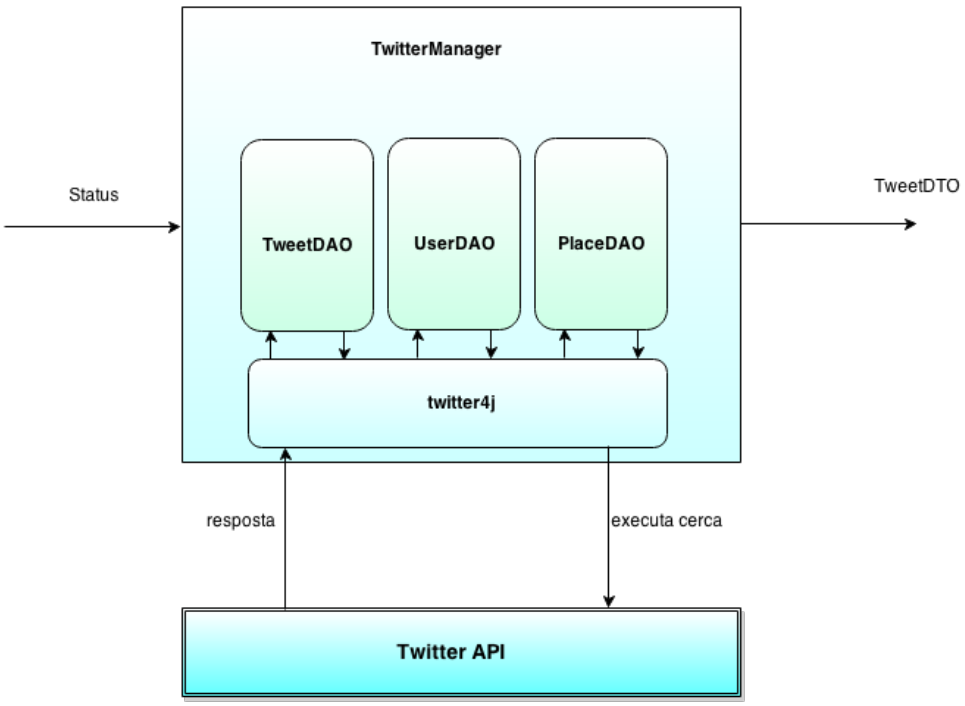


Fig. 4.2.2.1.1 – Esquema de TwitterManager

4.2.2.2 – OWLService

Un cop tenim el *TweetDTO* amb totes les dades que volem emmagatzemar a la ontologia entra en joc el component *OWLService*.

Aquest servei, bàsicament, fa servir el OWLDAO per crear instàncies de cada una de les classes de la nostra ontologia que representa una entitat al món real de *Twitter*. També el fa servir per crear (i assignar) els valors de les propietats de tipus de dades d'aquestes instàncies i les relacions entre elles i la resta de d'instàncies, creant i assignant, també, les propietats de tipus objecte que les relaciona.

En realitat aquest component *OWLDAO* estén un altre, *BaseDAO*, que es qui en realitat sap com crear instancies i propietats, omplir valors i relacions, buscar instàncies existents, carregar la ontologia, guardar els canvis i tota la resta d'operacions que es poden fer amb les instàncies de la ontologia. L'*OWLDAO* fa servir aquestes accions per crear, omplir, relacionar i guardar les instancies amb la informació que contenen els *DTO*s.

Per realitzar totes aquestes operacions, el *BaseDAO*, fa servir la llibreria java per tractar una ontologia en format OWL, OWL API de sourceforge [31].

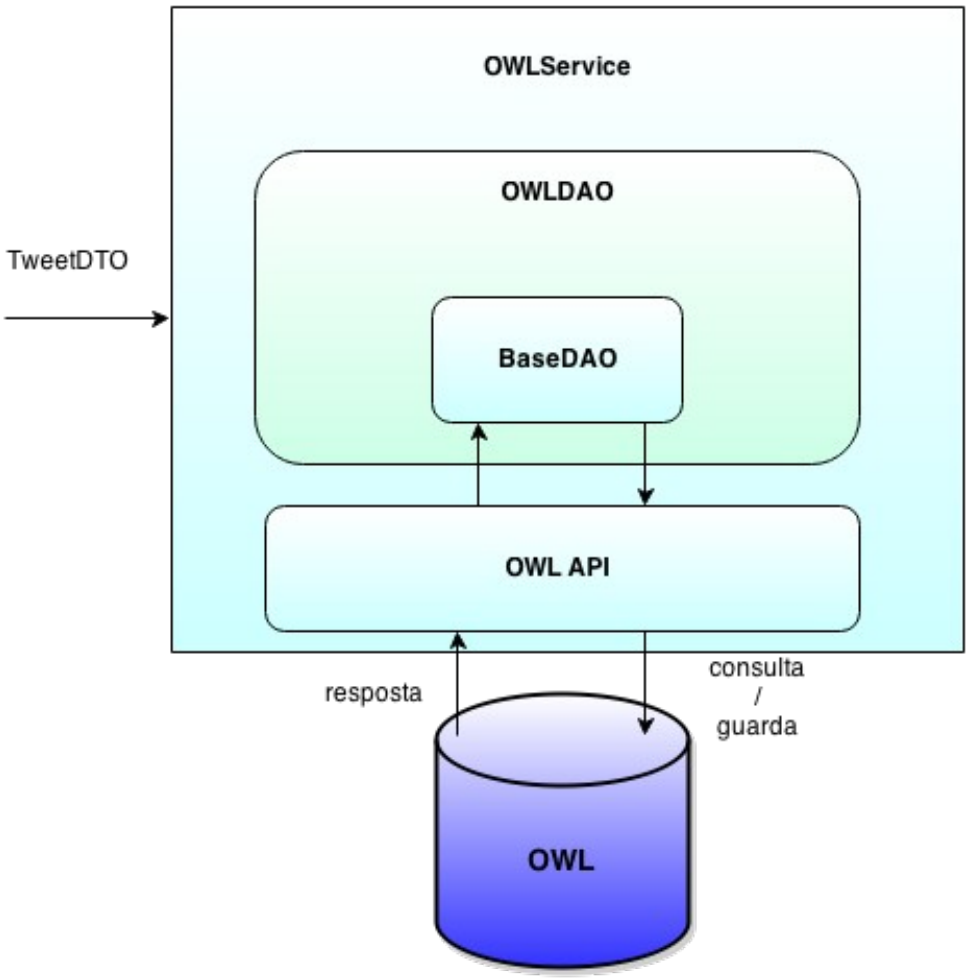


Fig. 4.2.2.2.1 – Esquema de OWLService

4.2.3 – Sistema de Cues

El sistema de cues que s'ha implementat per aquesta aplicació ha sigut el més senzill possible. En part per falta de temps i en part per les necessitats concretes de l'aplicació.

La implementació consta d'un component que es diu *SingletonQueue*. Està format per una cua (implementada amb un *ArrayBlockingQueue*) i te dos mètodes: *put* i *read*. Es un *singleton*, que vol dir que només es pot tenir una instància d'aquesta classe, que es comparteix entre els diferents components del sistema.

Per altra banda tenim la implementació del *MessageSender*, *SingletonQueueMessageSender*, i la implementació del *MessageReader*, *SingletonQueueMessageReader.* Aquestes dues implementacions comparteixen la mateixa instancia de SingletonQueue on envien missatges i els llegeixen respectivament.

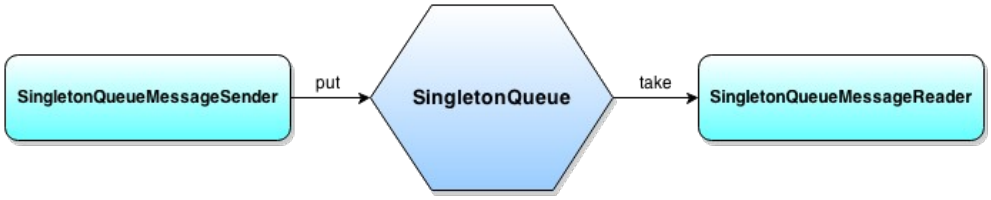


Fig. 4.2.3.1 – Esquema del sistema de cues

S'ha escollit aquest esquema fent servir les interfícies al Productor i al Consumidor per que hi ha una amplia gama de possibilitats per triar un sistema de cues segons les necessitats de fiabilitat i escalabilitat del sistema: es pot fer servir aplicacions de tercers (com per exemple activeMQ [32]) o millor, fer servir sistemes de cues al núvol (per exemple Cloud Queue de Rackspace [33] o Amazon SQS [34]).

Si es vol canviar el sistema de cues, simplement es poden crear noves implementacions de les interfícies *MessageSender* i *MessageReader* que facin servir els sistemes anomenats abans, o qualsevol altre, i tota la aplicació continuarà funcionant correctament.

4.2.4 – Twitter API

El component anomenat *Twitter API* de l'esquema general no es més que la API pròpia de *Twitter*, a la que accedeix la llibreria escollida per obtenir informació sobre els *Tweets*, ja sigui a la cerca inicial com a la posterior consulta per omplir les dades no obtingudes en aquesta. Per a més informació es pot consultar la documentació oficial [21].

4.2.5 – OWL

L'ultim component de l'esquema general que ens falta per comentar es l'anomenat *OWL*, que fa referència a la ontologia creada per representar el domini escollit per aquest treball: ontologia de *Twitter* (veure apartat 3).

4.3 Implementació

A continuació parlarem de la implementació d'alguns aspectes importants per cada component descrit al disseny:

4.3.1 – Productor

Tal i com s'explica al apartat del disseny, el Productor esta format per dos threads interns, un fa de productor intern de resultats de consulta a *Twitter Search API* i l'altre es el consumidor intern, i es comuniquen per una cua interna.

El productor executa queries a *Twitter* mentre quedin més resultats per demanar i no arribem al limit de resultats marcat per l'usuari (si l'usuari no marca límit, queryLimit=-1, continuarà fins portar tots els resultats de la consulta. Els resultats els va posant a la cua interna:

```
while (!noMoreResults && queryLimit!=-1?resultsProcessed<queryLimit:true)
{

    try {
         result = twitterQuerier.executeQuery(query);
         if (result.hasNext()) query=result.nextQuery();
         resultsProcessed = resultsProcessed + result.getCount();
         noMoreResults=!result.hasNext();
         workQueue.put(result);
        }catch (TwitterException e) {
         TwitterManager.handleTwitterException(e);
        }
}
//sending no more results signal
result = new PoisonedPillQueryResult();
workQueue.put(result);
```

Quan arribem al límit o no quedin més resultats per demanar, el productor envia un resultat a la cua (en forma d'implementació buida de *QueryResult, PoisonedPillQueryResult*) com a senyal de final de la consulta.

El tractament de la *TwitterException* es comentarà a l'apartat 4.3.2.1.

La cua interna no es més que una *LinkedBlockingQueue* d'elements *QueryResult*:

```
final LinkedBlockingQueue<QueryResult> workQueue = new
LinkedBlockingQueue<QueryResult>(50);
```

El consumidor intern llegeix de la cua interna i va enviant els *Tweets* (en format de *Status*) a la cua externa per ser processats posteriorment.

```
while (true){
    try {
           QueryResult result = workQueue.take();
           if (result.getTweets()!=null){
             sendMessages(result.getTweets());
           } else {
             log.info("No more results");
             //sending no more results signal
             sendMessages(result.getTweets());
             break;
           }
         } catch (InterruptedException e) {
```

```
            log.error("Interrupted! ", e);
            this.interrupt();
          }
 }
```

Repeteix aquest procés fins que arriba el senyal de final de consulta (el métode *getTweets()* del *QueryResult* restorna *null*), que es quan enviarà una implementació buida de *Status,* anomenada *PoisonedPillStatus,* a la cua externa:

```
if (processedResults!=null) {
     for (Status processedResult : processedResults) {
          messageSender.sendMessage(processedResult);
          messagesSent++;
     }
     log.info("Sent " + messagesSent + " messages to the Queue.");
} else {
     log.info("No more messages to send");
     messageSender.sendMessage(new PoisonedPillStatus());
}
```

En aquest punt el Productor ha acabat amb la seva feina i el procés es para.

El component que fa la consulta a *Twitter* es diu *TwitterQuerierImpl*, implementant la interfície *Querier* que l'obliga a tenir el mètode *executeQuery().* Aquest component només fa servir una classe *Twitter* de *twitter4j* per fer les consultes:

```
...
twitter = TwitterFactory.getSingleton();
...
QueryResult result = twitter.search(pQuery);
```

4.3.2 – Consumidor

El consumidor s'encarrega de llegir els missatges de la cua externa i processar-los:

```
TweetDTO tweet = twitterManager.process(message);
log.info("Processed tweet: "+tweet.getTweetExtract());
owlService.store(tweet);
log.info("Tweet stored.");
```

Com es veu, el processament del missatge corre a càrrec de *TwitterManager*, que genera un *TweetDTO* a partir del missatge processat, i del *OWLService* que s'encarrega d'emmagatzemar la informació que conté a la ontologia.

4.3.2.1 – TwitterManager

El *TwitterManager* omple els atributs de *TweetDTO* fent servir les dades contingudes dins d'un *Status:*

```
TweetDTO tweet = new TweetDTO();

tweet.setAuthor(parseAuthor(pOriginalStatus.getUser()));
tweet.setCreated(pOriginalStatus.getCreatedAt());
tweet.setFavouritesCount(pOriginalStatus.getFavoriteCount());
tweet.setCoordinates(parseCoordinates(pOriginalStatus));
tweet.setId("" + pOriginalStatus.getId());
tweet.setLang(pOriginalStatus.getLang());
tweet.setRetweetsCount(pOriginalStatus.getRetweetCount());
tweet.setSource(pOriginalStatus.getSource());
tweet.setText(pOriginalStatus.getText());
```

També n'extreu les *Entities* del *Status*:

```
tweet.setMedias(fetchMediaEntities(pOriginalStatus));
tweet.setHashtags(fetchHashtagEntities(pOriginalStatus));
tweet.setUrls(fetchUrlEntities(pOriginalStatus));
tweet.setUserMentions(fetchUserMentionEntities(pOriginalStatus));
```

Mira si té alguna *Place* associada:

```
tweet.setRelatedPlace(parsePlace(pOriginalStatus));
```

I comprova si es un *retweet* o un *reply* i fa el mateix procés recursivament per omplir el *TweetDTO* que representarà el *Tweet* al que es respon o es fa *retweet*:

```
if (!pIsRetweetOrReply)
tweet.setTweetReplied(parseReplyStatus(pOriginalStatus));

if (!pIsRetweetOrReply)
tweet.setRetweeted(parseRetweetedStatus(pOriginalStatus));
```

Aquest control *if* es per evitar que la recursivitat avanci més d'un nivell, complint amb les especificacions de disseny comentades previament. Es a dir que els mètodes *parseReplyStatus* i *parseRetweetStatus* faran una crida a aquest mateix mètode de process dels *Status* pero amb el control *pIsRetweetOrReply* a *true*, i per tant aquestes dues sentències no s'executaràn per anar a trobar el *reply/retweet* del *reply/retweet*.

Un punt important d'aquest *TwitterManager* es el maneig de la *TwitterException*, que fa servir la *API* de *Twitter* per indicar que s'ha superat el màxim de peticions per període de temps i que ens hem d'esperar per continuar fent peticions, això es fa en el mètode estàtic *handleTwitterException()*:

```
log.warn("Twitter API rate limit exceeded");
RateLimitStatus rateLimitStatus =
pTwitterException.getRateLimitStatus();
int waitingTime=10;
if (rateLimitStatus!=null){
   waitingTime = rateLimitStatus.getSecondsUntilReset();
}
log.warn("Waiting "+ waitingTime + " seconds for retry.");
long timeToRetry = waitingTime * 1000l;
Thread.sleep(timeToRetry);
```

Com es pot veure, la mateixa *Exception* conté la informació per saber quant de temps hem d'esperar per tornar a fer una consulta, encara que de vegades no conté aquesta dada i hem definit 10 segons d'espera per tornar a provar.

4.3.2.2 - OWLService

El component que s'encarrega de guardar la informació en la ontologia es *OWLService*, que fa servir el *OWLDAO* per fer-ho:

```
//process Tweet and store it and all it's inner classes into OWL:
OWLIndividual owlIndividualTweet = OWLDAO.storeFullTweet(tweet);

//store Tweet replied (with all its entities) and link with this
tweet
if (tweet.getTweetReplied()!=null) {
    OWLIndividual owlIndividualTweetReplied =
    OWLDAO.storeFullTweet(tweet.getTweetReplied());
    OWLDAO.setObjectProperty("inReplyOfTweet", owlIndividualTweet,
    owlIndividualTweetReplied);
}
```

```
//store Tweet retweeted (with all its entities) and link with this
tweet
if (tweet.getRetweeted()!=null) {
    OWLIndividual owlIndividualTweetRetweeted =
    OWLDAO.storeFullTweet(tweet.getRetweeted());
    OWLDAO.setObjectProperty("isRetweet", owlIndividualTweet,
    owlIndividualTweetRetweeted);
}

OWLDAO.saveOntology();
```

Si el *Tweet* es un *reply/retweet,* també guarda l'original i crea la propietat objecte *inReplyOfTweet* o *isRetwet* que els relaciona.

El nucli de *OWLDAO*, qui fa les operacions bàsiques sobre la ontologia, es la classe abstracta *BaseDAO*, qui s'encarrega d'obrir la ontologia, crear instàncies de classes, les seves propietats objecte i de tipus de dades i assignar-hi els valors fent servir assercions:

```
public OWLIndividual createNewClassIndividual(String pClassName,
String pIndividualName){
   OWLClass owlClass = owlDataFactory.getOWLClass(":"+pClassName,
   prefixManager);
   OWLNamedIndividual owlIndividual =
   owlDataFactory.getOWLNamedIndividual(":" + pIndividualName,
   prefixManager);
   OWLClassAssertionAxiom classAssertion =
   owlDataFactory.getOWLClassAssertionAxiom(owlClass, owlIndividual);
   manager.addAxiom(twitterOWL, classAssertion);
   return owlIndividual;
}
...
public void setObjectProperty(String pObjectPropertyName,
OWLIndividual pIndividualA, OWLIndividual pIndividualB){
   OWLObjectProperty objectProperty =
   owlDataFactory.getOWLObjectProperty(pObjectPropertyName,prefixMana
   ger);
   OWLObjectPropertyAssertionAxiom objectPropertyAssertion =
   owlDataFactory.getOWLObjectPropertyAssertionAxiom(objectProperty,p
   IndividualA, pIndividualB);
   manager.addAxiom(twitterOWL, objectPropertyAssertion);
}
...
public void setIntDataTypeProperty(String pPropertyName, int
pPropertyValue, OWLIndividual pNamedIndividual) {
   OWLDataPropertyExpression idProperty =
   owlDataFactory.getOWLDataProperty(pPropertyName,prefixManager);
   OWLLiteral propertyValue =
   owlDataFactory.getOWLLiteral(pPropertyValue);
   OWLDataPropertyAssertionAxiom assertion =
   owlDataFactory.getOWLDataPropertyAssertionAxiom(idProperty,
   pNamedIndividual, propertyValue);
   manager.addAxiom(twitterOWL,assertion);
}
```

Llavors *OWLDAO* estén aquesta classe abstracta i fa servir aquests mètodes per crear els individus i omplir els seus atributs amb les dades que li arriba dins cada *TweetDTO*:

```
...
//createdAt
setDateTimeDataProperty("createdAt", tweet.getCreated().toString(),
owlIndividualTweet);
...



//create OLW Tweet Class instance, fill all its DataType properties
and store it in ontology
OWLIndividual owlIndividualTweet = storeTweet(tweet);
//create OLW User Class instance, fill all its DataType properties
```

```
    and store it in ontology
    OWLIndividual owlIndividualUser = storeUser(tweet.getAuthor());
    //set Tweet.authoredBy Object property with the User instance
    setObjectProperty("authoredBy",owlIndividualTweet,owlIndividualUser);
    //set User.isAuthor Object property with the Tweet instance
    setObjectProperty("isAuthor",owlIndividualUser,owlIndividualTweet);
```

4.3.3 – Sistema de cues

La implementació del sistema de cues es molt simple, es composa d'una cua que no es més que un component singleton *SingletonQueue* que embolica:

```
private ArrayBlockingQueue<Status> messageQueue;
```

I te dos mètodes públics per enviar un missatge a la cua i per llegir un missatge de la cua:

```
/**
 * Send a message to the queue
 * @param pMessage Status to put in the queue
 * @throws InterruptedException
 */
public void put(Status pMessage) throws InterruptedException {
    messageQueue.put(pMessage);
    log.debug("message in q");
    log.debug(messageQueue.toString());
}

/**
 * Read a message from queue
 * @return Status message read from the queue
 * @throws InterruptedException
 */
public Status read() throws InterruptedException {
    log.debug("Reading message from q");
    return messageQueue.take();
}
```

Els components que implementen les interfícies per llegir i escriure missatges a la cua, *SingletonQueueMessageReader* i *SingletonQueueMessageSender,* simplement fan servir aquesta els mètodes anteriors per posar o agafar un missatge de la cua.

4.4 Comprovació dades inserides

Un cop executada l'aplicació podem veure les dades que s'han inserit a la ontologia. La figura següent mostra el número d'individus de cada classe que s'han inserit amb executant l'aplicació amb una query curta (uns 50 resultats):

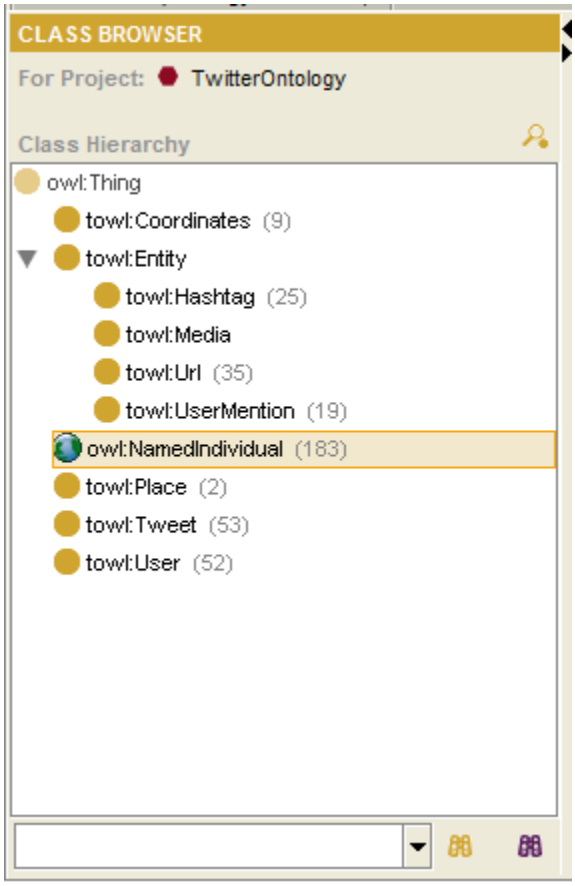


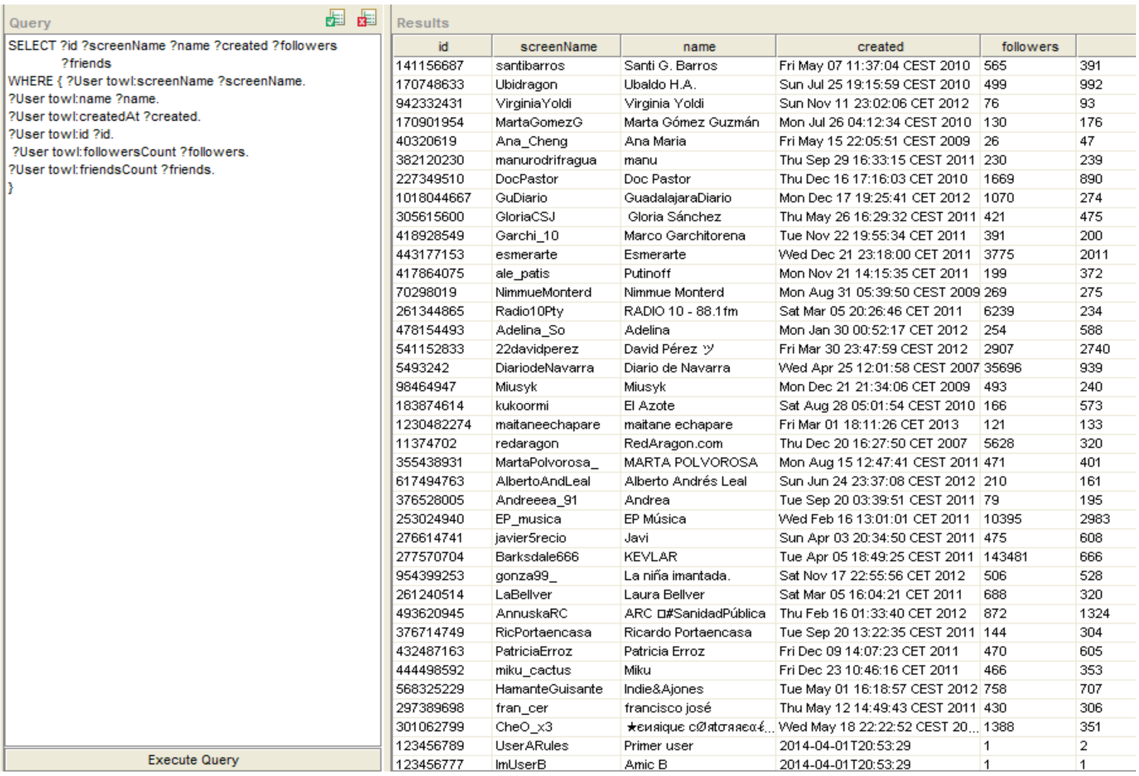
Fig 4.4.1 – Número d'individus inserits de cada classe.

Aquesta classe estranya que es veu, *owl:NamedIndividual*, s'ha generat automàticament a la ontologia des de l'aplicació. Això es degut a la diferencia de versions de *OWL* entre *Protégé* i *owlapi* (veure 4.1, punt 1)
A continuació presentem alguns exemples de les dades inserides fent servir *SPARQL* a *Protégé:*

4.4.1 – Users

Alguns dels usuaris inserits a la ontologia els podem veure a la figura 4.4.1.1, la sentència *SPARQL* utilitzada es la següent:

```
SELECT ?id ?screenName ?name ?created ?followers ?friends
WHERE { ?User towl:screenName ?screenName.
?User towl:name ?name.
?User towl:createdAt ?created.
?User towl:id ?id.
?User towl:followersCount ?followers.
?User towl:friendsCount ?friends.
}
```



Query

```
SELECT ?id ?screenName ?name ?created ?followers
        ?friends
WHERE { ?User towl:screenName ?screenName.
?User towl:name ?name.
?User towl:createdAt ?created.
?User towl:id ?id.
 ?User towl:followersCount ?followers.
?User towl:friendsCount ?friends.
}
```

Execute Query

Results

id	screenName	name	created	followers	
141156687	santibarros	Santi G. Barros	Fri May 07 11:37:04 CEST 2010	565	391
170748633	Ubidragon	Ubaldo H.A.	Sun Jul 25 19:15:59 CEST 2010	499	992
942332431	VirginiaYoldi	Virginia Yoldi	Sun Nov 11 23:02:06 CET 2012	76	93
170901954	MartaGomezG	Marta Gómez Guzmán	Mon Jul 26 04:12:34 CEST 2010	130	176
40320619	Ana_Cheng	Ana Maria	Fri May 15 22:05:51 CEST 2009	26	47
382120230	manurodrifragua	manu	Thu Sep 29 16:33:15 CEST 2011	230	239
227349510	DocPastor	Doc Pastor	Thu Dec 16 17:16:03 CET 2010	1669	890
1018044667	GuDiario	GuadalajaraDiario	Mon Dec 17 19:25:41 CET 2012	1070	274
305615600	GloriaCSJ	Gloria Sánchez	Thu May 26 16:29:32 CEST 2011	421	475
418928549	Garchi_10	Marco Garchitorena	Tue Nov 22 19:55:34 CET 2011	391	200
443177153	esmerarte	Esmerarte	Wed Dec 21 23:18:00 CET 2011	3775	2011
417864075	ale_patis	Putinoff	Mon Nov 21 14:15:35 CET 2011	199	372
70298019	NimmueMonterd	Nimmue Monterd	Mon Aug 31 05:39:50 CEST 2009	269	275
261344865	Radio10Pty	RADIO 10 - 88.1fm	Sat Mar 05 20:26:46 CET 2011	6239	234
478154493	Adelina_So	Adelina	Mon Jan 30 00:52:17 CET 2012	254	588
541152833	22davidperez	David Pérez ツ	Fri Mar 30 23:47:59 CEST 2012	2907	2740
5493242	DiariodeNavarra	Diario de Navarra	Wed Apr 25 12:01:58 CEST 2007	35696	939
98464947	Miusyk	Miusyk	Mon Dec 21 21:34:06 CET 2009	493	240
183874614	kukoormi	El Azote	Sat Aug 28 05:01:54 CEST 2010	166	573
1230482274	maitaneechapare	maitane echapare	Fri Mar 01 18:11:26 CET 2013	121	133
11374702	redaragon	RedAragon.com	Thu Dec 20 16:27:50 CET 2007	5628	320
355438931	MartaPolvorosa_	MARTA POLVOROSA	Mon Aug 15 12:47:41 CEST 2011	471	401
617494763	AlbertoAndLeal	Alberto Andrés Leal	Sun Jun 24 23:37:08 CEST 2012	210	161
376528005	Andreeea_91	Andrea	Tue Sep 20 03:39:51 CEST 2011	79	195
253024940	EP_musica	EP Música	Wed Feb 16 13:01:01 CET 2011	10395	2983
276614741	javier5recio	Javi	Sun Apr 03 20:34:50 CEST 2011	475	608
277570704	Barksdale666	KEVLAR	Tue Apr 05 18:49:25 CEST 2011	143481	666
954399253	gonza99_	La niña imantada.	Sat Nov 17 22:55:56 CET 2012	506	528
261240514	LaBellver	Laura Bellver	Sat Mar 05 16:04:21 CET 2011	688	320
493620945	AnnuskaRC	ARC □#SanidadPública	Thu Feb 16 01:33:40 CET 2012	872	1324
376714749	RicPortaencasa	Ricardo Portaencasa	Tue Sep 20 13:22:35 CEST 2011	144	304
432487163	PatriciaErroz	Patricia Erroz	Fri Dec 09 14:07:23 CET 2011	470	605
444498592	miku_cactus	Miku	Fri Dec 23 10:46:16 CET 2011	466	353
568325229	HamanteGuisante	Indie&Ajones	Tue May 01 16:18:57 CEST 2012	758	707
297389698	fran_cer	francisco josé	Thu May 12 14:49:43 CEST 2011	430	306
301062799	CheO_x3	★єияique c∅яtσяaєα£...	Wed May 18 22:22:52 CEST 20...	1388	351
123456789	UserARules	Primer user	2014-04-01T20:53:29	1	2
123456777	ImUserB	Amic B	2014-04-01T20:53:29	1	1

Fig. 4.4.1.1 Exemple d'Users inserits

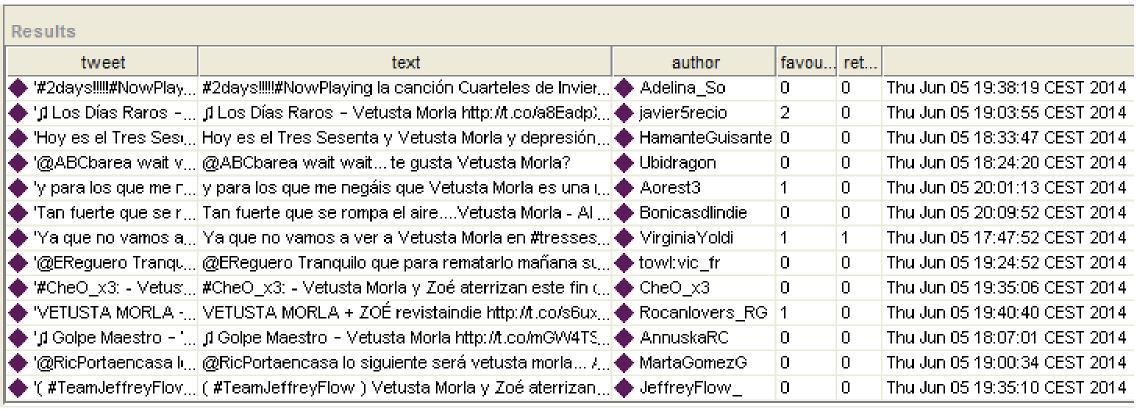
Mostrem només alguns dels camps de la classe *User* ja que si els mostréssim tots la imatge no seria gaire clara.

4.4.2 – Tweets

Fent servir la query següent:

```
SELECT ?tweet ?text ?author ?favourites ?retweets ?created
WHERE{ ?tweet towl:text ?text;
towl:authoredBy ?author;
towl:favouritesCount ?favourites;
towl:retweetsCount ?retweets;
towl:createdAt ?created.
}
```

Obtenim els resultats de la figura:



Results

tweet	text	author	favou...	ret...	
'#2days!!!!!#NowPlay...	#2days!!!!!#NowPlaying la canción Cuarteles de Invier...	Adelina_So	0	0	Thu Jun 05 19:38:19 CEST 2014
'♫ Los Días Raros -...	♫ Los Días Raros – Vetusta Morla http://t.co/a8Eadp...	javier5recio	2	0	Thu Jun 05 19:03:55 CEST 2014
'Hoy es el Tres Ses...	Hoy es el Tres Sesenta y Vetusta Morla y depresión...	HamanteGuisante	0	0	Thu Jun 05 18:33:47 CEST 2014
'@ABCbarea wait v...	@ABCbarea wait wait... te gusta Vetusta Morla?	Ubidragon	0	0	Thu Jun 05 18:24:20 CEST 2014
'y para los que me r...	y para los que me negáis que Vetusta Morla es una ...	Aorest3	1	0	Thu Jun 05 20:01:13 CEST 2014
'Tan fuerte que se r...	Tan fuerte que se rompa el aire....Vetusta Morla - Al ...	Bonicasdlindie	0	0	Thu Jun 05 20:09:52 CEST 2014
'Ya que no vamos a...	Ya que no vamos a ver a Vetusta Morla en #tresses...	VirginiaYoldi	1	1	Thu Jun 05 17:47:52 CEST 2014
'@EReguero Tranq...	@EReguero Tranquilo que para rematarlo mañana s...	towl:vic_fr	0	0	Thu Jun 05 19:24:52 CEST 2014
'#CheO_x3: - Vetus...	#CheO_x3: - Vetusta Morla y Zoé aterrizan este fin ...	CheO_x3	0	0	Thu Jun 05 19:35:06 CEST 2014
'VETUSTA MORLA -...	VETUSTA MORLA + ZOÉ revistaindie http://t.co/s6ux...	Rocanlovers_RG	1	0	Thu Jun 05 19:40:40 CEST 2014
'♫ Golpe Maestro – '...	♫ Golpe Maestro – Vetusta Morla http://t.co/mGW4TS...	AnnuskaRC	0	0	Thu Jun 05 18:07:01 CEST 2014
'@RicPortaencasa l...	@RicPortaencasa lo siguiente será vetusta morla... ...	MartaGomezG	0	0	Thu Jun 05 19:00:34 CEST 2014
'(#TeamJeffreyFlow...	(#TeamJeffreyFlow) Vetusta Morla y Zoé aterrizan...	JeffreyFlow_	0	0	Thu Jun 05 19:35:10 CEST 2014

Fig. 4.4.2.1 Exemple de Tweets inserits

4.4.3 – Place i Coordinates

Per veure les dos instancies de *Place* i les seves *Coordinates*, farem servir la següent query:

```
SELECT ?place ?attributes ?nom ?latitude ?longitude
WHERE{ ?place towl:attributes ?attributes;
towl:fullName ?nom.
?coordinates towl:inPlace ?place;
towl:latitude ?latitude;
towl:longitude ?longitude.
}
```

Que dona per resultat:

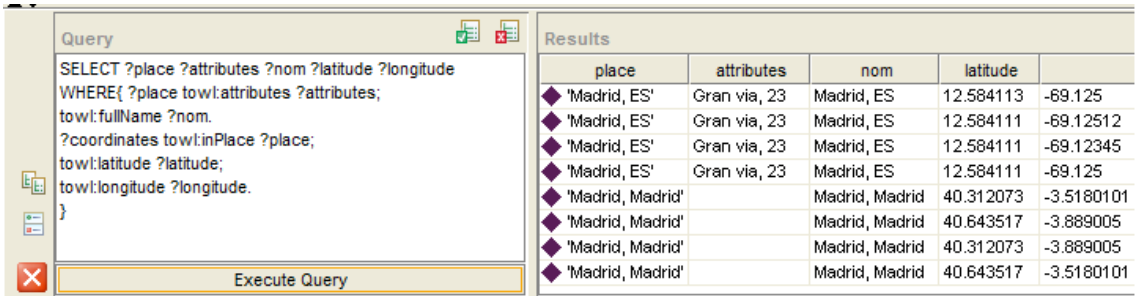


Fig. 4.4.3.1 Exemple de Places amb les seves Coordinates inserides

4.4.4 – UserMention

Les instàncies de *UserMention* les podem veure amb aquesta sentencia de SPARQL:

```
SELECT  ?name ?inici ?fi ?user ?tweet
WHERE{ ?userMention towl:screenName ?name;
towl:indexA ?inici;
towl:indexB ?fi;
towl:mentionsUser ?user;
towl:comesFrom ?tweet.
}
```

Que ens dona el resultat:

Results

name	inici	fi	user	tweet
VirginiaYoldi	3	17	VirginiaYoldi	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474578470425878528
esmerarte	125	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	125	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	125	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	125	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	125	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	125	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	125	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	125	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	125	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	139	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	139	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	139	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	139	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	139	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	139	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	139	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	139	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	139	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	127	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	127	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	127	135	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	127	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	127	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	127	140	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
esmerarte	127	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474591904349569025
esmerarte	127	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474590227395530752
esmerarte	127	137	towl:esmerarte	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474589566264168448
DiariodeNava...	3	19	DiariodeNavarra	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474578164480749568
maitaneechap...	87	122	towl:maitaneechapare	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474578470425878528
maitaneechap...	87	122	towl:maitaneechapare	'Ya que no vamos a ver a Vetusta Morla en #tressesenta habrá que escuchar el ens
maitaneechap...	87	103	towl:maitaneechapare	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474578470425878528
maitaneechap...	87	103	towl:maitaneechapare	'Ya que no vamos a ver a Vetusta Morla en #tressesenta habrá que escuchar el ens
maitaneechap...	106	122	towl:maitaneechapare	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474578470425878528
maitaneechap...	106	122	towl:maitaneechapare	'Ya que no vamos a ver a Vetusta Morla en #tressesenta habrá que escuchar el ens
maitaneechap...	106	103	towl:maitaneechapare	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474578470425878528
maitaneechap...	106	103	towl:maitaneechapare	'Ya que no vamos a ver a Vetusta Morla en #tressesenta habrá que escuchar el ens
AlbertoAndL...	3	18	towl:AlbertoAndLeal	http://www.owl-ontologies.com/OntologyTwitter.owl#TweetId: 474608739065274368

Fig 4.4.4.1 UserMention inserits.

4. Conclusions

Durant la realització d'aquest treball s'ha comprovat la gran ambició que comporta el concepte de la web semàntica i la gran feina que encara queda per fer, no només de cara a la definició dels estàndards, fins i tot a l'hora de posar-se d'acord els diferents corrents de la comunitat respecte a que és necessari i que no.
També ens hem adonat de la complexitat que te abordar el disseny d'una ontologia per primera vegada, encara que el domini no sigui gaire gran. Hem conegut la importància de la iteració en la construcció de la ontologia, ja que en fases posteriors hem vist que potser el disseny inicial era massa ambiciós i potser s'hauria pogut eliminar una mica més d'informació no rellevant pel cas que ens ocupa.
Tot i aixi, a pesar de que ens hem afrontat a reptes importants en tecnologies no gaire conegudes, penso que hem assolit tots els objectius plantejats inicialment (apartat 1.2), havent descartat per manca de temps l'objectiu opcional sobre crear una aplicació o un conjunt de regles sobre l'ontologia que permeti inferir informació nova a partir de la informació dels *tweets* emmagatzemats a la ontologia.
En tot projecte un punt molt crític des de l'inici es la planificació: es molt complicat planificar amb detall un projecte del que no se sap gairebé res de la majoria d'aspectes implicats en ell. Seguint el fil de la planificació de la mateixa assignatura en 3 parts, hem aconseguit fer un pla de treball bastant realista (tot i que el temps total disponible ha sigut molt reduït i la disponibilitat real no ha sigut la esperada per diferents motius) que fins a dia d'avui s'està respectant sense gaires desviacions (fora de lo normal en projectes d'aquest tipus). On hi ha hagut una desviació del pla més acusada ha sigut en la part de investigació sobre l'"estat de l'art" de la web semàntica: no ha sigut fàcil trobar informació coherent i confiable més enllà de la que ja oferia els materials de l'assignatura. Després alguns problemes relacionats amb la implementació de l'aplicació i l'adaptació de components de terceres parts (llibreries) han obligat a invertir-hi més temps del que es pensava inicialment.
Sempre els últims dies del projecte son els mes intensos de cara a no deixar caps per lligar però, tot i aixo, el pla i la metodologia escollida ha tingut força èxit.

4.1 Millores necessàries

Degut a la manca de temps ens trobem amb alguns problemes i punts a millorar que no s'han pogut solucionar:

1. **Problema de versions:** El principal problema que queda pendent de resoldre ha estat causat per una diferència de versions, no del tot compatibles entre si, del format *OWL* que fa servir *Protégé* 3.5 i el format OWL que fa servir la llibreria usada a l'aplicació per omplir la ontologia: owlapi 3.4.10. Aquesta diferencia de versions provoca que, un cop s'ha fet servir l'aplicació per omplir la ontologia de *tweets* aquesta es guarda en un format que pot entendre *Protégé* (com s'ha

pogut comprovar anteriorment en aquest document) però la mateixa aplicació es incapaç d'obrir-la de nou i continuar afegint dades.

2. **Millora pels fitxers de configuració:** Tant el fitxer de configuració de les credencials per accedir a la *API* de *Twitter*, com el de configuració dels *logs* de l'aplicació son a dins del fitxer *JAR* executable que conforma l'aplicació. S'haurien de poder treure fora i aixi poder canviar les propietats com es necessitin en cada moment.
3. **Millora d'escalabilitat:** El disseny de l'aplicació avança les possibilitats d'escalabilitat, fent servir interfícies en parts concretes com el sistema de cues per exemple, però la implementació no es gaire escalable, degut es clar a la falta de temps. Per ser més escalable hauria de poder executar-se els 3 components principals en diferents màquines, o per exemple implementar les interfícies per enviar missatges al sistema de cues per fer servir alguns dels sistemes més sofisticats i escalables que ja hem comentat en aquest document. També es podria afegir la possibilitat del *multithreading,* el consumidor per exemple podria tenir un número de fils configurable que llegissin de la cua i processessin els missatges en paral·lel.
4. **Millora per evitar la limitació de peticions a *Twitter*:** Es podria pensar en alguna forma per evitar, o disminuir en un grau alt, la limitació de peticions per espai de temps que té la *API* de *Twitter*. Ja sigui creant diferents comptes i alternar les seves credencials en cada cerca (o quan s'arriba al limit amb un compte, fer servir un altre), o buscant una solució més pactada amb la tercera part.

5. Glossari

- **Agent automàtic**: procés o programa d'ordinador que realitza alguna tasca automàticament.
- **API:** De l'anglés *Application Programming Interface*, conjunt de mètodes que ofereix una llibreria per ser utilitzada per un altre programari.
- ***Contributors:*** Concepte nou a *Twitter*, més d'un usuari podrà publicar un mateix *tweet* conjuntament.
- **Dades enllaçades (*linked data)*:** Mètode de publicació de dades estructurades interconnectades entre si, per facilitar la lectura i comprensió per agents automàtics.
- ***DAO (Data Access Object)*:** Paradigma de programació *java*, es tracta de classes que s'encarreguen de l'accés a dades emmagatzemades de qualssevol tipus, repositoris i formats.
- ***DTO (Data Transfer Object):*** Paradigma de programació java, son classes que representen alguna entitat al món real, pero de forma més lleugera per facilitar l'intercanvi i transferència de dades entre components d'una aplicació.
- **Escalabilitat:** Propietat d'un sistema informàtic que indica habilitat per canviar la seva mida o configuració per adaptar-se a circumstàncies canviants.
- **Estat de l'art:** Anglicisme derivat de *"state-of-art"* que significa, en tecnologia, l'estat actual d'aquesta o tecnologia puntera en algun tema concret.
- **Graf Global Gegant (*Giant Global Graph, GGG*):** Concepte que deriva de la *WWW* aplicat als desitjos de la web semàntica: unificar tota la *Web* en un graf gegant format per triplets, interconnectant tots els recursos disponibles.
- ***HTML (HyperText Markup Language)*:** Llenguatge d'etiquetes per hipertext. Es el llenguatge amb el que escriure la estructura de les pàgines web.
- **Interfície:** Contracte que pren una classe java, que defineix les accions que ha de implementar sense tenir en compte com s'ha fet aquesta implementació.
- ***JAR*:** Arxiu java (***Ja**va **Ar**chive*) comprimit que conté i permet executar aplicacions *java.*
- ***Java*:** Llenguatge de programació de proposit general, concurrent, orientat a objecte i basat en classes.
- **Metadades:** Conjunt de dades que conté informació sobre si mateix, com pot ser quantitat, tipus de dades, etc.
- ***Microblogging*:** servei de publicació de missatges breus via llocs web, SMS, missatgeria instantània, etc.
- ***Online*:** Anglicisme que vol dir "en línia", connectat a la xarxa.
- **Ontologia:** Exhaustiu i rigorós esquema conceptual dintre un o varis dominis de coneixement.

- ***OWL, Web Ontology Language*:** Llenguatge per definir ontologies sobre RDF i codificat en XML.
- **Owlapi:** *API d'OWL de sourceforge.*
- **Pila de la web semàntica (*Semantic Web Stack)*:** Disseny per capes del concepte de la web semàntica.
- ***Query*:** De l'anglés, significa consulta o cerca.
- **Reply:** A *Twitter* resposta al *tweet* d'un altre usuari.
- **Retweet:** A *Twitter*, reenviament o publicació del *tweet* d'un altre autor pels teus seguidors.
- **Seguidor (*follower*):** A *Twitter*, un altre usuari que llegeix (o al menys li arriben) els *tweets* que publica un usuari.
- **Seguit (*friend*):** A *Twitter*, usuari a qui un usuari segueix i, per tant, llegeix les seves publicacions.
- ***Singleton*:** A *java*, classe de la que només pot existir una sola instància que comparteixen la resta de classes que la fan servir.
- **Tesaurus:** Llista de paraules o termes empleats per representar conceptes.
- ***Thread*:** Fil d'execució d'un proces informàtic. Si el sistema operatiu i el llenguatge de programació es concurrent, permet de tenir varis fils a la vegada.
- ***Timeline*:** A *Twitter*, llistat de publicacions d'altres usuaris (seguits) que li arriba a un usuari concret.
- **Triplet:** expressió RDF de subjecte-predicat-objecte.
- **Tweet:** A *Twitter*, publicació d'un usuari de 140 caràcters màxim.
- **Twitter:** Servei de microblogging que permet publicar missatges de text de 140 caràcters.
- **Twitter4j:** llibreria per accedir a la *API* de *Twitter.*
- **UNICODE:** Estàndard de codificació de caràcters per facilitar el tractament informàtic, la transmissió i visualització de textos en qualsevol idioma.
- **URI, *Uniform Resource Identifier:*** cadena de caràcters curta que identifica univocament un recurs (servei, pàgina, document, etc).
- **URL, *Uniform Resource Locator*:** tipus de URI que serveix per identificar i localitzar recursos a Internet.
- **UserMention:** A *Twitter*, un usuari menciona a un altre dins un *tweet* que ha publicat.
- **W3C, *World Wide Web Consortium*:** Organisme internacional encarregat d'establir els estàndards web.
- ***Web 2.0,* web social:** llocs web que faciliten compartir informació i la col·laboració entre usuaris.
- **Web semàntica:** afegir metadades semàntiques i ontologies a la web actual per que agents automatics puguin trobar i entendre la informació.
- **WWW, *World Wide Web*:** Xarxa informàtica mundial.
- **XML, eX*tensible Markup Language*:** format flexible i extensible de llenguatge d'etiquetes per emmagatzemar dades en format llegible.

6. Bibliografia

[1] http://es.wikipedia.org/wiki/World_Wide_Web – 08/03/2014

[2] http://es.wikipedia.org/wiki/Tim_Berners-Lee – 08/03/2014

[3] Jordi Duran Cals, Jordi Conesa i Caralt, Robert Clarisó Viladrosa, "Representació del coneixement", Materials UOC.

[4] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web", Scientific American, pàg. 34-43, 2001, maig.

[5] http://es.wikipedia.org/wiki/Twitter – 09/03/2014

[6] http://www.w3.org/ -12/03/2014

[7] http://www.w3.org/RDF/ - 12/03/2014

[8] http://www.w3.org/People/Ivan/CorePresentations/SW_QA/Slides.html – 13/03/2014

[9] http://www.w3.org/2001/sw/wiki/RDFa -13/03/2014

[10] http://www.w3.org/2001/sw/wiki/SPARQL – 13/03/2014

[11] http://www.w3.org/OWL/ - 14/03/2014

[12] http://en.wikipedia.org/wiki/Semantic_Web – 20/03/2014

[13] http://en.wikipedia.org/wiki/Giant_Global_Graph – 20/03/2014

[14] Aurona Gerber, Alta van der Merwe, Andries Barnard, "A Functional Semantic Web Architecture" http://ksg.meraka.org.za/~agerber/Paper152.pdf

[15] http://en.wikipedia.org/wiki/Semantic_Web_Stack – 27/03/2014

[16] http://es.wikipedia.org/wiki/Microformato – 25/05/2014

[17] http://www.slideshare.net/signer/web-information-systems-wedinf11912-lecture-08-semantic-web – 25/05/2014

[18] http://en.wikipedia.org/wiki/GRDDL – 25/05/2014

[19] http://en.wikipedia.org/wiki/N-Triples – 25/05/2014

[20] http://stackoverflow.com/questions/11601856/what-is-unifying-logic-within-the-semantic-stack-model-and-who-is-supposed-to-ta – 25/05/2014

[21] https://dev.twitter.com/docs/platform-objects – 18/03/2014

[22] https://dev.twitter.com/docs/twitter-ids-json-and-snowflake – 18/03/2014

[23] https://dev.twitter.com/docs/platform-objects/tweets – 19/03/2014

[24] https://dev.twitter.com/docs/platform-objects/users – 19/03/2014

[25] https://dev.twitter.com/docs/platform-objects/entities – 19/03/2014

[26] https://dev.twitter.com/docs/platform-objects/places – 19/03/2014

[27] https://dev.twitter.com/docs/rate-limiting/1.1 – 20/03/2014

[28] http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html – 21/03/2014

[29] http://twitter4j.org/javadoc/twitter4j/Status.html – 18/04/2014

[30] http://twitter4j.org/en/index.html – 18/04/2014

[31] http://owlapi.sourceforge.net/ -18/04/2014

[32] http://activemq.apache.org/ - 18/04/2014

[33] http://www.rackspace.com/es/cloud/queues/ - 18/04/2014

[34] https://aws.amazon.com/es/sqs/ - 18/04/2014

7. Annexos

7.1 Instal·lació

Els requeriments necessaris per instal·lar aquesta aplicació son mínims:

- **Software:** es necessita que la màquina tingui instal·lat java 1.6 o superior.
- **Hardware:** després de fer proves exhaustives d'hores de funcionament (una cerca amb limit de 1000 resultats) s'ha comprovat que de consum de memòria no ha passat dels 500Mb i de consum de *CPU* no ha pujat del 5% (i només al principi de la execució, després no passava del 2%).

Per tant en qualsevol màquina domestica actual es pot executar l'aplicació.

Passos per la instal·lació:

1. S'ha de guardar l'arxiu de la ontologia (TwitterOntology.owl) en alguna carpeta coneguda per després poder fer servir la ruta a l'execució.
2. Guardar també el fitxer JAR executable en lloc conegut i accessible des de terminal.

7.2 Execució

L'aplicació està pensada per ser una llibreria ampliable per una interfície d'usuari a mida, però es pot executar individualment per línia de comandes. Si ens trobem a la carpeta on està guardat el JAR, haurem d'executar:

```
$>java -jar -Dontology.path=[ruta_fitxer_ontologia]/TwitterOntology.owl tfc-1.0-jar-with-dependencies.jar "[consulta que volem fer a Twitter]" [limit de resultats]
```

Adaptant les parts entre claudators a la nostra instal.lació i a la consulta que volem realitzar a *Twitter* i el límit de resultats que volem com a màxim.

Important: Per que l'aplicació funcioni s'ha de utilitzar una ontologia buida, per problemes de incompatibilitats entre versions (veure 4.1, punt 1). Si s'executa amb una ontologia que ja s'ha fet servir abans donarà error de format.

7.3 Codi font

El codi font de l'aplicació està publicat a gitHub:

https://github.com/esau/tfc

Es pot consultar tot el codi font a més de descarregar-ho i muntar el projecte en un IDE de java per millor consulta.