ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA · CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Progettazione e Realizzazione di una Applicazione per l'Estrazione, l'Analisi e l'Integrazione di Social Data e Linked Open Data

Tesi di Laurea Magistrale in Web Semantico

Relatore

Prof.ssa ANTONELLA CARBONARO

Presentata da

FABRIZIO MASINI

Sessione III

Anno accademico 2013/2014

Indice

Introduzione

Nell'ultimo decennio si è visto nascere un crescente interesse per le procedure di analisi e classificazione dei dati provenienti dai *Social Network*. Grazie ad una fiorente e continua espansione di questi, infatti, le aziende e gli enti di ricerca hanno iniziato a comprendere quanto sia elevato il potenziale che si cela dietro alle svariate moli di dati che ogni giorno vengono condivise su internet da milioni di utilizzatori.

Le nuove generazioni sono nate nel pieno del progresso tecnologico informatico, ed il mercato è stato inondato con la produzione di numerosi apparati innovativi. Smartphone, tablet, accessori ed elettrodomestici "intelligenti", hanno dato vita al concetto di *Internet Of Things (IoT)*, ovvero "l'internet delle cose". Questo meccanismo impone che ogni dispositivo venga dotato di un *indirizzo IP* e che possa scambiare dati attraverso la rete, per trasmettere informazioni o ricevere degli ordini.

La nascita di questi dispositivi e la continua ricerca in ambito di *domotica*, ha anche permesso di definire un nuovo concetto di *Smart City* [1]. Quest'ultima riguarda la trasformazione di processi di vita quotidiana, inserendo al loro interno l'utilizzo delle sopracitate tecnologie moderne, al fine di aumentare l'efficienza e diminuirne i costi di esecuzione, e più in generale, di incrementare la qualità della vita.

In secondo luogo, i suddetti apparati tecnologici hanno contribuito alla nascita di quello che viene denominato da alcuni, il *"**Web 3.0**"*.

Un concetto di Web [2], nel quale non sono più le aziende e le organizzazioni, le sole produttrici del contenuto informativo, ma sono gli stessi utenti "comuni" a divenire i creatori-fruitori dei contenuti.

Questo procedimento naturale, che caratterizza la struttura e la conformazione degli stessi ***social network***, spesso non viene totalmente compreso nemmeno dagli utenti che li utilizzano quotidianamente.

In effetti, rispetto agli anni passati, la situazione si è invertita, e sono le stesse aziende a "correre" sul mercato, per cercare di studiare ed analizzare i dati che vengono "prodotti" sui mezzi *sociali*, dagli stessi utenti che spesso non sono consci di quello che stanno realmente facendo.

La stessa natura dei *social network* infatti favorisce ed incentiva gli utenti, nella creazione e condivisione della maggior parte dei loro eventi giornalieri, contemplando al loro interno, informazioni altamente rilevanti quali la posizione dalla quale sono stati pubblicati, la foto relativa al prodotto o alla locazione a cui si sta facendo riferimento, ed eventualmente alla propria opinione personale sul soggetto o sullo stato d'animo che da esso ne deriva.

Tutte queste informazioni, estratte, filtrate e opportunamente analizzate, possono portare alla creazione di statistiche ed indagini altamente rilevanti sul mercato, le quali possono spaziare dalla moda, alla valutazione di prodotti commercializzati, fino ad arrivare alle previsioni sulle elezioni politiche o sull'andamento dei titoli di borsa.

In questi casi la parola *"social"* è stata utilizzata, a volte anche impropriamente, per abbinarla ai numerosi ambiti applicativi ai quali è stata associata, si parla ad esempio di *"Social Marketing"*, *"Social Opinion"*, *"Social Sentiment"*, e così molte altre.

Questo studio, si è posto di fronte ad una di queste problematiche, in particolare si è scelto di considerare il problema della comunicazione e gestione di ***eventi critici****. Indipendentemente dalla loro origine (catastrofi naturali, eventi climatici, attentati e atti terroristici), tutti questi eventi richiedono una comunicazione tempestiva alla massa, e l'adempimento di interventi immediati.*

Dal 2010 al 2015 si è visto crescere in Italia, come negli altri paesi del Mondo, un profondo rinnovamento nei mezzi di comunicazione pubblica. Rispetto agli altri paesi più flessibili e dinamici, l'Italia ha iniziato a studiare a rilento la possibilità di sfruttare le piattaforme dei *social media* come strumento di comunicazione **bidirezionale**. Queste infatti possiedono una doppia valenza, concedendo alle pubbliche amministrazioni di comunicare tempestivamente le informazioni alla popolazione, e allo stesso tempo, permettendo di ricevere ***feedback*** e contro-risposte immediate, da parte dei loro lettori.

Si sono creati veri e propri enti e organizzazioni, atti allo scopo di pubblicare informazioni e messaggi relativi a contesti specifici. Per avvalorare la credibilità e l'autenticità di questi messaggi, i maggiori *social network* hanno predisposto la possibilità di verificare *profili* e *gruppi*, in modo tale da apporre su di essi un marchio che ne certifichi la veridicità, e li distingua da eventuali pagine omonime.

Studiosi e ricercatori confermano, che dal 2015 al 2020 [3], si avrà ancora un incremento nell'utilizzo dei ***social media***, e che quindi le misure e le accortezze adoperate fino ad oggi, porteranno questi strumenti a diventare parte fondamentale della comunicazione sociale quotidiana.

Lo ***scopo*** *prefissato di questo progetto di tesi è stato dunque quello di studiare ed analizzare i fenomeni che caratterizzano la comunicazione e la diffusione dei Social Data, per poi poter andare a progettare e sviluppare un applicativo software che permettesse la gestione e l'analisi di queste informazioni. Oltre alla possibilità di estrarre, manipolare e memorizzare i dati provenienti dai social network, è stata richiesta la possibilità di presentare una visualizzazione di questi dati, ed una loro possibile integrazione con altri dati reperibili sulla rete, e provenienti dal mondo degli Open Data, al fine di aumentare la capacità espressiva e di analisi dei risultati ottenuti.*

Il documento di tesi verrà strutturato come segue.

Il **primo capitolo** descriverà i fenomeni e le caratteristiche della comunicazione per mezzo di *Social Media*, andando dapprima a presentare un excursus storico dei *Social Network*, ed in un secondo momento andando ad elencare le principali caratteristiche e definizioni relative ai *Social Data*, e alle strutture di comunicazione ad essi legate.

Nel **secondo capitolo** verrà introdotto e descritto il *social network* selezionato per la fase di estrazione delle informazioni, ovvero ***Twitter***. Verrà presentato un breve riassunto delle caratteristiche che lo contraddistinguono dagli altri *social media*, e verrà motivata la scelta che è stata impiegata.

Nel **terzo capitolo** verrà descritto un breve riassunto sull'analisi logico-concettuale del problema affrontato, al fine di introdurre l'applicazione da realizzare per raggiungere lo scopo del progetto. Verranno descritti i principali requisiti imposti dalle specifiche e verranno brevemente motivate le scelte e le tecnologie realizzative adottate.

Nel **quarto capitolo** verranno discusse e presentate tutte le soluzioni implementative realizzate al fine di rispettare gli obiettivi prefissati. Inoltre verranno spiegate in dettaglio le funzionalità peculiari implementate, andando ad accompagnare le descrizioni con brevi parti di codice sorgente, e annesse immagini relative ai componenti grafici realizzati.

Nel **quinto capitolo** verranno presentati i reali casi di studio affrontati, mostrando tabelle e grafici per la valutazione dei risultati ottenuti.

Infine il documento si concluderà con una parte **conclusiva**, nella quale verranno descritti i risultati ottenuti, evidenziando eventuali problematiche riscontrate e descrivendo alcuni dei possibili ambiti e sviluppi futuri.

Capitolo 1

Semantic Social Web

In questo capitolo verrà introdotto l'ambito di studio di questa tesi, andando ad elencare quelli che sono i concetti e le definizioni fondamentali, che saranno utilizzate per descrivere i problemi che si sono dovuti affrontare nel corso di questo studio. Il capitolo verterà principalmente sul sempre più diffuso utilizzo dei Social Network, sullo tecniche di analisi delle grandi moli di dati che in essi vengono prodotti, ed infine sulle strutture e architetture di comunicazione che vengono utilizzate per rappresentarli e per condividerli.

1.1 Introduzione al Web 3.0

Per quanto si stia parlando di concetti innovativi, e non esistano definizioni ufficiali riguardanti questi argomenti, si proverà a definire il processo evolutivo che ci sta portando dal *"web dei contenuti interattivi"* ovvero il **Web 2.0** al sempre più attuale *"web semantico dei dati"* ovvero il **Web 3.0**.

Partendo dall'espressione coniata da Tim O'Reilly, in occasione della *"Web 2.0 Conference di O'Reilly Media"* [4], tenutasi al termine del 2004, si è definito il Web 2.0 come un'evoluzione del World Wide Web, rispetto alla sua condizione precedente.

Wikipedia definisce il concetto di ***Web 2.0*** come:

> *«Si* indica *come Web 2.0 l'insieme di tutte quelle applicazioni online che permettono un elevato livello di interazione tra il sito web e l'utente come i blog, i forum, le chat, i wiki, le piattaforme di condivisione di media come Flickr, YouTube, Vimeo, i social network come Facebook, Myspace, Twitter, Google+, Linkedin, Foursquare, ottenute tipicamente attraverso opportune tecniche di programmazione Web e relative applicazioni web afferenti al paradigma del Web dinamico in contrapposizione al cosiddetto Web statico o Web 1.0.»*

A partire da questa definizione, e tenendo conto che da questa fonte non sono stati considerati minimamente i dispositivi e le apparecchiature utilizzate dagli utenti fruitori dei servizi. Un fattore che probabilmente ha favorito la nascita del Web 3.0 è stata proprio la diffusione sul mercato di massa, di *smartphone* e dispositivi mobili. Questi hanno fatto sì che il mercato si spostasse verso una sempre crescente richiesta di applicazioni mobili e servizi web accessibili da dispositivi eterogenei. Le aziende *leader* sul mercato, si sono dovute adoperare, proponendo *software* e servizi altamente dinamici, e in grado di adattarsi alle diverse conformazione *hardware* e *software* di questi dispositivi.

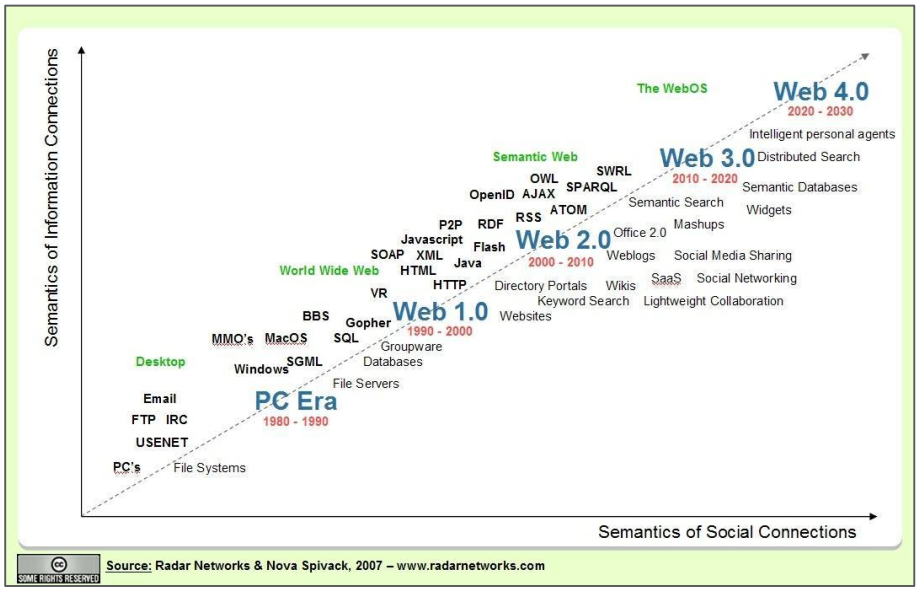


Figura 1: Semantics of Information And Social Connections.
(http://blog.law.cornell.edu/voxpop/files/2010/02/radarnetworkstowardsawebos.jpg)

Come si può vedere in [**Figura 1**], si è passati ad una fase del web moderno, denominata *Web 3.0*. Sebbene in letteratura ci siano pareri contrastanti, tra chi non condivide l'utilizzo di queste "etichette" per denominare le evoluzioni del web e preferirebbe utilizzare direttamente un termine generico come *"Web moderno"* per comprenderle tutte, e chi invece cerca di classificare ognuna di queste fasi di evoluzione; è possibile risalire alle prime comparse di questo termine.

Agli inizi del 2006 infatti, *Jeffrey Zeldman*, web designer critico nei confronti del web 2.0 e della tecnologia associata *AJAX*, e *John Markoff*, giornalista statunitense del *New Yrok Times*, hanno rispettivamente coniato ed utilizzato per la prima volta il termine *Web 3.0*, all'interno di documenti ufficiali [2].

Questo termine ha alimentato un dibattito che ha avuto per protagonisti, figure storiche di *Internet*, da *Tim Berners-Lee* (co-fondatore del World Wide Web) a *Reed Hastings* (fondatore e CEO di *Netflix*) fino a *Jerry Yang* (fondatore e presidente di *Yahoo!*) [5].

Prendendo spunto da questi e da altri esperti in letteratura, è possibile fornire una definizione per il *Web 3.0*, in linea con quella del Web 2.0, e nella quale vengano fatti emergere quelli che sono i tratti salienti di questa evoluzione del web.

Il ***Web 3.0*** si riferisce ad un terza generazione di servizi *internet-based* che compongono quello che viene definito *"The Intelligent Web"*, ovvero un web composto da servizi quali: *web semantico* (del quale verrà fatta una trattazione nel successivo paragrafo), *microformat o mark-up semantici,* ricerche e *query* in linguaggio naturale, *data-mining, machine learning, recommendation agents* e tecnologie che sfruttano l'*intelligenza artificiale*.

Ad accompagnare questa definizione, vi sono alcune tecnologie che stanno raggiungendo un livello di maturità tale, da rendere il *Web 3.0*, una realtà [6]:

- ***"The Intelligent Web"***
    - *Tecnologie Semantiche (RDF, OWL, SWRL, SPAQL)*
    - *Database Distribuiti ("World-Wide-Database")*

    - *Applicazioni Intelligenti (interrogazioni in linguaggio naturale, machine learning, machine reasoning, agenti autonomi, …)*
- ***Ubiquitous Connectivity***
    - *Dispositivi mobili (e possibilità di accedere ad internet da essi)*
    - *Connessione (dalla banda-larga alla fibra ottica)*
- ***Network Computing***
    - *Interoperabilità tra Web Service*
    - *Computazione Distribuita (P2P, Grid Computing, …)*
    - *Cloud Computing (IaaS, PaaS, SaaS)*
- ***Open Technologies***
    - *Protocolli e API Open*
    - *Open Data e Linked Open Data (formato dei dati, piattaforme open-source, "Creative Commons" e licenze di utilizzo)*
- ***Open Identity***
    - *Open Identity (OpenID)*
    - *Open Reputation*
    - *Portabilità della propria identità e dei dati personali (all'interno di servizi differenti)*

1.2 Semantic Web

Tim Berners-Lee ha coniato il termine ***Semantic Web***, considerandolo come il componente fondante della struttura del *Web 3.0*, e promuovendo l'idea che si possa considerare il web come una grande collezione di *database*, sui quali andare ad operare le proprie operazioni.

Seguendo questa corrente di pensiero, lo stesso *Berners-Lee, nel 2006*, ha descritto il *Web Semantico* con una frase che è diventata celebre in letteratura ed è stata presa come punto di partenza di molti documenti di ricerca:

> *«People keep asking what Web 3.0 is. I think maybe when you've got an overlay of scalable vector graphics - everything rippling and folding and looking misty - on Web 2.0 and access to a semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource.»*

Con il termine *Web Semantico*, si intende la trasformazione del *World-Wide-Web*, nella quale ogni documento pubblicato, debba essere accompagnato da informazioni e **metadati** che ne specifichino il contesto semantico, in un formato ***machine-readable***, ovvero leggibile, interrogabile ed interpretabile anche attraverso l'elaborazione automatica delle macchine.

Parte fondamentale di questo processo evolutivo, è la definizione dello standard ***Resource Description Framework (RDF)*** da parte del *W3C*. Questa non è altro che è una particolare applicazione del linguaggio *XML*, che cerca di standardizzare le relazioni tra i concetti, ispirandosi ai principi della logica del primo ordine (ovvero la logica dei predicati), e utilizzando gli strumenti tipici del web, quali *URI (Uniform Resource Identifier)* e *Namespace*.

L'*RDF Data Model*, è nato su tre principi chiave:

- Qualunque oggetto o risorsa sulla rete, può essere identificata attraverso l'utilizzo di un URI.
- Occorre utilizzare sempre il linguaggio meno espressivo che possa definire la risorsa (*"least-power"*).
- Qualunque soggetto può dire qualsiasi cosa su qualunque oggetto.

Per memorizzare queste informazioni, secondo la logica dei predicati, esse possono essere rappresentate attraverso l'utilizzo di asserzioni *statement-based*, costituite da ***triple***, composte da soggetto, predicato e oggetto; dove il soggetto è una risorsa, il predicato è una proprietà, e l'oggetto è un valore (quindi eventualmente anche un URI, che punta ad una differente risorsa).

Nella [**Figura 2**] si vede un possibile esempio di architettura *software*, per la realizzazione di una applicazione di *Semantic Web*, che sfrutti il potere espressivo e la conoscenza, contenuti in ***Ontologie*** condivise sulla rete. Queste *ontologie* non sono altro che rappresentazioni formali, condivise ed esplicite, di una concettualizzazione riguardante il dominio da esprimere. In modo che queste informazioni possano essere sfruttate per applicare tecniche automatiche o semi-automatiche di ragionamento induttivo, di classificazione o di risoluzione di problemi.

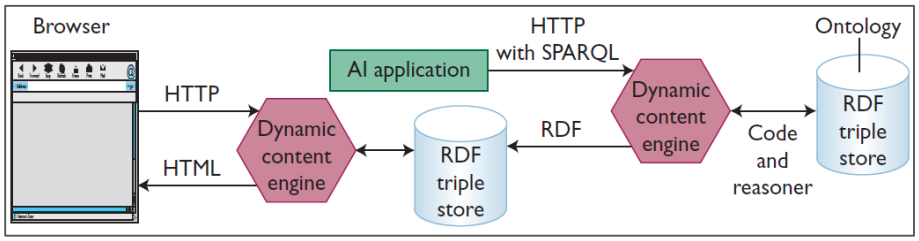


Figura 2: Esempio di architettura di una *Semantic Web Application*, dove lo scambio di *"semantic-data"*, avviene per mezzo di un sistema che permette operazioni dà e verso Ontologie. Queste ultime memorizzano le informazioni con Triple RDF. (Immagine prelevata da [2]).

Continuando a seguire questo ragionamento, è doveroso aggiungere che in una più recente apparizione del sopracitato fondatore del *W3C*, l'attenzione si sia focalizzata sull'organizzazione e la condivisione di queste grandi **moli di dati**, che oggi vedono luce nella diffusione di piattaforme di condivisione e *social network*.

Di seguito sono state estrapolate alcune delle frasi più rilevanti del discorso di *Tim Berners-Lee* in una *TED Conference* tenutasi nel Febbraio del 2011 [7]:

> *«World Wide Web has a new priority: Raw Data Now. … Data drives a huge amount of what appens in our lives … Because somebody takes the data and does something with it … I Want you to put your data on the Web. Not only should we share our data, but we should demand Governments and Businesses share the data they prepare as well …»*

Questo discorso risulta essere una sorta di esortazione diretta agli utenti, ma anche agli Enti Pubblici, ai Governi, e alle Pubbliche Amministrazioni, di cercare di condividere il più possibile i propri dati, in maniera aperta, trasparente e fruibile da chiunque necessiti di tali informazioni.

E da questo è stata definita una nuova evoluzione del web, una specie di transizione *«dal web attuale document-sharing, al web del futuro data-sharing*", ponendo come fondamenta di questo ragionamento, tre fattori chiave:

- Ogni *URL* sul web deve puntare a dati presenti sulla rete.
- Chiunque acceda a tali *URL*, deve ricevere in risposta i dati puntati.
- Le relazioni tra i dati, possono contenere *URL* addizionali, tra di essi.

1.3 Big Data

L'anno 2014 è stato caratterizzato da numerosi studi relativi alle previsioni e alle statistiche che inquadrano l'andamento della rete e la relativa produzione di documenti e risorse web.

Per esempio, l'associazione *Alumniportal Deutschland*, la quale consiste in una vera e propra "rete sociale" che promuove il collegamento e la condivisione di informazioni tra tutte le compagnie, le università e le organizzazioni che aderiscono al progetto *"Germany-Alumni"*, al quale possono iscriversi tutti le persone che hanno studiato, lavorato o fatto ricerca in Germania; ha condotto un'intervista con un noto esperto nell'ambito *Big Data* [8].

Daniel Pfirrmann, Consigliere Delegato e Co-Fondatore della *DiOmega GmbH*, una società che sviluppa a tutto campo sul ramo IT, dai *Web Services* alle *Mobile Application*, ha risposto ad alcune domande che gli sono state poste.

In particolare gli è stata posta una domanda relativa alla *XoCommunications*, una compagnia che, in linea con le correnti stime di utilizzo della rete, avrebbe predetto che, vista la **crescita esponenziale** dei dati che si sta avendo in questi anni, nel 2016 si raggiungerà un *"zettabyte"* di dati prodotti e trasferiti su internet. Considerando che, un *zettabyte (ZB)* equivale a $\mathbf{10^{21}}$bytes, la domanda che sorge spontanea è capire se col passare degli anni, saremo ancora in grado di trovare ciò che stiamo cercando, in quello che è stato definito *"the ocean of data"* (letteralmente, *"l'oceano dei dati"*).

La risposta dell'esperto, ha tirato in causa il *Web Semantico* (Paragrafo 1.2), dicendo che nella nuova era del web, si è passati da un'internet creato per le persone, ad un'internet creato e adeguato alle **macchine**, ovvero un internet dove i contenuti siano collegati da riferimenti logici e semantici, in modo che le macchine possano interpretarli e classificarli autonomamente.

Una previsione che a questo punto non può far altro che rafforzare e convincere anche i più accaniti dissuasori delle previsioni che fino ad oggi erano state fatte da aziende e esponenti leader delle tecnologie *IT* sul mercato. Si pensi per esempio alla meno recente conferenza "*Google's 2010 Atmosphere Convention"* alla quale hanno preso parte numerosi esperti di fama mondiale.

In tale occasione, era stata da molti criticata l'affermazione di *Eric Schmidt, CEO di Google*, il quale aveva asserito che nel 2010, nell'arco di due giorni si producevano in media 5 *exabytes (EB)* di dati, equivalenti a $\mathbf{10^{18}}$bytes, e li aveva comparati alla quantità di informazioni prodotta dagli albori della civilità fino al 2003. Questa è solo una delle tante affermazioni che vanno a descrivere un mondo letteralmente sommerso dai dati (eterogenei, non strutturati e provenienti da svariate fonti), i quali necessitano di strumenti adeguati per poter essere maneggiati.

Se si dovesse definire in maniera intuitiva il mondo dei ***Big Data***, lo si descriverebbe come un vasto insieme di dati digitali, organizzati in *dataset* circoscritti a settori distinti. Tutti questi dati eterogenei, arrivano per esempio dai profili degli utenti nei *social network*, dalla loro posizione geografica mentre effettuano delle telefonate, dagli indirizzi internet che essi assumono e da quelli delle pagine che visitano navigando su internet, fino ad arrivare a tutti quei dati sensibili, economici e sanitari, che vengono affidati ad applicazioni e servizi, spesso *geo-distibuiti* in *cloud*, sparsi per il pianeta.

Più in generale con il termine *Big Data*, si considera una grande mole di dati dai quali risulta difficile estrapolare informazioni utili, se non attraverso l'utilizzo di strumenti non convenzionali, che permettano di gestire e processare queste grandi quantità di dati in un tempo ragionevole.

Per definire i *Big Data*, si sono andati a delineare, incrementalmente nel tempo, 6 concetti chiave fondamentali, dei quali i primi 4 rappresentano le cosiddette "*quattro V dei Big Data*":

1. ***Volume***, considerando l'enorme dimensione effettiva dei *dataset*.
2. ***Velocità***, si riferisce alla velocità di generazione dei dati (alla quale generalmente corrisponde una velocità di analisi dei dati in tempo reale, o quasi).
3. ***Varietà***, considera l'eterogeneità e la provenienza dei dati, che possono essere strutturati o meno.
4. ***Veridicità***, intesa in termini di qualità dei dati (rispetto al valore informativo che può essere estratto da essi).
5. ***Affidabilità***, si riferisce alla possibilità di avere dati inconsistenti.

6. ***Complessità***, quest'ultima è direttamente proporzionale con la dimensione del dataset, maggiore è il volume, maggiore è la difficolta di processare ed estrarre informazioni qualitative da essi.

1.3.1 Semantic Web Mining

Definiti i *Big Data*, è intuibile comprendere quanto potenziale ci sia dietro a queste sconfinate "miniere di informazioni", dalle quali si possono estrarre strutture di conoscenza e di sapere, informazioni statistiche, profili di trend e previsioni nel tempo.

Come evidenziato dal Professore Universitario e Direttore Scientifico del "Complexity Education Project", Valerio Eletti, dell'Università Sapienza di Roma, nei suoi articoli di riflessione sui *Big Data* [9]; spesso non si tiene conto di due fattori fondamentali:

1. Connettendo i diversi *dataset*, si ottiene un ***insieme reticolare iper-complesso***, dal quale può essere estratto e dedotto un contenuto informativo che è di gran lunga maggiore rispetto al contenuto che si otterrebbe sommando letteralmente i singoli contenuti, dei singoli *dataset*.
2. Lo studio e l'estrazione del contenuto celato all'interno dei *Big Data*, assume un'importanza strategica non irrilevante, considerando i diversi possibili ambiti di interesse. Negli ultimi anni infatti, sia la finanza privata che quella pubblica, hanno attuato una corsa frenetica per predisporre efficaci strumenti semantici, in grado di analizzare e processare questi dati, via via crescenti, in tempi ragionevolmente ridotti. Basti pensare che, nel Marzo del 2012, sotto l'Amministrazione Obama, negli Stati Uniti d'America, sia stato istanziato un fondo di 200 milioni di dollari per la *"Big Data Research And Development Initiative"* [10]. E la stessa situazione sia avvenuta, da parte dell'Unione Europea, per finanziare un progetto decennale da un miliardo di euro (100 milioni annui), il quale comprende numerose opere, tra cui il *"Living Earth Simulator"*, ovvero un'enorme rete di calcolo in grado di aggregare le moli di *Big Data*, provenienti da tutto il mondo [11].

Tutta la precedente analisi è servita ad introdurre quello che viene denominato *"**Semantic Web Mining**"*, il quale mira a collegare due importanti aree di sviluppo, il ***Semantic Web*** *(Paragrafo 1.2)* e il ***Web Mining***.

Da Wikipedia si può estrapolare la seguente definizione:

> *«Il Web Mining è l'applicazione delle tecniche di Data Mining per scoprire "pattern" dal Web. In accordo con il settore di analisi al quale esso viene applicato, è possibile suddividere questa applicazione in tre diverse tipologie, il Web Usage Mining, il Web Content Mining e il Web Structure Mining.»*

Di seguito verranno elencate e contraddistinte le tre sopracitate tipologie:

1. Il *"Web Usage Mining"*, analizza le azioni degli utenti nella rete al fine di capire cosa stanno cercando e perché.
2. Il *"Web Structure Mining"*, mira allo studio delle architetture di connessione tra i diversi nodi della rete, sfruttando i grafi degli *hyperlink* e le pagine *XML-based*, con una struttura ad albero.
3. Infine il *"Web Content Mining"*, quello più diffuso, che riguarda l'estrazione, l'analisi e l'integrazione dei dati, delle informazioni e della conoscenza, presenti all'interno dei documenti e delle risorse web.

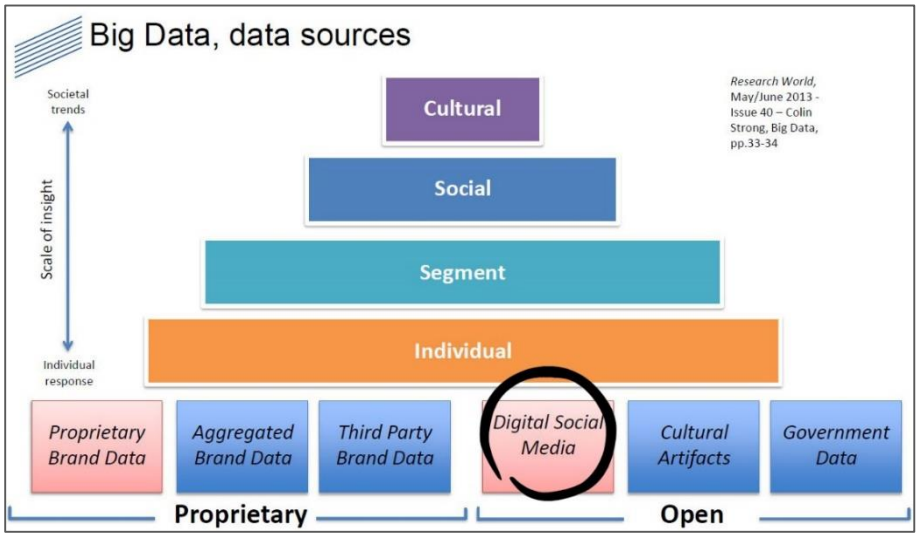


Figura 3: Classificazione delle sorgenti di *Big Data,* sull'asse verticale la scala d'interesse, su quello orizzontale le sorgenti Proprietarie e quelle Open.
(Immagine da *Research World 2013, Issue 40, Colin Strong, Big Data, pp.33-34).*

1.4 Social Media

Partendo dal paragrafo precedente e ricollegandosi all'ultima figura presentata [**Figura 3**], si nota come, sull'asse orizzontale del grafico venga espressa una classificazione delle sorgenti di *Big Data*, ed essa possa essere suddivisa in due macro-famiglie, quella dei *dati proprietari*, e quella dei *dati open*.

Una delle tre sotto-sorgenti di dati pubblici, viene denominata *"Digital Social Media"*, e rappresenta tutti quei dati che possono essere reperiti gratuitamente dai *social network* e dalle piattaforme di condivisione online.

Wikipedia definisce i ***Social Media*** nel modo seguente:

> *«Social Media (in italiano si riferisce ai media sociali) è un termine generico che indica che indica tecnologie e pratiche online che le persone adottano per condividere contenuti testuali, immagini, video e audio.»*

I *social media* in particolare, hanno portato un cambiamento radicale nel sistema di comunicazione e di apprendimento umano, in quanto hanno rivoluzionato il modo con cui le persone possono leggere, apprendere e condividere informazioni.

Un'altra importante conseguenza dell'avvento dei *social media*, è dato dalla loro intrinseca proprietà di ***bidirezionalità***. Quest'ultima infatti, ha facilitato un processo, che era già partito con la nascita delle pagine web e dei blog, ovvero ha dato la possibilità di trasformare un canale di comunicazione, che prima era rappresentabile come un "monologo" (da uno a molti) ad un "dialogo" (da molti a molti), dove gli utenti finali, che prima erano rilegati ad essere solamente fruitori dei contenuti, si sono trasformati in possibili editori. A tale proposito infatti, diversi documenti in letteratura, questi mezzi sociali assumono l'acronimo di *CGM (Consumer-Generated Media)* o *UGC (User-Generated Content)*.

Di seguito verrà presentata una tabella contenente un confronto tra i media industriali "classici" ed i nuovi *social media*, in modo da far emergere gli aspetti vincenti di questa nuova forma di comunicazione [**Tabella 1**].

	Media Classici	Media Sociali
Bacino di utenza	Audience Globale (limiti fisici e di nazionalità)	Audience Globale
Accessibilità	Statali o di Enti privati (costi medio/alti)	Pubblici (gratuiti o con costi d'accesso contenuti)
Fruibilità	Richiedono formazione e competenze tecniche	Non richiedono particolari competenze
Velocità	Tempi di produzione e messa in onda lenti	Tempi di pubblicazione e risposta praticamente istantanei
Permanenza	Mezzi fisici (video, audio, libri, riviste, ...) difficilmente modificabili	Possono sempre essere soggetti a modifiche e revisioni
Responsabilità	Soggetti a vincoli editoriali, conflitti d'interesse, audience, ...	Non soggetti a vincoli esterni particolari

Tabella 1: Confronto tra Media Classici e Media Sociali, con le relative peculiarità.

1.4.1 Social Network

La diffusione dei *Social Media*, ha visto nell'ultimo decennio, un grande e progressivo aumento nell'utilizzo dei cosiddetti *Social Network*.

Wikipedia permette di disambiguare correttamente il termine, dal momento che esso può essere utilizzato con due diverse accezioni del termine.

In primo luogo, ci si riferisce alla traduzione letterale inglese del termine, intendendo la definizione storica e sociologica di "rete sociale", considerata nel senso più fisico del termine, come un insieme di individui connessi tra loro per mezzo di legami sociali. Essi possono infatti essere legati da diverse tipologie di relazione, e condividere tra loro informazioni e interessi.

Nella nostra realtà odierna, ci si riferisce al termine "internazionale" di ***Social Network*** *(SN)*¸ intendendo il suo significato informatico. Quest'ultimo viene utilizzato in ambito Web per definire strutture sociali virtuali, le quali si vengono a creare per mezzo di servizi e siti web di riferimento.

Sebbene il primo *social network*, risalga all'anno 1997 *(SixDegrees [12])*, la vera diffusione di questi strumenti di massa, è iniziata solamente dal 2003.

Uno studio di *Alexa*, leader mondiale nell'analisi dei *raw data* e dell'utilizzo di rete, permette di far emergere il processo evolutivo dei *SN*, passando dagli albori (Dicembre 2009) allo stato attuale delle cose (Dicembre 2014) [13]:

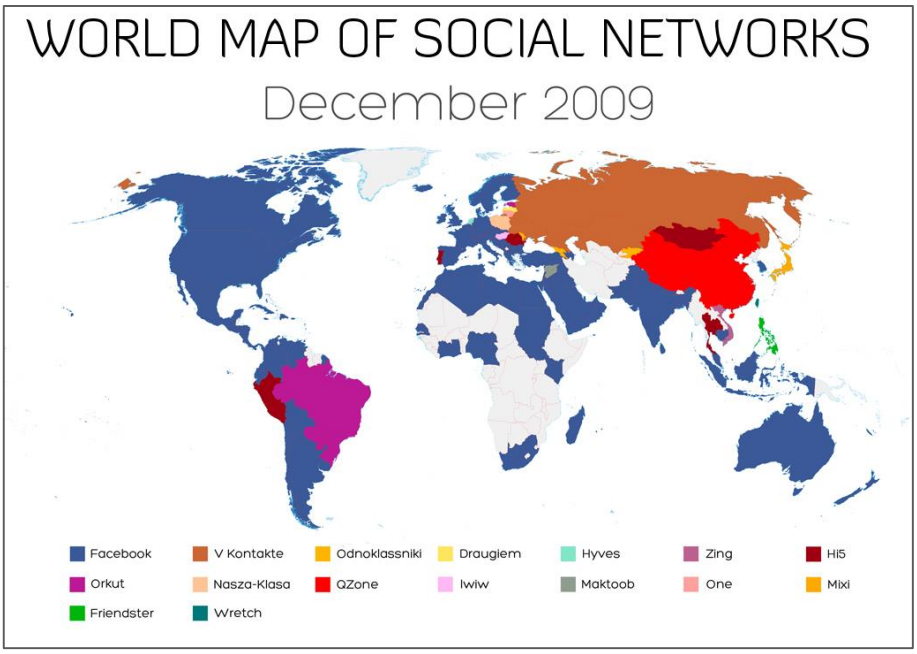


Figura 4: Rappresentazione su scala mondiale, della diffusione dei *Social Network*. (Dicembre 2009, Fonte: Alexa – http://www.alexa.com/)

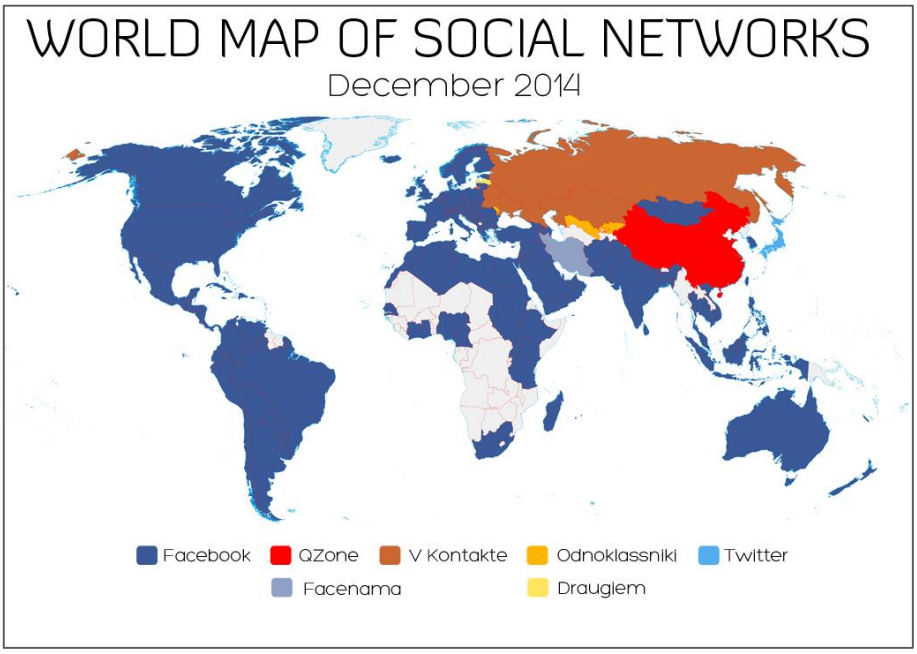


Figura 5: Rappresentazione su scala mondiale, della diffusione dei Social Network. (Dicembre 2014, Fonte: Alexa – http://www.alexa.com/)

Nelle due figure precedenti [**Figura 4** - **Figura 5**] si può notare come, al termine di una finestra temporale di soli 5 anni, la diffusione dei *SN*, sia passata dall'avere molte più scelte, frazionate sui diversi territori, all'avere il monopolio di pochi *SN*, altamente condivisi e apprezzati dalle masse, sulla quasi totalità delle nazioni mondiali (*Facebook*, primo tra tutti).

1.4.2 Semantic Social Computing

In ambito di *Social Media* e tecnologie semantiche, spicca una figura di fama mondiale. Questa persona è Mills *Davis*, Fondatore del progetto *"Project10X"*, ricercatore, analista e consulente per le aziende private e gli enti governativi, riguardo all'ambito di quelle che lui stesso definisce le *"Next-Wave Semantic Technologies"*. È stato co-fondatore e ha preso parte in maniera attiva in diversi progetti, quali ad esempio *SiCoP (Federal Semantic Interoperability Community of Practice)* e *NCOR (National Center for Ontology Research)*.

Quest'ultimo, in veste di Direttore Amministrativo del Progetto10X, ha anticipato i tempi, pubblicando nel 2007 un report denominato *"Semantic Wave 2007: Industry Roadmap to Web 3.0"* [14].

All'interno di questo studio, si ritrovano argomenti che già in precedenza sono stati citati nei paragrafi di questa trattazione; ci si riconduce ad esempio agli argomenti della definizione di *Web 3.0*, al *Semantic Web* e alle raccolte di *Big Data*.

Inoltre, l'autore ha voluto evidenziare una classificazione, suddividendo la ***Semantic Wave***, in 5 principali temi tecnologici di sviluppo:

1. ***Executable Knowledge***, segna la transizione del web da *information-centric* a *knowledge-centric* (ovvero orientata alla conoscenza, e non all'informazione fine a se stessa).
2. ***Semantic User Experience***, pone l'attenzione sull'esperienza vissuta dagli utenti, quindi considerando le valutazioni personali, la *user-friendly* delle interfacce, la facilità di accesso ai servizi, e così diversi altri criteri di accessibilità e fruibilità delle applicazioni.

3. ***Semantic Social Computing***, rappresenta il modo con cui gli utilizzatori comunicano e condividono informazioni (conoscenza), considerando parte del processo anche la *machine-comprehension.* (Di seguito verrà definita con un dettaglio maggiore).
4. ***Semantic Applications and Things***, consiste nella visione empirica della struttura dei prodotti, dei servizi e della *proprietà intellettuale*.
5. ***Semantic Infrastructure***, per lo studio di reti (intese come sistemi ed ecosistemi) scalabili, in grado di inferire conoscenza e risolvere problemi complessi.

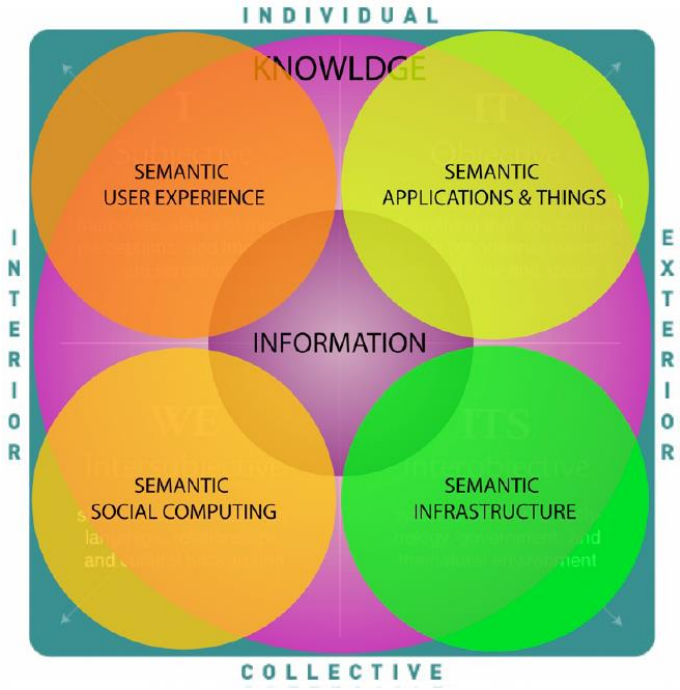


Figura 6: Rappresentazione dei 5 principali temi tecnologici della *Semantic Wave*. (Figura liberamente tratta da [14] pag.7/41)

La ***Semantic Social Computing***, ha permesso di portare la comunicazione sociale virtuale ad un nuovo livello, grazie all'utilizzo della rete sottostante (*Web 3.0*) e alle tecnologie semantiche. Il suo obiettivo è infatti quello di aggiungere un livello di rappresentazione della *conoscenza* sottostante ai dati, ai processi e ai servizi forniti dalla rete, contribuendo alla creazione di una *conoscenza condivisa e strutturata*, relativa ad ogni forma di risorsa presente in rete, dai contenuti dei documenti, ai modelli e le strutture di comunicazione, ai comportamenti e le abitudini degli utenti che ne fanno quotidianamente uso.

SOCIO-SEMANTIC WEB

Se il *Web Tradizionale* era stato definito come il web dei documenti (dalla limitata interoperabilità), il *Web Semantico* [Paragrafo 1.2] era stato definito come il web dei dati semantici, ovvero dei dati integrati e collegati tra loro, la *Semantic Social Computing*, quindi l'applicazione di tecniche semantiche al *Web 3.0* e ai *Social Network*, ha dato vita a quello che può essere definito ***Socio-Semantic Web*** [15].

1.5 Social Data Analysis

La *Social Data Analysis (SDA)* permette di dare un senso, e un significato ben preciso, alle moli di dati che si possono reperire dai contesti *social*. Questo termine è stato coniato da *Martin Wattemberg*, un noto ricercatore americano della *IBM Research*, specializzato nelle tecniche e nelle teorie di *Data Visualization*. In ambito di ricerca, essa è stato anche rinominata ***Big Social Data Analysis***, in relazione al suo contesto applicato ai *Social Big Data*.

La *SDA* è caratterizzata da due componenti specifiche:

- ***Social Data***, che devono essere estratti da *Social Network*, o altre applicazioni caratterizzate da condivisioni sociali.
- ***Tecniche Sofisticate di Analisi***, in molti casi il contesto richiede che vengano svolte delle computazioni in tempo reale (o quasi), in altri invece è sufficiente analizzare i dati nel momento in cui lo si ritiene più opportuno, si pensi ad esempio ad applicazioni statistiche che devono prima raccogliere campioni composti da un quantitativo di dati sufficienti, per poi far emergere dei valori significativi per l'intera popolazione.

È da tenere in considerazione inoltre, che esistono svariati **fattori chiave** che incidono sull'efficacia del processo di analisi.

Di seguito ne verranno elencati alcuni tra i principali:

- ***Profondità di analisi***, in quanto è possibile tenere in considerazione un numero davvero elevato di fattori qualitativi e quantitativi, tra cui: contesto, contenuti, utenti e informazioni di profilo, condivisioni, apprezzamenti e commenti (per il ramo della *Sentiment Analysis*) e reti di collegamenti (siano essi denominati "follower" o "friend" a seconda del *social network* di provenienza).
- **Finestra temporale**, la scelta dei tempi è fondamentale, soprattutto vista la natura dei *Social Network*, e dalle tipologie di risultati che ci si aspetta di reperire da essi. Spesso, si possiede una finestra di opportunità limitata nel tempo, affinché i risultati che si ottengono, abbiano una qualche forma di valenza, per questo motivo occorre realizzare dei *software* di analisi in grado di operare il più velocemente possibile.
- **Analisi dell'Influenza**, è un termine che cerca di evidenziare quanto sia importante identificare quali siano i concetti chiave, che permettono di estrarre la maggior parte dei risultati da un determinato contesto. Dal momento in cui si sta operando in un contesto così ampio come quello dei *Big Data*, non è possibile analizzare tutto ciò che viene condiviso sulla rete.
- **Analisi della Diffusione in Rete**, questa tecnica prevede l'estrazione di risultati, non solamente in maniera diretta attraverso l'analisi dei contenuti delle risorse, ma anche analizzando la struttura e le metodologie di condivisione, che vengono a crearsi naturalmente all'interno delle reti virtuali sociali. Nei paragrafi seguenti, verranno trattati alcuni di questi aspetti, come per esempio, lo studio delle caratteristiche che contraddistinguono i dati ***virali*** all'interno delle reti sociali.

1.5.1 Information Diffusion on Social Networks

Lo studio della *Information Diffusion* all'interno dei *Social Network*, incorpora quattro importanti componenti, le quali vengono espresse nella figura seguente:

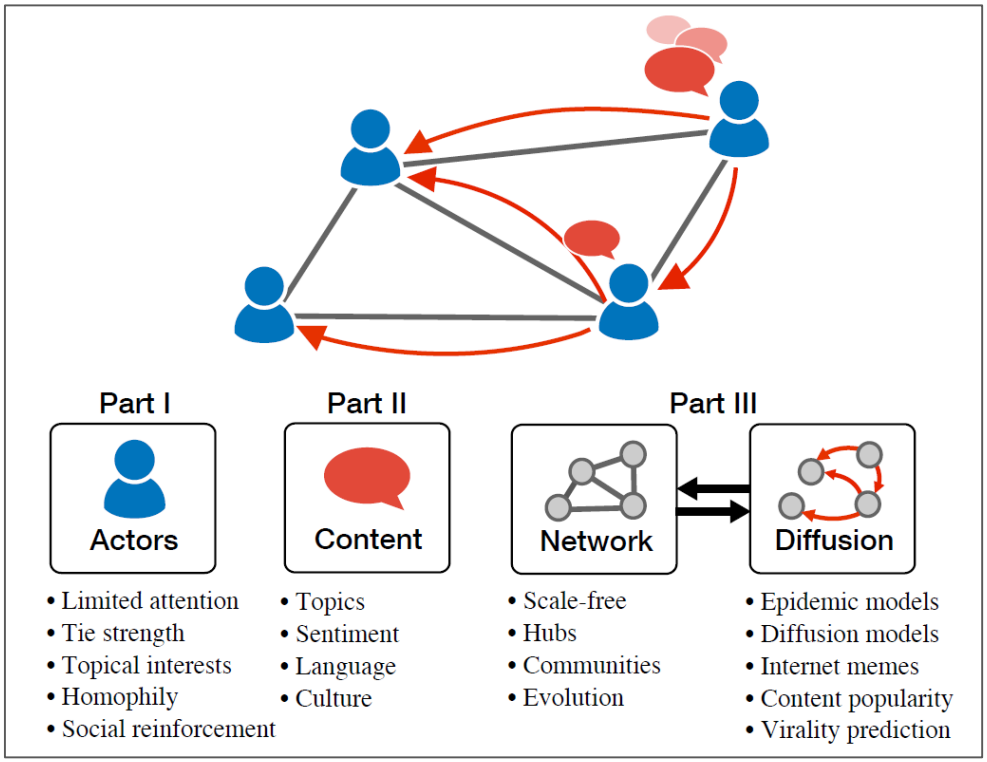


Figura 7: Classificazione della *Information Diffusion on Social Networks*.
(Immagine tratta da: http://cnets.indiana.edu/groups/nan/informationdiffusion/)

NOTA: Dal momento che, tra gli obiettivi di questa trattazione non vi è né l'implementazione, né lo studio dettagliato delle tecniche di *Social Data Analysis*; di seguito verrà introdotto e definito, a grandi linee, lo studio della *Information Diffusion*. Queste tecniche verranno prese come spunto, nella fase di progettazione e implementazione dell'applicazione [rispettivamente, Capitolo 3 e Capitolo 4] per realizzare lo studio e l'analisi dei dati reperiti dalla fase di estrazione. In particolare verranno analizzati alcuni aspetti, sia dei contenuti, che della diffusione dei messaggi nella rete.

Nella [**Figura 7**] si possono chiaramente evincere 3 macro-tipologie, all'interno della classificazione delle componenti dell'***Information Diffusion***:

1. ***Actors***, sono le persone, i reali utilizzatori dei *Social Network*, i quali svolgono sia la funzione di *fruitori*, quando leggono e consultano i messaggi e le risorse condivisi da altri, sia *creatori*, quando sono essi stessi i proprietari intellettuali di ciò che condividono sulla rete.
2. ***Content***, sono le caratteristiche stesse dei contenuti che vengono diffusi in rete (messaggi, commenti, foto, condivisioni, …).
3. ***Interplay between Network Structure and Diffusion***, consiste nell'esplorare l'interazione tra queste due componenti, e studiare come la *topologia di rete* influenzi la diffusione delle informazioni; e viceversa, come il *flusso del traffico* modifichi l'*evoluzione della rete*.

Avendone la possibilità, sarebbe auspicabile addentrarsi in dettaglio, all'interno di queste tre tipologie, in quanto ognuna di esse possiede caratteristiche e tratti che la contraddistinguono dalle restanti (si veda la seguente figura, che rappresenta una tassonomia trasversale rispetto alla precedente).

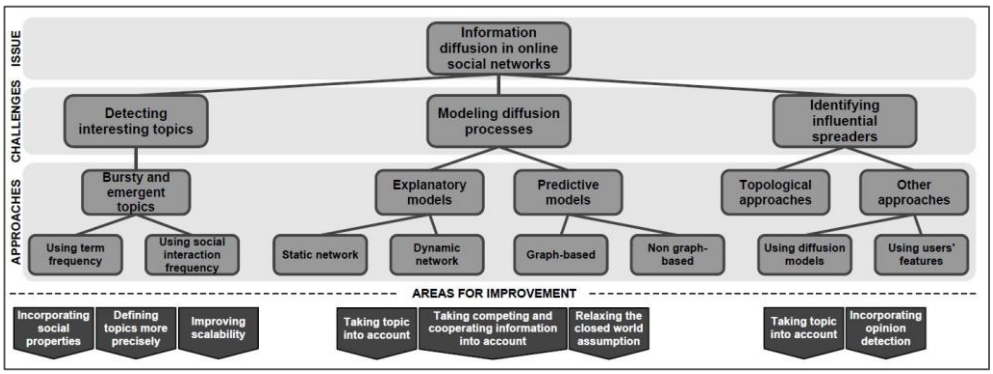


Figura 8: Raffigurazione dettagliata della tassonomia relativa alla *Information Diffusion* nell'ambito specifico dei *Social Network*. Nella parte alte sono presenti le "sfide della ricerca" a tal proposito, nella parte bassa, i possibili approcci e le aree di miglioramento. (Immagine tratta da [16]).

Seguendo l'obiettivo preposto nell'introduzione di questa trattazione, di seguito verrà trattata con maggiore attenzione rispetto alle altre, la "terza parte", la quale parla di tecniche di ***diffusione*** dei messaggi. In particolare lo studio si soffermerà sulla possibilità di studiare elementi quali la *popolarità* o la *viralità*, di un contenuto condiviso tra gli utenti si un *social network*.

1.5.2 Modelli di Diffusione

Tra le tecniche di analisi della *Diffusione di Informazioni* (presenti nella Parte III, **Figura 7** del paragrafo precedente), si possono evincere per esempio, lo studio degli *Epidemic Models (letteralmente, "modelli epidemici")* e dei *Diffusion Models ("modelli di diffusione")*, dei quali non verrà fatta un'ulteriore trattazione dettagliata. Mentre per le restanti voci, di seguito verrà proposta una breve e riassuntiva analisi delle principali caratteristiche:

- ***Internet Memes***, consiste nello studio delle caratteristiche e della propagazione di quelli che vengono denominati i *"meme"* della rete *(dall'inglese meme, dal greco mímēma «imitazione»)*. Il termine ***meme*** era stato coniato da *Richard Dawkins* nel 1976 [17], ovvero prima dell'avvento di internet e dei *social network*; nel 2013 lo stesso *R.Dawkins* ne ha fornito una nuova definizione [18], caratterizzata dai nuovi mezzi digitali di comunicazione, considerandola un'alterazione deliberata, scaturita dalla creatività umana. Direttamente dalla definizione di *Wikipedia*, è possibile scoprire che si tratta di *"unità informative auto-propaganti"*, caratteristiche dell'evoluzione culturale, e analoghe a ciò che è il ***gene*** per la *genetica*. Questi *meme* sono elementi che si trasmettono, spesso per ***imitazione***, attraverso mezzi non genetici come i *social network*. Altre fonti li definiscono più genericamente *"fenomeni di internet"*, ovvero azioni che si propagano attraverso la rete internet ed i suoi mezzi di condivisione sociale, diventando improvvisamente *"celebri"*. L'analisi di questi *internet meme*, è resa possibile dal fatto che essi lascino informazioni rilevanti, una sorta di firma o impronta digitale, all'interno dei *media* (immagini, link, video e *hashtag*) per mezzo dei quali essi si propagano.
- ***Content Popularity***, consiste nello studio della *popolarità*, ovvero dei fattori che favoriscono la condivisione di particolari ***topic***. Esso può considerare svariate tipologie di fattori, dalla frequenza dei termini maggiormente utilizzati, alle caratteristiche strutturali dei messaggi che ottengono maggior condivisione sulle masse. Spesso la *popolarità* di un determinato messaggio, non dipende solamente dal contenuto

informativo, ma viene fortemente influenzata dal contesto sociale in cui essa è inserita, e dalla reputazione dell'utente che la condivide.

- ***Virality Prediction***, consiste nello studio dei modelli di diffusione dei messaggi sociali, analizzando particolari criteri che permettono di inquadrare quali tipologie di utenti e contenuti, abbiano le giuste "qualità" per poter diventare *virali* all'interno delle reti sociali virtuali. Spesso queste analisi si basano su teorie empiriche, e su modelli probabilistici, e hanno il fine di fare predizioni nel tempo, relative alla diffusione di questi contenuti virali.
  Di seguito viene mostrato il confronto tra due *meme* condivisi sulla rete (uno virale, e l'altro no), mostrando i caratteri che li differenziano, in termini di tempi e di quantità di condivisioni.

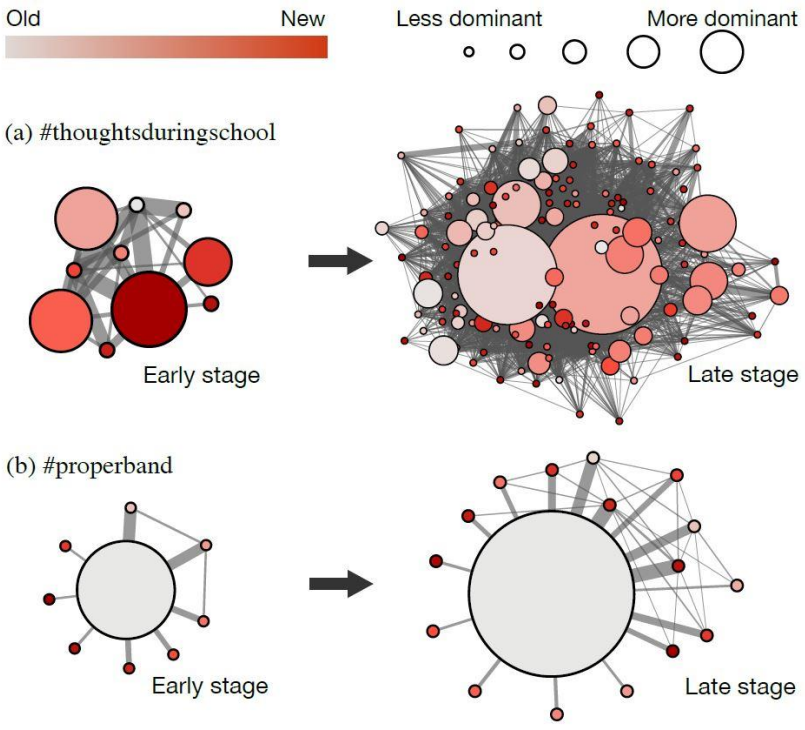


Figura 9: Evoluzione di due meme differenti (virale VS non-virale). Nella parte alta della figura, si vede la Legenda: l'indice di colorazione indica che i nodi con colore più scuro sono i più recenti (ovvero in cui l'*hashtag* è stato condiviso più di recente), mentre la dimensione dei nodi, indica il numero di condivisioni.

I nodi indicano ciascuno una differente *community* di utenti.
(a) diffusione di un *meme* virale, che passa da 30 a 200 condivisioni.
(b) diffusione di un *meme* non-virale, che arriva a 65 condivisioni.

Social Convergence

In letteratura, si nota come, la *diffusione delle informazioni* su *social network*, sia stata strettamente correlata ad un fenomeno "naturale" denominato ***Social* Convergence** [19] per il quale si favorisce la creazione di vere e proprie *community* di condivisione, che possiedono tutte caratteristiche favorevoli alla diffusione di contenuti virali ed epidemici.

A tal proposito, è nata una vera e propria teoria che studia questo fenomeno, denominata *"Symbolic Convergence Theory"*, la quale studia la coesione, le caratteristiche e le motivazioni che spingono gruppi di utenti a conformare queste *community*, nelle quali vengono condivisi contenuti e mezzi di comunicazione.

All'interno di questa teoria, si studiano per esempio fenomeni legati al comportamento degli utenti di queste *community*, durante eventi di coesione popolare o di tensione sociale.
Possono essere argomento di studio, le caratteristiche dei singoli utenti, le tipologie ed i contenuti dei messaggi condivisi, le topologie e le strutture delle reti sociali che si vengono a creare.

Community Structure

Un altro fattore fondamentale per la condivisione virale di contenuti sulle piattaforme di *social sharing*, è rappresentato dalla struttura delle *community*. Vengono a crearsi dei gruppi di utenti, i quali partecipano più o meno attivamente alla condivisione di contenuti attinenti ad uno stesso contesto, o a comuni interessi di ascolto.
Queste *social community*, possiedono caratteristiche intrinseche alle loro strutture, le quali favoriscono, o meno, la condivisione di contenuti al loro interno, e verso l'esterno.

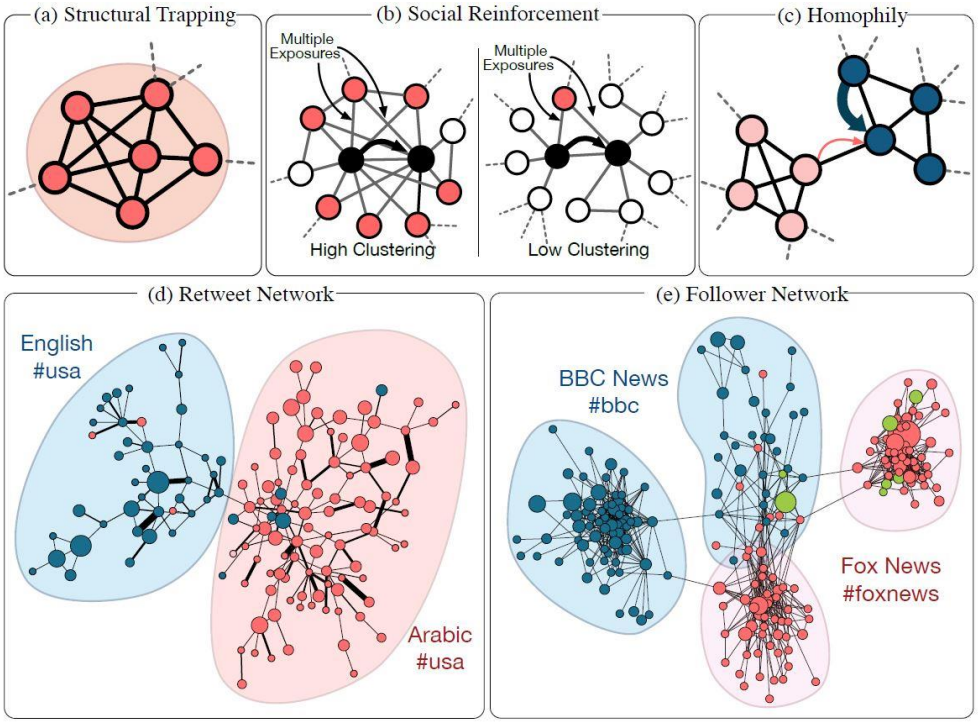


Figura 10: Caratteristiche delle *Social Community* che aumentano le possibilità di contagiare la rete con contenuti virali.

(a) ***Structural Trapping***: *le community con pochi link uscenti, trattengono naturalmente il flusso di informazioni al loro interno.*
(b) ***Social Reinforcement***: *gli utenti che hanno utilizzano un meme virale (nodi neri) innescano una "esposizione multipla" nei confronti degli altri utenti fruitori (nodi rossi). Nel caso di cluster densi, si ha una maggior possibilità di esposizione.*
(c) ***Homophily***, *utenti della stessa community (nodi dello stesso colore) hanno più probabilità di essere simili e quindi di condividere le stesse idee.*
(d) ***Diffusion Structure***, *nell'esempio si vede come utenti appartenenti a due nazionalità differenti, creino una divisione naturale nel grafo delle condivisioni (meme #usa, utenti inglesi in blue, utenti arabi in rosso).*
(e) ***Community Structure***, *analizzando la struttura delle community, presi due meme attinenti ad uno stesso contesto (#bbc e #foxnews) si vede come le due community siano relativamente divise (colori blu e rosso) al di fuori dei nodi che hanno utilizzato entrambi gli hashtag (colore verde).*

Descritte le tecniche di analisi e le strutture, che rendono possibili studi sulla metodologia di condivisione di messaggi sociali, e ne evidenziano i tratti fondamentali, affinché essi diventino virali sulla rete; l'enfasi della trattazione si sposterà sulla possibilità di sfruttare queste tecniche al fine di studiare messaggi di allerta legati ad eventi di crisi o catastrofi naturali.

Tornando alle premesse di questa trattazione, il *focus* fondamentale di questo studio si centra sull'analisi di contenuti relativi a messaggi di allerta meteo, comunicazioni di rischio attentati, e ogni altra forma di condivisione relativa ad eventi urgenti e di massima importanza, i quali richiedano interventi e comunicazioni tempestive.

Si pensi per esempio all'impatto che eventi mondiali come, l'*Attentato alle Twin Towers di New York, dell'11 Settembre 2001*, o lo *Tsunami nell'Oceano Indiano del 2004*, abbiano cambiato e fortemente influenzato il sentimento e la coesione popolare sui media e sugli strumenti di comunicazione sociale [20].

Eventi più recenti, come l'*Attentato terroristico alla Sede del giornale francese Charlie Hebdo, del 7 Gennaio 2015*, hanno confermato quanto l'utilizzo dei *social media*, sia diventato parte fondamentale della comunicazione moderna.

L'articolo [21], come molti altri articoli reperibili sulla rete, evidenzia la copertura mediatica che è stata oggetto delle comunicazioni avvenute dal momento stesso dell'attentato, ai giorni immediatamente successivi. In quest'ultimo caso, ad esempio, si è visto l'utilizzo del *meme* #JeSuisCharlie, un particolare *hashtag* per raggruppare tutti i messaggi relativi a tale contesto. Questo evento ha dimostrato come, atti di questo genere, che spaventano e terrorizzano la popolazione, spesso costituiscono il fattore chiave per una forte coesione popolare.

Di seguito viene presentata la rappresentazione grafica di 4 fenomeni analoghi ai precedenti, che hanno assunto un comportamento ***virale*** nell'anno 2011, vista la loro grande condivisione in rete, e soprattutto visti i contesti che stavano andando a manifestare.

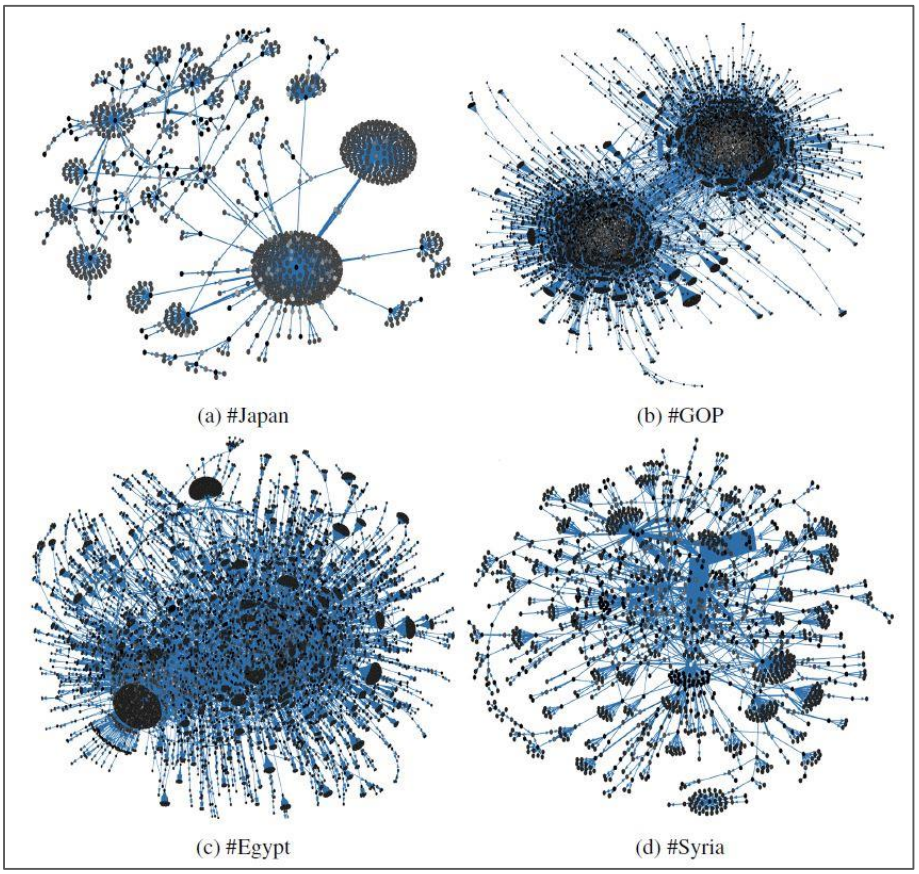


Figura 11: Raffigurazione delle reti di diffusione dei *meme*, relativi a *topic* differenti. I nodi rappresentano gli utenti, gli archi diretti rappresentano i messaggi o i post che trasportano i *meme*, la vivacità di ogni nodo indica il numero di condivisioni dell'utente, e il peso degli archi riflette il numero di condivisioni tra due utenti.

I quattro casi proposti in [**Figura 11**], trattano eventi mondiali relativi a calamità naturali ed eventi politici:

a) *#Japan*, terremoto propagatosi in territorio Giapponese nel Marzo del 2011.
b) *#GOP*, rappresenta l'hashtag della *"US Republican Party"*, e come altri *meme* politici, riporta l'interazione e i messaggi di differenti punti di vista sull'orientamento politico.
c) *#Egypt*, eventi di ribellione nel territorio Egiziano, nel 2011 (nel contesto della *"Primavera Araba"*, ovvero in un periodo caratterizzato da numerose sommosse popolari nei territori arabi, tra la fine del 2010 e l'inizio del 2011).
d) *#Syria*, eventi di protesta, analoghi al contesto egiziano in (c), relativi al contesto della *"Primavera Araba"* del 2011.

Capitolo 2

Twitter

In questo capitolo verrà innanzitutto presentata una panoramica sul *social network* Twitter, mettendo in evidenza le caratteristiche che hanno favorito la sua scelta rispetto ad altre piattaforme sociali. In secondo luogo, verranno discusse la terminologia e gli aspetti principali della sua struttura, che hanno fatto emergere Twitter come *Social Media* in ambito di *Emergency* e *Disaster Event Communication.*

2.1 Introduzione

Twitter è una piattaforma gratuita di *social networking* e *micro-blogging*, che fornisce ai suoi iscritti la possibilità di condividere messaggi di testo composti da un massimo di 140 caratteri ciascuno. Questi vanno a posizionarsi in ordine cronologicamente inverso, dal più recente al più lontano nel tempo, sulla pagina dell'utente proprietario, ed in maniera trasparente, anche nelle *timeline (letteralmente, "linee del tempo")* degli utenti che hanno deciso di seguire gli aggiornamenti di quest'ultimo.

Nella pagina di presentazione del *social network* (Twitter About, https://about.twitter.com/), è possibile leggere il motto che lo caratterizza:

> *«Twitter helps you create and share ideas and information instantly, without barriers.»*

Twitter si definisce infatti il miglior modo per poter connettere persone provenienti da ogni parte del mondo, permettendogli di esprimere tutto ciò che pensano, e di scoprire cosa sta succedendo agli altri nello stesso momento.

2.1.1 Storia

Twitter è nato nel Marzo del 2006, dalla *Obvious Corporation* di San Francisco, e ad oggi, che sono trascorsi 9 anni dalla sua nascita, è una delle reti sociali virtuali più utilizzati al mondo. Milioni di utenti lo utilizzano ogni giorno per condividere informazioni, e allo stesso tempo per osservare commenti, opinioni e news condivise da altri utenti.

Figura 12: Twitter Logo (Immagine tratta da https://twitter.com/)

Il 21 Marzo del 2006 *Jack Dorsey*, fondatore di Twitter e attuale Presidente del Consiglio di Amministrazione del *social network*, ha pubblicato il "primo tweet della storia", nell'arco di circa 3 anni, è stato pubblicato il "bilionesimo tweet".

Tra le curiosità e gli eventi storici legati a questa piattaforma, si deve segnalare un particolare evento avvenuto il 22 Gennaio 2010, quando dallo spazio, il Colonnello degli Stati Uniti d'America, nonché Astronauta della Nasa, *Timothy J. Creamer*, ha pubblicato il "primo tweet dallo spazio". Questo evento ha dato il via ad un fenomeno sempre crescente di condivisioni da parte di figure illustri, nei più svariati contesti, ottenendo un alto riscontro popolare di adesione sulla popolazione.

Figura 13: Primo tweet dallo spazio (T.Creamer, USA Colonel, NASA, 22-01-2010)

Un'ulteriore considerazione riguarda direttamente l'idea e le caratteristiche di Twitter. Esso infatti, rispetto ai suoi principali concorrenti (si veda per esempio Facebook), ha subito dato l'impressione che si potesse utilizzare non solamente per la condivisione di contenuti personali, ma anche come mezzo di **informazione di massa** e di condivisione di informazioni certificate, ovvero provenienti da pagine verificate di personaggi illustri, aziende e enti governativi.

Dal 2012 ad oggi, sono state attuate iniziative di ***"trasparenza"*** per favorire la condivisione, da parte di Enti Pubblici e Aziende Leader sul mercato, di report e informazioni relative a bilanci, progetti e investimenti, in modo da chiarire la loro situazione economica e sociale, migliorando l'immagine ed incrementando la fiducia dei loro sostenitori.

Per rendersi conto della reale diffusione del *social network*, e per capire quanto sia esteso il bacino di utenza e il numero di messaggi da essi condivisi quotidianamente; di seguito verranno elencate alcune statistiche relative a Twitter e alla sua crescita nel tempo.

2.1.2 Numeri: Trends & Statistiche Demografiche

Analizzare i numeri e le statistiche di utilizzo dei *social network*, è un compito sempre più arduo, vista la continua dinamicità di questi mezzi di comunicazione. Si stima infatti, che visti i tassi di crescita che essi possiedono, tabelle e risultati perdano la loro esattezza, nell'arco di ventiquattro ore. Risultati annuali o trimestrali dunque, possono servire solamente per fare chiarezze e attribuire un ordine di grandezza a questi fenomeni.

La stessa Twitter, pubblica annualmente report relativi all'andamento economico e sociale dell'anno appena trascorso. Senza addentrarsi nella questione economica, è comunque possibile analizzare le stime che da essi vengono presentate sulla pagina web di riferimento del social network *(https://about.twitter.com/it/company)*.

Di seguito vengono presentati alcuni risultati calcolati al termine del 2014:

- **288 milioni, la media degli utenti attivi mensilmente**.
  Questo valore risulta essere in linea con il grafico presentato di seguito, il quale ne segna la costante crescita avvenuta dal 2010 al 2014, nonostante nel 2010 fossero già passati quattro anni dalla sua fondazione e come è intuitivamente prevedibile, il fattore di crescita si sarebbe dovuto smorzare prima.

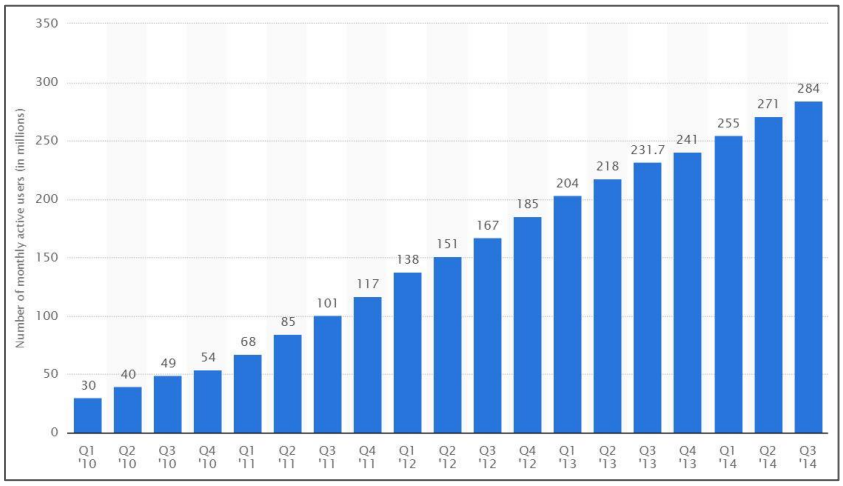


Figura 14: Numero di utenti mondiali attivi mensilmente su Twitter (dal primo trimestre 2010 al terzo del 2014). (Immagine tratta da Statista, *The Statistics Portal*: http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/)

- **500 milioni di tweet giornalieri**, questa è la stima calcolata mediando il numero dei *tweet* giornalieri nell'arco del 2014.
- **80% di utilizzo su piattaforma *mobile***, è la stima osservando i diversi mezzi di fruizione, distinguendo l'accesso da *app mobile*, da quello sulla pagina web tradizionale.
- **77% degli utenti, al di fuori degli USA**, questo valore possiede un alto valore di significatività, in quanto inizialmente si pensava che il fenomeno di Twitter fosse limitato e contestualizzato al territorio Americano.
- **35 le lingue supportate**, questo valore evidenzia lo sforzo dei *team* di sviluppo di Twitter, al fine di aumentare la globalizzazione e la diffusione mondiale della loro piattaforma sociale.

Oltre ai dati relativi al bacino d'utenza, Twitter presenta sulla sua pagina anche notevoli informazioni relativi all'azienda:

- **3600 dipendenti nel mondo, in 14 sedi internazionali**, tra queste: Amsterdam, Berlino, Dublino, Londra, Madrid, Parigi, Rio de Janeiro, San Paolo, Singapore, Sydney, Seul, Tokyo, Toronto e Vancouver.
- **50% dei dipendenti compone il settore tecnico**, tra i 3600 dipendenti, circa il 50% è composto dal personale con qualifiche tecniche.

Per monitorare questi valori, e per avere un'idea reale ed aggiornata di questi, è possibile consultare il sito:

Internet Live Stats (http://www.internetlivestats.com/twitter-statistics/)

Questo sito effettua un monitoraggio costante sulle maggiori tecnologie informatiche utilizzate quotidianamente, e riporta in tempo reale, dei contatori che mostrano il crescere di questi valori. Per esempio vengono riportati contatori e statistiche riguardanti il numero di utenti correntemente online, il numero di siti web raggiungibili, il numero di email inviate giornalmente, le interrogazioni effettuate sul motore di ricerca Google, e così molte altre informazioni relative ai principali *social network*.

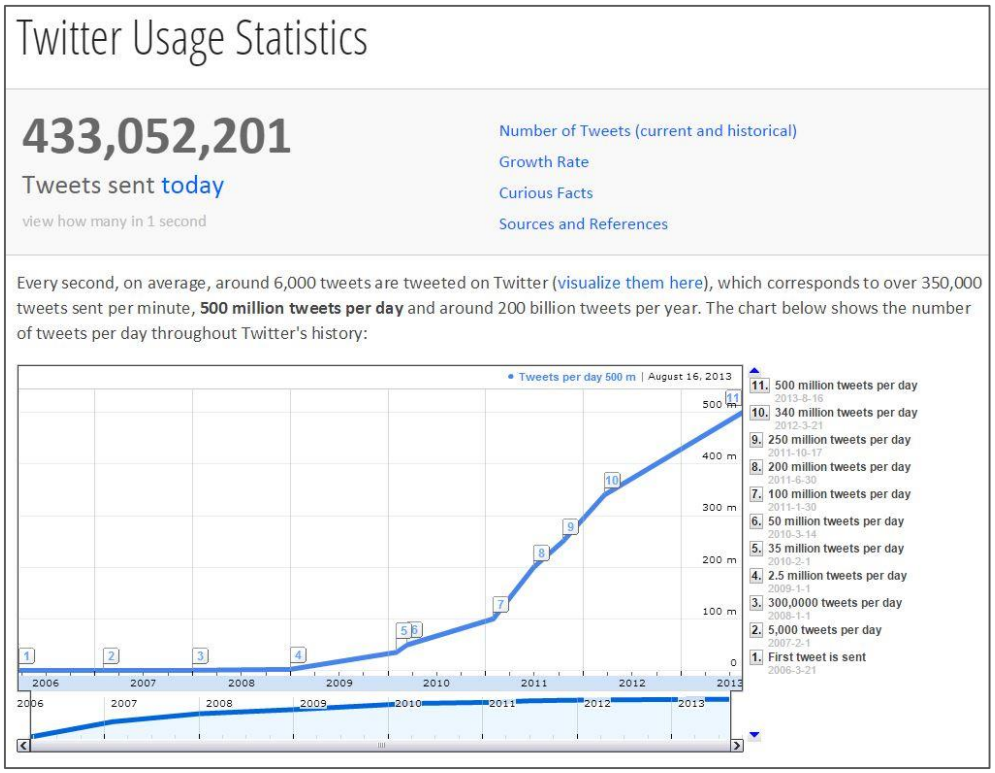


Figura 15: Contatore delle condivisioni di Twitter, in continuo aggiornamento.
Nella parte bassa, un grafico che mostra l'andamento di crescita del numero di tweet condivisi annualmente, dal 2006 al 2013.
(Fonte:http://www.internetlivestats.com/twitter-statistics/)

2.2 Perché Twitter?

Innanzitutto Twitter rispetta la linea di pensiero che è stata scelta per la realizzazione di questo progetto, ovvero, la decisione di utilizzare solamente tecnologie e strumenti *open-source*, o come in questo caso, gratuiti.

Twitter dal lato suo, è uno dei dieci siti web più visitati sulla rete, e come si è potuto vedere nel paragrafo precedente, possiede un bacino di utenti davvero elevato, che pubblicano e condividono una quantità di materiale sufficiente per garantire analisi sui più svariati ambiti.

Oltre a questo aspetto, occorre mettere in luce quali sono le caratteristiche che contraddistinguono la struttura di questo *social network*, perché anch'esse fanno parte dei motivi che hanno fatto propendere la scelta su questa piattaforma, rispetto che sulle sue concorrenti.

2.2.1 Struttura e caratteristiche Principali

Come già accennato in precedenza, Twitter è un servizio caratterizzato da un'idea semplice: ogni utente ha la possibilità di condividere un messaggio breve, formato da un massimo di 140 caratteri. Indipendentemente dallo scopo e dal contesto del messaggio, sia esso un commento o un pensiero, la descrizione di un evento, o la condivisione di un contenuto multimediale.

All'interno dei messaggi viene infatti data la possibilità di condividere oltre al testo, immagini, video, link a pagine e contenuti esterni.

Il meccanismo di pubblicazione e fruizione dei contenuti su Twitter, si basa su una logica triviale: ogni utente può "seguire" un gruppo di utenti, e esso stesso può "essere seguito" da un altro gruppo di utenti, che non per forza coincide con quello citato nel primo caso.

Nel gergo di Twitter, "seguire" un utente, significa ricevere e visualizzare tutti i messaggi che esso decide di pubblicare. Tutti questi messaggi vengono mostrati in una sezione denominata *"**timeline**"*, che è presente su ogni pagina profilo, e dove vengono inseriti tutti i messaggi di tutte le persone che si stanno seguendo.

Per ogni pagina profilo è dunque possibile evidenziare due gruppi o categorie di appartenenza, denominati secondo la terminologia di Twitter:

- ***followers**:* la lista degli utenti che seguono la pagina;
- ***following**:* la lista degli utenti seguiti dal profilo in questione.

Superando la logica di base, ed entrando in dettaglio sull'implementazione *software* di Twitter, si scopre che i suoi sviluppatori hanno deciso di fare affidamento, in gran parte, su software *open source*. Inizialmente la piattaforma era stata implementata sul *framework* "Ruby on Rails". Nell'anno 2009 l'architettura di Twitter ha subito un'importante *porting* verso una nuova soluzione *software*, in particolare si è deciso di sostituire l'implementazione del server per la gestione delle code dei messaggi denominato *"Starling"* (e realizzato in *Ruby*), con una realizzazione dello stesso, denominato *"Kestrel"* (e realizzato in *Scala*) [22].

Dal 2011 a questa parte, Twitter ha ufficialmente sostituito *Ruby on Rails*, nei confronti di una soluzione ibrida, ottenuta principalmente con *Java* e *Scala*.

Twitter è passato dalla sua realizzazione iniziale, che consisteva in un unico sistema monolitico, ad una realizzazione composta da moduli e servizi indipendenti, i quali comunicano per mezzo di chiamate **RPC (Remote Procedure Call).** Con la realizzazione di una propria API, ha permesso a milioni di sviluppatori nel mondo, di integrare funzionalità di Twitter all'interno di altre applicazioni e servizi web.

2.2.2 Vantaggi

Di seguito verranno elencati alcuni degli aspetti positivi, che hanno favorito la scelta di Twitter:

- Il vantaggio principale, che è anche la caratteristica che più lo contraddistingue dagli altri *social network*, è il fatto che Twitter permetta l'esistenza di **relazioni unidirezionali** tra gli utenti. Questo sta a significare, che non è necessario che entrambi gli utenti inizino "a seguirsi", o se lo si vuole spiegare con la terminologia di Facebook, che l'utente "accetti l'amicizia" dell'altro. Questa caratteristiche permette agli utenti di seguire i contenuti e gli aggiornamenti provenienti da ogni genere di fonte, senza preoccuparsi di altre limitazioni.
- I Tweet (anche denominati *Status*, per via della loro scelta di nome inziale, visto che rappresentano degli "aggiornamenti di stato") sono limitati ad una dimensione di **140 caratteri**. Per il caso di studio proposto, e per l'ambito di studio di questa trattazione, questa caratteristiche **favorisce l'analisi** dei contenuti di questi messaggi.
- Twitter permette l'inserimento di link esterni e contenuti multimediali, ma soprattutto, la possibilità di effettuare delle **menzioni** (l'equivalente dei *"tag"* di Facebook) anteponendo il simbolo @ ai nomi degli utenti (in particolare a quelli che vengono definiti nella terminologia Twitter, *screename*). Queste, oltre a condividere il messaggio con una o più persone specifiche, permettono di creare delle

conversazioni bilaterali, consentendo agli utenti menzionati di rispondere ai tweet altrui.

- Seguendo il *trend* degli ultimi anni, è stata inserita la possibilità di aggiungere degli ***hashtag*** ai messaggi, anteponendo al ***topic*** del dibattito sul quale si vuole andare a trattare, il simbolo #.
- Twitter, com'è già stato accennato nell'introduzione di questo capitolo, è stato scelto da **enti governativi** e **amministrazioni pubbliche**, per la comunicazione ufficiale di aggiornamenti ed eventi sociali.
- Infine Twitter è stato oggetto di molti studi e ricerche, ed avendo ottenuto questo grande riscontro popolare, è stato accompagnato da un forum di supporto agli sviluppatori (https://dev.twitter.com/) e da un sistema di **API** in costante aggiornamento. In particolare, sono stati effettuati *porting* delle stesse API, per i più noti linguaggi di programmazione, in modo da favorire l'utilizzo di queste all'interno di progetti trasversali.

2.3 API Twitter

Le API pubbliche fornite da Twitter, vengono rese disponibili a ricercatori ed utilizzatori, per la realizzazione di applicazioni e servizi che necessitino di interagire con lo stesso *social network*. L'utilizzo di tali API è gratuito, ma si richiede agli sviluppatori di sottostare a delle limitazioni, le quali vengono imposte sia in relazione al tipo di applicazione (quindi al tipo di servizio e di API utilizzato) sia in base agli utenti che effettuano l'autenticazione.

Le API per accedere ai dati di Twitter possono essere classificate in due categorie, basate sulla loro architettura e sul metodo di accesso implementato:

- **REST APIs**, realizzate secondo l'architettura REST, fortemente utilizzata negli ultimi anni per realizzare servizi web. Queste API si basano sulla strategia *"pull"*, ovvero l'utente riceve i dati solamente se ne ha fatto precedentemente richiesta.
- **Streaming APIs**, forniscono un flusso continuo di dati pubblici. Queste API infatti, si basano sulla strategia *"push"*, e dopo aver

ricevuto una richiesta di informazioni, esse restituiscono un flusso continuo di aggiornamenti, senza richiedere ulteriori input dall'utente.

È di fondamentale importanza specificare che, per questioni di sicurezza, dalla versione attuale delle API (1.1) viene richiesta l'autenticazione per svolgere qualsiasi tipo di azione.

Di seguito verranno analizzate in dettaglio le caratteristiche, del processo di autenticazione [Paragrafo 2.3.1] e quelle dei due metodi di utilizzo, REST e Streaming, rispettivamente [Paragrafo 2.3.2 e 2.3.2].

2.3.1 Open Authentication (OAuth)

Nella versione più aggiornata delle API (attualmente API v1.1), esiste un nuovo modello di autenticazione appartenente allo standard di autenticazione *open* denominato ***OAuth***. Questo modello prevede due tipologie differenti di autenticazione:

Application-user Authentication

Questa è la forma più comune di autenticazione, richiede che l'applicazione sia autenticata e autorizzata da Twitter, ed abbina ad essa, l'autenticazione dell'**utente** finale per il quale si sta svolgendo la chiamata alle API. Il processo di verifica delle **credenziali** dell'utente, restituisce un *User Access Token*, che viene abbinato lato applicazione.

Application-only Authentication

Questa forma permette all'applicazione di effettuare chiamate alle API, con la sola autenticazione dell'**applicazione**, quindi senza richiedere alcun contesto utente (alcuna credenziale agli utenti finali). Questa forma di autenticazione, presenta delle **limitazioni** rispetto alla *Application-user Authentication*, in quanto alcuni servizi delle API richiedono un contesto utente.

Indipendentemente dal metodo di autenticazione scelto, occorre tenere in considerazione che entrambi i contesti, possiedono determinati limiti di richieste e di chiamate effettuabili. Effettuando una richiesta con le API, si riceve infatti quello che viene denominato *"**Rate Limit Status**"*, ovvero la situazione aggiornata relativa alle chiamate e ai limiti ad esse legati.

Più in dettaglio, i *Rate Limit* di Twitter, vengono calcolati su *"**finestre temporali**"* di 15 minuti ciascuna. Ogni 15 minuti, i limiti vengono nuovamente aggiornati rispetto al loro quantitativo massimo. Quando possibile, utilizzando l'*Application-only Authentication*, si possiedono limiti maggiori, per esempio nel caso delle ricerche di *tweet* (*GET search/tweets,* il limite per *User Auth* è di 180 richieste, mentre per *App Auth* è di 450). Per maggiori informazioni relative a questi limiti, si rimanda alla documentazione sul sito di *Twitter*, alla pagina (https://dev.twitter.com/rest/public/rate-limits).

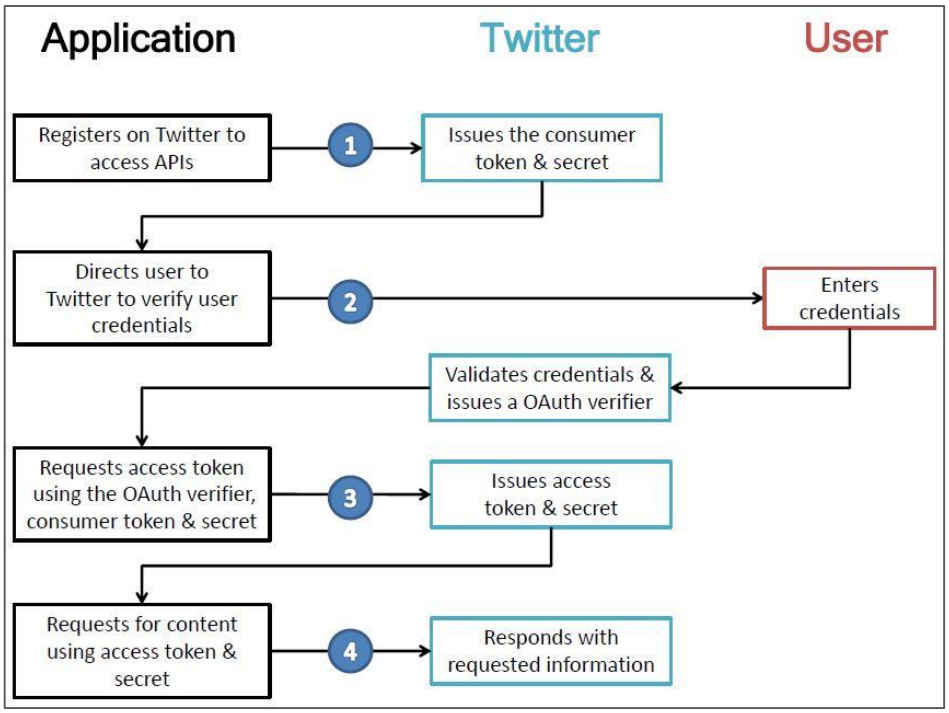


Figura 16: Twitter OAuth Wrokflow (rappresenta la sequenza ordinata di passi necessari per effettuare operazioni autenticate attraverso le API di Twitter).

Osservando [**Figura 16**] è possibile notare che, la procedura di autenticazione delle API, può essere scomposta in 4 differenti passaggi:

1. Ogni applicazione che utilizzi le API di Twitter, deve essere stata registrata sul sito: https://apps.twitter.com/.
   In questo modo, all'atto della registrazione, è possibile richiedere due informazioni necessarie per l'autenticazione dell'applicazione: ***Consumer Key*** *(API Key)* e ***Consumer Secret*** *(API Secret)*.

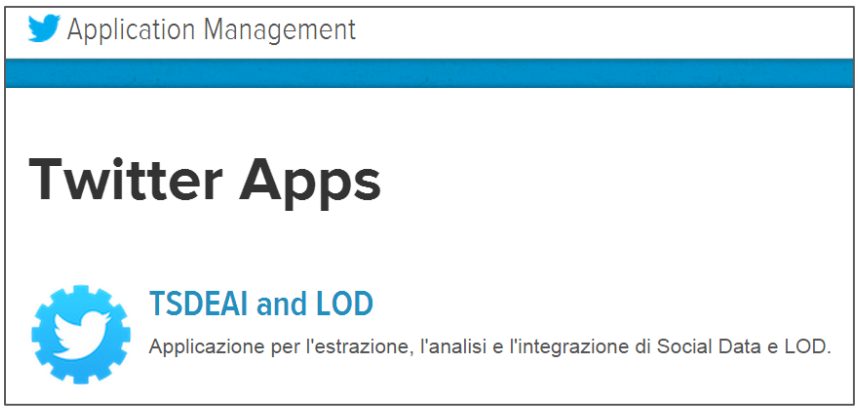


Figura 17: Registrazione dell'applicazione sulla pagina di riferimento Twitter.

2. L'applicazione utilizza le *Consumer Key e Consumer Secret*, per creare un link univoco, nel quale l'utente viene rediretto al processo di immissione delle credenziali. Dopo aver verificato le credenziali utenti, Twitter restituisce un *OAuth Verifier*, una sorta di PIN.
3. L'utente deve trasmettere il PIN all'applicazione, in modo che essa possa richiedere un *Access Token* e un *Access Secret* univoci, da assegnare all'utente che ha effettuato la *login*.
4. Per mezzo di questi *Token*, l'applicazione autentica effettivamente l'utente su Twitter, e fornisce la possibilità di effettuare chiamate alle API, utilizzando il contesto utente appena autenticato.

NOTA: Rispetto alla procedura appena completata, per effettuare un'autenticazione del tipo *Application-only Auth*, dalla stessa pagina in cui si è registrata l'applicazione su Twitter, è possibile ricevere anche l'*Application Access Token*. Questo *token* permette di effettuare chiamate alle API in maniera autenticata, utilizzando un contesto applicazione.

2.3.2 REST API

Le *REST API* forniscono un metodo di accesso programmato alle letture e scritture sui dati di Twitter. Queste API si basano sull'architettura *REST*, utilizzata in molti altri servizi web, e utilizzano la strategia *"pull"*.

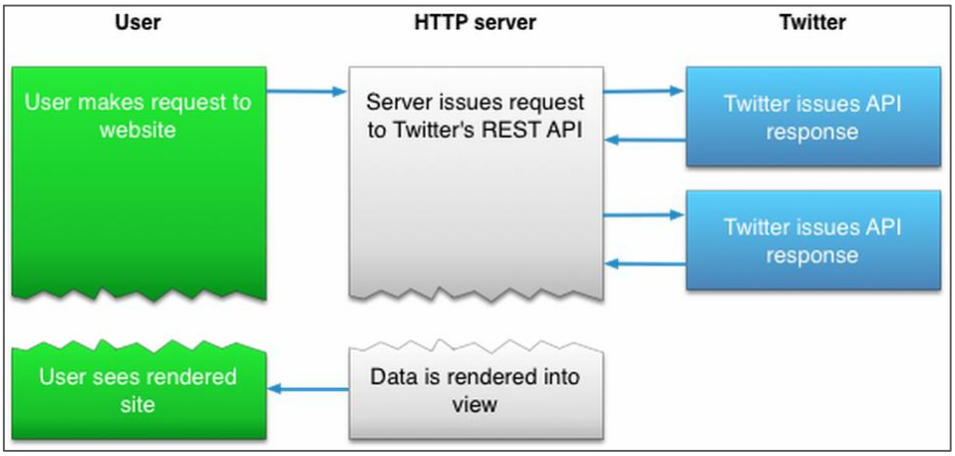


Figura 18: Processo di chiamata alle API REST.

Come mostrato in [**Figura 18**], per risolvere le richieste effettuate dall'utente, l'applicazione deve reindirizzare le richieste alle API di Twitter. Alla ricezione delle risposte, dovrà poi mostrare i risultati all'utente che ne ha fatto richiesta.

2.3.3 Streaming API

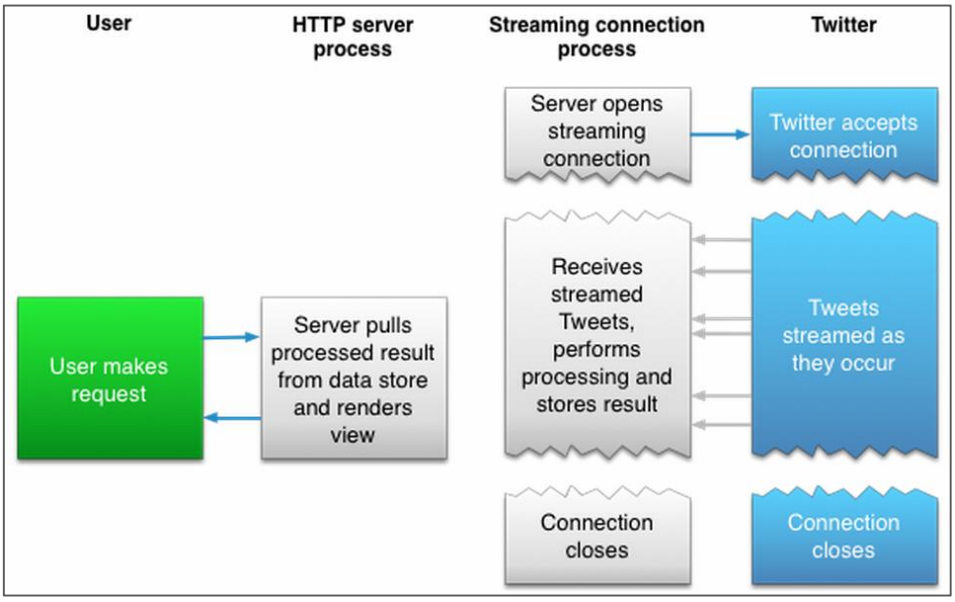


Figura 19: Processo di chiamata alle API Streaming.

Come mostrato in [**Figura 19**], il processo per le *STREAMING API*, è completamente diverso. L'applicazione deve aprire, e tenere aperta, una connessione con Twitter, per questo motivo spesso si delega ad un altro processo questa operazione. Nella suddetta figura si vede un processo denominato *Streaming Connection Process*, adetto all'apertura della connessione, e alla ricezione dei *tweet*, proveniente dal flusso continuo di aggiornamenti che vengono prodotti e condivisi sul *social network*. Per questo motivo occorre che i *tweet* così ricevuti vengano continuamente filtrati, aggregati, e infine memorizzati in un *data store*, in modo che alla necessita, possano essere reperiti e presentati all'utente. Questo modello è sicuramente più complesso del precedente [Paragrafo 2.3.2] ma permette di realizzare un processo di estrazione e analisi in *real-time*.

2.4 Twitter per la Gestione delle Emergenze

Il fattore che in assoluto ha influenzato maggiormente la scelta del *social network* di estrazione, è stato il fatto che Twitter sia stato preso e scelto dalle più grandi organizzazioni mondiali, come da enti governativi e aziende, sia nell'ambito privato che pubblico, come mezzo di digitalizzazione e di **comunicazione mediatica**.

Nella scena mondiale esistono numerosi siti che analizzano questo fenomeno sulle piattaforme sociali, e presentano i loro risultati, per fornire report, valutazioni e statistiche, riguardanti le stesse aziende o enti interessati.

Un esempio di queste, è:

Socialbakers (reperibile al link: http://www.socialbakers.com/statistics/).

Questo sito, che sta avendo una grande diffusione nel mondo (si pensi che possiede già 2500 clienti dislocati su oltre 100 nazioni differenti) permette di tracciare, analizzare ed effettuare benchmark, su oltre 8 milioni di profili *social* reperiti dai maggiori *social network*.

In questo caso è stato utilizzato, nella sua variante gratutia e limitata, per osservare l'andamento e la diffusione delle principali pagine e community mediatiche, sul *social network* Twitter.

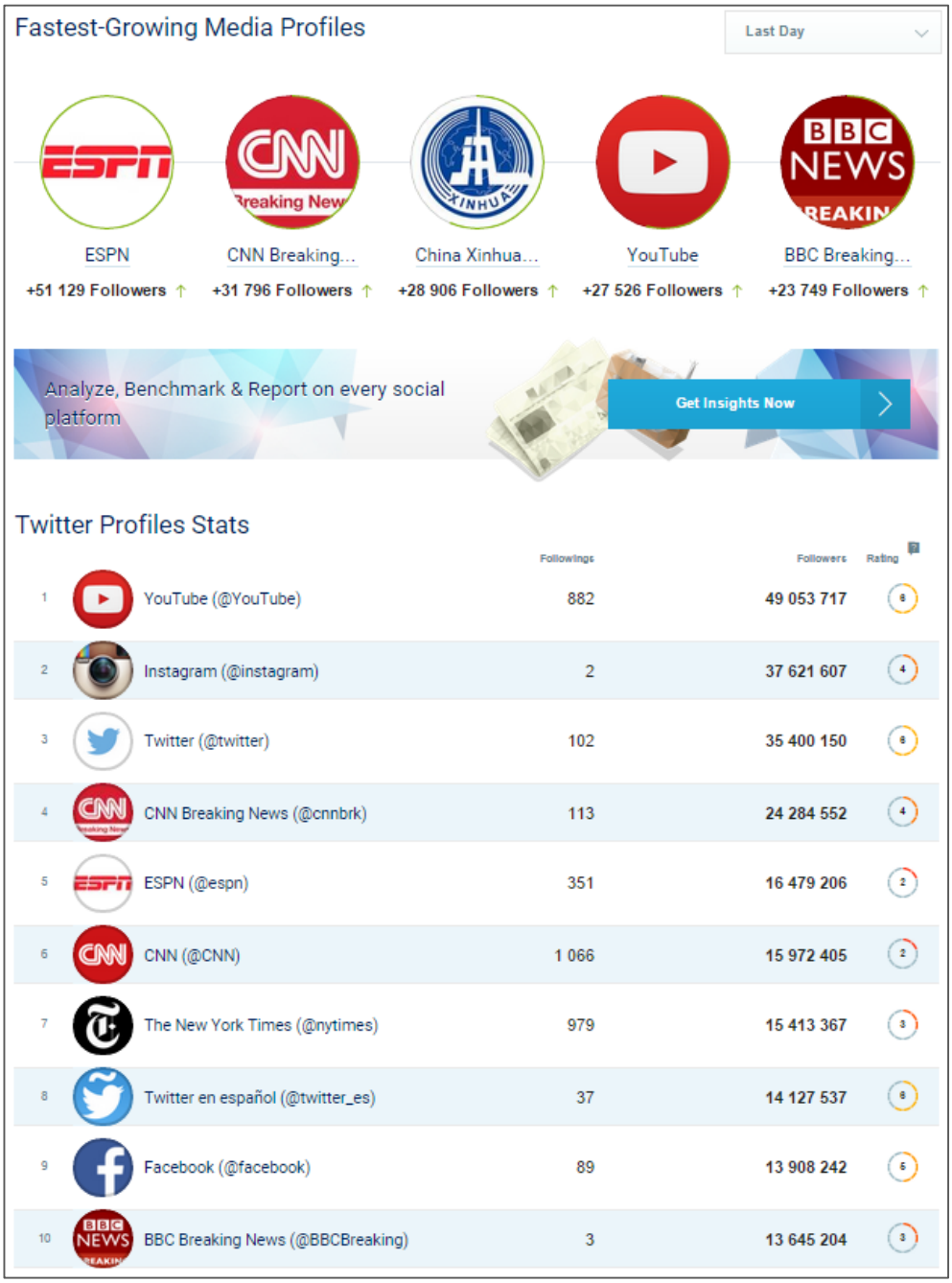


Figura 20: Statistica dei dieci profili twitter più utilizzati in ambito *media* e *comunicazione*. In alto si nota una statistica giornaliera, con i profili in crescita, in relazione agli ultimi *follower* acquisiti.
(Fonte: http://www.socialbakers.com/statistics/twit)

2.4.1 Comunicazione di un Evento Critico

La comunicazione mediatica per mezzo dei *social network*, può comprendere i più svariati campi d'interesse. Si pensi ad esempio all'utilizzo di essi in ambito di *marketing* e campagne pubblicitarie dei prodotti, oppure in ambito **politico**.

Nel corso degli ultimi anni infatti, si è visto un crescente utilizzo dei mezzi di comunicazione virtuale da parte di *leader* ed esponenti politici.

Primo fra tutti, il Presidente Americano *Barack Obama*, il quale, nella sua campagna elettorale del 2008, decise di rinforzare la propria immagine e le proprie adesioni popolari, facendo un largo uso dei *social network*, ed in particolare di Twitter [23].

La stessa situazione si è vista anche in Italia, durante gli ultimi due governi. Prima con il Presidente del Consiglio Enrico Letta, poi con l'attuale Presidente Matteo Renzi, il quale a differenza del primo, ha deciso di ufficializzare questo mezzo di comunicazione moderno. Infatti, dalla fine del 2012, oltre ai profili personali dei ministri, sono state create e certificate alcune pagine di Twitter, in modo da permettere la comunicazione di eventi e manovre ufficiali promosse dal governo italiano. (Si veda per esempio, la pagina ufficiale della Presidenza del Consiglio dei Ministri, *@Palazzo_Chigi*).

Allo stesso modo, anche nel contesto della comunicazione di **eventi critici** e di disastri naturali, sono state istituite nuove pagine ufficiali, nella maggior parte dei paesi del mondo.

Inoltre, per andare incontro a questo movimento, Twitter nel Settembre del 2013, ha istituito un nuovo mezzo di comunicazione dedicato alla comunicazione di eventi di allerta, in modo da favorirne la fruizione e la lettura da parte degli utenti finali. Questo nuovo sistema è stato denominato ***Twitter Alerts*** [24], ed è stato definito dalla stessa compagnia, nel seguente modo:

> *«Twitter Alerts is a new way to get accurate and important information when you need it most.»*

Questo meccanismo consiste nel permettere ad enti e istituzioni, locali, nazionali o internazionali, che forniscono informazioni urgenti alla popolazione, di poter pubblicare dei messaggi prioritari. Tra le categorie di enti che possono aderire a questo progetto, le seguenti possiedono priorità sulle restanti:

- Agenzie per la *sicurezza* e il rispetto delle regole.
- Agenzie per la gestione delle situazioni di *emergenza*.
- Enti governativi cittadini, municipali o regionali, che provvedono a servizi di comunicazione per la gente.
- Enti federali o statali, per la comunicazione e la prevenzione in ambiti di crisi o disastri.
- Enti umanitari o di volontariato, i quali forniscono supporto e informazione in situazioni di emergenza (si veda la definizione delle *NGOs*, ovvero delle *Non-Governmental Organization*).

Qualsiasi utente, *follower* di una pagina relativa ad un ente che abbia aderito al progetto *Alerts*, che decida di volerne seguire i "messaggi di allerta", acconsente alla possibilità che Twitter invii al suo *smartphone*, **notifiche *push*** o **sms**, per comunicargli questo genere di eventi.

Le comunicazioni ricevute con questo genere di sistema, sono state pensate per i seguenti scopi:

- Avvisi ed allerte, per imminenti pericoli.
- Istruzioni preventive e comunicazioni urgenti di sicurezza.
- Direzioni e modalità di evacuazione di luoghi a rischio.
- Informazioni per l'accesso e la fruizione di beni di prima necessità.
- Informazioni riguardo interruzioni di servizio e mezzi di comunicazione.
- Informazioni legate a vie di fuga, mezzi di trasporto e infrastrutture.

2.4.2 Processo di Analisi dei tweet di Allerta

Il processo di analisi dei *tweet* di allerta, è fortemente condizionato dall'ambito nel quale il messaggio viene lanciato, ed anche dallo scopo che tale comunicazione vuole avere sulla popolazione.

Per prima cosa infatti, occorre definire gli ***stadi socio-temporali*** di un ***disastro***.

Stage 0 PRE-DISASTER	Stato di informazione sociale: analisi e studio della situazione che precede il disastro.	0
Stage 1 WARNING	Attività di precauzione, includono messaggi ed avvisi di comunicazione agli utenti.	1
Stage 2 THREAT	Percezione del cambiamento (punto di svolta), che suggerisce azioni di sopravvivenza.	2
Stage 3 IMPACT	**Evento di crisi in atto**, comunicazioni in diretta da parte di enti e singoli individui.	4
Stage 4 INVENTORY	Calcoli e constatazioni dei danni provocati a persone, risorse, mezzi e infrastrutture.	3
Stage 5 RESCUE	Interventi spontanei e non-organizzati, di recupero e aiuto alle persone colpite.	3
Stage 6 REMEDY	Azioni ed interventi da parte delle organizzazioni competenti al recupero e alla messa in sicurezza.	2
Stage 7 RECOVERY	Recupero e ricostruzione dei danni. Preparazione di mezzi e misure preventive per eventi analoghi.	1

Tabella 2: Fasi *socio-temporali* di un disastro o un evento di crisi, con relativa descrizione delle azioni da compiere. A destra, una legenda verticale, presenta un indice di colorazione accompagnato da numeri, i quali indicano l'impatto e la gravità di ogni fase. (Dati ottenuti traducendo e rivisitando le informazioni di [25]).

In secondo luogo, occorre pensare a quali azioni si vogliono attuare sui dati estratti dal *social network*, durante le fasi di gestione dell'evento critico.

Esistono due macro-aree di intervento, che si differenziano principalmente per lo scopo e la missione da compiere.

- **Gestione dell'evento in *"real-time"***: consiste nell'estrazione dei messaggi reperiti dalle piattaforme sociali, e nella **repentina** (pressoché immediata) analisi di tali dati. Questo meccanismo richiede alte capacità di computazione, e un'infrastruttura che consenta l'analisi e la rappresentazione dei risultati, e delle azioni di intervento successive, nel modo più rapido possibile. È sottointeso, che, nel caso di Twitter, a questo scopo si debbano utilizzare le *API Streaming*, per ottenere i *tweet* in tempo reale.

- **Gestione dell'evento *"a posteriori"*:** questa consiste nella realizzazione di un'infrastruttura che permetta di svolgere, in una prima fase, un'attività di estrazione su messaggi e informazioni di eventi che si sono conclusi recentemente o che sono ancora in atto. E in un secondo tempo, di poterli analizzare e rappresentare, facendo uno studio a posteriori, il quale mostri analisi, statistiche e valutazioni sul modo in cui si è gestita la crisi e su quali effettive comunicazioni siano state attuate. Questo rispetto al precedente, non permette di intervenire immediatamente sull'evento di crisi, ma permette di effettuare azioni correttive per i successivi casi potenzialmente simili. Inoltre questa soluzione non richiede particolari capacità computazionali, in quanto non necessita che le azioni di analisi siano svolte in concomitanza con l'evento, e permette di utilizzare le *API REST*, anche per reperire contenuti condivisi nelle giornate precedenti, fino ad un massimo di circa 8-9 giorni antecedenti la data di estrazione.

Una volta scelta la tipologia di servizio che si vuole svolgere, occorre determinare su quali **aree di analisi** si andrà ad operare. Le azioni applicabili in fase di analisi dei dati estratti da Twitter, sono classificabili secondo tre principali categoria:

- ***Descriptive Analytics (DA)**:* consiste essenzialmente nell'applicare delle metriche per effettuare valutazioni e statistiche relative a *tweet, utenti e URL* (in merito ad esempio alla loro quantità, alla loro distribuzione, alla loro visibilità o alla conformazione di gruppi o classi).
- ***Content Analytics (CA):*** consiste principalmente nell'analizzare il contenuto dei *tweet*, applicando tecniche di *Word Analysis, Hashtag Analysis e Sentiment Analysis* (affidandosi principalmente a fattori quali la frequenza dei termini o le loro associazioni).
- ***Network Analytics (NA)*:** consiste nello studiare le conformazioni di reti e sotto-reti che si vengono a creare, dall'interazione tra gli utenti e la condivisione dei loro messaggi. Si distingue in *Topological Analysis, Centrality Analysis e Community Analysis.*

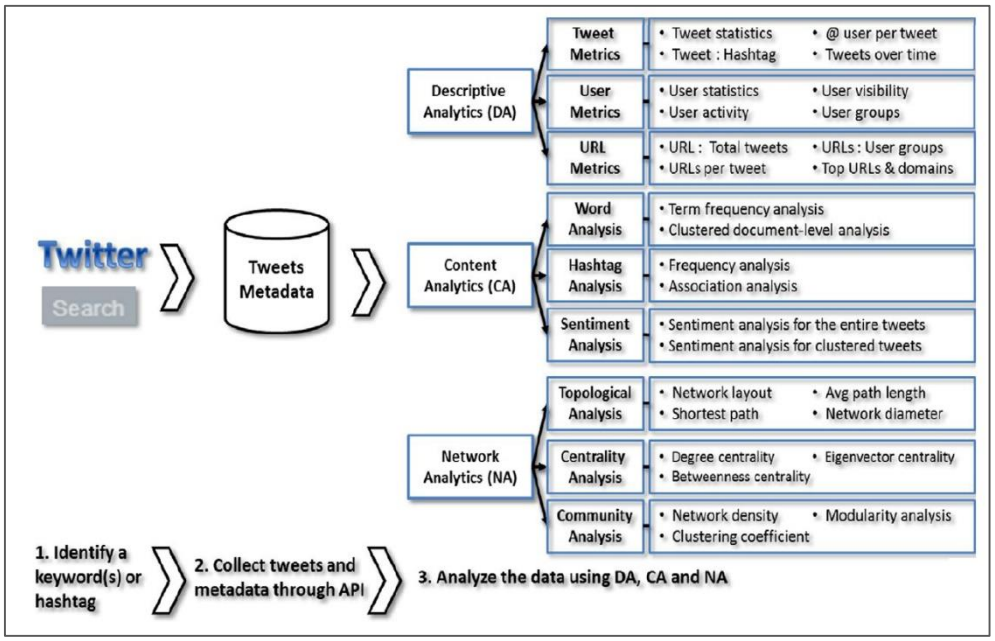


Figura 21: Classificazione delle principali tecniche di estrazione ed analisi dei dati reperiti da Twitter: *Descriptive Analytics (DA), Content Analytics (CA), Network Analytics (NA).* (Immagine tratta da [26]).

2.4.3 Affidabilità dei Tweet in caso di Emergenza

Prima di iniziare a trattare un argomento così delicato come la gestione di informazioni critiche durante le fasi di un evento di emergenza, e poter effettuare delle considerazioni veritiere sui risultati ottenuti estraendo dati da Twitter; si sono dovuti fare studi sull'**affidabilità** e la **credibilità** delle informazioni che circolano in rete.

In letteratura è infatti possibile reperire studi ed analisi che hanno valutato il comportamento degli utenti, e le tipologie di messaggi, ricevuti nell'ambito di un evento critico.

Le ipotesi fatte nella fase di ideazione del progetto sono state confermate e motivate nei risultati presentati in [27]. Nel corso di tale studio infatti sono stati studiati in dettaglio il contenuto dei messaggi, le caratteristiche degli utenti coinvolti (considerando anche *followers* e *following*) e le caratteristiche delle reti di collegamento e di condivisione che si sono create. L'evento studiato, appartiene alla categoria scelta anche da questa trattazione, infatti si tratta di un evento di terremoto tenutosi in Cile nell'anno 2010.

I risultati ottenuti da tale studio comprovano che:

- Le caratteristiche topologiche delle **reti** di utenti non variano durante eventi critici, rispetto alla loro quotidiana conformazione.
- Passando da piccole o medie *community*, a grandi ***community***, non si ha alcuna perdita di generalità.
- Il **vocabolario** utilizzato nel corso di un crisi, assume una "bassa varianza". Questo fatto permette di assumere che i *tweet* di allerta, tendono a descrivere ***topic*** e concetti **globali** e **comuni**, riducendo così l'entropia di rete.
- Il risultato più atteso di questa analisi, riguarda lo studio sulla propagazione di *"verità confermate" (**confirmed truths**)* rispetto alle *"false affermazioni" (**false rumors**)*. Seppur tali considerazioni derivino solamente da un solo evento critico, e la quantità di informazioni sulle quali sono state effettuate le valutazioni, non risulta essere molto elevata; i risultati ottenuti sembrano confermare che le voci false tendono ad essere **messe maggiormente in discussione**, rispetto alle vere affermazioni.

Questi risultati supportano la tesi ipotizzata, confermando che in casi di emergenza come questi, nel quale viene messa a rischio l'incolumità di tutta la collettività, le informazioni reperite possono essere considerate vere, quasi nella totalità dei casi. I pochi o sporadici casi di utenti che vogliono volutamente creare scompiglio o effettuare azioni di "disinformazione" (a volte anche in maniera involontaria), inviando messaggi di false allerte, vengono comunque "**auto-identificati**" dalla stessa rete, che attraverso lo strumento dei commenti e delle condivisioni, mette in discussione tali avvenimenti.

I risultati e l'efficacia delle analisi sui *tweet*, dipendono quindi in gran parte, dalla bontà degli algoritmi di analisi e classificazione utilizzati, e dalla capacità degli osservatori, nel riuscire a contestualizzare i pochi, e precedentemente citati, casi di *false rumors*.

2.4.4 Casi di Successo

Lo stesso Twitter, ha dedicato una sezione del suo sito di riferimento, per raccontare le vicende e le storie delle realtà che hanno avuto successo grazie allo stesso *social network*. In queste pagine si possono trovare casi di successo organizzati secondo diversi criteri, dal marchio all'industria di provenienza, dalla tattica agli obiettivi da raggiungere sul mercato, fino ad arrivare ad una classificazione per dimensione di azienda.

In relazione all'ambito di questa trattazione, quindi alla condivisione di contenuti mediatici ed informazioni relative ad eventi di importanza critica, è possibile visitare la sezione *"Success Stories: Media, News & Publishing"* (https://biz.twitter.com/success-stories/industry/media-news-publishing) di Twitter, nella quale vengono proposti i principali successi da parte di compagnie e aziende di informazione.

Osservando le interviste e gli articoli dei principali giornali mondiali, nonché gli articoli di ricerca, si trovano svariati casi di eventi critici, che hanno avuto una forte condivisione sociale su Twitter.

Di seguito vengono elencati alcuni di questi eventi, appartenenti a categorie di eventi critici differenti (terremoti, uragani, esondazioni, attentati) ordinati cronologicamente rispetto alla loro data di apparizione:

- 2010, Aprile, 14 - Terremoto in Yushu (Qinghai, Cina), con 2698 vittime, 270 scomparsi, e migliaia di persone ferite, senza contare i danni ad abitazioni e infrastrutture (si veda l'analisi in [28]).
- 2011, Marzo, 11 – Tsunami e Terremoto in Tohoku (Giappone), con 15703 morti accertate, 5314 feriti e 4647 dispersi.
- 2012, tra 31 Gennaio e 28 Febbraio – **Iniziativa Italiana**, durante le nevicate che hanno colpito la città di Firenze nel mese di Febbraio. Questo è uno dei primi casi di utilizzo di *social network*, da parte degli enti pubblici italiani, avvenuto nel Comune di Firenze, con sindaco Matteo Renzi. Questo evento non può essere paragonato agli altri eventi citati (per gravità e importanza dei danni) ma risulta essere comunque un esempio importante di coesione sociale. Nel corso di

questa iniziativa è stato ad esempio proposto l'*hashtag*, *#firenzeneve*. (Per un'analisi più dettagliata si veda [29]).

- 2012, Ottobre, 22-29 – Uragano Sandy, soprannominato *"Superstorm Sandy"*, colpisce Giamaica, Cuba, Hispaniola, Bahamas, Florida, con 182 vittime accertate e circa 50 miliardi di danni. Questo è stato uno degli eventi storici e mediatici che ha avuto più riscontro sulle piattaforme sociali virtuali, vista l'entità e la vastità dei danni arrecati da tale forza della natura (si veda [30]).
- 2013, Aprile, 15 – Attentato alla Maratona di Boston (Stati Uniti d'America), caratterizzato dall'esplosione di due bombe, con 3 morti accertate, 264 feriti.
- **2015, Gennaio, 7-8-9** – Attentato alla sede del giornale ***Charlie Hebdo*** a Parigi (Francia), caratterizzato da un attacco armato alla redazione del giornale e in altre due località vicine, con 20 morti e 11 feriti. Nonostante abbia avuto conseguenze di molto inferiori (in termini numerici) rispetto alle precedenti, ha segnato drasticamente la sicurezza e la serenità della capitale francese, alimentando un movimento enorme su tutti i maggiori *social network*, ed avviando una vera e propria campagna definita dall'*hashtag #JeSuisCharlie*.

Capitolo 3

Applicazione TSDEAI: Analisi e Progettazione

In questo capitolo verrà descritto il contesto del problema, andando ad affrontare il processo di analisi logico-concettuale, che porterà al vero e proprio processo realizzativo, successivamente descritto nel seguente capitolo. Le scelte realizzative ipotizzate, verranno presentate e motivate, tenendo conto delle problematiche e dei vincoli imposti, ed evidenziando i vantaggi e gli svantaggi riscontrati.

3.1 Descrizione del problema

Vista l'analisi effettuata nella prefazione introduttiva di questa tesi, e nei successivi due capitoli, e considerato il crescente interesse di enti di ricerca ed aziende, riguardo l'analisi di dati prelevati dai *social network*, si è scelto di progettare e realizzare un'applicazione che permettesse di sperimentare e realizzare tecniche fini a questo scopo.

In particolare si è deciso di studiare il contesto e le forme di comunicazione e trasmissione di informazioni relative ad eventi catastrofici ed emergenze, per mezzo di piattaforme virtuali di condivisione sociale.

All'interno di questo studio, si è evinto che il *social network* più adatto, e maggiormente utilizzato in forma "ufficiale" (da enti governativi e amministrazioni pubbliche), e anche dai singoli individui e dalle *community* di utenti (appassionati e devoti alle differenti cause), possa essere identificato in *Twitter*. (Si veda Capitolo 2, per le informazioni e le motivazioni dettagliate).

Per quanto riguarda invece lo **scopo** realizzativo di questa tesi, sono state delineate fin da subito, quattro principali richieste da soddisfare:

1) Permettere l'estrazione ed il reperimento di dati ed informazioni salienti, riguardanti il contesto di emergenza scelto.
2) Fornire funzionalità di classificazione automatica dei dati ottenuti, per valutare ed identificare i dati "realmente interessanti" per lo scopo desiderato.
3) Consentire l'integrazione e l'arricchimento dei contenuti estratti, introducendo informazioni e nozioni prelevate dal mondo degli *open data*.
4) Fornire strumenti per l'analisi e la presentazione di dati statistici, e informazioni dedotte da essi, permettendo di suddividere modularmente la fase di estrazione da quella di analisi e presentazione.

3.1.1 Requisiti, Vincoli e Problematiche

Vista la natura del problema, e le richieste imposte nella fase iniziale del progetto, si è evinto subito quanto fosse importante effettuare uno studio e un'analisi accurata di ogni singolo requisito, in modo tale da ridurre al minimo i possibili conflitti e le possibili problematiche future, eventualmente riscontrabili in fase di realizzazione.

Implementare un'applicazione di questo genere, che richiede l'integrazione e la comunicazione tra servizi differenti, sia per origine che per tipologia di funzionalità fornita, richiede uno sforzo maggiore in fase di progettazione.

Di seguito vengono elencati alcuni aspetti e requisiti, che si sono dovuti prendere in considerazione, nell'effettuare le scelte che hanno portato alla soluzione *software* finale.

Si noti come le **quattro principali "macro-funzionalità"** precedentemente citate, siano state utilizzate per raggruppare le sotto-funzionalità evinte dal processo di analisi.

Estrazione di contenuti (da *social network*)
• Permettere l'esecuzione di interrogazioni e richieste di informazioni, interagendo ed interfacciandosi con i servizi offerti da *Twitter*. • Consentire l'esecuzione e la messa in attesa di "operazioni di scaricamento", in modo da permettere di programmare le operazioni e lasciare il programma in esecuzione come processo in *background* (senza richiedere l'intervento dell'utente). • Permettere la memorizzazione dei dati reperiti dal *social network*, garantendone la consistenza, e consentendo di suddividerli e organizzarli secondo criteri utili. • Consentire la creazione di liste di utenti/pagine o la memorizzazione delle interrogazioni più frequenti, in modo da poterle ripetere in tempi successivi.
Analisi e Classificazione
• Permettere la categorizzazione e classificazione dei testi reperiti dal *social network*, in maniera automatica o semi-automatica. In particolare consentire l'identificazione dei *topic* più ricorrenti. • Consentire l'esecuzione di test su questi dati, in modo da valutare la miglior parametrizzazione delle richieste di classificazione. • Valutare la diffusione e la viralità di particolari informazioni selezionate, in modo da ricostruire la rete delle loro condivisioni tra gli utenti dei social network.
Integrazione di informazioni esterne (da *open data*)
• Fornire informazioni aggiuntive, reperite da banche dati attinenti al bacino dei *linked-open-data*. • Integrare gli *open-data* con le entità estratte nella fase di classificazione, permettendone l'abbinamento e la valutazione.
Presentazione dei risultati
• Effettuare delle stime sui risultati ottenuti, e permettere la visualizzazione di tali dati con l'apposita rappresentazione grafica (esempio diagrammi, grafici, tabelle, …). • Fornire ogni possibile informazione aggiuntiva, che possa accrescere il contenuto informativo espresso (considerato il contesto).

Tabella 3: Funzionalità principali dell'applicazione, emerse in fase di analisi.

Occorre inoltre tenere in considerazione che, alle funzionalità richieste (espresse in **Tabella 3**), sono da aggiungere tutte quelle funzionalità che fanno parte delle "buone regole di realizzazione" di un *software*. Quindi si deve garantire, nel migliore modo possibile, la consistenza e l'integrità dei dati reperiti, la fruibilità e la responsività dell'applicazione, anche durante l'esecuzione di compiti onerosi. Includendo in questa parte, tutta la gestione delle interazioni con l'utente finale (utilizzatore dell'applicazione) e la sincronizzazione tra i diversi processi. Tenendo conto di quelle che sono le risorse in condivisione tra di essi, applicando delle tecniche di gestione dei conflitti e di mutua esclusione.

3.2 Casi d'uso

Analizzando le funzionalità richieste, ed i requisiti precedentemente citati, è stato possibile far emergere i casi d'uso dell'applicazione.

Per motivi di spazio, e di presentazione, di seguito verrà espressa solamente una vista aggregata dei casi d'uso principali, che caratterizzano il problema.

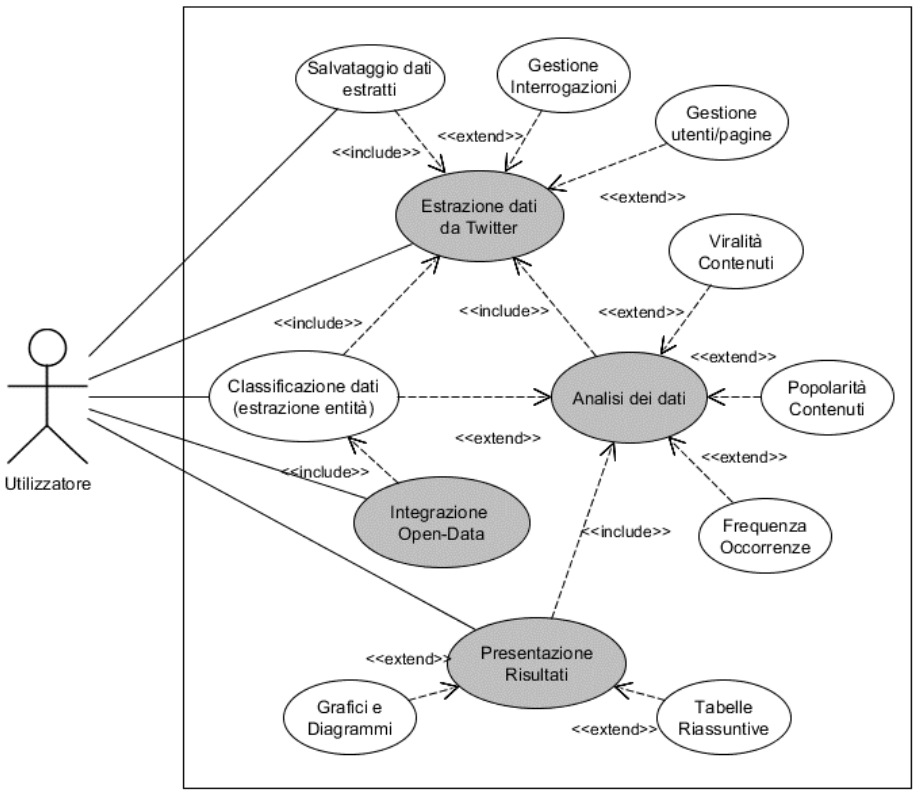


Figura 22: Diagramma UML dei casi d'uso.

3.3 Architettura Logica

Tutta l'architettura logica dell'applicazione, è stata fondata sul *design pattern* ***Model-View-Controller (MVC)***. Questo *pattern architetturale* è particolarmente adatto per la realizzazione di sistemi software di questo tipo, nei quali l'interfaccia utente o *GUI (Graphical User Interface)* possiede un ruolo fondamentale, per la gestione ed evacuazione delle richieste dell'utente.

Questo pattern che si può posizionare nel livello presentazione di un'architettura *multi-tier (o multi-strato)*, vuole rafforzare la suddivisione logica tra i diversi comparti dell'applicazione. In particolare consente di separare logicamente il livello di presentazione dei dati, da quello della logica di business.

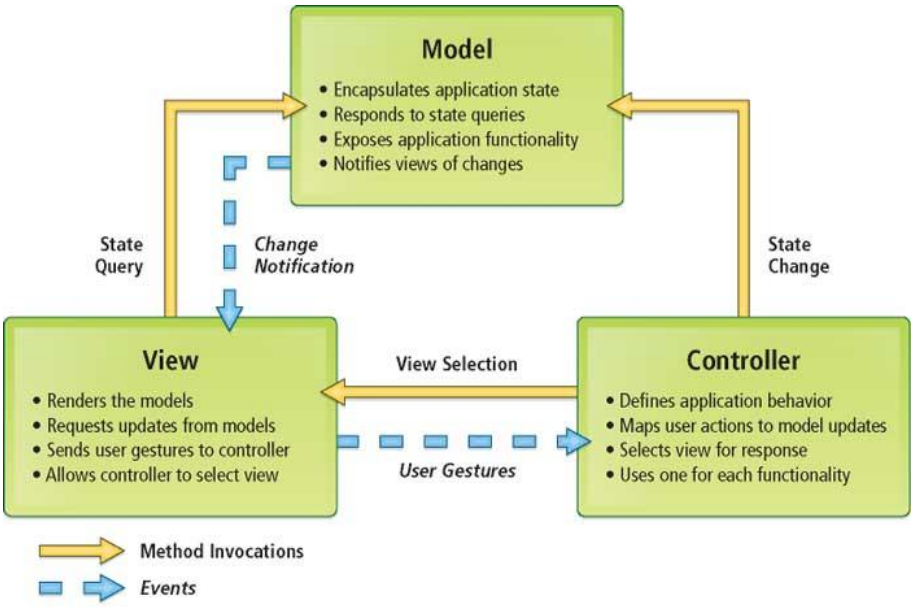


Figura 23: *MVC (Model View Controller).*

In particolare, in questa applicazione:

- Il ***model*** (che solitamente rappresenta il *core* dell'applicazione) incapsulerà lo stato dell'applicazione e definirà le operazioni che potranno essere eseguite sui dati. Esponendo alla *view* e al *controller*, le funzionalità di accesso e aggiornamento dei dati, definite secondo le regole di *business*. Il *model* dovrà inoltre notificare ai componenti della *view* gli eventuali aggiornamenti apportati in seguito alle richieste del

controller, al fine di mantenere sempre allineata la vista dell'utente con i dati del sistema.

- La ***view*** (è responsabile della logica di presentazione) si occuperà della *GUI* di interazione con gli utenti, mantenendo aggiornati i dati presentati all'utente. Nel caso particolare, permetterà di gestire ed eseguire le interrogazioni per l'estrazione dei dati *social*, consentirà la visualizzazione e l'analisi di tali dati, mostrando informazioni aggiuntive legate alle entità classificate, o reperite dagli *open-data*. Infine servirà anche per presentare visivamente i dati statistici e i risultati d'analisi, per mezzo di grafici e tabelle riassuntive.
- Il ***controller*** (il quale svolge la logica di controllo) avrà la responsabilità di "trasformare" le interazioni dell'utente sulla *view*, in azioni eseguite dal *model*.

Per quanto riguarda invece il vero e proprio diagramma delle classi, vista l'entità del progetto e la dimensione che ha raggiunto, e visto anche l'obiettivo preposto, si è deciso di andare ad introdurre porzioni del diagramma delle classi direttamente nel capitolo successivo [Capitolo 5].

Questo permetterà di introdurre gradualmente le parti più interessanti di tutto il diagramma, in modo da poterle spiegare in dettaglio, accompagnandole a porzioni di codice che ne faciliteranno la comprensione.

3.4 Scelte Realizzative e Tecnologie Utilizzate

Tutto il progetto è stato caratterizzato da una scelta realizzativa stringente. Visto l'ambito di studio, e il contesto universitario in cui si trova, e ipotizzando possibili sviluppi futuri per questo progetto, si è deciso di utilizzare solamente ***software open-source*** o **gratuito**, in modo da non dover incorrere in licenze proprietarie e costi di utilizzo.

Come detto fin dal capitolo introduttivo di questa tesi, vista l'ampiezza di questo progetto, e le numerose funzionalità richieste, si è chiaramente scelto di cercare di riutilizzare tutto il codice e le librerie presenti in rete, in grado di svolgere tali funzionalità. Questo ha permesso di concentrarsi sull'integrazione

di questi servizi e sulla messa in opera dell'applicazione, in modo da effettuare dei test reali su dati concreti, che potessero far emergere le potenzialità di questa idea.

Questo è il normale meccanismo oggigiorno, per realizzare *software* all'avanguardia, sia esso in contesti aziendali privati o in ambiti di ricerca pubblici. Tranne in particolari contesti dove venga richiesto uno sviluppo *ad hoc*; viste le numerose fonti *online* e le corpose librerie che implementano le più svariate funzionalità, sarebbe inutile e ridondante, andare a ricreare tutte le differenti funzionalità dal loro inizio.

Visti i tempi di realizzazione e la vastità del progetto, si è scelto a maggior ragione di integrare e riutilizzare quante più librerie gratuite possibili. Nei paragrafi successivi verranno brevemente descritte le tecnologie e le librerie utilizzate, motivando le ragioni che hanno portato alla loro scelta.

3.4.1 Java in ambiente Eclipse

JAVA 8 – Linguaggio di programmazione Java, nella sua recente versione 8 (http://docs.oracle.com/javase/8/).

Eclipse Luna 4.4 – Ambiente di sviluppo multi-linguaggio e multi-piattaforma (https://projects.eclipse.org/releases/luna).

Inutile entrare in dettaglio nella descrizione dell'ambiente di sviluppo Eclipse (tra i più utilizzati) e del linguaggio di programmazione Java. Unica nota da tenere in considerazione, riguarda la versione Java utilizzata, che dalla 8, ha introdotto importanti nuove funzionalità, dalle espressioni in *lambda calcolo* alle *Streaming API* (per facilitare operazioni sulle collezioni di dati). Particolarmente utile in questo contesto, e per la gestione dei dati reperiti da *Twitter* (memorizzati con identificativi a 64 bit), sarà il pieno supporto per le operazioni con dati primitivi di grandi dimensioni (Java ha introdotto ad esempio il confronto tra *long* a 64 bit, includendo la possibilità di utilizzarli come quantità *unsigned*).

3.4.2 Twitter4J (Twitter for Java)

Libreria per integrare i servizi di *Twitter* nelle applicazioni Java (supportando dalla versione Java 5, alle successive). Tra i vantaggi di questa libreria, vi sono:

- Zero dipendenze da librerie esterne.
- Supporto *built-in* (integrato) dell'autenticazione *OAuth*.
- Compatibilità completa con le API di *Twitter* aggiornate (1.1 version).

3.4.3 Google GSON (Json)

Libreria che permette di convertire Oggetti Java nella loro rappresentazione *JSON (Javascript Object Notation)*. E' anche possibile convertire una stringa JSON nell'equivalente Oggetto Java. Vantaggi di questa libreria, rispetto alle svariate altre:

- Supporta la conversione di "oggetti arbitrari" (contemplando i casi con gerarchie di ereditarietà innestate, e l'utilizzo di tipi generici).
- Fornisce metodi semplici per la conversione: *toJson()* e *fromJson()*.
- Rispetto agli altri progetti open-source per la conversione di JSON, questa libreria consente di convertire oggetti preesistenti, di cui non si possiede il codice, senza dover immettere delle annotazioni per la conversione, all'interno delle classi Java.

3.4.4 MongoDB (Non-Relational Database)

MongoDB è un software libero e *open source*, concepito per realizzare un database non relazionale, orientato ai documenti. (http://www.mongodb.org/).

Questo software gratuito, realizza un database di nuova concezione, che viene spesso classificato come *"NoSQL"*, e più in generale, come *database* non relazionale, che fa uso di documenti invece di utilizzare le tradizionali tabelle relazionali, e un linguaggio di *query* espressivo che non fa utilizzo di sql.

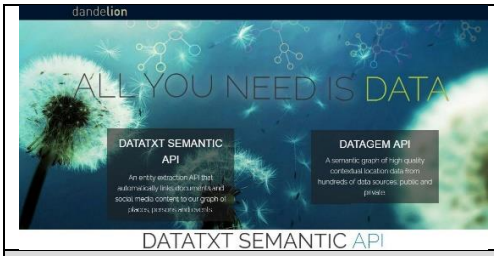
Il vantaggio chiaro di questa scelta sta nella possibilità di memorizzare documenti *JSON* aventi uno "schema dinamico", in quelli che vengono denominati documenti *BSON*. Il formato *JSON* già precedentemente citato e fortemente utilizzato in molti servizi web, anche in Twitter). Il formato *BSON (Binary JSON)* invece è stato ideato per MongoDB, con lo scopo di memorizzare strutture dati ed array associativi.

Questo genere di *database*, favorisce e velocizza numerose operazioni di gestione e memorizzazione di queste tipologie di dati, e per questo motivo è stato utilizzato in numerose piattaforme e servizi web, si pensi ad esempio a Ebay, Paypal o Foursquare.

Un ulteriore vantaggio di questa soluzione, sta nelle sue qualità intrinseche, quali l'alta scalabilità e flessibilità, e la robustezza nelle comuni operazioni di lettura e scrittura.

In questo particolare progetto, verrà integrato nella soluzione Java, e verrà utilizzato un ide grafico gratuito denominato MongoVue, per la gestione e visualizzazione dei documenti memorizzati dall'applicazione di estrazione dei *social data (MongoVUE, http://www.mongovue.com/).*

3.4.5 DataTXT-NEX (Named Entity eXtraction)



SPAZIODATI

DataTXT-NEX, è un servizio all'interno del progetto *Dandelion.eu* (https://dandelion.eu/products/datatxt/), gestito da *SpazioDati*. (http://www.spaziodati.eu/it/).

SpazioDati è un'azienda operante nel territorio italiano, la quale sfrutta tecnologie in ambito di *BigData* e *SemanticWeb*, con l'intento di realizzare due grandi servizi di API distinti. Il primo riguarda le *DataTXT Semantic API* (che verranno utilizzate in questo progetto) che forniscono funzionalità di classificazione ed estrazione automatica di contenuti. Il secondo, *Datagem API*, riguarda la creazione di un grande grafo di conoscenza semantica con dati reperiti da centinaia di sorgenti dati, siano esse pubbliche o private.

Questa libreria di API è stata scelta, visti e confrontati i suoi risultati, nella classificazione ed estrazione di *topic* da brevi testi provenienti da piattaforme *social*. È infatti stato riscontrato quanto le comuni tecniche di *Natural Language Processing (NLP)* non sempre riescano ad ottenere i migliori risultati, in testi brevi e caratterizzati da numerosi possibili errori sintattici o grammaticali.

Questa libreria risulta particolarmente efficiente proprio nel contesto dei Tweet (vista la loro breve natura di massimo 140 caratteri) e permette non solamente di estrarre le entità fondamentali contenuti in questi testi, ma anche di collegarle ai contenuti reperiti da DBPedia (e quindi dal mondo degli *OpenData*). Questo è reso possibile da un sistema di API REST, efficiente e scalabile, indipendente da linguaggi esterni e dalle tecnologie di *NLP*.

DataTXT-NEX supporta diverse lingue, e date le sue caratteristiche, si definisce l'unica *"Named Entity Extraction – Linking API"* sul mercato.

3.4.6 GeoNames (World GeoLocation DB)

GeoNames è un *database* geografico, che copre tutti i paesi del mondo, e contiene oltre otto milioni di locazioni geografiche (http://www.geonames.org/).

Questo database geografico mondiale, è completamente grautito, e racchiude al suo interno la quasi totalità dei paesi e delle cittadine presenti nel mondo, comprendenti di dati di localizzazione, nomi alternativi (in differenti lingue) e dati sulla popolazione e sulle dimensioni.

Questo database risulta essere un'ottima scelta per tutte quelle soluzioni che non vogliano dipendere da altri servizi di API (come possono essere le ben note API di GoogleMaps).

Nonostante questi servizi di API siano fortemente utilizzati, in svariati contesti, vista la loro affidabilità e diffusione, e vista la possibilità di non dover dipendere da una sorgente (per quanto possa essere aggiornata) memorizzata in locale.

Per questo progetto si è scelto comunque di preferire una soluzione come GeoNames, per non dover sottostare ad ulteriori limitazioni sulle richieste (che si sarebbero incontrate utilizzando un nuovo servizio di API in forma gratuita). Avendo ipotizzato un gravoso quantitativo di richieste necessario per analizzare tutte le locazioni dei differenti dati estratti da *social network*.

Verrà dunque scaricata una versione aggiornata di questo database e verrà integrata al database memorizzato con tecnologia MongoDB (precedentemente introdotta). Questo permetterà di velocizzare e di facilitare le ricerche sui dati geo-localizzati.

3.4.7 ApacheHttpClient (RESTful Java Client)

Apache HttpClient è una libreria Java robusta ed efficiente, per la realizzazione di servizi che effettuano operazioni *http* (http://hc.apache.org/httpcomponents-client-ga/index.html).

Apache HttpClient è una libreria basilare per la realizzazione di applicazioni *client*, che debbano raggiungere risorse tramite il protocollo *http*. Tra queste viene data la possibilità di realizzare dei servizi *RESTful*.

Prima di decidere di utilizzare ***Apache HttpClient*** per realizzare questo servizio *RESTful Client*, sono state effettuate diverse prove con altre librerie analoghe (ad esempio *RESTLET, http://restlet.com/).* Alla fine si è optato per tale libreria, per la sua alta flessibilità, per la robustezza, e per la sua semplicità nel realizzare tali tipologie di chiamate.

3.4.8 JFreeChart (Chart and Diagram Presentation)

JFreeChart è una libreria Java per l'integrazione di diagrammi e contenuti grafici all'interno delle interfacce grafiche delle applicazioni (http://www.jfree.org/jfreechart/).

JFreeChart è una libreria Java gratuita e *open-source*, per la presentazione di grafici e diagrammi statistici, la quale possiede una documentazione molto ricca, e numerosi strumenti e componenti pre-realizzati che possono essere riutilizzati riducendo lo sforzo implementativo, e garantendo comunque un'alta possibilità di parametrizzazione. Tra i vantaggi di questa vi sono inoltre, la possibilità di esportare i dati ottenuti in differenti formati di immagine (dalle comuni JPG e PNG, ai formati vettoriali quali EPS e SVG).

Capitolo 4

Applicazione TSDEAI: Implementazione

In questo capitolo verrà descritta l'applicazione nella sua interezza, verranno spiegate l'interfaccia utente e le funzionalità offerte all'utilizzatore, e verranno caratterizzate le classi principali, accompagnate dalle porzioni di codice più rilevanti al fine di facilitare la spiegazione di ciò che si è andato a realizzare.

4.1 Introduzione

Per quanto riguarda la spiegazione e descrizione dettagliata delle funzionalità implementate, si è scelto di adottare un metodo ed un ordine di presentazione non propriamente tradizionale. In particolare, come si è già anticipato nel capitolo precedente [Capitolo 3], non si andranno a presentare in ordine cronologico le differenti parti che compongono solitamente un progetto di ingegneria del software. Piuttosto, viste le numerose funzionalità da spiegare, e considerata l'entità di questa tesi, la quale vuole solamente descrivere e accompagnare il progetto dell'applicazione, ma non vuole sostituire quello che sarebbe una giusta e completa documentazione di progetto; nei successivi

paragrafi verranno descritte dettagliatamente le funzionalità principali, accompagnate da porzioni di codice chiarificatrici e immagini dell'applicazione nella sua esecuzione. La scelta di accorpare porzioni di codice, porzioni del diagramma delle classi, e infine *screenshot* dell'*IDE* in esecuzione, vuole cercare di spiegare e descrivere al meglio tutto il lavoro che è stato compiuto al fine di realizzare le funzionalità prefissate.

4.1.1 Perché TSDEAI?

L'applicazione, che è stato correttamente, e doverosamente, registrata nella sezione *Developer* di *Twitter* (si veda il Paragrafo 2.3.1 - Open Authentication (OAuth)), è stata denominata con l'acronimo ***TSDEAI***.

Questo acronimo ricorda le principali funzionalità richieste dagli obiettivi di progetto:

> ***T****witter* ***S****ocial* ***D****ata –* ***E****xtraction* ***A****nalysis and* ***I****ntegration.*

La registrazione di tale applicazione sulla piattaforma *Developer* di *Twitter*, ha permesso di ricevere i *token* di accesso ai servizi, rispettivamente *Consumer Key (API Key) e Consumer Secret (API Secret)* per l'autenticazione *OAuth-Application-Only*.

Per permettere di utilizzare anche le funzionalità accessibili solamente con una *OAuth-User-Authentication*, è stato abbinato il mio account personale a quello dell'applicazione, ricevendo due ulteriori *access token*.

In questo modo si è implementata un'applicazione che permette di effettuare richieste sia con il *contesto-application* sia con il *contesto-user*, consentendo la possibilità di passare da un metodo di *login* all'altro (in *runtime*).

Per definire un ordine cronologico di spiegazione delle funzionalità, si sfrutterà direttamente la struttura dell'applicazione, partendo dalla *main class* di avvio del programma, e andando ad analizzare i differenti **moduli software** in successione.

4.1.2 Perché API REST?

Per l'esecuzione e la risoluzione delle richieste di estrazione, si è scelto di utilizzare le ***API REST*** *di Twitter (v1.1)*, in quanto, visto l'obiettivo della tesi e il contesto di *"emergency"* più volte citato, si è pensato che un'analisi a posteriori, degli eventi avvenuti, sarebbe stata più favorevole allo scopo, rispetto ad un'analisi fatta in *live* (concomitante all'evento) utilizzando la versione delle *API* in *streaming*.

Rispetto all'utilizzo delle *Streaming API*, infatti, con quelle *REST* è possibile scaricare *tweet*, dal momento corrente in cui si effettua la chiamata, fino ad una data antecedente di circa 7-9 giorni.

Questa finestra temporale delle *API REST*, permette di studiare il contesto e le *query* da effettuare, senza doversi preoccupare di catalogare tutti gli *status* che vengono condivisi. Una volta scelto l'argomento, il contesto, ed eventualmente una o più pagine di utenti dai quali andare ad effettuare le estrazioni; è possibile effettuare *query* ripetute (le quali restituiscono fino ad un massimo di 100 *tweet*, raccolti in "pagine") per andare a scaricare tutti gli *status* pubblicati nel lasso di tempo.

4.2 Struttura dell'applicazione

Come per la maggior parte delle applicazioni Java che utilizzano componenti *Swing* (per la loro interfaccia grafica), questa possiede una classe principale denominata **MainTSDEAI**, nella quale è presente il metodo *main*, che viene invocato ed eseguito da un *thread iniziale*.

La realizzazione di un'applicazione con interfaccia *Swing,* coinvolge normalmente tre differenti macro-tipologie di ***thread***:

1. *Initial Thread (o thread iniziali)* che servono per eseguire il codice nella fase di avvio dell'applicazione.
2. *L'EDT (Event Dispatch Thread)* che esegue il codice di gestione degli eventi, gestendo l'interazione con l'utente utilizzatore. Particolare attenzione nella progettazione di un'applicazione, va posta proprio sulla

gestione dell'*EDT*, il *thread* fondamentale che si occupa di effettuare il "*drawing*" dei componenti grafici (e quindi di mostrare e "disegnare" l'interfaccia utente). Questo *thread* di sistema si occupa della gestione degli eventi generati dall'interazione tra l'utente e i componenti della *GUI*. Questo *thread* viene avviato dalla *Java Virtual Machine (VM)* e gestisce una coda di eventi, restando in attesa che nuovi eventi da gestire vengano inseriti in coda. Esso resta in vita fin quando esistono componenti visibili, e ogni qualvolta preleva un evento dalla coda, lo notifica ai *Listener* corrispondenti, in modo da eseguire il codice di gestione degli stessi.

3. *I "Worker Thread" (o Background Thread)* i quali consistono in *thread* dedicati a scopi specifici, per l'esecuzione di operazioni *time-consuming* "sottostanti" all'interfaccia grafica. Questi vengono istanziati secondo differenti *pattern* di allocazione (vedremo per esempio l'utilizzo di *Executor Service*, per l'allocazione di una *thread pool* con un numero prefissato di *thread*).

Di seguito viene riportata una porzione del diagramma delle classi, rappresentante la classe *MainTSDEAI* appena citata:

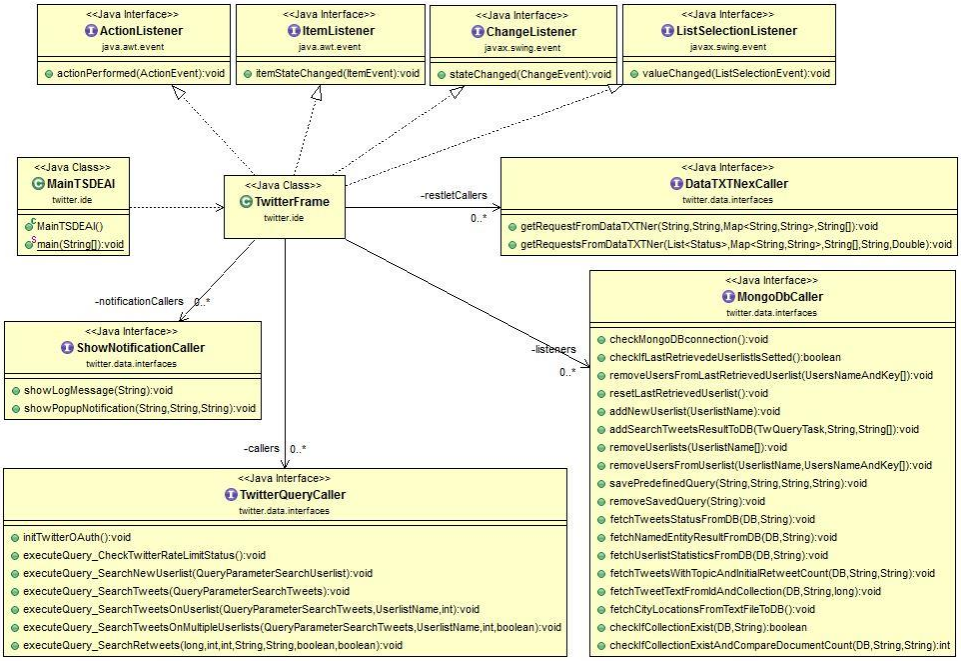


Figura 24: Diagramma UML delle Classi (parziale) relativo alle classi coinvolte nella fase di avvio e gestione dell'interfaccia utente.

Nella precedente [**Figura 24**] si possono evincere le classi di interazione con *MainTSDEAI*, tra queste per esempio la classe *TwitterFrame*, che implementa la finestra principale dell'applicazione, e le interfacce dei *Listener* abilitati alla gestione ed evacuazione degli eventi di interazione con gli utenti.

Di seguito viene mostrato il breve codice della classe *main*, che istanzia e definisce le differenti classi sopracitate, aggiungendone alcune che verranno spiegate accuratamente nei paragrafi successivi.

MainTSDEAI.java

```
package twitter.ide;

import mongodb.util.MongoDBConfiguration;
import mongodb.util.MongoDBContext;
import twitter.data.classes.TwitterContext;
import twitter.util.EnvironmentsCall;
import twitter.util.Util.OAuth_AuthenticationType;

public class MainTSDEAIandLOD{

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        // Per scegliere lo stile del frame
        EnvironmentsCall.AdjustFrameLookAndFeel();

        // Parte con user-auth (personal account)
        OAuth_AuthenticationType authenticationStartMode =
                        OAuth_AuthenticationType.USER_OAUTH;

        TwitterContext twContext =
                        new TwitterContext(authenticationStartMode);
        MongoDBContext mongoDBcontext =
                        MongoDBConfiguration.initializeConnection
                        ("localhost", 27017,"twitter");

        TwitterFrame tf =
                        new TwitterFrame(twContext, mongoDBcontext);

        // Per posizionare correttamente il frame
         EnvironmentsCall.AdjustFrameLocation(tf);


        Controller controller =
                        new Controller(tf, twContext, mongoDBcontext);

        CursorToolkit.startWaitCursor(tf);

        tf.addInputListener(controller);
        tf.addQueryCaller(controller);
        tf.addRestletCaller(controller);
        tf.addNotificationCaller(controller);

        tf.setVisible(true);
        tf.firstAuthentication();    // AUTO-LOGIN TWITTER E MONGODB
    }
}
```

Codice 1: Codice sorgente completo della classe *main* (MainTSDEAI.java).

Osservando il [**Codice 1**]:

- Si possono denotare due *classi statiche* (con funzionalità marginali), le quali servono ad adempiere a scopi generici per l'impostazione e la presentazione dell'interfaccia. Ci si riferisce in particolare a **EnvironmentCall** e **CursorToolkit**.
- Oltre a queste si denotano due classi che hanno funzionalità di "contenitore" per la trasmissione e referenziazione dei dati e delle istanze di classe, che servono ai differenti moduli applicativi per comunicare e modificare lo *stato* corrente dell'applicazione. Queste due classi sono rispettivamente **TwitterContext** e **MongoDBContext**.
- In aggiunta a queste si denota una classe principale, **TwitterFrame**, la quale viene istanziata dal metodo *main*. Questa classe si occuperà di tutta la gestione della *view* della finestra principale.
  Essa è stata progettata, suddividendo i **3 moduli di funzionalità** principali, in 3 ***tabpage*** differenti, in modo da rendere compatta e uniforme la vista della stessa applicazione.

In seguito verrà mostrato come siano state impiegate anche altre finestre, per la presentazione di contenuti temporanei (ad esempio le notifiche *popup*) o informazioni aggiuntive (ad esempio la *history delle query* o le richieste pendenti), che vedremo nei paragrafi conclusivi di questo capitolo. Ciò nonostante, tutta la logica applicativa principale e la maggior parte delle interazioni con l'utente, avviene con le componenti grafiche definite nella classe principale *TwitterFrame*.

NOTA:
Per motivi di spazio, e per questioni di leggibilità del codice, non è stata riportato il codice sorgente della classe, in quanto sarebbe di difficile navigazione e comprensione in un documento testuale come questo.

La classe *TwitterFrame* di fatto possiede 3 tipologie di metodi:

1. Un primo metodo principale, con i relativi sotto-metodi annessi, per l'inizializzazione e la parametrizzazione dei componenti grafici della *GUI.*
2. Possiede inoltre numerosi metodi per permettere le azioni di interazione con l'utente, che verranno poi gestite e risolte dal *Controller* (si veda architettura *MVC*, nel Paragrafo 3.3 - Architettura Logica), il quale verrà descritto di seguito, andando ad analizzare le interfacce da esso implementate.
3. In aggiunta a questi metodi, ne possiede altrettanti relativi alla notifica (e seguente presentazione) dei risultati ottenuti dall'esecuzione delle operazioni lanciate dall'utente attraverso la *GUI,* e risolte secondo la logica applicativa espressa nelle classi di *Model (in rispetto del MVC).*

Nei paragrafi successivi:

- Verrà dapprima descritto il ***Controller***, con le relative *signature* dei metodi implementati per rispettare le interfacce dichiarate.
- Successivamente verranno analizzate le funzionalità di: **Estrazione** [**Errore. L'origine riferimento non è stata trovata.**], **lassificazione e Integrazione** [0] e infine **Analisi dei contenuti** [4.5] implementate nei tre moduli *software* principali.

4.2.1 Controller

Il *Controller* rispettando l'architettura *MVC*, più volte citata, deve implementare tutti quei metodi che gli permettono di gestire le richieste dell'utente. Realizzando le interfacce demandate dalla classe *TwitterFrame*, implementa tutti i metodi che permettono di gestire gli eventi di interazione tra utente e *GUI*, andando a ridirigere le richieste sul *Model*, e quindi sulla logica del *business*.

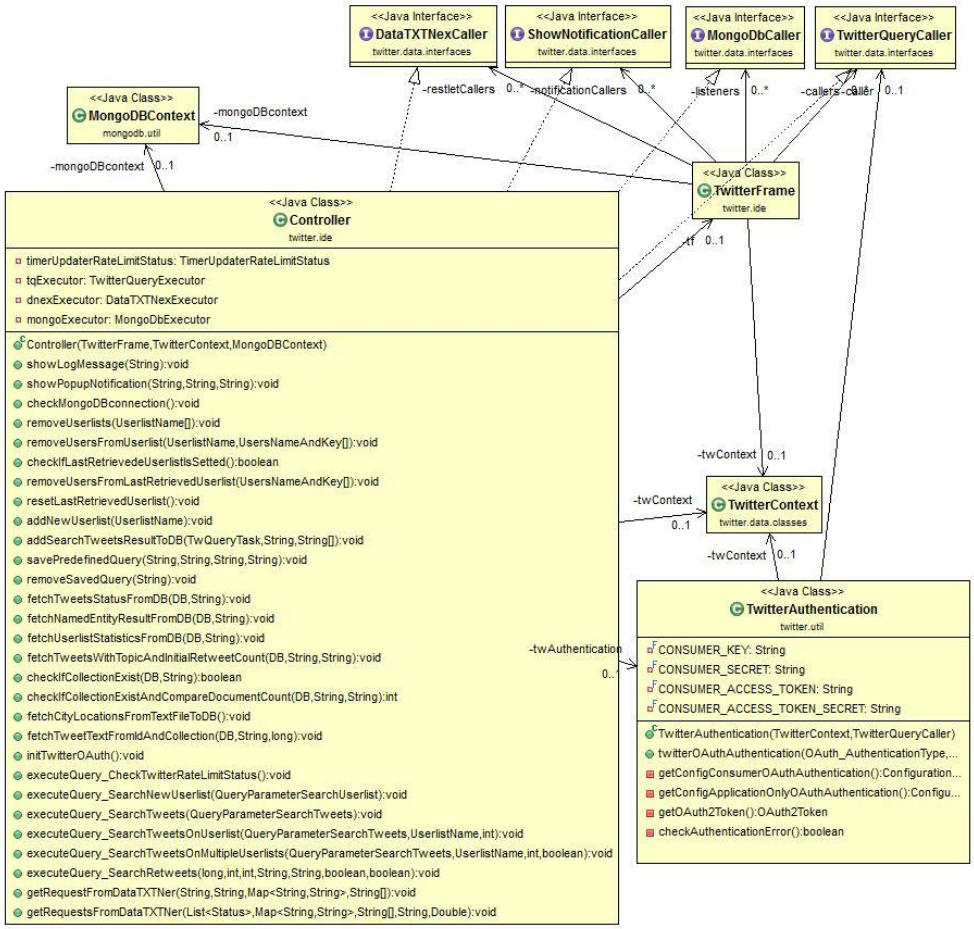


Figura 25: Diagramma UML (parziale) relativo alla classe Controller, con riferimento alle interfacce realizzate da essa, alle classi di *context*, e alla classe di autenticazione di *Twitter*.

In particolare, nei paragrafi successivi verrà descritto il *Model*, suddiviso in tre macro-funzionalità: *Estrazione, Classificazione e Integrazione, e Analisi.*

4.3 ESTRAZIONE

Il modulo *software* dedicato all'***Estrazione dei tweet***, è rappresentato graficamente dalla prima finestra in avvio, dopo il lancio dell'applicazione. L'interfaccia è infatti suddivisa, innanzitutto in tre differenti *tabpage*, uno per ogni modulo *software*. Successivamente, all'interno di ogni pagina, sono presenti diverse sezioni contenenti i componenti per la parametrizzazione e l'esecuzione delle azioni richieste.

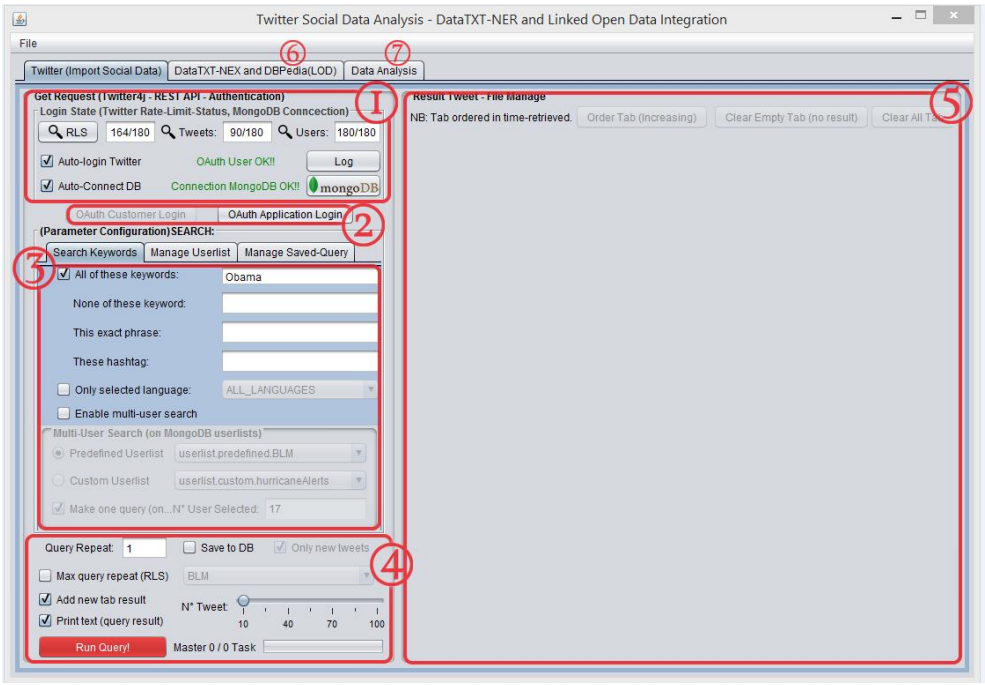


Figura 26: TwitterFrame (finestra principale all'avvio dell'applicazione). All'avvio vengono avviate altre finestre minori, laterali, che sono state volutamente omesse, perché verranno trattate e spiegate nei paragrafi successivi.

NOTA:
vista la natura dell'applicazione, e l'alta richiesta di parametrizzazione, per quanto si sia cercato di ottimizzare la disposizione dei componenti grafici, ordinandoli in sezioni secondo quelli che sono i loro scopi; l'IDE risulta essere comunque molto ricco di strumenti, per questo motivo ogni screenshot (aggiunto per facilitare la comprensione dell'applicazione) verrà etichettato sfruttando delle frecce e dei marcatori numerici, per poter andare a descrivere in dettaglio le funzionalità implementate.

Osservando la [**Figura 26**] che rappresenta uno *screenshot* della prima finestra di avvio dell'applicazione, si può notare che sono state delimitate e numerate 7 aree differenti. Di seguito verranno introdotte singolarmente:

1. La sezione **(1)** presenta le informazioni legate allo stato della sessione di *login*, per l'autenticazione di *Twitter* e la connessione con il *service* in locale di *MongoDB*. All'avvio dell'applicazione, viene lanciata una procedura di *login* automatico, con i dati precedentemente settati, e viene confermato la corretta autenticazione all'utente. Inoltre vengono

presentati tre campi di testo, contenenti i principali *rate-limit-status* utilizzati in questa schermata. (Nel Paragrafo 4.3.7, verrà trattata in dettaglio la questione dei limiti suddetti, imposti dalle *API* di *Twitter*).

2. La sezione **(2)** presenta due pulsanti che permettono di passare dall'autenticazione *OAuth-User-Authentication* alla differente *OAuth-Application-Only-Authentication*. Questo cambiamento chiaramente, oltre a dover necessariamente mostrare il buon esito dell'autenticazione, deve modificare il contesto dell'applicazione, andando in particolare ad aggiornare immediatamente i contatori di *rate-limit-status* precedentemente citati ed esposti nella sezione (1).
3. Questa sezione **(3)** presenta un'ulteriore suddivisione in *tabpage*:
    a. La prima scheda denominata *"Search Keywords"* è dedicata all'impostazione dei parametri per l'esecuzione delle *query* di *search-tweets*.
    b. La seconda *"Manage Userlist"* verrà illustrata in seguito [Paragrafo 4.3.9] e permetterà la ricerca di utenti e liste di utenti, e la successiva memorizzazione di esse all'interno del *database*, in modo da poterle riutilizzare nelle *query* programmate.
    c. La terza *"Manage Saved Query"* verrà anch'essa discussa in seguito [Paragrafo 4.3.10], e in breve, permetterà all'utente di memorizzare all'interno del *database*, delle *query* parametrizzata, in modo da poter ripetere in tempi successivi la stessa interrogazione.
4. La sezione **(4)** comprende una serie di parametri per l'esecuzione di *query* ripetute sulla stessa pagina, per l'abilitazione alla stampa testuale dei risultati ottenuti dalle stesse *query* sotto forma di *JSON*, e per l'abilitazione dei *popup* di notifica o della pagina *tab-result* (le quali verranno spiegate in dettaglio in seguito [Paragrafo 4.3.11]).
5. La sezione **(5)** servirà a presentare i risultati ottenuti dalle *query* e in maniera dinamica andrà ad aggiungere *tabpage* univoche per la loro visualizzazione (analogamente al punto (4), verrà discussa successivamente [Paragrafo 4.3.11]).

6. Il *tabpage* indicato dal marcatore **(6)** permetterà di aprire la nuova scheda relativa al secondo modulo di *"Classificazione e Integrazione"*. Questo verrà dettagliatamente descritto in [Paragrafo 0].
7. Il *tabpage* **(7)** indica la scheda relativa all'ultimo modulo *software*, dedicato alla *"Analisi dei contenuti"*, e verrà trattato in [Paragrafo 4.5].

4.3.1 Modulo Software e Packages Implementati

Per introdurre tutte le funzionalità di questo *modulo software*, si è deciso di presentare la lista dei *package* implementati, al fine di introdurre anche quelle classi che per ovvi motivi di spazio non potranno essere riportate in questa tesi.

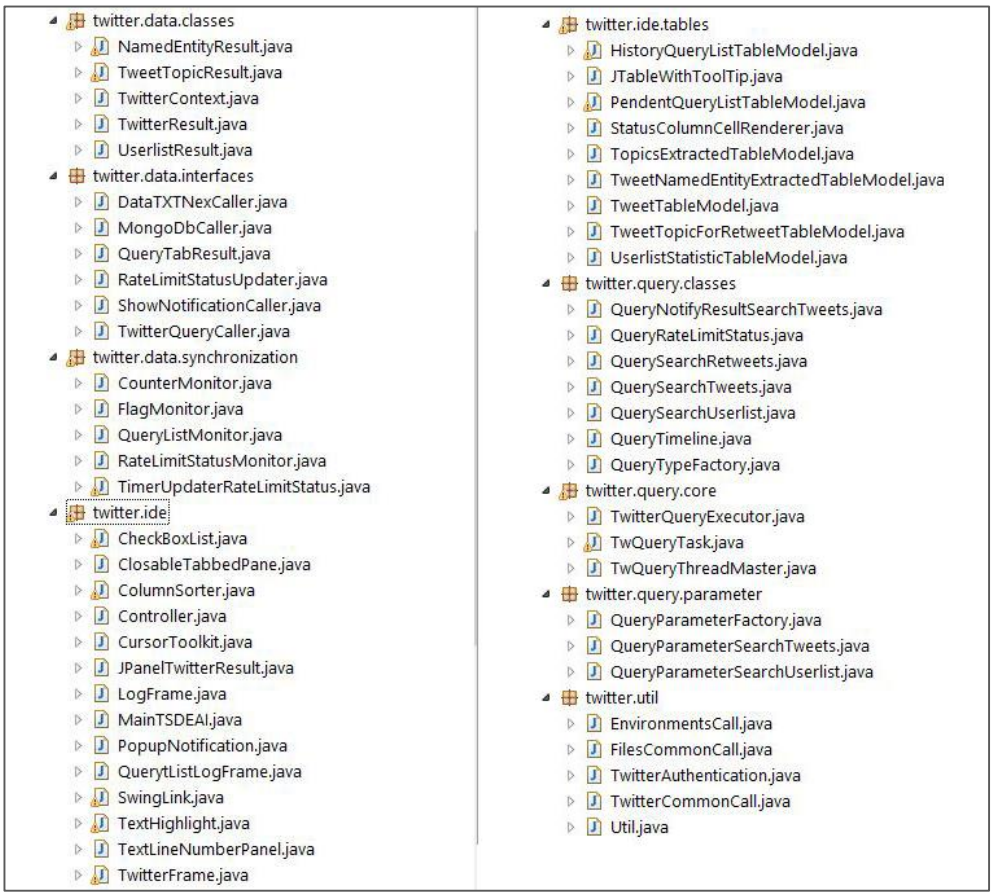


Figura 27: Elenco completo dei *package* relativi al *namespace "twitter.*"*.

Dalla [**Figura 27**] si nota che tutti i pacchetti del modulo sono caratterizzati da nomi identificativi, appartenenti tutti allo stesso *namespace* (*twitter.**).

Questi pacchetti sono stati divisi secondo quelli che sono i loro scopi e le caratteristiche che li contraddistinguono:

- ***twitter.data.classes:*** *comprende tutte le classi per la trasmissione e l'aggiornamento della stato della view, da parte del model.*
- ***twitter.data.interfaces:*** *presenta tutte le interfacce realizzate per definire i metodi di gestione degli eventi della view (questa classe comprende anche le interfacce già citate, relative ad altri servizi differenti da Twitter, come ad esempio DataTXT-Nex o MongoDB).*
- ***twitter.data.synchronization:*** *contiene tutte le classi appositamente create per estendere il concetto di Semafori e Monitor, utilizzati per gestire le operazioni di scritture e letture, su risorse condivise tra thread e istanze differenti.*
- ***twitter.ide:*** *comprende tutte le classi realizzate per presentazione dell'interfaccia grafica.*
- ***twitter.ide.tables:*** *comprende le classi definite per estendere le JTable implementate da Java, e definire nuovi modelli per ordinare ed effettuare il rendering dei contenuti delle tabelle in maniera differente da quelle tradizionali. Le tabelle implementate infatti permettono di ordinare i risultati secondo colonne specifiche e permettono la visualizzazione di icone all'interno delle celle.*
- ***twitter.query.classes:*** *racchiude al suo interno tutta la gerarchia di classi per la rappresentazione delle query di Twitter, e dei relativi parametri (si veda anche twitter.query.parameter).*
- ***twitter.query.core:*** *racchiude la parte centrale della computazione delle richieste a Twitter, e comprende una struttura ideata per la suddivisione delle richieste secondo il pattern Master-Worker.*
- ***twitter.query.parameter:*** *in abbinamento al pacchetto alla gerarchia di classi in twitter.query.classes, racchiude i parametri necessari per le due tipologie che ne necessitano, rispettivamente Search-Tweets e Search-User. Queste ultime verranno descritte dettagliatamente nei successivi paragrafi.*
- ***twitter.util:*** *comprende le classi contenenti metodi e proprietà statiche, per il settaggio dell'ambiente, delle interazioni e della comunicazione tra i differenti moduli.*

4.3.2 Advanced Search

Twitter permette, attraverso le API, di effettuare delle richieste denominate *Advanced Search*. Esse sono analoghe alle ricerche che si possono effettuare direttamente sul sito web di riferimento, ed includono in particolare, un serie numerosa di parametri che permettono di restringere e contestualizzare il campo effettivo di ricerca dei *tweet*.

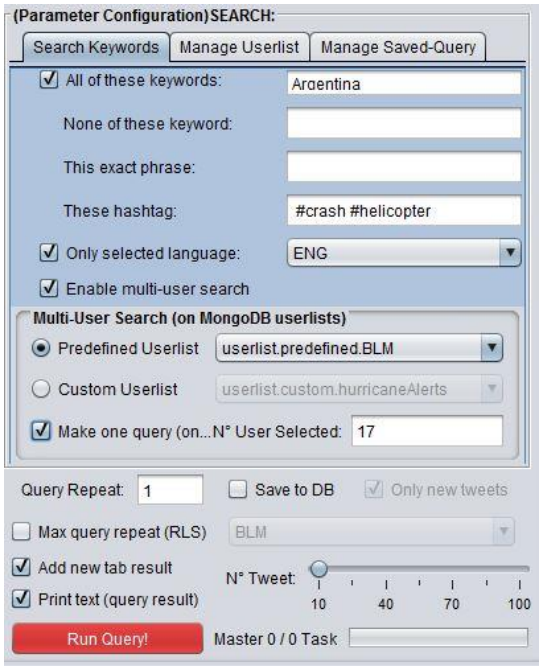


Figura 28: Sezione di parametri per l'esecuzione di richieste di *Search-tweets*.

Tra i parametri studiati, e inseriti all'interno dell'*GUI*, si trovano:

- La possibilità di cercare in maniera "tradizionale", un insieme di parole.
- La possibilità di cercare una frase esatta (condizione di *AND*).
- Cercare almeno una delle parole indicate (condizione di *OR*).
- Inserire parole che non devono comparire (condizione di *NOT*).
- Impostare la lingua di scrittura del tweet (auto-identificata da *Twitter*).
- La possibilità di cercare degli *hashtag* (con l'apposito simbolo #).

Per quanto riguarda le persone coinvolte nel *tweet*:

- La possibilità di definire gli utenti dai quali i messaggi provengono, oppure gli utenti destinatari di tali messaggi.

- Inoltre la possibilità di menzionare degli utenti (con il simbolo @).

Esistono poi dei parametri trasversali, che non sono stati inseriti direttamente nella *GUI*, che permettono di definire:

- Una località, per ricevere solamente *tweet* geo-localizzati, provenienti da zone vicine al punto specificato.
- Un intervallo di date precise, tra le quali cercare i *tweet*.
- Un sentimento (auto-identificato da *Twitter*), sia esso positivo o negativo, in riferimento al contenuto espresso all'interno del *tweet*.
- La possibilità di cercare solamente domande (quindi messaggi identificati da *Twitter*, visto il loro tono interrogativo.
- La richieste di includere non solamente i *tweet*, ma anche i *retweet*, all'interno degli *status* restituiti. Questa funzionalità ad esempio, non è stata inserita come componente grafico, ma è stata utilizzata per effettuare l'analisi delle condivisioni, e verrà più volte citata all'interno del successivo [Paragrafo 4.5.3].

4.3.3 Interrogazione delle Timeline

Le *API* di *Twitter* possiedono svariati metodi per la ricerca ed il reperimento di informazioni legate ai *tweet* o ai loro utenti. Tra questi metodi esistono:

- *"GET statuses/user_timeline"*.
- *"GET statuses/home_timeline"*.
- *"GET search/tweets"*.

Tutti questi metodi restituiscono rispettivamente una ***timeline*** di *tweet*.

Dal momento che, queste *timeline* possono crescere e diventare davvero eccessive (per il numero di *tweet* su di esse pubblicato), esistono limiti per ogni tipologia di richiesta, sul numero restituito da ogni singola chiamata.

Per poter reperire tutti i *tweet*, occorre allora adottare tecniche che iterano il procedimento, eseguendo *query* ripetute con la stessa parametrizzazione.

La complessità è dovuta dalla stessa natura di *Twitter*, infatti è possibile che, anche durante l'esecuzione delle interrogazioni, siano aggiunti in dei *tweet* in *realtime* (andando così a modificare il volume dei dati sulle *timeline*).

Per questo motivo, le normali tecniche di *paginazione* spesso non riescono ad ottenere i risultati sperati. Osservando la documentazione, si trova una sezione apposita per la gestione delle operazioni sulle *timeline*.

Di seguito verrà riportato l'esempio reperito dalla documentazione, andando a spiegare quale è stato il metodo implementato all'interno di questa applicazione.

Gestire Paging e Cursoring sulle Timeline

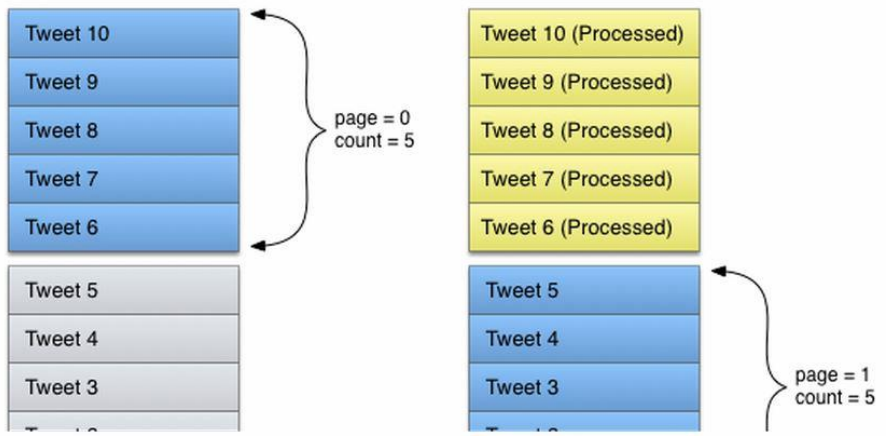


Figura 29: Gestione delle *timeline* in un "mondo ideale", senza aggiornamenti.

In un mondo ideale, la paginazione sarebbe molto facile da implementare. Considerando l'esempio in [**Figura 29**], prendendo come caso d'esempio una *timeline* con 10 *tweet* ordinati secondo il loro *id univoco* (memorizzato come *long* a 64bit) in ordine cronologico (e dunque rispetto agli *id*, in maniera descrescente).

Se si ipotizzano due richieste, impostando come dimensione di pagina solamente 5 (anche se nel caso reale, si può richiedere fino ad un massimo di 100 *tweet* per le richieste di tipo *GET search/tweets*).

Effettuando le due chiamate in sequenza, si otterrebbe l'intera *timeline* composta di 10 *tweets*.

Purtroppo questa è solamente una situazione ideale, che non tiene conto del fatto che, le timeline siano in continuo aggiornamento, e che vengano aggiunti frequentemente tweet in cima ad esse.

Nella figura seguente, viene preso il primo caso d'esempio, e si ipotizza che tra la prima richiesta e la seconda, vengano inseriti due nuovi *tweet* in cima alla pagina.

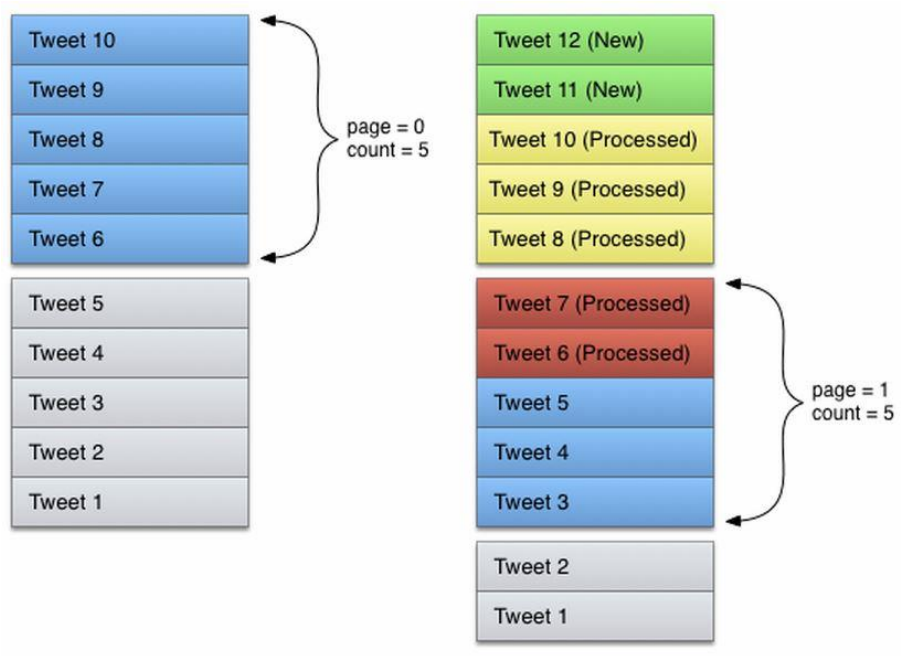


Figura 30: Gestione delle *timeline*, caso d'esempio iniziale in un contesto reale, con l'aggiunta di nuovi *tweet* tra una richiesta e la seguente.

Come si può notare in figura, in questo caso, la seconda richiesta, restituisce due *tweet* già restituiti dalla prima, andando così a riportare contenuto superfluo e non restituendo gli ultimi due *tweet* della pagina.

Si noti come, nello stesso esempio, se nel lasso di tempo tra la prima e la seconda richiesta, dovessero essere pubblicati più di 5 tweet, almeno una delle richieste effettuate sarebbe completamente ridondante e superflua.

Per risolvere questo problema, le *API* permettono l'utilizzo di due parametri fondamentali:

- *Max_id (per richiedere tweet con un id inferiore rispetto a quello impostato in tale parametro).*
- *Since_id (per richiedere tweet con un id maggiore rispetto a quello impostato).*

Un'applicazione che tenga in considerazione entrambi i parametri e si avvalga di una buona strategia per l'estrazione di tali *tweet*, può ottimizzare le performance e andando a ridurre al minimo le richieste superflue.

In particolare, dopo aver effettuato almeno una richiesta, e quindi aver ottenuto già dei *tweet*, dalla stessa o da richieste effettuate in passato, è possibile impostare i due parametri come segue:

- Sfruttando la tecnica di ***cursoring*** (che ad ogni *response* ricevuta in merito ad una *request*, assegna un *"cursore"* che punta alla successiva pagina di risultati) e inizializzando il parametro ***max_id*** (ad ogni ricezione di una *response*) con ***l'ID inferiore***, tra quelli ricevuti; è possibile andare a richiedere solamente i *tweet* mancanti, in coda rispetto a quelli già ottenuti.

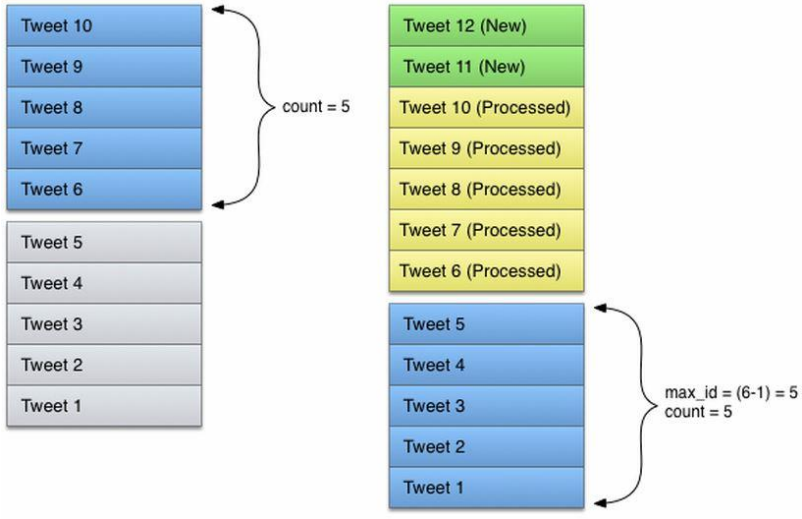


Figura 31: Gestione delle *timeline*, esempio di successo sfruttando ***max_id***.

NOTA:
*Per migliorare ulteriormente questa tecnica, in [**Figura 31**] e nell'applicazione realizzata, è stato settato il valore di max_id all'ID inferiore (sottraendo ancora 1). In questo modo, si vanno a restituire esattamente tutti e **soli** i tweet non ancora ricevuti. Questa operazione è possibile solo in ambienti che permettono di adoperare id con precisione a 64bit. In caso non sia possibile effettuare questa operazione, la tecnica ottimizzerà comunque le query, andando nel peggiore dei casi, a restituire le pagine di tweet (mettendo come primo status, l'ultimo della richiesta precedente).*

- Sfruttando la stessa tecnica di *cursoring*, ed andando ad impostare anche il parametro *since_id*, è possibile andare a reperire anche i nuovi *tweets*, che sono stati aggiunti dopo che l'applicazione abbia effettuato almeno una richiesta.

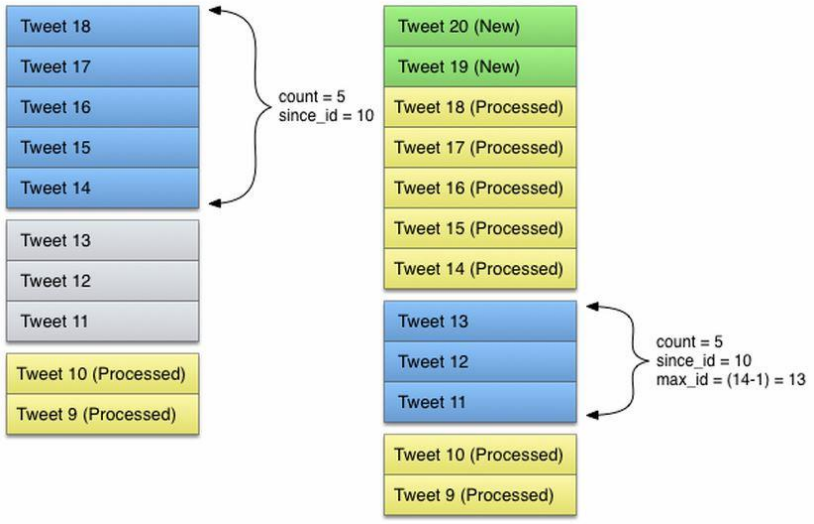


Figura 32: Gestione delle *timeline*, due esempi di successo utilizzando ***since_id***.

Questo parametro risulta essere molto utile in questo contesto, nel quale le interrogazioni relative agli eventi di *emergency* individuati, possono avvenire a distanza di ore o di giorni dall'evento stesso.

Uno dei sotto-obbiettivi dell'applicazione è proprio quello di cercare di raccogliere tutti i *tweet*, anche se essi possono essere stati pubblicati con un certo ritardo rispetto all'evento stesso. Inizializzando questo parametro di ***since_id*** con l'***ID maggiore***, tra quelli degli *status* già ricevuti in precedenza, è possibile andare a prelevare solamente i "nuovi" *tweets*, senza restituire quelli ridondanti.

4.3.4 Gerarchia delle Classi Query

Per le differenti tipologie di query, è stata creata una gerarchia di classi, le quali possiedono proprietà caratteristiche che le contraddistinguono dalle altre. Tutte le classi figlie specializzano una classe madre denominata *QueryTypeFactory*, che viene utilizzata per generalizzare il concetto di *Query*, e forzare la presenza di alcune proprietà fonamentali.

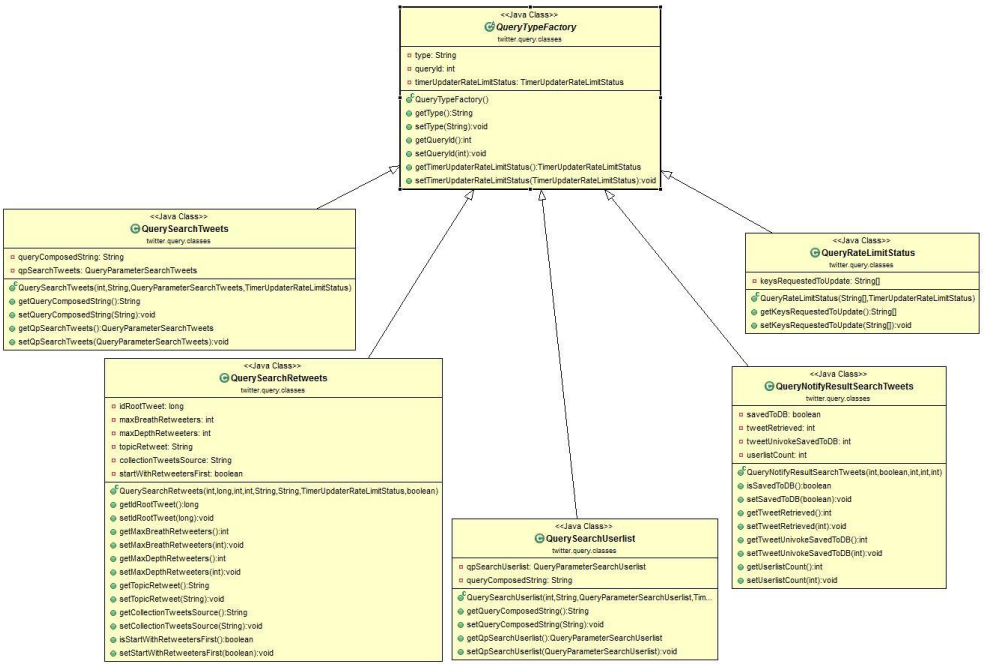


Figura 33: (Porzione) Diagramma UML delle classi, relativo alla *Gerarchia Query*.

Di seguito viene mostrato il codice di una delle classi figlie, a fini di esempio.

QuerySearchTweets.java

```
package twitter.query.classes;

import twitter.data.synchronization.TimerUpdaterRateLimitStatus;
import twitter.query.parameter.QueryParameterSearchTweets;
import twitter.util.Util;

public class QuerySearchTweets extends QueryTypeFactory{

    private String queryComposedString;
    private QueryParameterSearchTweets qpSearchTweets;

    public QuerySearchTweets(int queryID,
                      String queryComposedString,
                      QueryParameterSearchTweets qpSearchTweets,
                      TimerUpdaterRateLimitStatus
                      timerUpdaterRateLimitStatus){

        setType(Util.QUERY_SEARCH_KEY);
        setQueryId(queryID);
        setTimerUpdaterRateLimitStatus(timerUpdaterRateLimitStatus);
        this.setQueryComposedString(queryComposedString);
        this.setQpSearchTweets(qpSearchTweets);
    }

    public String getQueryComposedString() {
              return queryComposedString;
    }
    public void setQueryComposedString(String queryComposedString) {
        this.queryComposedString = queryComposedString;
    }

    public QueryParameterSearchTweets getQpSearchTweets() {
        return qpSearchTweets;
    }

    public void setQpSearchTweets(
                      QueryParameterSearchTweets qpSearchTweets) {
        this.qpSearchTweets = qpSearchTweets;
    }
}
```

Codice 2: Codice sorgente (completo) relativo alla classe *QuerySearchTweets*.

4.3.5 Gerarchia delle Classi QueryParameter

Analogamente alla gerarchia sopracitata, relativa alle *Query*, si è dovuto creare una gerarchia di classi per il passaggio di parametri dalla *View* al *Model*.

In particolare per gestire le due tipologie di richieste, che necessitano di numerosi parametri, sia per l'impostazione delle *query* vere e proprie, sia per la restituzione da parte del *Model*, dei risultati; è stata ideata una classe madre, denominata *QueryParameterFactory* dalla quale ereditano sia la classe *QueryParameterSearchTweets*, che *QueryParameterSearchUserlist* (la prima

per rappresentare i parametri delle richieste di *tweet*, e la seconda quelle degli utenti).

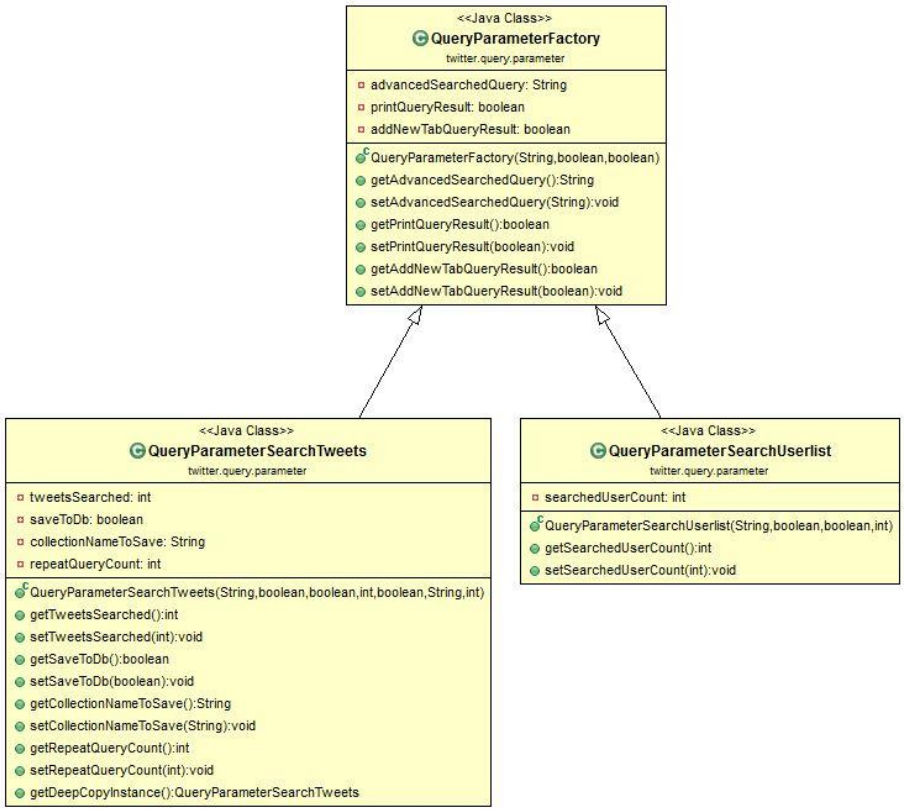


Figura 34: (Porzione) Diagramma UML delle classi, *Gerarchia Query Parameter*.

4.3.6 Task Allocation (CORE APPLICATION LOGIC)

La parte centrale della logica applicativa, per la gestione e attuazione delle richieste effettuate tramite le *API Twitter v1.1*, risiede nel *package* *twitter.query.core* (si veda Paragrafo 4.3.1).

Per istanziare richieste al servizio di *Twitter* (come del resto anche per i servizi di *DataTXT* e *MongoDB*) sono state istanziate tre classi ***Executor*** denominate:

- ***TwitterQueryExecutor***, per la gestione delle *query* a *Twitter*.
- ***DataTXTNexExecutor***, per le richieste a *DataTXT-NEX*.
- ***MongoDbExecutor***, per le operazioni sul *database MongoDB*.

I primi due sono servizi online, e richiedono entrambi l'interazione con un servizio di *API REST*, e necessitano che vengano rispettate regole di autenticazione e di limitazione di utilizzo. Il terzo invece opera in un contesto locale, interagendo con il *database* con tecnologia non relazionale, *MongoDB*. (Nulla vieta di estendere questo ultimo terzo servizio, e di spostare il *database* in rete, per renderlo in un prossimo futuro, accessibile da diverse locazioni).

Il primo *Executor*, rispetto agli altri due, vista la sua natura e la possibilità di suddividere l'insieme delle richieste in *Task* differenti, è stato realizzato secondo il *pattern multi-threading Master-Worker*.

Le richieste dirette al servizio di *Twitter*, vengono dunque precedentemente suddivise ed allocate grazie ad una successiva classe *TwitterQueryThreadMaster*.

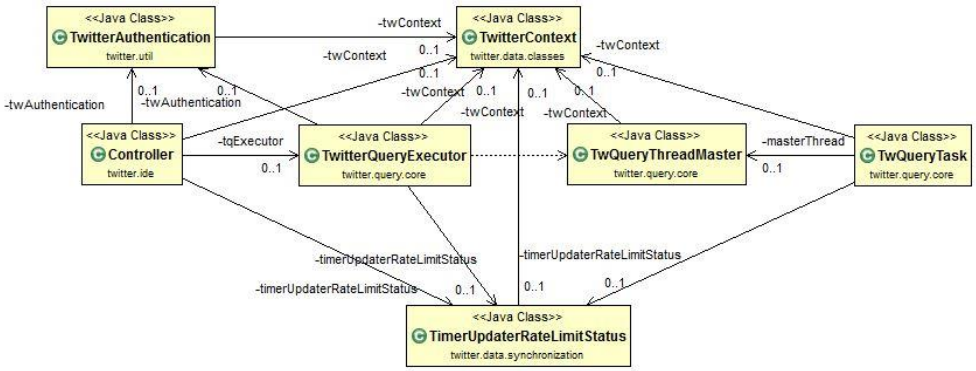


Figura 35: (Porzione) Diagramma UML delle classi, riferito a *TwitterQueryExecutor*, *TwQueryThreadMaster* e *TwQueryTask*, per esprimere le dipendenze tra le classi che effettuano le richieste su *Twitter*.

Si noterà come, attenendosi ad una buona programmazione ad oggetti, e cercando di rendere indipendenti i differenti moduli *software*, si sia cercato di spostare tutta la complessità e la conoscenza verso il *model*. In particolare, in questa catena di chiamate per la risoluzione di richieste al servizio di *API* di *Twitter*, a partire dalla *view* e dal *controller*, si sia cercato di non richiedere alcuna conoscenza relativa alla *business logic* di risoluzione di tali richieste. Scendendo lungo la catena di chiamate ai metodi, passando da *TwitterQueryExecutor* a *TwQueryThreadMaster*, ed infine arrivando a *TwQueryTask*, si sia via via andati a richiedere maggiore conoscenza.

Di seguito verranno presentate in dettaglio, le classi coinvolte in questo processo (introdotte in [**Figura 35**]).

TWITTER QUERY EXECUTOR

Questa è la classe che permette al *Controller* di demandare le richieste, pervenute da eventi gestiti ed avvenuti sulla *View*, al vero e proprio *Model* del sistema.

TwitterQueryExecutor possiede tutti i metodi per lanciare le seguenti differenti tipologie di richieste:

CHECK RLS	***RunCheckTwitterRateLimitStatus**, per effettuare richieste di aggiornamento del Rate-Limit-Status legato al contesto utente correntemente loggato (verrà trattato in seguito, [Paragrafo 4.3.7]).*
SEARCH USERS	***RunSearchNewUserlist**, per effettuare richieste di "ricerca di utenti" secondo precisi parametri (verrà tratto successivamente, [Paragrafo 4.3.9]).*
SEARCH TWEETS	***RunSearchTweets**, per lanciare richieste di "ricerca di tweets", secondo i parametri della Advanced Search, ma senza aver impostato nessun utente o lista di utenti specifici (verranno dettagliatamente spiegate, dopo aver introdotto il rate-limit-status e le liste di utenti, [Paragrafo 4.3.10]).* ***RunSearchTweetsOnUserlist**, per lanciare una richiesta specificando l'utente proprietario dello status o del retweet.* ***RunSearchTweetsOnMultipleUserlist**, analoga alla precedente, con l'aggiunta di una specifica lista di utenti, precedentemente selezionati.*
SEARCH RETWEETS GRAPH	***RunSearchRetweets**, per effettuare il calcolo del grafo dei retweet, secondo certi criteri e certe supposizioni. Questo algoritmo verrà trattato in dettaglio in seguito, [Paragrafo 4.5.3].*

Tabella 4: Classificazione delle tipologie di *search* implementate in *TwQueryTask*.

TWITTER QUERY THREAD-MASTER

Per ognuna delle richieste da effettuare, *TwitterQueryExecutor* genera un'istanza della classe *TwQueryThreadMaster*, in modo tale da separare completamente l'esecuzione di ciascuna di esse.
Questa classe *"Master"*, all'interno dell'architettura ***Master-Worker*** precedentemente citata, nel metodo costruttore istanzia una ***FixedThreadPool*** (con un numero prefissato di *thread*, limitato al numero massimo di processori disponibili sulla macchina, e una coda illimitata di ***task*** condivisi tra di essi) che cercherà di distribuire equamente il lavoro, su questi *thread* istanziati. Il costrutto che permette di utilizzare questo metodo è denominato ***ExecutorService*** ed è fornito direttamente dalla libreria *java.util.concurrent.executors*.

Di seguito viene riportato il codice sorgente relativo all'implementazione della classe sopracitata.

TwQueryThreadMaster.java

```
package twitter.query.core;

import … …

public class TwQueryThreadMaster extends Thread {

    private TwitterFrame tf;
    private TwitterContext twContext;
    private MongoDBContext mongoDbContext;
    // SEMAFORI per fare wait se si raggiunge il RATE-LIMIT
       STATUS (CONDIVISO TRA I NUOVI TASK)
    private HashMap<String, Semaphore> availableTaskTillRLS;
    private List<QueryTypeFactory> queryList;
    private ExecutorService executor;
    private CountDownLatch latch;// BARRIERA ATTESA FINE TASK
    private int nThread;
    private boolean saveToDbQueryResult;
    private String collectionNameToSave;
    private int[] tweetsRetrieved;
    private int[] tweetsSaved;
    private boolean searchOnlyNewTweets;
    private long lowestIDfetchedFromSavedTweet;
    private long greatestIDfetchedFromSavedTweets;

    public TwQueryThreadMaster(List<QueryTypeFactory> queryList,
               TwitterFrame tf, TwitterContext twContext,
               MongoDBContext mongoDbContext, HashMap<String,
               Semaphore> availableTaskTillRLS, boolean
               searchOnlyNewTweets) {
        this.tf = tf;
        this.twContext = twContext;
        this.mongoDbContext = mongoDbContext;
        this.availableTaskTillRLS = availableTaskTillRLS;
        this.queryList = queryList;
        this.searchOnlyNewTweets = searchOnlyNewTweets;
```

```
        // FIXED THREAD POOL (AVAILABLE PROCESSORS)
        nThread = Runtime.getRuntime().availableProcessors();
        this.executor = Executors.newFixedThreadPool(nThread);
        // Creazione ThreadPool con un numero fissato di thread
        // BARRIERA viene settata con numero di TASK da lanciare
        this.latch = new CountDownLatch(queryList.size());
    }

    @Override
    public void run() {
        try {
            // Incremento contatore MASTER THREAD in esecuzione!
            twContext.incMasterInExecutionCounterMonitor();

       tf.updateQueryInExecutionCountView(twContext.getMasterInExec
       utionCounter(), twContext.getTaskInExecutionCounter());

            if (queryList.get(0).getType() ==
                    Util.QUERY_SEARCH_KEY) {
                saveToDbQueryResult =
                    ((QuerySearchTweets)queryList.get(0)).getQpSea
                    rchTweets().getSaveToDb();
                collectionNameToSave =
                    ((QuerySearchTweets)queryList.get(0)).getQpSea
                    rchTweets().getCollectionNameToSave();
                tweetsRetrieved = new int[queryList.size()];
                tweetsSaved = new int[queryList.size()];

                if (searchOnlyNewTweets) {
                    // query mongodb, per lowest e greatest id
                      // (tra i tweet misti, da pag. diverse)
                    long[] searchIDs =
                              MongoDbOperation.getLowestGreatestIdsF
                              romTweetCollection(mongoDbContext.getD
                              b(), collectionNameToSave);
                    lowestIDfetchedFromSavedTweet =
                              searchIDs[0];
                    greatestIDfetchedFromSavedTweets =
                              searchIDs[1];
                }
            }

            int indexQueryTask = 0;
            // Per ogni QUERY nella QUERY LIST ricevuta
            // ExecutorService invoca un nuovo THREAD(TASK)
            for (QueryTypeFactory qtf : queryList) {
                switch (qtf.getType()) {
                case Util.QUERY_RATE_LIMIT_STATUS:
                    executor.execute(new TwQueryTask(latch, tf,
                              twContext, (QueryRateLimitStatus)qtf,
                              availableTaskTillRLS));
                    break;
                case Util.QUERY_SEARCH_KEY:
                    executor.execute(new TwQueryTask(this,
                              latch, tf, twContext,
                              (QuerySearchTweets)qtf,
                              availableTaskTillRLS, indexQueryTask,
                              searchOnlyNewTweets,
                              lowestIDfetchedFromSavedTweet,
                              greatestIDfetchedFromSavedTweets));
                    indexQueryTask++;
                    break;
                case Util.QUERY_SEARCH_USERLIST_BY_KEYWORD:
                    executor.execute(new TwQueryTask(latch, tf,
                              twContext,
                              mongoDbContext,(QuerySearchUserlist)qt
                              f, availableTaskTillRLS));
                    break;
                case Util.QUERY_SEARCH_RETWEET:
                    executor.execute(new TwQueryTask(latch, tf,
```

```
                                twContext, mongoDbContext,
                                (QuerySearchRetweets)qtf,
                                availableTaskTillRLS));
                        break;
                    }
                }
                // Barriera in attesa di tutti i THREAD-TASK
                latch.await();

                // Prima di uscire
                executor.shutdownNow();

                // Mostra notifica popup al termine di tutti i TASK
                switch (queryList.get(0).getType()) {
                case Util.QUERY_RATE_LIMIT_STATUS:

                   tf.showPopupNotificationForRateLimitStatusResult(twCo
                   ntext.getLastRateLimitStatusChecked());
                    break;
                case Util.QUERY_SEARCH_KEY:
                    int sumTweetRetrieved =
                           Arrays.stream(tweetsRetrieved).sum();
                    int sumTweetSaved =
                           Arrays.stream(tweetsSaved).sum();
                    if (saveToDbQueryResult) {
                           tf.updateSavedTweetsResultToDb(collectionNameT
                           oSave, sumTweetRetrieved, sumTweetSaved);
                    }

                   tf.showPopupNotificationForSearchResult(Util.QUERY_SE
                   ARCH_KEY, Arrays.stream(tweetsRetrieved).sum());
                  // Rimuovo query da LISTA PENDENTI
                   QueryTypeFactory removedQuery =
                           twContext.findAndRemoveFromPendentList(queryLi
                           st.get(0).getQueryId());
                  tf.updateQueryPendentList();
                    // Aggiorno lista query HISTORY (PER LOG FRAME)
                    twContext.getHistoryQueryList().addQuery(new
                           QueryNotifyResultSearchTweets(removedQuery.get
                           QueryId(), saveToDbQueryResult,
                           sumTweetRetrieved, sumTweetSaved,
                           queryList.size()));
                    tf.updateQueryHistoryList();
                    break;
                case Util.QUERY_SEARCH_USERLIST_BY_KEYWORD:
                   tf.showPopupNotificationForSearchResult(Util.QUERY_SE
                   ARCH_USERLIST_BY_KEYWORD, 0);
                    break;
                }

                // Decremento il contatore dei MASTER THREAD
                twContext.decMasterInExecutionCounterMonitor();
            tf.updateQueryInExecutionCountView(twContext.getMasterInExec
            utionCounter(), twContext.getTaskInExecutionCounter());

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        public void updateQueryTaskTweetsCountResult(
                    int tweetsRetrievedFromTask, int indexQueryTask){
            tweetsRetrieved[indexQueryTask] = tweetsRetrievedFromTask;
        }

        public void updateQueryTaskTweetsSavedCount(
                    int tweetsSavedFromTask, int indexQueryTask){
            tweetsSaved[indexQueryTask] = tweetsSavedFromTask;
        }
    }
```

Codice 3: Codice sorgente completo, della classe *TwQueryThreadMaster*.

TwitterQueryTask

La classe *TwQueryTask* è stata ideata con l'intento di rappresentare i *task* svolti dai *thread "worker"*, nell'architettura precedentemente esposta. In particolare, questa classe, realizza l'interfaccia *Runnable*, e in base ai differenti metodi costruttori implementati, inizializza e modifica lo stato attuale del *model* per poter risolvere le richieste che vengono demandate al *TwQueryThreadMaster*.

Di seguito verrà presentata una porzione del codice, che presenta la dichiarazione dei *metodi costruttori* sopracitati, e del metodo *Run()*.

NOTA
Per l'implementazione delle singole richieste, si vedano i paragrafi successivi a questo.

TwQueryTask.java

```
package twitter.query.core;

import … …

public class TwQueryTask implements Runnable {

    private TwitterFrame tf;
    private TwitterContext twContext;
    private MongoDBContext mongoDbContext;
    private TimerUpdaterRateLimitStatus timerUpdaterRateLimitStatus;

    private QueryRateLimitStatus queryRateLimitStatus;
    private QueryTimeline queryTimeline;
    private QuerySearchTweets querySearchTweets;
    private QuerySearchUserlist querySearchUserlist;
    private QuerySearchRetweets querySearchRetweets;

    private String typeQuery;
    private List<String> jsonSplittedTweetsToSave;
    private HashMap<String, Semaphore> availableTaskTillRLS;
    private CountDownLatch latch;

    private int queryTaskIndex;
    private TwQueryThreadMaster masterThread;

    private boolean searchOnlyNewTweets;
    private long lowestIDfetchedFromSavedTweets;
    private long greatestIDfetchedFromSavedTweets;

    private RetweetTree tree;
    private List<GraphUser> retweetersInfos;
    private long univokeExactlyRetweetFakeId;
    private StringBuilder textRetweetGraphAnalysisResult;
    private String timeRetweetGraphCalulateStarted;
    private long userAnalyzedCount;
```

```
/*** CONSTRUCTOR METHODS ***/
//*****************************************************

//  QUERY TIPO 0: CHECK RATE-LIMIT-STATUS
public TwQueryTask(CountDownLatch latch, TwitterFrame tf,
    TwitterContext twContext, QueryRateLimitStatus query,
    HashMap<String, Semaphore> availableTaskTillRLS) {
     this.tf = tf;
     this.twContext = twContext;
     this.typeQuery = query.getType();
     this.queryRateLimitStatus = query;
     this.latch = latch;
     this.timerUpdaterRateLimitStatus =
            query.getTimerUpdaterRateLimitStatus();
     this.availableTaskTillRLS = availableTaskTillRLS;
}

//  QUERY TIPO 1: TIMELINE UTENTE
public TwQueryTask(CountDownLatch latch, TwitterFrame tf,
            TwitterContext twContext, QueryTimeline query) {
     this.tf = tf;
     this.twContext = twContext;
     this.typeQuery = query.getType();
     this.queryTimeline = query;
     this.latch = latch;
     this.timerUpdaterRateLimitStatus =
                    query.getTimerUpdaterRateLimitStatus();
}

// QUERY TIPO 2: SEARCH PAROLE CHIAVE NEI TWEET
public TwQueryTask(TwQueryThreadMaster masterThread,
            CountDownLatch latch, TwitterFrame tf, TwitterContext
            twContext,
           QuerySearchTweets query,
            HashMap<String,Semaphore> availableTaskTillRLS, int
            queryTaskIndex, boolean searchOnlyNewTweets, long
            lowestIDfetchedFromSavedTweets,
            long greatestIDfetchedFromSavedTweets) {
     this.tf = tf;
     this.twContext = twContext;
     this.typeQuery = query.getType();
     this.querySearchTweets = query;
     this.availableTaskTillRLS = availableTaskTillRLS;
     this.latch = latch;
     this.queryTaskIndex = queryTaskIndex;
     this.masterThread = masterThread;
     this.searchOnlyNewTweets = searchOnlyNewTweets;
     this.lowestIDfetchedFromSavedTweets =
                    lowestIDfetchedFromSavedTweets;
     this.greatestIDfetchedFromSavedTweets =
                    greatestIDfetchedFromSavedTweets;
     this.timerUpdaterRateLimitStatus =
                    query.getTimerUpdaterRateLimitStatus();
}

// QUERY TIPO 3: SEARCH LISTA UTENTI TRAMITE PAROLE CHIAVE
public TwQueryTask(CountDownLatch latch, TwitterFrame tf,
            TwitterContext twContext, MongoDBContext
            mongoDbContext, QuerySearchUserlist query,
            HashMap<String, Semaphore> availableTaskTillRLS) {
     this.tf = tf;
     this.twContext = twContext;
     this.mongoDbContext = mongoDbContext;
     this.typeQuery = query.getType();
     this.querySearchUserlist = query;
     this.availableTaskTillRLS = availableTaskTillRLS;
     this.latch = latch;
     this.timerUpdaterRateLimitStatus =
                    query.getTimerUpdaterRateLimitStatus();
}
```

```
    // QUERY TIPO 4: SEARCH LISTA RETWEET
        public TwQueryTask(CountDownLatch latch, TwitterFrame
                    tf, TwitterContext twContext, MongoDBContext
                    mongoDbContext, QuerySearchRetweets query,
                    HashMap<String, Semaphore>
                    availableTaskTillRLS) {
            this.tf = tf;
            this.twContext = twContext;
            this.mongoDbContext = mongoDbContext;
            this.typeQuery = query.getType();
            this.querySearchRetweets = query;
            this.latch = latch;
            this.availableTaskTillRLS = availableTaskTillRLS;
            this.timerUpdaterRateLimitStatus =
                    query.getTimerUpdaterRateLimitStatus();
            retweetersInfos = new ArrayList<GraphUser>();
            textRetweetGraphAnalysisResult = new
                    StringBuilder();
        }

    @Override
    public void run() {

        // Incremento il contatore dei TASK THREAD
        twContext.incTaskInExecutionCounterMonitor();

       tf.updateQueryInExecutionCountView(twContext.getMasterInExec
       utionCounter(), twContext.getTaskInExecutionCounter());

        switch (typeQuery) {
        case Util.QUERY_RATE_LIMIT_STATUS:

       checkTwitterRateLimitStatus(queryRateLimitStatus.getKeysRequ
       estedToUpdate());
            break;
        case Util.QUERY_TIMELINE:
            //getUserTimeline();
            break;
        case Util.QUERY_SEARCH_KEY:
            searchKeywordsInTweets(searchOnlyNewTweets,
                    lowestIDfetchedFromSavedTweets,
                    greatestIDfetchedFromSavedTweets);
            break;
        case Util.QUERY_SEARCH_USERLIST_BY_KEYWORD:
            searchUserlistByKeyword();
            break;
        case Util.QUERY_SEARCH_RETWEET:
            searchFollowersRetweetsGraph();
            break;
        }

        // TASK al TERMINE dell'ESECUZIONE, comunica al MASTER
        latch.countDown();

        // Decremento il contatore dei TASK THREAD
        twContext.decTaskInExecutionCounterMonitor();

       tf.updateQueryInExecutionCountView(
                    twContext.getMasterInExecutionCounter(),
                    twContext.getTaskInExecutionCounter());
    }
... ...
```

Codice 4: (Porzione) Codice sorgente relativo alla classe *TwQueryTask*, limitato alla sola presentazione dei metodi costruttori e del metodo *run()*.

4.3.7 Gestione dei Rate-Limit-Status (Semafori e Monitor)

Le *API di Twitter* (nella versione utilizzata, v1.1) forniscono un metodo di accesso all'omonimo *social network*, in maniera del tutto gratuito, sottostando però a determinate limitazioni. Ogni richiesta infatti deve essere autenticata secondo il protocollo *OAuth-Authentication* (definito nel [Paragrafo 2.3.1]).

Definito dunque il contesto, quindi la risorsa utilizzata, sia essa *user-authentication o application-authentication*, occorre che l'applicazione che utilizza tali *API*, limiti le richieste effettuate da tale utente, in modo da sottostare a quello che viene definito il ***Rate-Limit-Status (RLS)***.

Ogni 15 minuti (rappresentanti la finestra temporale di riavvio del *RLS*) i parametri relativi alla richiesta in questione, vengono resettati al loro valore iniziale. Ogni *rate-limit-status* ricevuto, contiene i seguenti campi:

- **X-Rate-Limit-Limit***:* il valore limite (massimo) per le richieste effettuabili da tale contesto, nell'arco della finestra temporale.
- **X-Rate-Limit-Remaining:** il numero rimanente di richieste possibili, prima dello scadere della finestra.
- ***X-Rate-Limit-Reset:*** il tempo rimanente prima che la finestra temporale del *RLS* si riavvii (espresso in *UTC epoch seconds*).

Di seguito viene mostrato il codice relativo alla richiesta di effettuare una *CheckTwitterRateLimitStatus*, all'interno della classe *TwQueryTask*.

TwQueryTask.java (checkTwitterRateLimitStatus)

```
//  QUERY TIPO 0: CHECK RATE-LIMIT-STATUS
    private void checkTwitterRateLimitStatus(
                              String[] keysRequestedToUpdate){
        try {
            String lastRateLimitStatusCheckedParsed =
               twContext.getLastRateLimitStatusCheckedParsedToString();
               availableTaskTillRLS.get(
                      Util.RATE_LIMIT_STATUS_APPLICATION).acquire();

                Map<String,RateLimitStatus> limitStatus =
                      twContext.getTwitter().getRateLimitStatus();
                twContext.setLastRateLimitStatusChecked(limitStatus);
                updateViewRateLimitStatus(keysRequestedToUpdate);

                if (lastRateLimitStatusCheckedParsed.equals("")) {
                    CursorToolkit.stopWaitCursor(tf);

               timerUpdaterRateLimitStatus.startRLSDaemonPersistentNotifi
```

```
                cation(limitStatus.get(Util.RATE_LIMIT_STATUS_APPLICATION)
                .getSecondsUntilReset());
                 }
                 for (String keyRLS : Util.RATE_LIMIT_TO_UPDATE) {
                     if ((limitStatus.containsKey(keyRLS))
                       &&(limitStatus.get(keyRLS).getRemaining()<limitStat
                       us.get(keyRLS).getLimit())&&(limitStatus.get(keyRLS
                       ).getSecondsUntilReset()>0)) {

                               timerUpdaterRateLimitStatus.addRlsUpdaterTim
                               erToDaemon(keyRLS,
                               limitStatus.get(keyRLS).getSecondsUntilReset
                               ());
                     }
                 }
                 checkAvailableTaskTillRLS(keysRequestedToUpdate);

         } catch (TwitterException e) {
             twContext.setErrorOAuthStatusCode(e.getStatusCode());
             launchPopupNotificationErrorQuery(e.getErrorMessage());
         } catch (InterruptedException e) {
             launchPopupNotificationInterruptedQuery();
         }
     }

     private void checkAvailableTaskTillRLS(
                String[]keysRequestedToUpdate)
                throws InterruptedException{
         if (availableTaskTillRLS!=null) {

             for (String key : keysRequestedToUpdate) {
                 Semaphore sem = availableTaskTillRLS.get(key);

                 if (sem != null) {
                     int currentAvailablePermits =
                       availableTaskTillRLS.get(key).availablePermits();
                     int releasePermits =
                       twContext.getLastRateLimitStatusChecked().get(key).
                       getRemaining()
                       - Util.LIMIT_MINIMUM_RATE_LIMIT_STATUS
                       - currentAvailablePermits;
                     if (releasePermits>0) {
                       availableTaskTillRLS.get(key).release(releasePermit
                       s);
                     }else if (releasePermits<0){
                       if(twContext.getLastRateLimitStatusChecked().get(ke
                       y).getSecondsUntilReset()>0){
                               availableTaskTillRLS.get(key).acquire(Math.a
                               bs(releasePermits));
                         }
                     }
                 }
             }

         }
     }
```

Codice 5: (Porzione) Codice sorgente riferito al metodo *checkTwitterRateLimitStatus()*, all'interno della classe *TwQueryTask*.

Come ogni altro servizio di *API REST* di questo genere, che offre un numero limitato di richieste gratuite effettuabile da ogni utente autenticato; nel caso in cui questo utente (sia esso un normale *user* o direttamente l'intero contesto *application*), superi il limite di richieste consentito, verrà prima di tutto avvisato, ricevendo in risposta un messaggio con un codice di errore (il quale potrà essere decodificato in base alla documentazione esposta dal sito).

Se questo comportamento dovesse venire reiterato più volte nell'arco di un breve lasso di tempo, il servizio potrebbe decidere di inserire tale utente (o tale applicazione) direttamente in una *"**blacklist**"*, dove gli utenti inseriti non possono più effettuare alcun genere di richiesta, almeno fino a quando questi non vengano riabilitati.

---

*Vista la natura dell'applicazione, e data la necessità di voler mettere in coda un numero multiplo di richieste a Twitter, considerando anche la possibilità che l'applicazione stia in funzione anche in un periodo di tempo non supervisionato dall'utente utilizzatore, sono state prese numerose precauzioni ed è stata implementata una "**strategia**" per evitare si superare i limiti consentiti.*

Nel paragrafo successivo, verrà descritta questa strategia, con la relativa implementazione e la presentazione dei componenti grafici appositamente predisposti, per permettere all'utente utilizzatore di monitorare la situazione del suo *RLS*.

4.3.8 Gestione delle Finestre Temporali (Timer-Task)

Per la gestione dei *RLS* è stata realizzata una strategia, che cerca di prevenire ogni possibile "sforamento" dei limiti consentiti.

Questa soluzione fa utilizzo di un package appartenente al namespace (twitter.), denominato **twitter.data.synchronization**. Le classi contenuti all'interno di questo package, sfruttano meccanismi di sincronizzazione e di mutua esclusione (come semafori e monitor) per mettere in attesa i differenti thread che tentano di effettuare le richieste al servizio, nei casi sopracitati, in cui si raggiunga il limite minimo di richieste disponibili.*

In particolare, come era facile ipotizzare, ogni qualvolta un *thread* si trovi nella situazione in cui debba effettuare una richiesta alle *API di Twitter*, si richiede a tale *thread* di effettuare un'operazione di *acquire* su un'apposita *hashmap* composta da ***counter semaphore***, contenenti i singoli quantitativi limite di richieste. Chiaramente, nel caso in cui un semaforo (relativo ad una richiesta specifica) non possieda più *permits* disponibili, il *thread* si bloccherà in una situazione di attesa finché qualcun altro non notificherà lui e i restanti *thread*

bloccati, dell'inserimento di nuovi *permits* dovuti alla chiusura di una finestra temporale. Questa "entità" che si deve occupare della riabilitazione dei *thread* bloccati in attesa di permessi, è stata realizzata con una classe denominata **TimerUpdaterRateLimitStatus**.

Tale classe, permette di istanziare ed avviare, differenti ***TimerTask***, ovvero dei *task* programmati, che permettono di svolgere determinate azioni allo scadere del loro tempo di attesa. Tale classe viene fornita dalla libreria *java.util.Timer.schedule*, e distingue il metodo di chiamata *Schedule()* in due differenti tipologie.

Innanzitutto sono stati differenziati i *RLS* in due tipologie:

- Il primo tipo è relativo alla sola richiesta *search-rate-limit-status* (ovvero la richiesta che viene effettuata tutte le volte che si verifica l'attuale stato dei limiti di contesto).
  Per tale tipologia di *RLS*, è stato deciso di istanziare un "**timer daemon**" (ovvero un "demone") che esegue il codice implementato nel metodo Run() del TimerTask, dopo un determinato **delay** (espresso in millisecondi), e poi continua la sua esecuzione, alternando un tempo di attesa di **period** (anch'esso espresso in millisecondi) all'esecuzione del metodo Run().
- La seconda tipologia di chiamata di *RLS*, riguarda tutte le restanti richieste a *Twitter* (esempio: *search-tweets, search-user, search-followers*, ...).
  Per tali richieste, a differenza del caso specifico precedente, è stato istanziato un "tradizionale" *TimerTask*, con il solo utilizzo di ***delay***. Tali richieste infatti, una volta chiamate, avviano una finestra temporale di *RLS*, e al termine di essa, riattivano tutti i permessi per la tipologia di risorsa, senza riaprire nuovamente una finestra temporale (non richiedendo dunque l'avvio di un *"demone"*, bensì di un semplice *task* temporizzato).

Di seguito verrà, dapprima mostrato il componente grafico ideato per la rappresentazione visiva e l'aggiornamento dello *status* (visto dall'utente), e in un secondo momento, verrà presentato il codice della stessa classe.

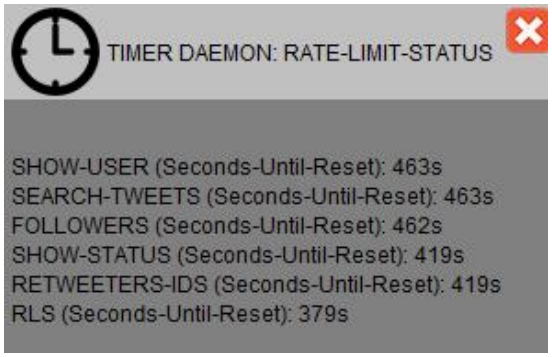


Figura 36: Per la rappresentazione visiva del *timer*, è stata realizzata una classe *PopupNotification (che verrà discussa nei paragrafi successivi)*, che in questo caso viene richiamata con una particolare parametrizzazione.

La classe *PopupNotification* infatti, è stata realizzata per dare la possibilità di far apparire piccole finestre di notifica (con un tempo di vita, di alcuni secondi). In questo caso viene utilizzata per rappresentare una piccola finestra (non decorata, e posizionata in maniera fissa, nell'angolo in alto a destra, rispetto allo schermo), la quale non scompare dopo un determinato lasso di tempo, e ogni secondo, aggiorna i rispettivi contatori dei *RLS* avviati.

TimerUpdaterRateLimitStatus.java

```
package twitter.data.synchronization;
import … …

/*** TWITTER API 1.1 ********************************************/
/*** LA FINESTRA TEMPORALE DEL RATE-LIMIT-STATUS di 15 MINUTI.  ***/
/*** Ogni 15 minuti, riavvia permessi Customer e Application    ***/
/*** "search/tweets" -> customer = 180/15min -> appl=450/15min  ***/
/*** NB: OGNI RICHIESTA HA IL SUO RLS,                          ***/
/***     OGNI FINESTRA TEMPORALE SI AVVIA IN TEMPI DIVERSI      ***/
/****************************************************************/

// Richiesta forza di aggiornare Rate-Limit-Status (ogni 15 minuti,
finestra di tempo di Twitter API 1.1)
public class TimerUpdaterRateLimitStatus {

    private TwitterFrame tf;
    private TwitterContext twContext;
    private HashMap<String,Semaphore> availableTaskTillRLS;
    private Timer uploadCheckerTimer;
    private Timer uploadRlsSemaphores;
    private HashMap<String,TimerTask> timerUpdaterStarted;
    private long timeWindowUntilReset;
    private TwitterQueryCaller controller;

    private PopupNotification currentPopupNotificationTimer;

    public TimerUpdaterRateLimitStatus(TwitterQueryCaller controller,
               TwitterFrame tf, TwitterContext twContext,
               HashMap<String,Semaphore> availableTaskTillRLS) {
        this.tf = tf;
        this.twContext = twContext;
        this.availableTaskTillRLS = availableTaskTillRLS;
        this.controller = controller;

        uploadCheckerTimer = new Timer(true);   // demone (per RLS)
        timeWindowUntilReset =
                    Util.TIME_WINDOW_TWITTER_UNTIL_RESET_RLS;
        timerUpdaterStarted = new HashMap<String,TimerTask>();
```

```
        uploadRlsSemaphores = new Timer(false); // per semafori
    }

    public void initTimerUpdaters(){
        // elimino task dai timer
        if (uploadCheckerTimer!=null) {
            uploadCheckerTimer.cancel();
            uploadCheckerTimer.purge();
        }
        if (uploadRlsSemaphores!=null) {
            uploadRlsSemaphores.cancel();
            uploadRlsSemaphores.purge();
        }
        if (timerUpdaterStarted!=null) {
            for (String key : timerUpdaterStarted.keySet()) {
                timerUpdaterStarted.get(key).cancel();
            }
        }

        uploadCheckerTimer = new Timer(true);   // demone (per RLS)
        timeWindowUntilReset =
                    Util.TIME_WINDOW_TWITTER_UNTIL_RESET_RLS;
        if (currentPopupNotificationTimer!=null) {
            currentPopupNotificationTimer.dispose();
        }
        timerUpdaterStarted = new HashMap<String,TimerTask>();
        uploadRlsSemaphores = new Timer(false); // per semafori
    }

    // Richiede (TASK, DELAY, PERIOD)
    // delay = tempo preso dal RLS in modo attendere e SINCRONIZZARSI
    // period = tempo trascorso tra ogni esecuzione successiva
    public void setTimeToResetdAndRunDaemon(
        long milliSecondsUntilFirstReset){
        showPopupNotificationTimer(milliSecondsUntilFirstReset);
    }

    public void startRLSDaemonPersistentNotification(
                            long secondsUntilFirstReset){
        currentPopupNotificationTimer.addRlsUpdater(
        Util.RATE_LIMIT_STATUS_APPLICATION, secondsUntilFirstReset);
        System.out.println("!!!! AVVIATO DEMONE RLS !!!!");
        TimerTask task = new TimerTask() {
            @Override
            public void run() {
               controller.executeQuery_CheckTwitterRateLimitStatus();
            }
        };

        timerUpdaterStarted.put(Util.RATE_LIMIT_STATUS_APPLICATION,task);

        if (secondsUntilFirstReset>0) {
            uploadCheckerTimer.schedule(task,
               (secondsUntilFirstReset+5)*1000,timeWindowUntilReset);
        }else {
            uploadCheckerTimer.schedule(task, timeWindowUntilReset-
     (Math.abs(secondsUntilFirstReset))*1000+5000, timeWindowUntilReset);
        }
    }

    public void addRlsUpdaterTimerToDaemon(String typeRLS,
                            long secondsUntilFirstReset){
        if (currentPopupNotificationTimer != null) {
            if(typeRLS.compareTo(Util.RATE_LIMIT_STATUS_APPLICATION)!=0){
                currentPopupNotificationTimer.addRlsUpdater(
                            typeRLS, secondsUntilFirstReset);
                final String type = typeRLS;
                if (!timerUpdaterStarted.containsKey(typeRLS)) {
                    System.out.println("AVVIATO "+type+" TIMER!");

                    TimerTask task = new TimerTask() {
                        @Override
                        public void run() {
```

```
                            int releasePermits = 0;
                            if (twContext.getAuthenticationMethod()
                              == Util.OAuth_AuthenticationType.USER_OAUTH)
                              {
                                 switch (type) {
                                 case
                                     Util.RATE_LIMIT_STATUS_SEARCH_TWEETS:
                                      releasePermits =
                                       Util.LIMIT_RLS_SEARCH_TWEETS_USER;
                                      break;
                                  case … … altri casi … break;
                                  }
                            }else {
                                 switch (type) {
                                 case
                                  Util.RATE_LIMIT_STATUS_SEARCH_TWEETS:
                                      releasePermits =
                                  Util.LIMIT_RLS_SEARCH_TWEETS_APPLICATION;
                                      break;
                                  case … … altri casi … break;
                                  }
                            }
                     releaseNewRateLimitStatusAvailable(type,releasePermits);
                       controller.executeQuery_CheckTwitterRateLimitStatus();
                                    timerUpdaterStarted.remove(type);
                                }
                            };
                            timerUpdaterStarted.put(typeRLS,task);

                            if (secondsUntilFirstReset>0) {
                                     uploadRlsSemaphores.schedule(task,(secondsUn
                                     tilFirstReset+5)*1000);
                            }else {
                                     uploadRlsSemaphores.schedule(task,timeWindow
                                     UntilReset+5000);
                            }
                        }
                    }else {
                        currentPopupNotificationTimer.addRlsUpdater(
                                    typeRLS, secondsUntilFirstReset);
                    }
                }else {
                    System.out.println("Impossibile aggiungere RLS");
                }
            }

            private void showPopupNotificationTimer(long duration){
                final long d = duration;
                SwingUtilities.invokeLater(new Runnable() {
                    @Override
                    public void run() {
                        currentPopupNotificationTimer = new
                              PopupNotification(twContext,
                              Util.NOTIFICATION_TIMER_DAEMON_RATELIMITSTATUS_HEAD
                              ER, Util.RATE_LIMIT_STATUS_APPLICATION, d);
                    }
                });
            }

            // Aggiorna il nuovo RLS, e rilascia semafori bloccati
            private void releaseNewRateLimitStatusAvailable(String typeRLS,
                int maxReleasePermits){
                int currentAvailablePermits =
                       availableTaskTillRLS.get(typeRLS).availablePermits();
                int releasePermits = maxReleasePermits -
                       Util.LIMIT_MINIMUM_RATE_LIMIT_STATUS -
                       currentAvailablePermits;
                availableTaskTillRLS.get(typeRLS).release(releasePermits);
            }
        }
```

Codice 6: Codice sorgente completo, relativo alla classe *TimerUpdaterRateLimitStatus*.

Occorre tenere in considerazione il fatto che, ad ogni richiesta effettuata alle *API di Twitter*, venga restituito, annesso alla risposta, anche il *RLS relativo a tale richiesta*. Per questo motivo, ad ogni richiesta viene fatto un *"duplice controllo"* e si verifica che lo stato locale dei *RLS* sia allineato con quello reale, monitorato da *Twitter* (per evitare ogni possibile errore).

4.3.9 Gestione delle Pagine di Utenti (Search Userlist)

Un'ulteriore funzionalità implementata per fornire maggiori strumenti all'utilizzatore di questa applicazione, garantisce la possibilità di **ricercare utenti** (per mezzo di parole chiave), concatenando diverse *query*, per le richieste che necessitano di più di 20 utenti restituiti. (In quanto 20, è il numero limite imposto da Twitter, per le risposte di questa tipologia di richiesta).

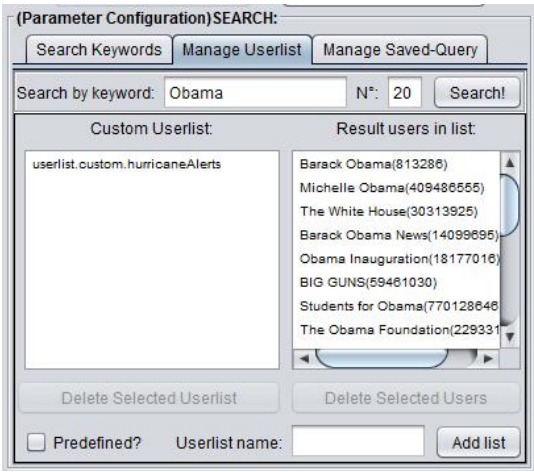


Figura 37: Sezione dedicata alla Gestione delle *Userlist* (esempio: ricerca per nome "Obama" e inserimento di una nuova lista).

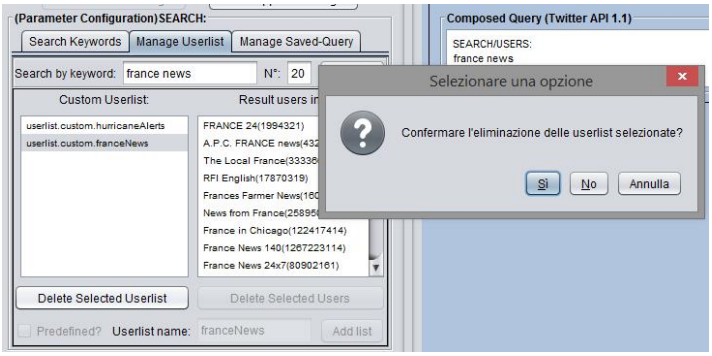


Figura 38: Sezione dedicata alla Gestione delle *Userlist* (esempio di cancellazione di una lista precedentemente memorizzata, con richiesta di conferma).

Di seguito viene mostrato il codice per la ricerca di utenti.

TwQueryTask.java (searchUserlistByKeyword)

```
// QUERY TIPO 3: SEARCH LISTA UTENTI TRAMITE PAROLE CHIAVE
    private void searchUserlistByKeyword() {
        UserlistResult res = null;
        ResponseList<User> queryResult = null;
        DocumentListHashMapByField resultHashList =
              new DocumentListHashMapByField();
        int remainingUser =
       querySearchUserlist.getQpSearchUserlist().getSearchedUserCount();
        int i = 1;
        int userCount = 0;

        try {
         /* ATTENZIONE! MAX RESULT 1000 PERSONE (1000 NOMI = 50 PAG) */
       querySearchUserlist.getQpSearchUserlist().setSearchedUserCount(get
       SearchedUserCountInvariants());
            // Calcolo le pagine equivalenti al numero di persone
               int searchedUserPages =
               getSearchedUserPagesCountInvariants();
            String stringParsed = "PAGINE RICHIESTE(QUERY DA EFFETTUARE):
                                   "+searchedUserPages+
                                  "\nTOTALE USER RICHIESTI:
                                   "+remainingUser +
                                  "\nUSER-LIST RETRIEVED:\n";
            String secondPartStringParsed = "";

            // IMPORTANTE: VERIFICA CHE NON SI SFORI RLS RICHIESTE!!!!
            RateLimitStatus lastUserSearchedRLS =
                      twContext.getLastRateLimitStatusChecked().get(
                      Util.RATE_LIMIT_STATUS_SEARCH_USERS
            boolean end = false;

            while ((i<=searchedUserPages)&&(!end)){

               availableTaskTillRLS.get(Util.RATE_LIMIT_STATUS_SEARCH_USE
               RS).acquire();// <--- MI FERMO SUL SEMAFORO (SE NON HO
               RAGGIUNTO IL LIMITE DEI RSL vado avanti!)

                queryResult =
                      twContext.getTwitter().searchUsers(querySearchUserl
                      ist.getQpSearchUserlist().getAdvancedSearchedQuery(
                      ), i);  // Leggo gli USER della PAGINA(i)

               updateSingleRateLimitStatus(queryResult.getRateLimitStatus
               (), Util.RATE_LIMIT_STATUS_SEARCH_USERS);

                // IMPORTANTE!!! AGGIORNO lastUserSearchedRLS
                lastUserSearchedRLS = queryResult.getRateLimitStatus();

                int userToRead = (queryResult.size() > remainingUser)?
                      remainingUser : queryResult.size();

                if (userToRead>0) {
                    userCount = ((i-1)*20);
                    for (int j = 0; j < userToRead; j++) {
                        User user = queryResult.get(j);
                        String rawJson =
                              TwitterObjectFactory.getRawJSON(user);
                        Long id = user.getId();
                        userCount++;
                        resultHashList.put(id, rawJson);

                        stringParsed +=
                              userCount+")"+user.getName()+"("+id+")\n";
                        secondPartStringParsed +=
                              "\n"+JsonUtilOperation.parseSTRINGJSONtoPRET
                              TYSTRING(rawJson);
                    }
```

```
                    remainingUser -= userToRead;
                }else {
                    end = true; // non ho trovato pi risultati (MI FERMO)
                }
                i++;
            }

        stringParsed += "\nJSON RAW RETRIEVED:"+secondPartStringParsed;
            res = new UserlistResult(stringParsed, resultHashList,
               userCount, querySearchUserlist.getQueryComposedString());
            informTwitterFrameEndQueryTask(querySearchUserlist, res);

        } catch (TwitterException e) {
            System.out.println(e.getMessage());
            launchPopupNotificationErrorQuery(e.getMessage());
        } catch (InterruptedException e) {
            launchPopupNotificationInterruptedQuery();
            System.out.println(e.getMessage());
        }
    }
```

Codice 7: (Porzione) Codice sorgente riferito al metodo searchUserlistByKeyword(), all'interno della classe TwQueryTask.

Salvataggio delle liste di utenti

Viene in aggiunta, data la possibilità di memorizzare all'interno del database, le liste di utenti ricercate ed opportunamente modificate, assegnando ad esse un nome identificativo per poterle riconoscere in futuro, nella fase di selezione dei parametri per l'effettuazione di richieste di *search*.

Vengono forniti inoltre componenti per la **modifica** e **cancellazione** di nomi utente, dalle liste appena ricercate, o da quelle precedentemente memorizzate.

Ed infine, viene data la possibilità di memorizzare la lista in due differenti famiglie:

- Le ***"custom userlist"*** che sono comuni liste di utenti, ricercate e poi memorizzate all'interno del database.
- Le ***"predefined userlist"*** per la memorizzazione di liste "protette", che non possono essere eliminate in futuro, se non dagli amministratori del database, e che saranno le uniche utilizzabili per le *query* di *search-tweets* che permetteranno di salvare gli status, sempre all'interno del database. (Queste ultime *query* verranno trattate in seguito, [Paragrafo 4.3.10]).

4.3.10 Gestione delle Query (Search Tweets)

Come già precedentemente introdotto in [Paragrafo 4.3.6] parlando della classe *TwitterQueryExecutor*, esistono differenti tipologie di "ricerche di *tweets*".

In particolare viene data la possibilità all'utente, di ricercare *status* senza indicare alcun utente specifico, oppure indicando un singolo utente (quindi una singola pagina o *timeline*, su cui andare ad effettuare le ricerche) oppure indicando una lista di utenti (precedentemente memorizzati, attraverso l'apposito strumento).

Oltre ai parametri di *advanced search* definiti in [Paragrafo 4.3.2] e alla specifica delle *userlist* coinvolte [Paragrafo 4.3.9], è presente una sezione dedicata ad altri ulteriori parametri.

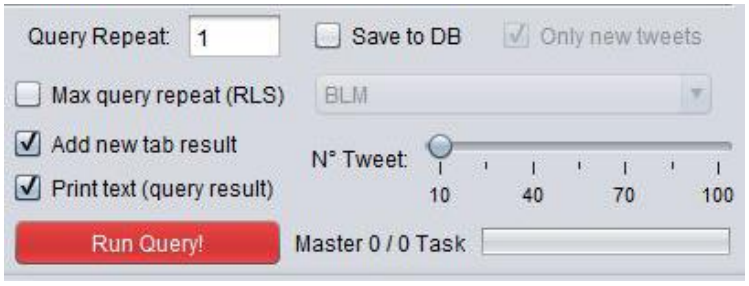


Figura 39: Sezione di parametri specifici per le richieste di Search-tweets.
Come si può vedere in figura, viene data la possibilità all'utente di:
- Specificare il numero di ripetizioni della stessa query.
  (Eventualmente, anche di massimizzare tale numero esaurendo RLS),
- Specificare se presentare a video una tabpage contenente i risultati.
- Specificare se si vuole visualizzare il testo del JSON ricevuto in risposta.
- Decidere se effettuare una query tra quelle "salvate" e di memorizzare i dati ottenuti in tale modo, all'interno del database MongoDB.
- Prefissare un numero di status massimo per ogni richiesta (val≤100).

Salvataggio delle query

Per facilitare il compito dell'utilizzatore, nel catalogare e ripetere eventuali interrogazioni nel tempo, in relazione a quelli che sono gli eventi che egli vuole monitorare; è stata predisposta una sezione che permette di salvare i parametri di ricerca di una precisa *query*.

Queste stesse *query "salvate"* appaiono in [**Figura 39**] nella quarta voce in didascalia, e vengono rappresentate graficamente con un menù a discesa.

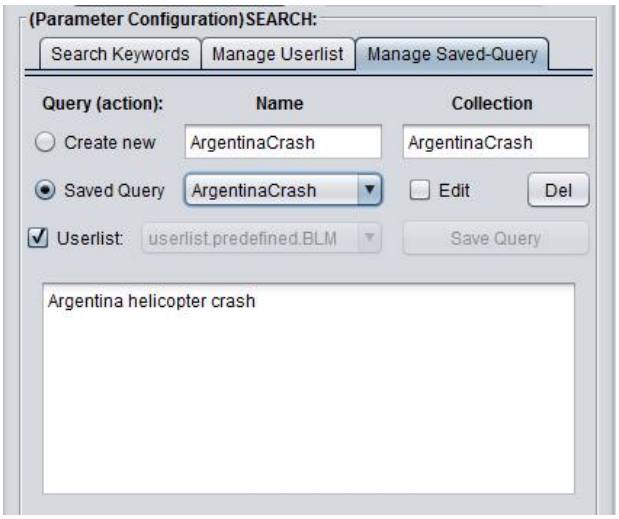


Figura 40: Sezione dedicata al salvataggio delle *query* (esempio: modifica di una *query* precedente, denominata "ArgentinaCrash", con relativi parametri impostati).

Come si può vedere in [**Figura 40**], viene data la possibilità di memorizzare **nuove query** o di **modificare** *query* precedentemente salvate nel *database*.

Tra i campi di impostazione, occorre inserire un nuome univoco per la *query*, ed un nome univoco per la *category* (ovvero la *collection* di *MongoDB*, in cui andare effettivamente a salvare tali parametri di ricerca). Vengono poi richieste, eventuali *userlist* (facoltative) su cui andare ad effettuare l'interrogazione, e direttamente un campo contenente la *query* in formato *advanced search*. Tale *query* può essere composta manualmente seguendo la documentazione di *Twitter*, oppure andando a copiare una *query*, precedentemente lanciata nella prima schermata legata alle *search-tweets*, la quale permette l'impostazione di tali parametri attraverso l'utilizzo di componenti grafici.

SCHEDULING DELLE QUERY

Come già detto nel [Paragrafo 4.3.7], sono state implementate tecniche per la gestione dei *rate-limit-status*, attraverso l'utilizzo di strutture come semafori e monitor.

Grazie a queste strutture, si è creato un sistema modulare e indipendente, che permette, senza arrecare problemi all'utente utilizzatore, di avviare molteplici *query* in successione, e di aspettare che l'applicazione finisca di effettuare le sue computazioni, senza preoccuparsi dei limiti o della gestione immediata dei risultati ricevuti.

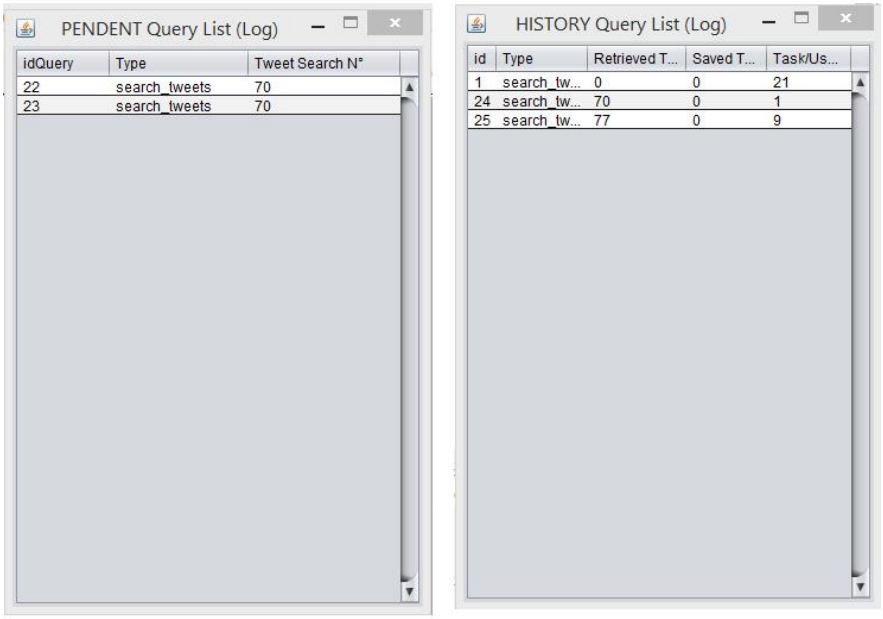


Figura 41: Doppio ***log frame*** dedicato alla visualizzazione delle *query* messe in coda, e della *history* delle *query* effettuate.

Per facilitare la visualizzazione [**Figura 41**] e la gestione delle interrogazioni messe in coda, o eventualmente terminate, sono stati create due finestre con funzionalità di ***"log"***, le quali affiancano la finestra principale fin dall'avvio del programma. La prima finestra si occupa delle *query "ancora pendenti"* e quindi messe in coda, e non ancora risolte, mentre la seconda presenta la *"history"* di tutte le *query* di *search-tweets* e *search-userlist* effettuate, con annessi alcuni dei parametri riassuntivi più salienti (come ad esempio, il numero di *status* restituiti, o il numero di *document* inseriti nel *database MongoDB*).

4.3.11 Notifica dei Risultati

Per poter avvisare l'utente dell'avvenuta computazione, e mostrare l'esito positivo o negativo dell'esecuzione delle *query*, sono stati ideati principalmente due componenti.

1. Direttamente incorporate all'interno di *TwitterFrame* (nella schermata principale dell'applicazione) è stata implementata una sezione dedicata alla presentazione dei risultati ottenuti nelle operazioni di *search-tweets* e *search-userlist*.

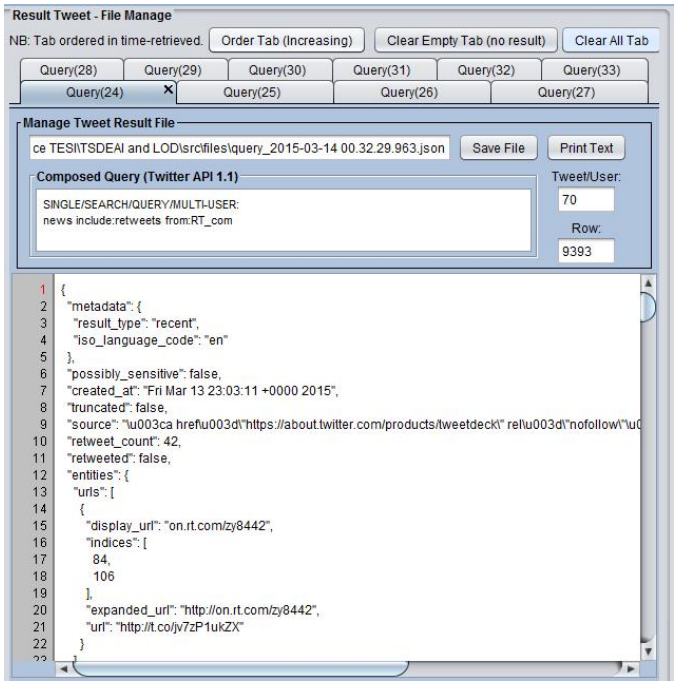


Figura 42: Sezione dell'applicazione dedicata alla presentazione dei risultati, con aggiunta di *tabpages*, e con la possibilità di chiudere tutti i *TabResults*, ordinarli in ordine crescente secondo il loro identificativo (notare che essi vengono restituiti in ordine temporale,

*Osservando in [**Figura 42**] si nota la situazione dell'IDE in un particolare istante temporale, preso successivamente ad alcune chiamate di ricerca di tweets.*

Nella sezione adibita, in risposta alle notifiche ricevute (secondo il *pattern MVC*) direttamente dal *Model* (in particolare da istanze di *TwQueryTask* che vogliono comunicare alla *View* dell'avvenuto completamento di richieste di *search*) corrisponde la **creazione dinamica di *tabpages***, per poter presentare i dati e il risultato di ciascuna *query* lanciata.

In particolare infatti, vengono creati dinamicamente a *runtime*, dei componenti denominati *TabResult*, che permettono la visualizzazione dei dati in risposta da *Twitter* (ricevuti direttamente in formato *JSON*) e di alcuni dati riassuntivi indicanti per esempio il numero di *status* realmente reperiti, o il numero di righe di testo presenti nella risposta.

2. Il secondo metodo permette di notificare l'utente dell'avvenuta computazione, sfruttando delle ***notifiche popup***.
   È stata implementata una classe apposita, denominata *PopupNotification*, che realizza piccole finestre dinamiche, che appaiono ordinatamente nella parte bassa dello schermo, incolonnate sul lato destro, e svaniscono dopo un numero prefissato di secondi (di *default* inferiore ai 10 secondi). Queste notifiche, sono state personalizzate presentando un'icona caratteristica dello stato da notificare, un *header message* (con un titolo esplicativo) e un *body message* (che descrive l'avvenuta computazione, riportando talvolta dati significativi).

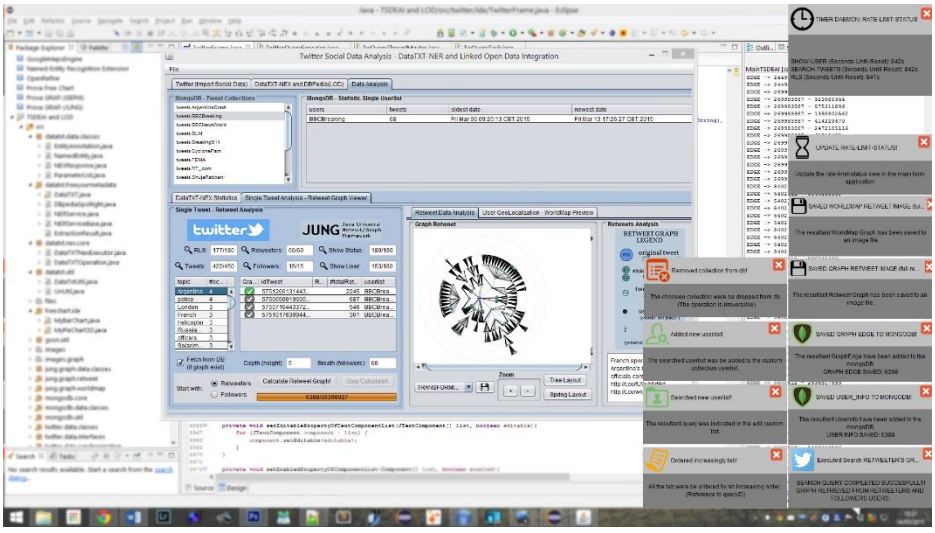


Figura 43: *Screenshot* (modificato per mostrare differenti *Notification Popup*).

Vista la dinamicità del servizio, e la possibilità di avere ipoteticamente decine di operazioni messe in coda e notificate quasi contemporaneamente, si è dovuto gestire dinamicamente il posizionamento di queste finestre. Queste infatti non si sovrappongono mai, partono dall'angolo dello schermo in basso a destra (esattamente superiore alla barra delle applicazioni del sistema operativo) e vanno a posizionarsi nello spazio vuoto dello schermo, andando via via crescendo. Il *timer daemon* (di aggiornamento dei *rate-limit-status*) introdotto in [Paragrafo 4.3.8] è stato volutamente posizionato in maniera fissa, nell'angolo dello schermo in alto a destra, e le notifiche *popup*, sono state progettate per evitare tale finestra.

NOTA

*Entrambe queste funzionalità di notifica dei risultati, possono essere disabilitate per fare in modo che l'applicazione non disturbi il lavoro dell'utilizzatore se questi non vuole esserne notificato. Tutto l'IDE è stato progettato per avere un'***alta configurabilità** *di parametri.*

Di seguito verrà presentato il codice di *TwQueryTask* relativo all'implementazione di *Search-Tweets*.

TwQueryTask.java (searchKeywordsInTweets)

```
// QUERY TIPO 2: SEARCH PAROLE CHIAVE NEI TWEET
    private void searchKeywordsInTweets(boolean searchOnlyNewTweets,
              long lowestIDfetchedFromSavedTweets, long
              greatestIDfetchedFromSavedTweets){

        TwitterResult res = null;
        String stringParsed = "";
        QueryResult queryResult = null;

        int tweetCount = 0;
        int remainingTweet =
             querySearchTweets.getQpSearchTweets().getTweetsSearched()
             *querySearchTweets.getQpSearchTweets().getRepeatQueryCount();

        jsonSplittedTweetsToSave = new ArrayList<String>();
        int currentSubTask = 1;

        Query query = new Query(querySearchTweets.getQpSearchTweets().
                          getAdvancedSearchedQuery());
        query.setCount(querySearchTweets.getQpSearchTweets().
               getTweetsSearched());

        Long lowestId;
        Long greatestId;
        boolean searchingOldestTweets = true;
        boolean searchingNewestTweets = false;
        boolean finito = false;
```

```
        // NOTA BENE: Se ho richiesto di cercare solamente nuovi tweets
        (vengono passati come parametri, lowestID e greatestID)

        if (!searchOnlyNewTweets) {
            lowestId = (long)-1;
            greatestId = (long)-1;
        }else {
            lowestId = lowestIDfetchedFromSavedTweets;
            greatestId = greatestIDfetchedFromSavedTweets;
        }

        List<Status> listTweets = new ArrayList<Status>();

        try {
            while ((tweetCount < remainingTweet)
               &&(!finito)&&(currentSubTask <=
               querySearchTweets.getQpSearchTweets().
               getRepeatQueryCount())) {


       availableTaskTillRLS.get(Util.RATE_LIMIT_STATUS_SEARCH_TWEETS).
       acquire();// <--- MI FERMO SUL SEMAFORO

                if (searchOnlyNewTweets) {
                    // Prima TWEETS VECCHI (LOWEST_ID=DATA PIU' VECCHIA)
                    if (searchingOldestTweets) {
                            query = new Query(querySearchTweets.
                              getQpSearchTweets().
                              getAdvancedSearchedQuery());

                              query.setCount(querySearchTweets.getQpSearch
                              Tweets().getTweetsSearched());
                        query.setMaxId(lowestId+1);

                    // Poi NUOVI TWEETS (GREATEST_ID = DATA PIU' NUOVA)
                    if (searchingNewestTweets) {
                        query = new Query(querySearchTweets.
                      getQpSearchTweets().getAdvancedSearchedQuery());

                      query.setCount(querySearchTweets.getQpSearchTweets(
                      ).getTweetsSearched());
                        query.setSinceId(greatestId);
                    }
                }

                queryResult = twContext.getTwitter().search(query);

               updateSingleRateLimitStatus(queryResult.getRateLimitStatus
               (), Util.RATE_LIMIT_STATUS_SEARCH_TWEETS);

                int currentQueryTweetCount =
                      queryResult.getTweets().size();

                if (currentQueryTweetCount > 0) {
                    for (Status tweet : queryResult.getTweets()) {
          //IMPORTANTE!!! (X DATATXT-NEX <-- SOLO LINGUE: DE,EN,FR,IT,PT
              if (Util.DATATXT_LANG_ACCEPTED.contains(tweet.getLang())) {
                 // IMPORTANTE!!!
                 // Se saveToDb settato, allora memorizzo il vettore dei
               TWEETS(JSON) per poterli passare al metodo di salvataggio!
               if (querySearchTweets.getQpSearchTweets().getSaveToDb()) {

               jsonSplittedTweetsToSave.add(JsonUtilOperation.parseOBJECT
               JSONtoPRETTYSTRING(tweet));
                            }
              stringParsed +=
               JsonUtilOperation.parseOBJECTJSONtoPRETTYSTRING(tweet);
                  // Aggiungo tweet alla lista da restituire
                  listTweets.add(tweet);

                 // TODO (JDK 1.8) CONFRONTO UNSIGNED 64bit
                lowestId = (Long.compareUnsigned(tweet.getId(),lowestId)
                < 0) ? tweet.getId() : lowestId;   // Prendo id minore
```

```
                greatestId = (Long.compareUnsigned(tweet.getId(),greatestId)
                  > 0) ? tweet.getId() : greatestId; // Prendo id maggiore
                    }
                }
                 // IMPORTANTE: dalla query precedente, prendo NextQuery
                  if (queryResult.hasNext()) {
                          query = queryResult.nextQuery();
                       }else {
                       // Se CERCAVO TWEET VECCHI, devo ancora cercare NUOVI
                       // altrimenti, se stavo cercando TWEET NUOVI e NON CI
                          SONO ALTRE QUERY successive possibili, MI FERMO!
                            if (searchOnlyNewTweets) {
                             finito = (searchingNewestTweets) ? true : false;
                            } else {
                                finito = true;
                            }
                       }
                       // Aggiorno contatore totale
                       tweetCount += currentQueryTweetCount;
                   }else {
                       if (searchOnlyNewTweets) {
                           if (searchingOldestTweets) {
                               searchingOldestTweets = false;
                               searchingNewestTweets = true;
                           }else {
                            // Se cercavo NUOVI, e ZERO RISULTATI, MI FERMO!
                               searchingNewestTweets = false;
                               finito = true;
                           }
                       } else {
                           finito = true;
                       }
                   }

                   if (finito) {
                       System.out.println("Query INTERROTTA: non ci sono
                         altri nuovi tweet."); // Esco anche se non ho
                         finito il numero di sub-task ripetizioni! Finito i
                         tweet disponibili sulle pagine recenti!
                   }
                   currentSubTask++;
               }

               // Al termine restituisco il risultato con struttura apposita
               res = new TwitterResult(stringParsed, listTweets, tweetCount,
                  querySearchTweets.getQueryComposedString());
               informTwitterFrameEndQueryTask(querySearchTweets, res);

           } catch (TwitterException e) {
               System.out.println(e.getMessage());
               launchPopupNotificationErrorQuery(e.getMessage());
           } catch (InterruptedException e) {
               launchPopupNotificationInterruptedQuery();
               System.out.println(e.getMessage());
           }
       }
```

Codice 8: (Porzione) Codice sorgente riferito al metodo searchKeywordsInTweets(), all'interno della classe TwQueryTask.

4.4 CLASSIFICAZIONE E INTEGRAZIONE

Per realizzare la *classificazione automatica* delle ***entità***, si è scelto di utilizzare il servizio di *API REST* di *Dandelion.eu*, denominato ***DataTXT-NEX***.

Questo servizio permette di effettuare richieste di ***Named Entity eXtraction*** (quindi *recognition,* e poi estrazione) dei principali *topic* presenti all'interno dei testi sottomessi al servizio.

Questa libreria di API è stata scelta, avendo visto e confrontato i diversi risultati ottenuti dalle altre librerie presenti in rete. Questo servizio in particolare, rispetto agli altri, non utilizza le normali tecniche di *Natural Language Processing (NLP)* per riconoscere ed estrarre i *topic*, bensì si affida al suo *"grafo di conoscenza"* ottenuto integrando e i concetti e le nozioni, prelevate dai *linked-open-data* (ed in particolare, da *DBPedia*).

Questa scelta implementativa, permette di superare i limiti della tecnica *NLP* sopracitata, che in testi brevi (come possono essere i *tweet* estratti dal *social network* omonimo) caratterizzati da numerosi possibili errori, menzioni ed *hashtag*, potenziali errori sintattici e grammaticali, non permette di ottenere sempre dei risultati soddisfacenti.

Il sito di riferimento di tale *libreria,* citato in [Paragrafo 3.4.5], definisce il suo servizio come l'unica *"Named Entity Extraction – Linking API"* sul mercato.

Questo nome fa chiaramente riferimento ad un altro importante fattore positivo, che ha fatto propendere sulla sua scelta. Questo servizio infatti, oltre a fornire funzionalità di *NEX*, permette di collegare i ***topic*** così estratti, alle rispettive definizioni presenti su *DBPedia* (e quindi di soddisfare una delle richieste progettuali inziali, integrando i contenuti direttamente estratti, con conoscenze aggiuntive reperite dal mondo degli *OpenData*).

Per quanto riguarda l'interfaccia, anche in questo *modulo*, verrà presentata la finestra dell'*IDE*, evidenziando le differenti sezioni che la compongono.

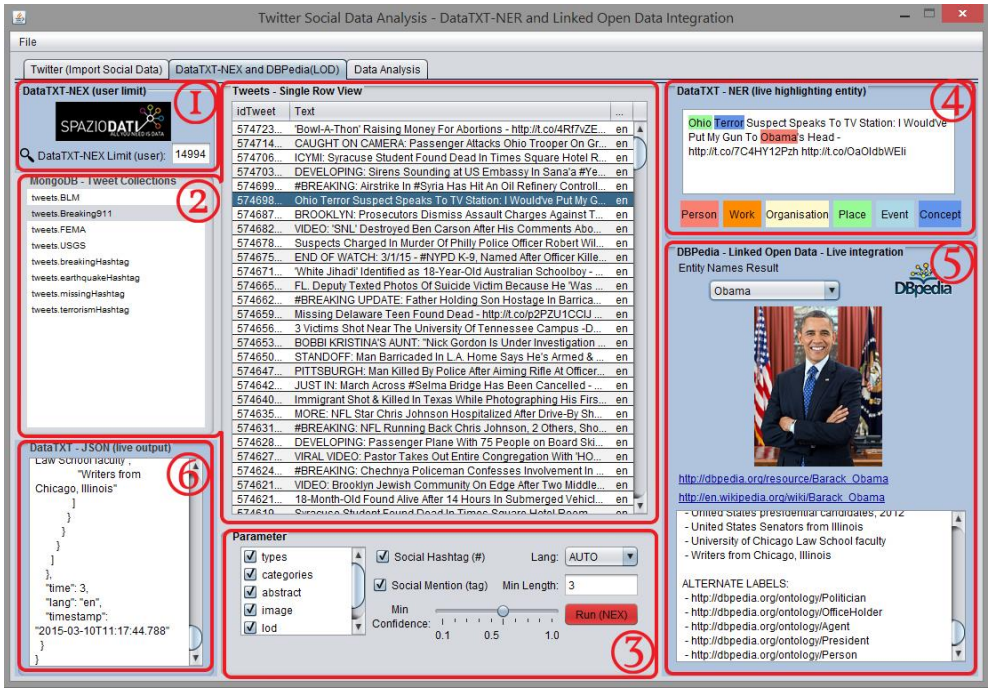


Figura 44: Finestra del secondo *modulo software*, dedicata alla *Classificazione e Integrazione automatica di topic*, realizzata sfruttando il servizio ***DataTXT-NEX***.

Osservando la [**Figura 44**] che rappresenta uno *screenshot* della finestra del modulo di classificazione ed integrazione dei *topic*, si può notare che sono state delimitate e numerate 6 aree differenti. Di seguito verranno introdotte singolarmente:

1. La sezione **(1)**, analogamente al modulo precedente, presenta la visualizzazione dei limiti imposti dal servizio di *API REST*.

   NOTA:
   Normalmente la limitazione di richieste a DataTXT-NEX, per quanto riguarda il servizio gratuito, comprende 1000 richieste giornaliere. Contattando gli sviluppatori del servizio, ed esponendo le caratteristiche del progetto intrapreso a fini di ricerca universitaria, è stata concesso un adeguamento delle limitazioni a 15000 richieste giornaliere (per la durata di un mese). Questo upgrade ha permesso di aumentare il numero di tweet analizzati all'interno dei casi di studio che verranno presentati nel [Capitolo 5].

   ---

2. Nella sezione **(2)** viene presentata una lista aggiornata delle ***collection di tweets*** *(presenti sul database MongoDB)*, reperite ed estratte dal primo modulo *software*. Alla selezione di una delle liste, viene

automaticamente aggiornata la vista adiacente, andando a riempire le righe della tabella con la totalità dei *tweet* estratti, andando a presentare per ogni *status*, solamente l'***idTweet*** (univoco), il ***text*** (testo contenuto all'interno del *tweet*) e ***lang*** (una sigla identificativa del linguaggio, automaticamente classificato da *Twitter*). Quest'ultimo campo verrà utilizzato anche al momento del lancio della richiesta di *estrazione dei topic*, in quanto il servizio *DataTXT-NEX* presenta alcune limitazioni sulle lingue consentite ed ottiene migliori risultati, impostando direttamente la lingua di estrazione, senza utilizzare quella automatica (AUTO("auto")). In particolare, allo stato odierno della libreria, è consentita solamente l'estrazione di testi appartenenti ai seguenti linguaggi: ITA("it"), ENG("en"), FRA("fr"), GER("de"), POR("pt").

3. La sezione **(3)** è caratterizzata da un insieme di componenti grafici per l'impostazione dei parametri di *recognition & extraction* dei ***topic***. Viene ad esempio impostata la lingua, la lunghezza minima accettata, l'eventuale ricerca di *mention (tag di persone)* o di *hashtag*, l'inclusione di diversi contenuti direttamente reperiti da *DBPedia*, ed infine, il parametro più importante, relativo al livello di ***confidenza***. Tale valore è stato impostato di *default* a *0.6 (su un massimo di 1.0)*, seguendo il valore consigliato sulla documentazione. Effettuando dei test con valori modificati, si è chiaramente riscontrato che:
    - Con valori ridotti di *confidence (0.1-0.3)* si ottiene un numero molto elevato di *topic* riconosciuti all'interno del testo analizzato, ma spesso tali entità evidenziate, non presentano una corretta definizione all'interno del contesto del messaggio.
    - Con valori alti di *confidence (0.8-1.0)* si richiede la massima precisione alla libreria, riducendo notevolmente il numero di *topic* riconosciuti, evitando spesso di evidenziare entità fondamentali all'interno del messaggio.
4. Le sezioni **(4-5)** saranno dettagliatamente esposte nel successivo paragrafo dedicato all'integrazione con i contenuti di *DBPedia* [Paragrafo 4.4.3].

5. La sezione **(6)** presenta la risposta completa, ricevuta dal servizio di *API REST di DataTXT-NEX*, sotto forma di *JSON* (la quale è stata opportunamente "tradotta" in un oggetto di classe *NEXResponse*, creato appositamente per contenere tutti i dati reperiti dall'operazione di *"parsing"*).

4.4.1 Modulo Software e Package Implementati

Per introdurre tutte le funzionalità di questo *modulo software*, in maniera analoga al precedente modulo [Paragrafo 4.3], si è deciso di presentare la lista dei *package* implementati.

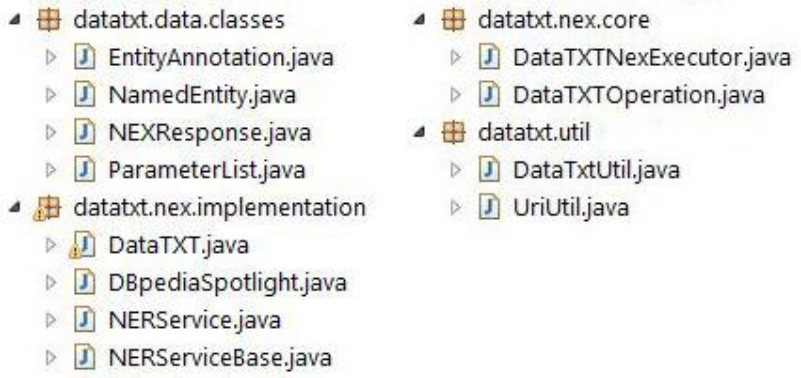


Figura 45: Elenco dei *package* relativi al *namespace (datatxt.*)*.

Osservando [**Figura 45**] si possono evincere 4 differenti *package*:

- ***datatxt.data.classes***, contiene le classi per la definizione di oggetti utili alla trasmissione dei risultati dell'operazione di classificazione di *topic* e alla loro presentazione.
- ***datatxt.nex.implementation***, questo *package* è strettamente legato al successivo (*datatx.nex.core*) e rappresenta la gerarchia di classi per la realizzazione del vero e proprio servizio che effettua le chiamate parametrizzate alle *REST API* di *DataTXT-NEX*.
- ***datatxt.nex.core***, (analogamente al *package twitter.query.core*) presenta una classe *DataTXTNexExecutor* e una *DataTXTOperation*, per rispettare la struttura *Master-Worker* definita per effettuare le

chiamate ai differenti servizi, e per realizzare le vere e proprie chiamate a *DataTXT*.

- ***datatxt.util***, è composto da due classi *DataTxtUtil e UriUtil*, le quali contengono metodi e proprietà statiche, per la parametrizzazione delle richieste, e ottenere una buona leggibilità del codice.

Di seguito, verrà presentato il diagramma UML per la rappresentazione delle classi coinvolte nella catena di richieste per l'estrazione di *topic* e la loro integrazione con gli *open-data*. E successivamente, verrà presentato e brevemente discusso, il codice relativo alle classi principali di questo modulo *software*.

4.4.2 Named Entity Extraction

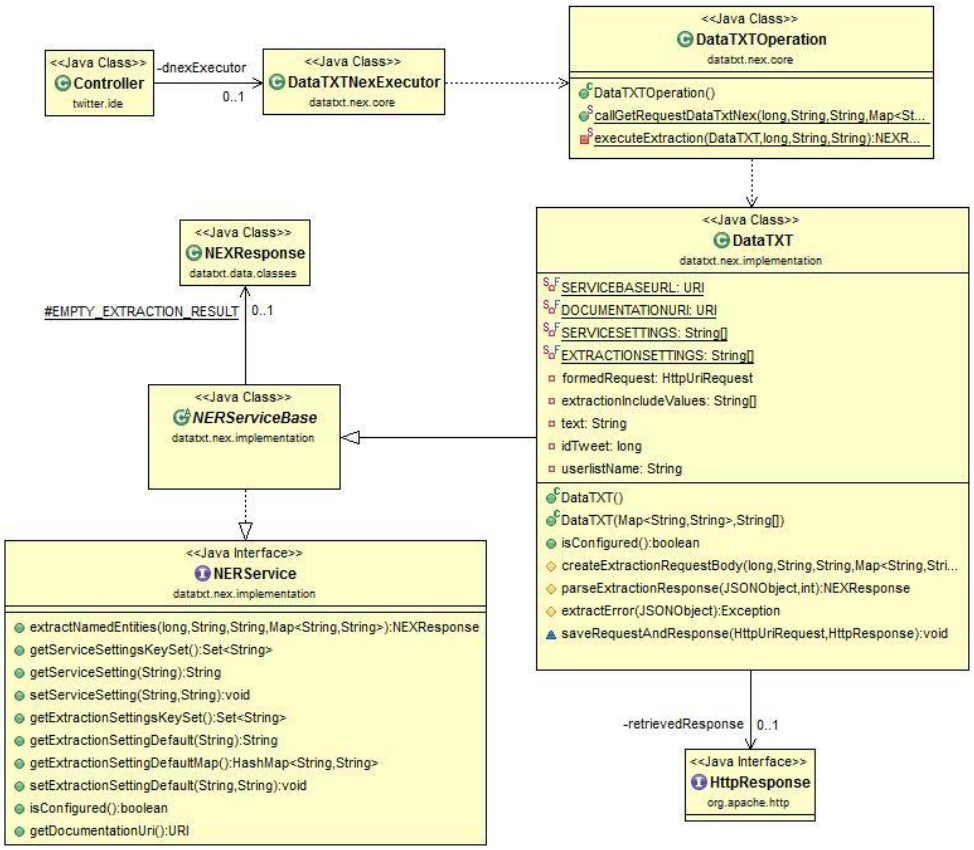


Figura 46: (Porzione) Diagramma UML delle classi, rappresentante le classi coinvolte nella catena di chiamate per l'estrazione di *topic* tramite il servizio ***DataTXT-NEX***.

Analogamente al modulo *software* legato all'estrazione dei *tweet*, anche in questo viene presentato il codice dell'*Executor*.

DataTXTNexExecutor.java

```
package datatxt.nex.core;
import … …

public class DataTXTNexExecutor {

    private TwitterFrame tf;
    private MongoDBContext mongoDBcontext;

    public DataTXTNexExecutor(TwitterFrame tf,
                              MongoDBContext mongoDBcontext) {
        this.tf = tf;
        this.mongoDBcontext = mongoDBcontext;
    }

    public void runGetRequestFromDataTXTNex_SingleTweet(String text,
                      String lang, Map<String,String>
                      extractionParameters, String[] extractionInclude) {
        final String txt = text;
        final Map<String,String> exPar = extractionParameters;
        final String[] exIncl = extractionInclude;
        final String language = lang;

        new Thread(new Runnable() {
            @Override
            public void run() {
                // FORZO LA LINGUA (PER NON INCORRERE IN ERRORI
                      NELL'AUTO-RECOGNITION DI DATATXT-NEX) (2)*
                exPar.remove(DataTxtUtil.PAR_LANG_LABEL);
                exPar.put(DataTxtUtil.PAR_LANG_LABEL, language);

                NEXResponse nexResponse =
                              DataTXTOperation.callGetRequestDataTxtNex(-
                              1, "", txt, exPar, exIncl);
                String prettyStringResponse =
                      JsonUtilOperation.parseSTRINGJSONtoPRETTYSTRING(nex
                      Response.getJsonHttpResponse());
                tf.updateJsonResultStringFromDataTxtNer(nexResponse,
                              prettyStringResponse, txt,
                              nexResponse.getAnnotationLabelsWithIndices()
                              , nexResponse.getxDlUnitLeftLimit());
            }
        }).start();
    }

    public void runGetRequestFromDataTXTNex_ListOfTweet(
              List<Status> listTweets, Map<String, String>
              extractionParameters, String[] extractionInclude, String
              collectionNameWithoutNex, Double confidence) {

        final List<Status> listTw = listTweets;
        final Map<String,String> exPar = extractionParameters;
        final String[] exIncl = extractionInclude;
        final String collectionCompleteName =
                                        "nex."+collectionNameWithoutNex;
       final Double conf = confidence;

        new Thread(new Runnable() {
            @Override
            public void run() {

       // IMPORTANTE: PRIMA DI EFFETTUARE CHIAMATE A SERVIZIO DataTXT-NEX
       //
       //      1) CONTROLLO se ho NEXResponse nel DB (stessa CONFIDENZA)
       //        (TOLGO IDTweet di quelli presenti, EVITA CALL SUPERFLUE)
       //
```

```
        //      2) DATATXT, non gestisce lingue diverse da: IT-EN-DE-PT-FR
        //          (TWEET prelevati da Twitter solo con queste lingue)
        //          INOLTRE ---> FORZO IMPOSTAZIONE LINGUA (NON-AUTO)

                List<Long> idTweetsInNexDB =
                      MongoDbOperation.getIdTweetsWithSameConfidenceFromN
                      EXCollection(mongoDBcontext.getDb(),
                      collectionCompleteName, conf);
                List<Status> list = new ArrayList<Status>();

                for (Status s : listTw) {
                    if (!idTweetsInNexDB.contains(s.getId())) {
                        list.add(s);
                    }
                }

                if (list.size()>0) {
                    int i = 0;
                    List<NEXResponse> listNexResponse =
                                        new ArrayList<NEXResponse>();
                    int limitMinimumDatatxtNex =
                              Util.LIMIT_MINIMUM_DATATXT_NEX+1;

                      while((i<list.size()&&limitMinimumDatatxtNex>Util.L
                      IMIT_MINIMUM_DATATXT_NEX)){
                        // FORZO LA LINGUA (PER NON INCORRERE IN ERRORI
                              NELL'AUTO-RECOGNITION DI DATATXT-NEX) (2)*
                        exPar.remove(DataTxtUtil.PAR_LANG_LABEL);
                        exPar.put(DataTxtUtil.PAR_LANG_LABEL,
                              list.get(i).getLang());

                        NEXResponse nexResponse =
                              DataTXTOperation.callGetRequestDataTxtNex(li
                              st.get(i).getId(),
                              list.get(i).getUser().getScreenName(),list.g
                              et(i).getText(), exPar, exIncl);

                        // IMPORTANTE: Aggiorno il LIMITE per DATATXT-NEX
                        limitMinimumDatatxtNex =
                              nexResponse.getxDlUnitLeftLimit();
                        tf.updateDataTxtNEXlimit(limitMinimumDatatxtNex);

                        tf.updateDataTxtNEXprogressValueInc(false);

                        listNexResponse.add(nexResponse);
                        i++;
                    }

                    // Scrivo sul DB
                      MongoDbOperation.insertDocumentNex(mongoDBcontext.g
                      etDb(), collectionCompleteName,
                      listNexResponse.toArray(new
                      NEXResponse[listNexResponse.size()]), true);
                    // Aggiorno barra (TRUE = BARRA PIENA)
                      tf.updateDataTxtNEXprogressValueInc(true);
                    // Mostro risultati in tabella
                    HashMap<String,Double> hashResult =
                      MongoDbOperation.getHashmapWithUserlistAndTopicsCou
                      nt(mongoDBcontext.getDb(), collectionCompleteName);

                      tf.updateTableResultDataTXTnexUserlist(MongoDbOpera
                      tion.getVectorNEXfromListOfNexResponse(listNexRespo
                      nse),hashResult);
                }
            }
        }).start();
    }
}
```

Codice 9: Codice sorgente della classe *DataTXTNexExecutor.java*.

In particolare, osservando [**Codice 9**], si nota come la chiamata venga distinta in due tipologie:

1. ***runGetRequestFromDataTXTNex_SingleTweet()***
   prevede la classificazione automatica dei *topic* all'interno di un singolo *tweet*. È stata appositamente implementata per l'interfaccia del secondo modulo *software*, in quanto, a differenza del modulo successivo che prevederà la classificazione di tutta una *collection* di *status Twitter*, in questo viene richiesta solamente la presentazione dei dati di un singolo *status*.

   NOTA
   Tra gli scopi di questo modulo, vi è infatti anche la volontà di testare il servizio DataTXT-NEX, e la sua capacità di integrazione con i dati reperiti da DBPedia. Inoltre, si vuole dare la possibilità all'utilizzatore, di visualizzare singolarmente i dati e i topic classifcati dal servizio, per i singoli tweet, in modo ad esempio, da poter visualizzare in dettaglio tutte le informazioni aggiuntive reperite (esempio: foto, descrizioni, …). Questa funzionalità, verrà descritta accuratamente in seguito, [Paragrafo 4.4.3].

   ---

2. ***runGetRequestFromDataTXTNex_ListOfTweet()***,
   il secondo metodo, a differenza del primo, prevede la classificazione di una lista di *tweet* prelevati direttamente da una delle *collection* memorizzate all'interno del *database.*

   In questa seconda tipologia di chiamata, viene attuata una strategia di ***ottimizzazione***, che effettua il seguente controllo, prima di andare a sottomettere realmente la richiesta al servizio *DataTXT-NEX*

   1. Prima di effettuare la richiesta alle *API REST* infatti, si verifica se all'interno del *database MongoDB*, sia già presente una classificazione aggiornata della lista dei *tweet* richiesti (sempre identificati, anche all'interno del *database*, attraverso i loro *idTweet* univoci, a 64 bit). Se la ricerca restituisce un buon esito, e sia richiesta, che sorgente dati, sono allineate, viene aggiornata la vista dell'utente comunicando che non è

necessario effettuare una nuova classificazione, in quanto è possibile reperirla dal *database*.

2. Se il controllo al punto (1) ha dato esito negativo, viene mostrato nell'apposita casella, un messaggio (colorato di rosso, per richiamare l'attenzione dell'utilizzatore) indicante il "disallineamento" tra la lista di *tweet* richiesti (presenti nella *collection* reperita da *Twitter*) e i dati *NEXResponse* presenti sull'istanza del *database MongoDB*.

   Questo "disallineamento" potrebbe dipendere da diversi fattori:

   - Non è mai stata avviata una procedura di classificazione dei *topic* sulla *collection* di *tweet* correntemente selezionata.
   - In precedenza è stata effettuata una classificazione di *topic*, poi interrotta (che quindi non ha completato l'estrazione di *topic* per tutti i *tweet*).
   - Oppure potrebbero essere stati aggiunti dei *tweet*, in un periodo successivo alla prima richiesta di classificazione di essi. Questo può capitare molto spesso, se si monitora lo stesso evento a distanza di ore o giorni, e si richiede nuovamente di estrarre *status* da *Twitter*.
   - Infine, potrebbe essere stata richiesta una classificazione con un valore di *confidence* maggiore rispetto a quello memorizzato all'interno delle *NEXResponse*, salvate all'interno del *database*.

NOTA

Questa seconda tipologia di chiamata, effettuata su una collection di tweet, è stata ideata per il terzo modulo software, e per tale motivo, la rappresentazione grafica dei risultati verrà discussa nell'ultimo modulo [Paragrafo 4.5.2].

Seguendo l'ordine di chiamate, presentato in [**Figura 46**], di seguito verrà presentato il codice relativo all'implementazione di *DataTXTOperation*.

DataTXTOperation.java

```
package datatxt.nex.core;
import java.util.Map;
import datatxt.data.classes.NEXResponse;
import datatxt.nex.implementation.DataTXT;

public class DataTXTOperation {

    public static NEXResponse callGetRequestDataTxtNex(
                long idTweet, String userlistName, String text,
                Map<String,String> extractionParameters,
                String[] extractionInclude){

        DataTXT dataTXT = new
                        DataTXT(extractionParameters,extractionInclude);

        return executeExtraction(dataTXT, idTweet, userlistName, text);
    }

    private static NEXResponse executeExtraction(DataTXT dataTXT,
                long idTweet, String userlistName, String text){

        try {
            NEXResponse res = dataTXT.extractNamedEntities(
                                        idTweet, userlistName, text);
            return res;

        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Codice 10: Codice sorgente relativo alla classe *DataTXTOperation.java*.

Infine verrà presentato il codice relativo al metodo di creazione della richiesta, implementato all'interno della classe *DataTXT.java*, omettendo i restanti metodi e gli atributi della classe (per una questione di leggibilità e di spazio).

DataTXT.java (*createExtractionRequestBody*)

```
protected HttpEntity createExtractionRequestBody(final long idTweet,
                        final String userlistName, final String text, final
                        Map<String, String> extractionSettings) throws
                        UnsupportedEncodingException {

        final ParameterList parameters = new ParameterList();
        this.idTweet = idTweet;
        this.userlistName = userlistName;
        this.text = text;

        parameters.add("lang",
                extractionSettings.get(DataTxtUtil.PAR_LANG_LABEL));
        parameters.add("text", text);
        parameters.add("min_confidence",
            extractionSettings.get(DataTxtUtil.PAR_MIN_CONFIDENCE_LABEL));
        parameters.add("min_length",
                extractionSettings.get(DataTxtUtil.PAR_MIN_LENGTH_LABEL));
        parameters.add("social.hashtag",
            extractionSettings.get(DataTxtUtil.PAR_PARSE_HASHTAG_LABEL));
        parameters.add("social.mention",
            extractionSettings.get(DataTxtUtil.PAR_PARSE_MENTION_LABEL));

        if (extractionIncludeValues.length != 0) {
```

```
            String includes = extractionIncludeValues[0];
            for (int i = 1; i < extractionIncludeValues.length; i++) {
                includes += "," + extractionIncludeValues[i];
            }
            parameters.add("include", includes);
        }
        parameters.add("$app_id",
                getServiceSetting(DataTxtUtil.PAR_APP_ID_LABEL));
        parameters.add("$app_key",
                getServiceSetting(DataTxtUtil.PAR_APP_KEY_LABEL));
        return parameters.toEntity();
    }
```

Codice 11: (Porzione di codice) per motivi di spazio, si è deciso di riportare solamente il metodo *createExtractionRequestBody*, e di omettere i restanti metodi della classe *DataTXT.java*.

La classe *DataTXT*, della quale è stato presentato in [**Codice 11**] uno dei metodi principali, è stata realizzata estendendo la *classe astratta NERServiceBase*, la quale a sua volta, implementa l'interfaccia *NERService*. Queste ultime due, sono state realizzate, estendendo e modificando il codice *open-source* del progetto *freeyourmetadata*, al fine di adattarle a questa soluzione.

4.4.3 DBPedia (Linked-Open-Data)

Per quanto riguarda l'***integrazione*** dei *topic* con informazioni aggiuntive reperite dal mondo dei *Linked-Open-Data*, si è deciso di sfruttare la naturale funzionalità implementata dallo stesso servizio *DataTXT*, il quale permette di richiedere, insieme alle entità estratte, anche alcuni dati aggiuntivi reperiti dal portale *DBPedia*.

In questo progetto, la funzionalità degli open-data ha rivestito solamente un ruolo secondario, al fine di integrare i dati ottenuti dal processo di classificazione automatica, per aumentare il contenuto informativo proposto all'utente e permettere maggiori possibilità in fase di analisi.

In un futuro prossimo, è possibile pensare ad un'estensione del package, introducendo un vero e proprio servizio separato di integrazione dei linked open data, andando a prelevarli non solamente da DBPedia, ma anche da altre ontologie pubbliche.

Per terminare la spiegazione di questa sezione, occorre anche descrivere gli ultimi due gruppi di componenti, presenti nella finestra dell'interfaccia, introdotta in [Paragrafo 4.4]:

1. Per terminare la spiegazione precedente, nella sezione **(4)** è presente una casella di testo contenente il contenuto del *tweet*, e sottostante ad essa, vengono contrassegnate *6 categorie di contenuto*.

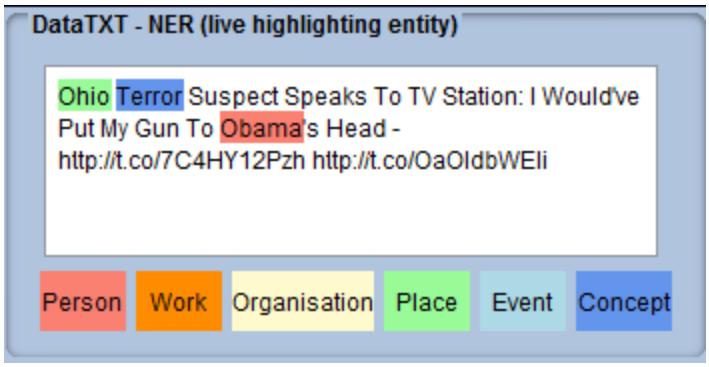


Figura 47: Particolare sezione, della finestra di interfaccia relativa al secondo modulo *software*, per la classificazione dei *topic*.

In particolare in [**Figura 47**] si notano 6 categorie, le quali sono state direttamente estratte dalle *"category" di DBPedia*, ad eccezione di ***concept***, la quale è stata creata accumulando tutte le restanti category.

Per far emergere maggiormente i topic estratti, all'interno del testo del tweet, è stato realizzato un componente che permette la "evidenziazione" di tali parole, all'interno del messaggio (rispettando fedelmente i colori indicati nella legenda di category sottostante al messaggio stesso).

---

2. La sezione **(5)**, invece permette la rappresentazione di tutte le restanti informazioni reperite ed abbinate, ai singoli *topic* estratti. Da queste informazioni si può ottenere, ad esempio: la foto del soggetto del *topic*, gli *url* a *DBPedia* e *Wikipedia* (cliccabili direttamente dall'interfaccia realizzata), le *label alternative* per esprimere lo stesso concetto, l'*abstract wikipedia,* una breve descrizione, ed una serie di altre cospicue informazioni.

4.5 ANALISI DEI CONTENUTI

Il modulo *software* per l'**Analisi dei Contenuti**, presenta come i due precedenti, una finestra dell'interfaccia ad esso dedicata. Tale finestra in questo caso, è stata opportunamente suddivisa in tre sotto-moduli che descrivono tre differenti macro-funzionalità implementate.

Di seguito verranno introdotte tutte le singole funzionalità realizzate, accompagnate dalla relativa interfaccia grafica e dalle parti di codice più rilevanti.

4.5.1 Modulo Software e Package Implementati

Per descrivere tutte le funzionalità di questo terzo modulo, occorre innanzitutto presentare i nuovi *package* che sono stati introdotti.

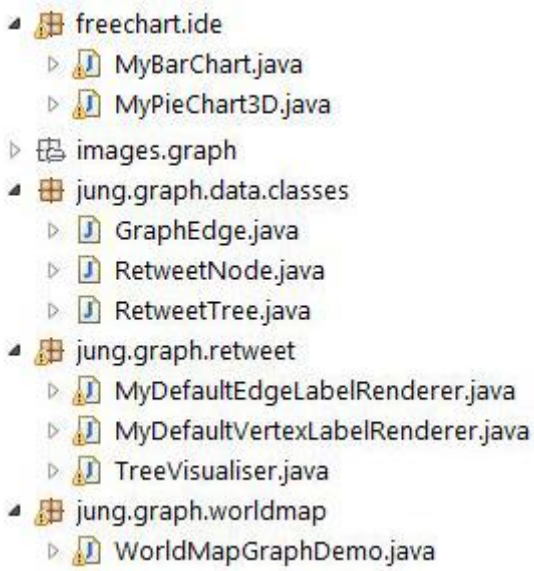


Figura 48: Elenco dei *package* relativi ai *namespace (jung.*)* e *(freechart.*)*.

I *package* presenti in [**Figura 48**] permettono di esprimere le seguenti funzionalità:

- ***freechart.ide***, contiene due classi realizzate appositamente, per fornire l'interfaccia di componenti grafici per la rappresentazione di dati attraverso diagrammi a torte e diagrammi a barre.

- ***jung.graph.data.classes***, è composto dalle classi di support alla rappresentazione del *"grafo dei retweet"* (o per meglio dire, di uno dei *"sotto-alberi"* del grafo) e degli archi di collegamento presenti tra i differenti nodi (queste classi verranno meglio spiegate, all'interno dei paragrafi successivi, [Paragrafo 4.5.3] e [Paragrafo 4.5.4].
- ***jung.graph.retweet***, presenta la classe che permette di visualizzare il grafo, e i relativi componenti grafici annessi.
- ***jung.graph.worldmap***, presenta il visualizzatore della mappa del mondo, con le relative funzionalità di rappresentazione dei nodi, e degli archi di collegamento tra di essi.

4.5.2 *Topic Analysis*

Per la prima *macro-funzionalità* di analisi e rappresentazione dei dati, relativi ai *topic* estratti, viene presentata di seguito la struttura dell'interfaccia.

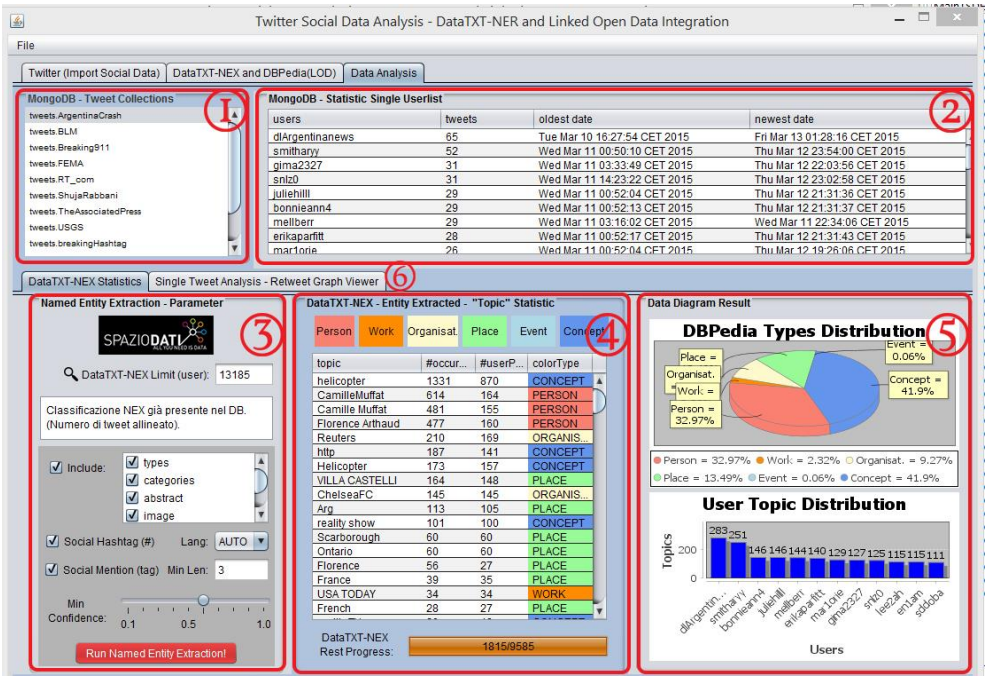


Figura 49: Finestra dell'interfaccia principale, dedicata al *terzo modulo software, per l'**analisi dei contenuti*** (e la presentazione dei grafici sui *topic*).

Osservando [**Figura 49**], che rappresenta uno *screenshot* della finestra, ottenuto durante l'esecuzione del programma, si possono notare 5 aree differenti, appositamente delimitate:

1. La sezione **(1)** presenta una lista contenente tutte le *collection* di *tweet*, reperite con il modulo uno, e memorizzate all'interno del *database*.
2. In sezione **(2)** viene presentata una tabella appositamente modificata, che presenta una vista aggregata dei *tweet* contenuti all'interno delle *collection*, suddivisi per utente di appartenenza, e con annesse le date relative al *tweet* più lontano nel tempo, e quello più recente (per dare subito un'idea immediata all'utilizzatore, riguardo al periodo complessivo di estrazione degli *status*). Inoltre le righe sono state appositamente ordinate in ordine decrescente secondo la colonna contenente il totale dei *tweet* per singolo utente.
3. La sezione **(3)** presenta nuovamente i componenti grafici relativi alla parametrizzazione delle richieste al servizio di *API REST di DataTXT-NEX*, con il relativo contatore dei permessi giornalieri. Questo contatore sarà sempre mantenuto aggiornato ed allineato con i server di *Dandelion.eu*, analogamente a quanto fatto per le *API di Twitter*, in modo da evitare di sforare il limite giornaliero concesso. Nella parte centrale di questa sezione è presente una nuova casella di testo, che in maniera trasparente, comunica all'utente se la *collection* selezionata al punto (1), sia o meno stata precedentemente classificata con *DataTXT* e se ci sia una classificazione dei *topic* per ogni *tweet* di tale *collection*.
4. La sezione **(4)** viene aggiornata solamente al completamento delle richieste di classificazione, lanciate dal punto (3). Nel caso in cui, localmente sia già presente una classificazione completa della *collection* selezionata al punto (1), questo avverrà immediatamente, in caso contrario occorrerà aspettare che tutte le richieste effettuate al servizio di *API* in questione, ricevano risposta e vengano memorizzate nel *database*. Nella tabella, anch'essa realizzata, andando ad estendere la classe *AbstractTableModel* di Java, vengono presentate quattro colonne: il ***topic*** estratto, il contatore delle **occorrenze** di tale *topic* all'interno di tutta la *collection*, il numero di **utenti** che hanno pubblicato *tweet* contenenti lo stesso *topic*, e infine la **categoria** di appartenenza. Per enfatizzare i risultati, le righe della tabella sono state ordinate in ordine decrescente in base alla colonna delle occorrenze.

5. Nella sezione **(5)** vengono **generati automaticamente** due diagramma per la visualizzazione di dati statistici relativi al processo di estrazione di *topic* appena concluso.
    a. Nel primo **diagramma a torta**, generato con una personalizzazione dei diagrammi a torta forniti dalla libreria *JFreeChart*, viene mostrata la **distribuzione delle categorie** di appartenenza dei *topic*. In questa rappresentazione vengono mantenute le colorazioni fin qui utilizzate e vengono mostrate delle etichette con i valori percentuali, ottenuti analizzando la classificazione sull'intero campione.
    b. Nel secondo **diagramma a barre** (o istogramma), sempre realizzato personalizzando le classi base della libreria *JFreeChart*, viene mostrato un numero limitato di utenti (per questioni di spazio di rappresentazione) ottenuti selezionando i primi 12, dal totale degli utenti che hanno condiviso *status* all'interno della *collection*, avendoli precedentemente ordinati in maniera decrescente, secondo il numero di occorrenze di *topic* citati.

    Il primo diagramma può servire per avere un'idea della distribuzione delle categorie di *topic* all'interno della *collection*, in modo da verificare quali siano gli argomenti più condivisi, rispetto alla *query* imposta in fase di estrazione dei *tweet da Twitter*.

    Il secondo diagramma può servire per fare una stima degli utenti più attivi all'interno del bacino di utenti coinvolto da tale *query*, e per avere un ordine di grandezza, sul numero massimo di condivisione di *topic* dei singoli utenti.

4.5.3 *Retweet Graph Analysis*

Per la seconda *macro-funzionalità* di analisi dei dati, occorre cliccare l'etichetta al punto **(6)** in [**Figura 49**]. In questo modo si apre una nuova scheda dell'applicazione dedicata allo studio delle ***ricondivisioni dei topic.***

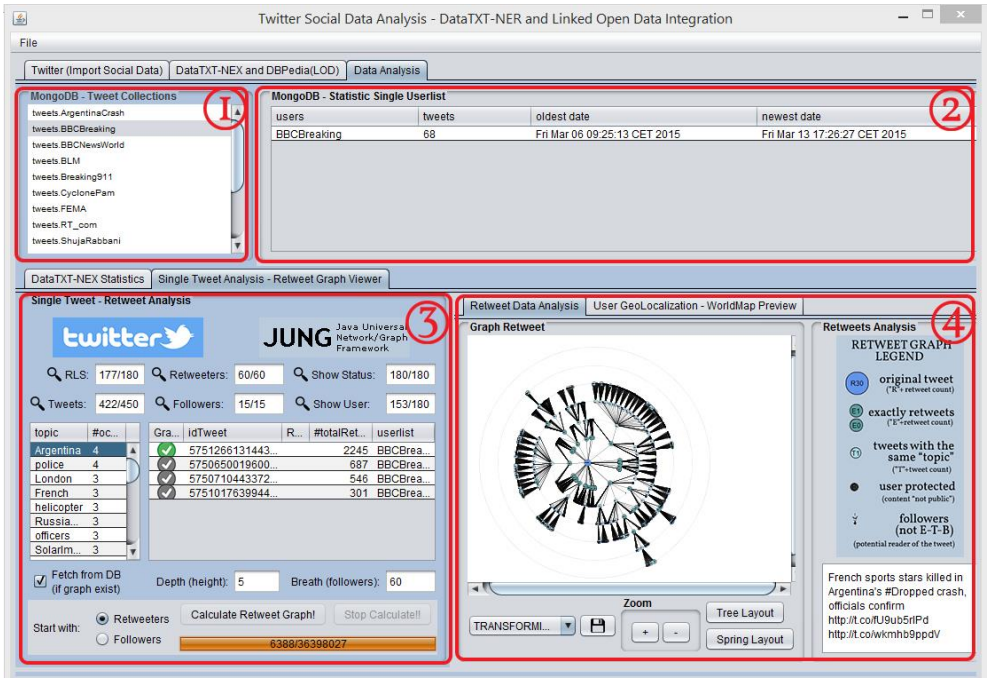


Figura 50: Finestra dell'interfaccia principale, dedicata al terzo modulo software, per l'analisi dei contenuti (e il calcolo del ***Grafo dei Retweet***).

In [**Figura 50**] si possono notare 4 sezioni evidenziate:

1. Le prime due sezioni **(1)** e **(2)** sono le stesse descritte nell'interfaccia precedente [**Figura 49**], e hanno ancora la funzionalità di selezione della *collection* e di presentazione dei dati complessivi relativi ad essa.
2. Le sezioni **(3)** e **(4)** sono dedicate alla parametrizzazione e presentazione dei dati relativi al calcolo del ***"Grafo dei Retweet"***.
   In (4) viene visualizzato il *rate-limit*-status relativo alle richieste che verranno utilizzate per la generazione del grafo, inoltre vengono presentate due tabelle opportunamente ordinate secondo l'occorrenza dei diversi *topic*, e secondo il numero di *ricondivisioni* (calcolato e fornito da *Twitter*). Nella parte inferiore della sezione, sono presenti componenti per la parametrizzazione della richiesta di calcolo del grafo, che verranno discussi nella parte seguente del paragrafo.
   In (5) vengono presentati dei componenti grafici, della libreria *Jung*, opportunamente modificati per la visualizzazione del grafo ottenuto dal processo di computazione. Nella parte laterale viene mostrata una *legenda* per l'interpretazione di tale grafico.

Scopo della sezione

Tra le tecniche di ***Social Data Analysis*** *(descritte brevemente nel capitolo introduttivo, [Paragrafo 1.5]) sono presenti diverse tipologie di analisi legate alla* ***Information Diffusion*** e ai fattori che influenzano la propagazione dei messaggi sulla rete (*Community Structure, Social Convergence, …).*

In questa sezione, si è deciso di cercare di ricostruire il GRAFO DEI RETWEET, ovvero il grafo contenente tutte le condivisioni di uno stesso tweet d'origine.

Twitter appiattisce la catena di condivisioni, in quanto, ogni qualvolta si va a ricondividere uno stesso status, sia esso presente sulla pagina originale in cui è stato creato, o sulla pagina di qualche altro utente che lo ha condiviso; tutti i nuovi retweet vengono fatti risalire fino al primo tweet, come se essi fossero tutti stati ricondivisi a partire dalla pagina d'origine.

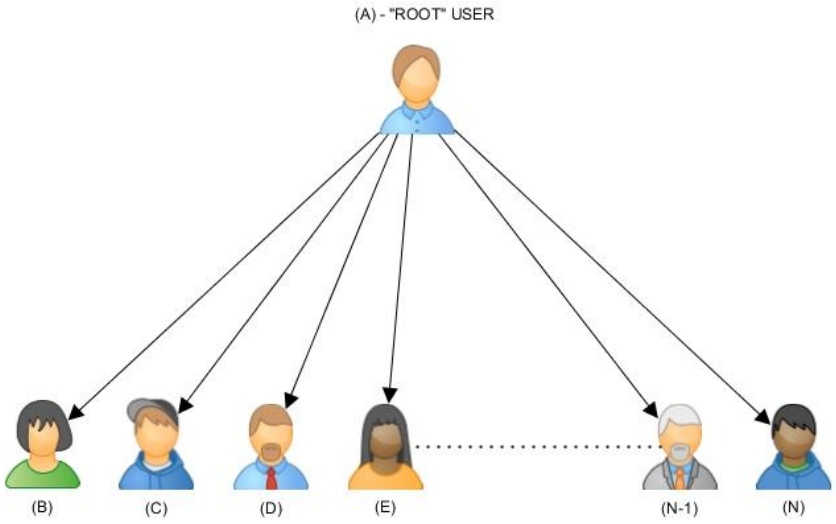


Figura 51: *Rappresentazione semplificata di uno degli Alberi di Retweet*, "appiattito" dal sistema di condivisioni *Twitter*.

Si viene a creare così una struttura ad **albero** *formata da un nodo radice (rappresentato dal* **tweet originale***) ed un solo livello di nodi figli (composti dalle foglie, che sono la totalità delle ricondivisioni, anche detti* **retweet***)* [**Figura 51**].

Questo fa sì che non sia possibile ricostruire, almeno direttamente, tutti i passaggi di condivisione di un tweet tra i diversi utenti.

Alcune aziende private forniscono servizi a pagamento per i loro utenti, nei quali riescono a tenere traccia del grafo delle ricondivisioni dei tweet dei loro clienti, in quanto a priori, implementano delle tecniche che comportano l'utilizzo di notificatori o listener, che tengono costantemente monitorato il comportamento di tale tweet sulla rete. Ad ogni condivisione, vengono notificati, in modo da poter salvare tale passaggio nelle loro strutture private.

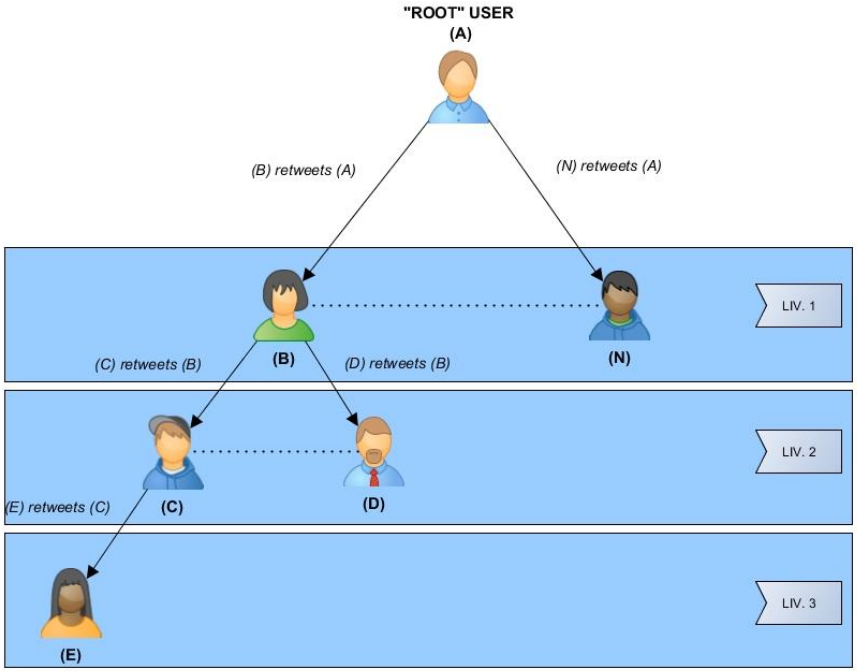


Figura 52: Rappresentazione semplificata di un *Albero di Retweet*, tenendo conto della catena di condivisioni "gerarchiche" dei *tweet*.

SCOPO di questa sezione è stato dunque quello di cercare di ricostruire tale grafo, trovando una "strategia alternativa".

La TECNICA IMPLEMENTATA si basa su un'IDEA semplice:

"Se un utente condivide un status, e non è follower della pagina che ha creato il tweet originale, allora con buona probabilità, avrà ricondiviso uno status che ha visto tra gli utenti che egli segue".

Questa è chiaramente una semplificazione, ma nella maggior parte dei casi, può essere assunta come verità, dal momento in cui, nella quasi totalità dei casi, gli utenti condividono tweet che essi vedono nelle loro bacheche (e quindi, che sono stati precedentemente condivisi o pubblicati dagli utenti che essi seguono).

FORMALIZZAZIONE DELL'IDEA PRECEDENTE:

Se un utente B ha condiviso uno status di un utente A (creatore di tale status), e un utente C (follower dell'utente B, e non di A) condivide nuovamente tale status, si può assumere che egli l'abbia ricondiviso dalla pagina di B.

OSSERVAZIONI:

1. *L'implementazione di tale tecnica, non permetterà di ricostruire l'intero grafo delle condivisioni, ma più semplicemente, di una sotto-parte di esso. In particolare verrà ricostruito uno degli* **alberi** *che parte dal* **nodo radice (tweet originale**) *e scende fino ad un livello di foglie specificato.*
2. *L'albero che si otterrà da tale implementazione, non potrà quindi essere considerato completamente corretto, dal momento in cui esso è stato generato da una forte assunzione. Potrà comunque servire per fare analisi relative all'entità delle condivisioni e al bacino di utenti toccato da tali messaggi.*
3. *Vista l'assunzione iniziale, si contempla la possibilità di:*
    a. *Perdere alcuni utenti che non hanno condiviso il messaggio dalla pagina di qualche loro "following", ma che hanno raggiunto direttamente (o per ricerca), la notizia sulla pagina dell'utente creatore.*
    b. *Prendere la ricondivisione "sbagliata" se, riprendendo l'esempio formalizzato e aggiungendo un utente D (agli utenti già presenti A,B,C), egli dovesse ricondividere lo status, avendo tra i suoi "following" sia l'utente B che l'utente C.*
4. *L'albero che si otterrà, considerando un valore di BREATH (ampiezza) di K, e un valore di DEPTH (altezza) di H, potrà avere un massimo di nodi ottenibile dalla seguente formula:*

$$\#nodi = \left[\frac{k^{h+1}-1}{k-1}\right]$$

Tale formula è stata reperita da Wikipedia, nel contesto dei "K-ary Tree" al quale questo albero fa riferimento.

PSEUDOCODIFICA (dell'algoritmo ideato):

i. *Preso l'UTENTE ROOT (creatore del tweet radice).*
ii. *Per ogni FOLLOWER dell'UTENTE, si verifichi se:*
    a. *Ha condiviso esattamente lo stesso tweet.*
    b. *Non ha condiviso lo stesso tweet, ma ha parlato dello stesso topic.*
    c. *Non ha effettuato né il caso (a) né il caso (b), inoltre egli è un utente "protetto" (ovvero possiede condizioni di privarcy che potrebbero impedire la leggibilità di tale status).*
    d. *Non ha effettuato nessuna delle precedenti, né (a), né (b), né (c).*
iii. *Gli utenti identificati nei punti (a) e (b), possono essere considerati POTENZIALI CONDIVISORI; per questi si andrà a ripetere il punto (ii).*
iv. *Per gli utenti identificati nei punti (c) e (d), si può solamente dire che essi sono dei POTENZIALI LETTORI di tale status.*

Codice 12: (Pseudocodifica) del codice relativo all'implementazione dell'algoritmo per la generazione dell'albero delle ricondivisioni.

Di seguito viene presentato il codice relativo al metodo ***SearchFollowersRetweetsGraph***, relativo alla classe *TwQueryTask*, presentata in [Paragrafo 4.3.6].

TwQueryTask(*searchFollowersRetweetsGraph*)**.java**

```
// QUERY TIPO 4: SEARCH LISTA RETWEETS
private void searchFollowersRetweetsGraph() {
try {
    timeRetweetGraphCalulateStarted = new SimpleDateFormat
        ("yyyy-MM-dd hh-mm-ss").format(new Date());

    univokeExactlyRetweetFakeId = -1;
    userAnalyzedCount = 0;

        availableTaskTillRLS.get(Util.RATE_LIMIT_STATUS_SHOW_STATUS).
                acquire();

    controlloRLS(Util.RATE_LIMIT_STATUS_SHOW_STATUS);
    Status rootTweet = twContext.getTwitter().showStatus(
                querySearchRetweets.getIdRootTweet());
```

```
updateSingleRateLimitStatus(
    rootTweet.getRateLimitStatus(),
    Util.RATE_LIMIT_STATUS_SHOW_STATUS);

appendRowToStringBuilder(
    "TWEET SOURCE (ROOT): "+rootTweet.getText());
appendRowToStringBuilder("RETWEETS:"+rootTweet.getRetweetCount());
appendRowToStringBuilder("\n");
    String startsFrom =
    (querySearchRetweets.isStartWithRetweetersFirst()) ?
    "(STARTS FROM RETWEETERS'IDS)" : "(STARTS FROM FOLLOWERS'IDS)";
appendRowToStringBuilder("DEPTH:
    "+querySearchRetweets.getMaxDepthRetweeters()+"   BREATH:
    "+querySearchRetweets.getMaxBreathRetweeters()+"   "+startsFrom);
// Creo nodo radice dell'albero dei retweet
tree = new RetweetTree(rootTweet.getUser().getId(),
    Util.NODE_ROOT_LABEL+rootTweet.getRetweetCount());
// Inserisco UTENTE ROOT nella lista GraphUser
if (rootTweet.getUser() != null) {
     retweetersInfos.add(getGraphUserInfoFrom(rootTweet.getUser()));
}
// AGGIORNO BARRA TF
informTwitterFrameUpdateProgressBarUserRetweetGraph(false);
userAnalyzedCount++;

int currentDepthLevel = 0;
/*** RICERCO TUTTI I FOLLOWER/RETWEETERS DELL'UTENTE CHE HA FATTO
    PARTIRE IL "ROOTTWEET" ***/
List<Long> followersIDs;
if (!querySearchRetweets.isStartWithRetweetersFirst()) {
     followersIDs = getFollowersIdsOf(rootTweet.getUser().getId(),
    querySearchRetweets.getMaxBreathRetweeters());
}else {
     followersIDs = new ArrayList<Long>();
    availableTaskTillRLS.get(Util.RATE_LIMIT_STATUS_RETWEETERS_IDS).ac
    quire();
     controlloRLS(Util.RATE_LIMIT_STATUS_RETWEETERS_IDS);
     IDs retwIDs = twContext.getTwitter().
            getRetweeterIds(rootTweet.getId(), -1);
     updateSingleRateLimitStatus(retwIDs.getRateLimitStatus(),
            Util.RATE_LIMIT_STATUS_RETWEETERS_IDS);
     boolean endRetweeters = false;

     while ((twContext.getLastRateLimitStatusChecked().get(
            Util.RATE_LIMIT_STATUS_RETWEETERS_IDS).getRemaining() >=
            Util.LIMIT_MINIMUM_RATE_LIMIT_STATUS)
              &&(!endRetweeters)) {

         LOOP: for (Long id : retwIDs.getIDs()) {
              followersIDs.add(id);
              if (followersIDs.size() >=
                     querySearchRetweets.getMaxBreathRetweeters()) {
                  endRetweeters = true;
                  break LOOP;
              }
         }
         if ((retwIDs.hasNext())&&(!endRetweeters)) {
              long nextCursor = retwIDs.getNextCursor();
              availableTaskTillRLS.get(
                     Util.RATE_LIMIT_STATUS_RETWEETERS_IDS).acquire();
              controlloRLS(Util.RATE_LIMIT_STATUS_RETWEETERS_IDS);
              retwIDs = twContext.getTwitter().getRetweeterIds(
                     rootTweet.getId(), nextCursor);
              updateSingleRateLimitStatus(
                     retwIDs.getRateLimitStatus(),
                     Util.RATE_LIMIT_STATUS_RETWEETERS_IDS);
         }else {
              endRetweeters = true;
         }
     }
}
currentDepthLevel++;
```

```
    /*** RICERCO IL RETWEET NELLA TIMELINE DEGLI UTENTI (FOLLOWER)
       PRECEDENTEMENTI REPERITI ******************************/
    for (int i = 0; i < followersIDs.size(); i++) {
        recursiveSearchOnFollower(rootTweet, currentDepthLevel,
       rootTweet.getUser().getId(), rootTweet.getId(),
       followersIDs.get(i));
    }
// AGGIORNO BARRA TF (RAGGIUNGO IL MASSIMO ANCHE SE
       L'ALBERO NON ERA PIENO = TRUE)
    informTwitterFrameUpdateProgressBarUserRetweetGraph(true);
    informTwitterFrameEndSearchRetweets();
} catch (TwitterException e) {
    System.out.println(e.getMessage());
    launchPopupNotificationErrorQuery(e.getMessage());
} catch (InterruptedException e) {
    launchPopupNotificationInterruptedQuery();
    System.out.println(e.getMessage());
}
}

private void recursiveSearchOnFollower(Status rootTweet, int
currentDepthLevel, Long parentUserId, Long parentTweetId, long userID){
try {
    // Se il FOLLOWER ha condiviso il RETWEET
       (lo AGGIUNGO all'albero, e CERCO NEI SUOI FOLLOWERS){
    availableTaskTillRLS.get(Util.RATE_LIMIT_STATUS_SHOW_USER).acquire();
    controlloRLS(Util.RATE_LIMIT_STATUS_SHOW_USER);
    User user = twContext.getTwitter().showUser(userID);
    if (user != null) {
        retweetersInfos.add(getGraphUserInfoFrom(user));
        // AGGIORNO BARRA TF
        informTwitterFrameUpdateProgressBarUserRetweetGraph(false);
        userAnalyzedCount++;
    }
    // Aggiorno RLS (RATE_LIMIT_STATUS_SHOW_USER)
    updateSingleRateLimitStatus(user.getRateLimitStatus(),
       Util.RATE_LIMIT_STATUS_SHOW_USER);
    // Cerco il RETWEET all'interno della TIMELINE dei FOLLOWER
    /*** QUERY ESATTA - RETWEET ***/
    availableTaskTillRLS.get(
       Util.RATE_LIMIT_STATUS_SEARCH_TWEETS).acquire();
    controlloRLS(Util.RATE_LIMIT_STATUS_SEARCH_TWEETS);
    // Se sono partito da RETWEET (QUERY NON AGGIUNGO "RT @userCorrente"
    Query query;
    if (rootTweet.getText().substring(0, 2).equals("RT")) {
        query = new Query(rootTweet.getText()+"
               from:"+user.getScreenName()+" include:retweets");
    }else {
        query = new Query("RT @"+rootTweet.getUser().getScreenName()+": "
              +rootTweet.getText()+" from:"+user.getScreenName()+"
               include:retweets");
    }
    QueryResult queryResult = twContext.getTwitter().search(query);
    // Aggiorno RLS (RATE LIMIT STATUS SEARCH TWEETS)
    updateSingleRateLimitStatus(queryResult.getRateLimitStatus(),
       Util.RATE_LIMIT_STATUS_SEARCH_TWEETS);
    int tweetCount = queryResult.getTweets().size();

    // Faccio la QUERY per TOPIC (T) SOLAMENTE se non ho trovato quella
       ESATTA (E) <---- RISPARMIO RLS SEARCH-TWEETS
    int tweetTopicCount = 0;
    QueryResult queryTopicResult = null;
    if (tweetCount <= 0) {
        /*** QUERY NON ESATTA - CERCO SOLAMENTE STESSO TOPIC ***/
        availableTaskTillRLS.get(
               Util.RATE_LIMIT_STATUS_SEARCH_TWEETS).acquire();
        controlloRLS(Util.RATE_LIMIT_STATUS_SEARCH_TWEETS);
        Query queryTopic = new
               Query(querySearchRetweets.getTopicRetweet()+"
               from:"+user.getScreenName()+" include:retweets");
        queryTopicResult = twContext.getTwitter().search(queryTopic);
        // Aggiorno RLS (RATE LIMIT STATUS SEARCH TWEETS)
        updateSingleRateLimitStatus(
```

```
                queryTopicResult.getRateLimitStatus(),
                Util.RATE_LIMIT_STATUS_SEARCH_TWEETS);
            tweetTopicCount = queryTopicResult.getTweets().size();
        }
        /*** SE L'UTENTE HA EFFETTUATO IL RETWEET DELLO STESSO STATUS ***/
        if (((Long.compare(rootTweet.getId(), parentTweetId) == 0)
            &&(querySearchRetweets.isStartWithRetweetersFirst())) // RETWEETS
            ||(tweetCount > 0)) {
            Long retweetId;
            if (tweetCount > 0) {
                Status tweet = queryResult.getTweets().get(0);
                retweetId = tweet.getId();
            }else {
                retweetId = univokeExactlyRetweetFakeId;
                univokeExactlyRetweetFakeId--;
            }
            // AGGIUNGO NODO
            tree.addNode(parentUserId, user.getId(),
                   Util.NODE_EXACTLY_RETWEET_LABEL+tweetCount);
            currentDepthLevel++;
            // Se non ho ragggiunto l'ultimo livello di FOLLOWER
            if ((currentDepthLevel <
                   querySearchRetweets.getMaxDepthRetweeters())
                   &&(!twContext.getStopRunningRetweetSearch())) {
                List<Long> followersIDs = getFollowersIdsOf(
                   user.getId(),
                   querySearchRetweets.getMaxBreathRetweeters());
                /*** RICERCO RETWEET IN TIMELINE (FOLLOWER) REPERITI ****/
                for (int i = 0; i < followersIDs.size(); i++) {
                    recursiveSearchOnFollower(rootTweet, currentDepthLevel,
                          user.getId(), retweetId, followersIDs.get(i));
                }
            }
        }
        /*** SE L'UTENTE HA CONDIVISO UNO/PIU' STATUS CON STESSO "TOPIC" ***/
        else if (tweetTopicCount > 0) {
            Status tweet = queryTopicResult.getTweets().get(0);
            // AGGIUNGO NODO
            tree.addNode(parentUserId, user.getId(),
Util.NODE_TOPIC_RETWEET_LABEL+tweetTopicCount);
            currentDepthLevel++;
            // Se non ho ragggiunto l'ultimo livello di FOLLOWER
            if ((currentDepthLevel <
                   querySearchRetweets.getMaxDepthRetweeters())&&(!twContext.
                   getStopRunningRetweetSearch())) {
                List<Long> followersIDs = getFollowersIdsOf(user.getId(),
                   querySearchRetweets.getMaxBreathRetweeters());
                /* RICERCO IL RETWEET NELLA TIMELINE DEI FOLLOWER REPERITI */
                for (int i = 0; i < followersIDs.size(); i++) {
                    recursiveSearchOnFollower(rootTweet, currentDepthLevel,
                          user.getId(), tweet.getId(), followersIDs.get(i));
                }
            }
        }
        /* SE NON HO NESSUNA DELLE PRECEDENTI, CONTROLLO SE NON E' PUBBLICO*/
        else if (user.isProtected()) {
            tree.addNode(parentUserId, user.getId(),
                   Util.NODE_NOT_PUBLIC_LABEL);
        }
        /* UTENTE DI CUI NON SI VEDE ALCUN TWEET (NESSUNA DELLE PRECEDENTI)*/
        else {
            tree.addNode(parentUserId, user.getId(), Util.NODE_ANY_TWEET);
        }
    } catch (TwitterException e) {
        launchPopupNotificationErrorQuery(e.getMessage());
    } catch (InterruptedException e) {
        launchPopupNotificationInterruptedQuery();
    }
```

Codice 13: (Porzione) Codice relativo all'implementazione del metodo *SearchFollowersRetweetsGraph* e dei metodi annessi, all'interno di *TwQueryTask.java*

OSSERVAZIONI:

1. *Nella reale implementazione, è stata contemplata la possibilità di* **partire** *sia dalla totalità dei* ***FOLLOWER****, che dagli utenti restituiti dal metodo* ***GET-RETWEETERS'IDS*** *(di Twitter) il quale permette di ottenere gli ultimi 100 RETWEETER più recenti.*
2. *È stata concessa la possibilità di imporre un massimo valore di* ***BREATH*** *(ampiezza), ovvero un valore limite per il numero di FOLLOWERS/RETWEETERS analizzati ad ogni nodo.*
3. *È stata concessa la possibilità di imporre un valore di* ***DEPTH*** *(profondità), ovvero un valore massimo per l'altezza dell'albero.*
4. *È stata concessa la possibilità di* **terminare la procedura** *di calcolo dall'esterno (gestendo un evento generato alla pressione di un apposito pulsante sull'interfaccia) andando a concludere le computazioni avviate, e non andando più ad aprire nuove computazioni.*
5. *È stata realizzata una* **struttura ad-hoc** *per la memorizzazione della struttura dell'albero, ed è stato realizzato un algoritmo ricorsivo per la ricerca dei potenziali ricondivisori, tra gli utenti figli dei singoli nodi.*

4.5.4 *User GeoLocation Analysis*

Per la terza *macro-funzionalità* viene innanzitutto presentata l'interfaccia:

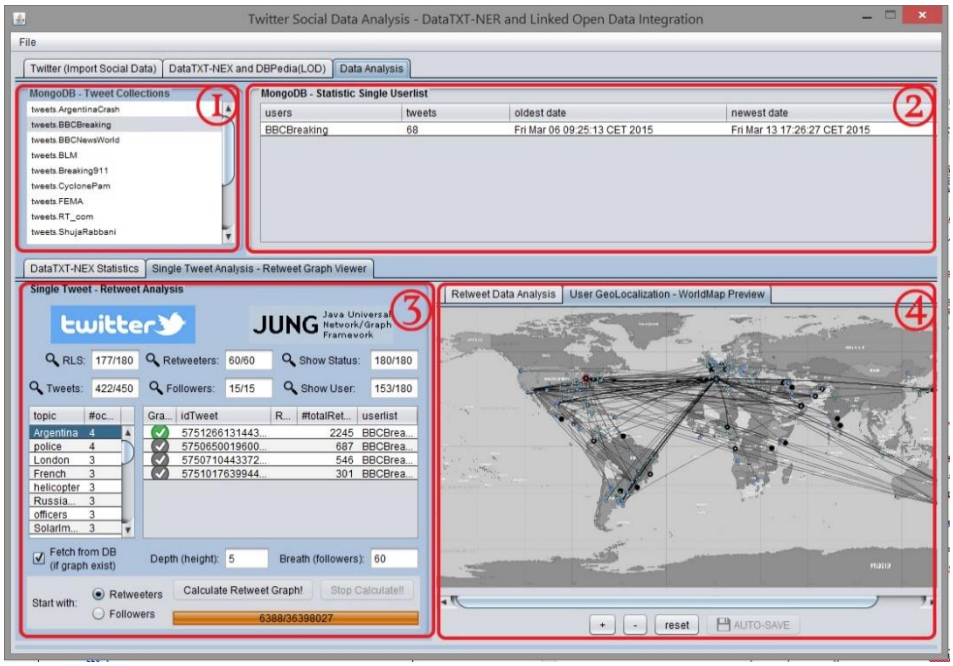


Figura 53: Finestra dell'interfaccia principale, dedicata al terzo modulo software, per l'analisi dei contenuti (e la presentazione della ***GeoLocation Worldmap***).

Osservando [**Figura 53**] si notano 4 sezioni evidenziate:

1. Le sezioni **(1)**, **(2)** e **(3)** sono relative a componenti già citati nei precedenti paragrafi, e mantengono le stesse funzionalità.
2. La sezione **(4)** presenta componenti grafici, opportunamente modificati, per permettere la visualizzazione di una mappa del mondo, con annessi i punti *geo-localizzati* di provenienza degli utenti coinvolti nel *"Grafo dei Retweet"*. Gli archi di collegamento visualizzati, rappresentano le vere e proprie condivisioni, tra i differenti nodi.

Scopo della sezione
Ottenere una mappa del mondo visitabile, contenente gli utenti coinvolti nella catena di ricondivisioni del tweet selezionato, sfruttando le informazioni reperite durante la computazione dello stesso Grafo dei Retweet.

OSSERVAZIONI:

1. *È stata realizzata una geo-localizzazione spaziale degli utenti, secondo quelle che sono le* ***località*** *da essi definite all'interno dei loro* ***profili Twitter****.*

   Questa scelta è stata confrontata in fase di progettazione, con l'analoga rappresentazione delle geo-localizzazioni dei tweet (che avrebbe sfruttato la posizione condivisa dagli stessi utenti in fase di pubblicazione degli stessi status).

   A favore di questa scelta, va tra le altre motivazioni, il fatto che si sia riscontrata una bassissima percentuale di tweet contenenti informazioni relative alla loro localizzazione (in quanto nella maggior parte dei casi, gli utenti disabilitano la connessione GPS per non rendere nota la loro posizione). Mentre è stata riscontrata una percentuale molto alta rispetto alla totalità dei casi analizzati, di profili contenenti informazioni relative alla località di provenienza dell'utente.

2. *Vista l'assunzione al punto (1) precedente, si contempla quindi la possibilità di:*
    a. *Reperire POTENZIALI RETWEETERS che non presentano un campo "località" del profilo Twitter completo o corretto.*
    b. *Localizzare gli utenti con un margine d'errore, dovuto ala precisione con cui li si sta posizionando sulla mappa del mondo. In particolare, viene sfruttata per la localizzazione, il nome della località (ottenuto dalle informazioni di profilo) e opportunamente disambiguato all'interno del* **database GeoNames,** *citato nel [Paragrafo 3.4.6], per ottenere le relative* **coordinate geo-spaziali** *(in termini di latitudine e longitudine).*
    c. *Non avere una mappa di utenti completa, in quanto l'assenza di alcune informazioni spaziali (nominate nel caso (a)) può portare alla mancanza di nodi, e quindi di archi di collegamento tra di essi.*

4.6 Moduli software trasversali

Di seguito vengono illustrati alcune classi "trasversali" al progetto, le quali sono state utilizzate all'interno di svariati metodi, appartenenti a *package* differenti. Queste classi consentono di richiamare operazioni generiche, componenti e proprietà statiche, utili in svariati contesti.

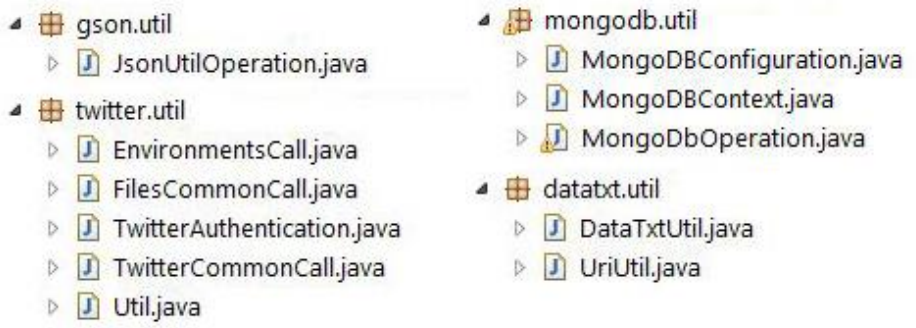


Figura 54: *Package* con relative classi *java*, in riferimento a *GSON (per le operazioni sul formato JSON)* e *twitter.util, mongodb.util, datatxt.util (per le operazioni e le proprietà inter-classe).*

4.6.1 Utility (GSON, Twitter, MongoDB, DataTXT)

Per ogni gruppo di *package* identificato da un apposito *namespace*, in relazione al contesto e allo scopo di utilizzo (esempio: *gson.* per le classi relative alle operazioni sul formato JSON, twitter.* per tutte le classi che compongono le interfacce ed i metodi di interazione con le API Twitter, …),* sono state create apposite classi di *"utility"* contenenti metodi ed attributi **statici**, in modo da permetterne un utilizzo massivo all'interno di tutte le altre classi, ottimizzando così la parametrizzazione e la riusabilità del codice.

4.6.2 Operazioni e Query su MongoDB

All'interno di tutti i paragrafi di questo Capitolo, si è sempre citata la memorizzazione di svariati dati all'interno del *database* (realizzato con tecnologia *MongoDB*), senza davvero entrare in dettaglio nell'implementazione di essa.

Questo paragrafo vuole quindi descrivere riassuntivamente le operazioni e le caratteristiche delle strutture utilizzate per effettuare le operazioni su *MongoDB (fetch dei dati, lettura/scrittura/modifica, ricerca e query, …).*

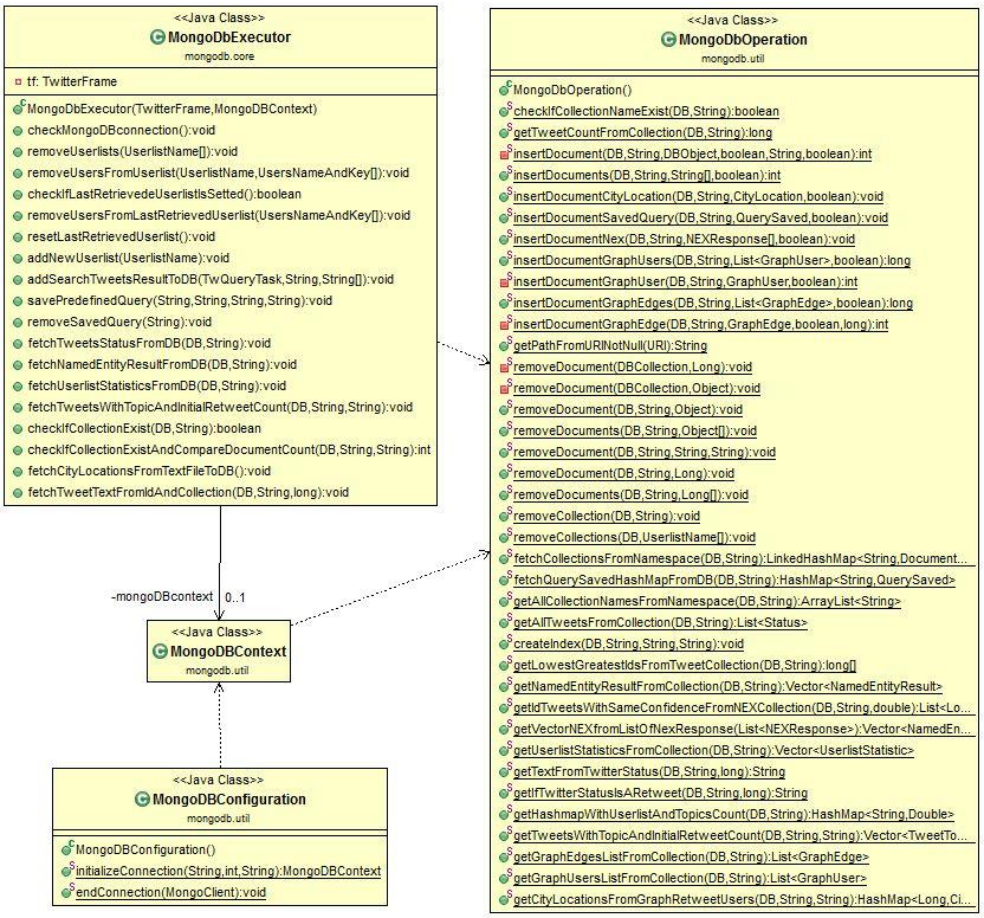


Figura 55: Diagramma UML (parziale) relativo alle classi realizzate per svolgere le operazioni sul *database* con tecnologia non-relazionale *MongoDB*.

Come si può notare in [**Figura 55**] all'interno della classe *MongoDBOperation*, sono state realizzate molteplici operazioni che operano su *collection* e *documents*, implementando in particolare operazioni di *fetch* e lettura dei dati, di scrittura o di modifica, e infine di ricerca, secondo chiave o campo, sfruttando le *query* offerte dal linguaggio interno a *MongoDB*.

Capitolo 5

Test e Analisi

In questo capitolo vengono presentati alcuni casi di studio affrontati, introducendo i contesti in cui gli eventi sono avvenuti, e presentando tabelle e grafici riassuntivi, che mostrano un effettivo utilizzo dell'applicazione sviluppata.

5.1 Introduzione

Per verificare l'effettivo funzionamento dell'applicazione, e per testare le sue potenzialità su dei casi reali, sono stati analizzati i più recenti avvenimenti critici, andando a selezionare quelli che hanno avuto maggior riscontro e più alta condivisione sui *social network*.

In particolare sono stati osservati i seguenti ambiti:

- *Disastri naturali, fenomeni ed eventi climatici critici.*
- *Attentati, minacce e rivendicazioni terroristiche*
- *Atti di guerriglia e scontri tra milizie.*
- *Incidenti ed avvenimenti di cronaca.*

Di seguito verranno riportati i risultati relativi a **3 casi di studio**, avvenuti negli ambiti sopra citati.

5.2 Caso di studio (1): Cyclone Pam (Australia)

Il primo caso di studio riguarda una calamità naturale, che si è abbattuta proprio in questi giorni, sulle isole australiane di Vanuatu.

L'EVENTO E LE SUE STIME

Il passaggio di quello che è stato soprannominato *"Ciclone Pam" (in lingua "**Cyclone Pam**")* ha provocato la morte di almeno 44 persone, e ferito altre 20. Il passaggio di tale ciclone, è avvenuto in data **13 Marzo 2015**, **nell'Arcipelago di Vanuatu**, distante circa 1750km dal continente Australiano. Secondo le stime, tale evento potrebbe aver colpito più di 260mila persone, e tutt'ora enti benefici e organizzazioni inernazionali, stanno intervenendo per aiutare nelle operazioni di recupero e salvataggio.

Cyclone Pam è stato classificato come un *"super-ciclone" di* **5^ categoria Saffir-Simpson**, i suoi venti hanno raggiunto velocità di oltre **300km/h** e dato il via ad un fenomeno denominato *"Storm Surge" (ovvero "onda di tempesta")*, il quale ha prodotto onde di altezza superiore ai **9 metri**, per alcune centinaia di miglia di distanza.

Figura 56: Notizia da *CNN Breaking News* relativa a *Cyclone Pam*, su *Twitter*

Estrazione dei Tweet e Classificazione

Dal processo di estrazione dei *tweets* con *query*: "cyclone pam" (senza aver specificato alcun utente), sono stati estratti **7861 tweet**.

Tutti questi *status* sono stato raccolti tra il **13 Marzo** e il **14 Marzo 2015**.

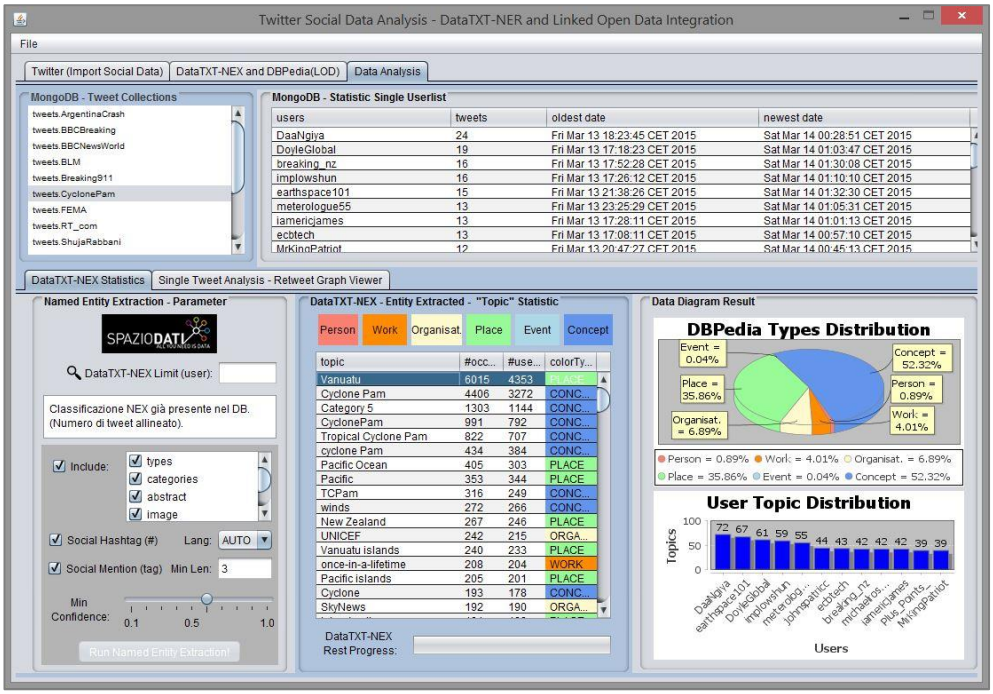


Figura 57: Classificazione dei *topic* estratti dalla totalità di *tweet* reperiti dalla *query "cyclone pam"*.

Osservando [**Figura 57**] si possono evincere le seguenti affermazioni:

- L'utente con maggior numero di *tweet* pubblicati (all'interno della finestra temporale analizzata) ha condiviso **24 tweet**.
- Sono presenti **15 utenti** con un numero di *tweet* superiore alla decina, rispetto alla **totalità di 5621 utenti**.
- Osservando la tabella *"User Topic Distribution"*, si nota che rispetto agli utenti totali di cardinalità 5621, solamente **10 utenti** presentano un quantitativo di *topic*, inerenti all'argomento (classificati con *confidence di almeno 0.6*), di almeno 40, all'interno delle loro pagine.
- Sono stati classificati **1211 *topic* differenti**, di cui **38 *topic*** con un numero di occorrenze superiore ai 100.

- I *topic* appartengono nel **52% dei casi a *Concept***, e nel **36% a *Place***, le restanti categorie possiedono percentuali inferiori al 10%.

Analisi della "Information Diffusion"

Per quanto riguarda lo studio dell'*Information Diffusion*, sono stati calcolati **l'*albero dei retweet*** (a partire dal nodo radice) e la ***mappa con gli utenti geo-localizzati.***

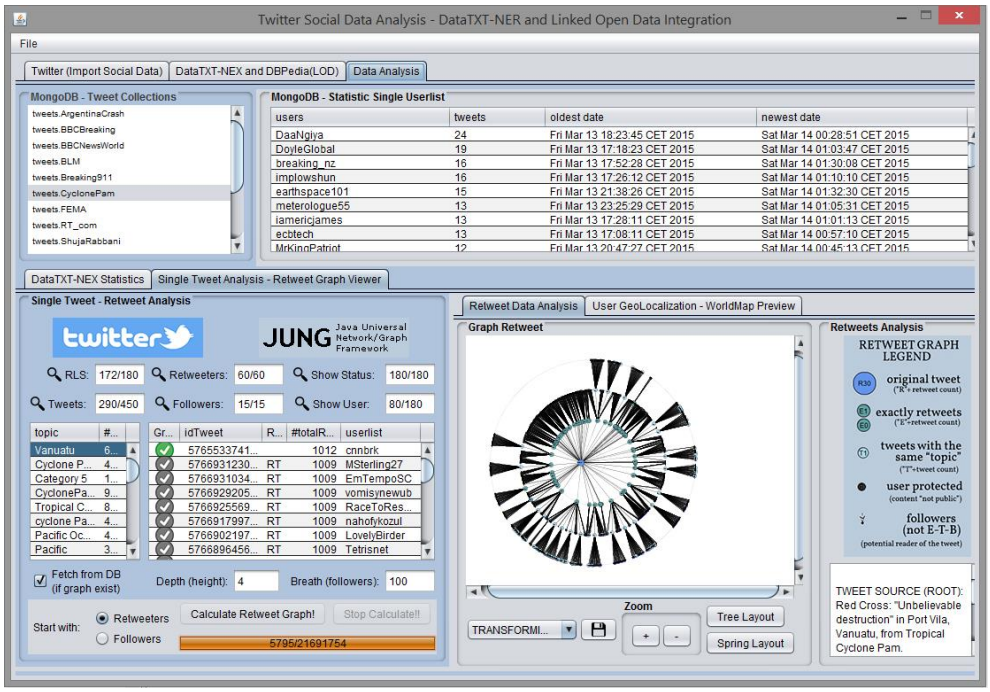


Figura 58: *Screenshot* dell'applicazione, al termine della computazione sul *topic* ***Vanuatu****, all'interno della collection* ***tweets.CyclonePam****.*

Partendo dal *tweet* in [**Figura 56**], al termine della *computazione* in [**Figura 58**], sono stati ottenuti i seguenti risultati:

- *Twitter* al momento dell'estrazione dello *status* indicava come numero totale di ricondivisioni per il *tweet* "radice" un valore di **1013**.
- *DATI DEL CALCOLO dell'Albero dei Retweet*:
  - *DEPTH (profondità): 3*
  - *BREATH (ampiezza): 100*
  - *Starts From (nodi di partenza): 100 retweeter.*

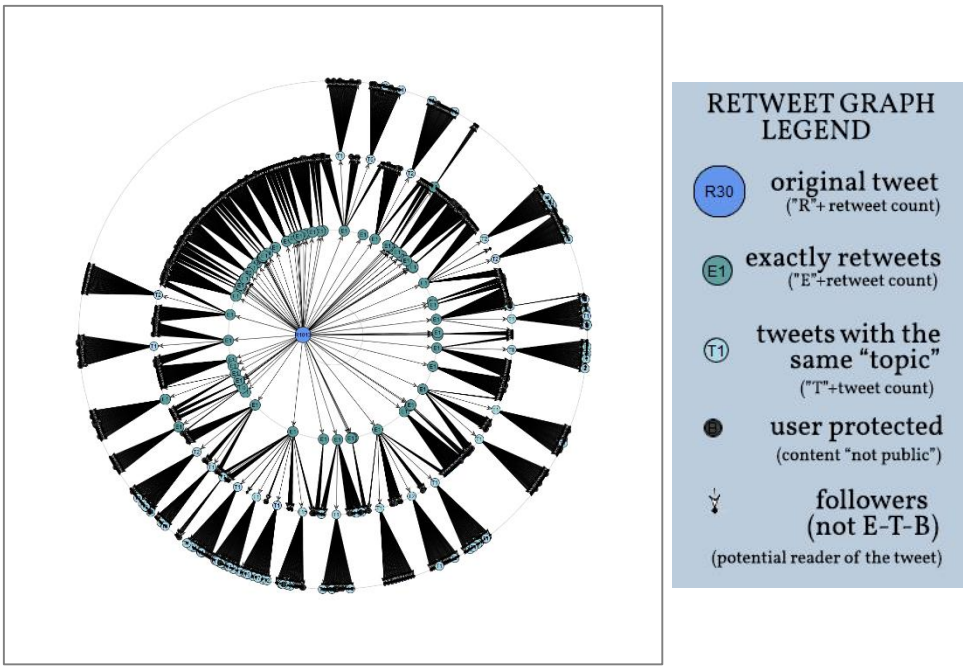


Figura 59: Albero di *Retweet* ottenuto partendo dal *tweet con topic* ***Vanuatu***.

Il processo di calcolo ha effettuato una computazione dell'ordine di:

- Ora d'inizio computazione: 14 Marzo 2015, ore 11.44.22 (hh/mm/ss).
- Ora di fine computazione: 14 Marzo 2015, ore 19.54.27 (hh/mm/ss).
- Ore di calcolo: **8.10.5 (hh/mm/ss).**
- Numero di utenti potenziali: **1.010.101** (calcolato secondo la formula presentata in [Paragrafo 4.5.3].
- Numero di utenti analizzati: **5796** (contando l'utente *root*).
  Di seguito espressi in dettaglio, secondo la *LEGENDA* in [**Figura 59**].
  - VERTICE (R): 1
  - VERTICI (E): 101
  - VERTICI (T): 107
  - VERTICI (A): 5407
  - VERTICI (B): 179
- Numero di utenti salvati nel *DB*: **5796**.
  - Numero di *"location"* trovate nel campo descrizione: **5126**.
  - Numero di "latitude/longitude" reperite ricercando tali *location* all'interno del *database GeoNames* locale: **4247**.

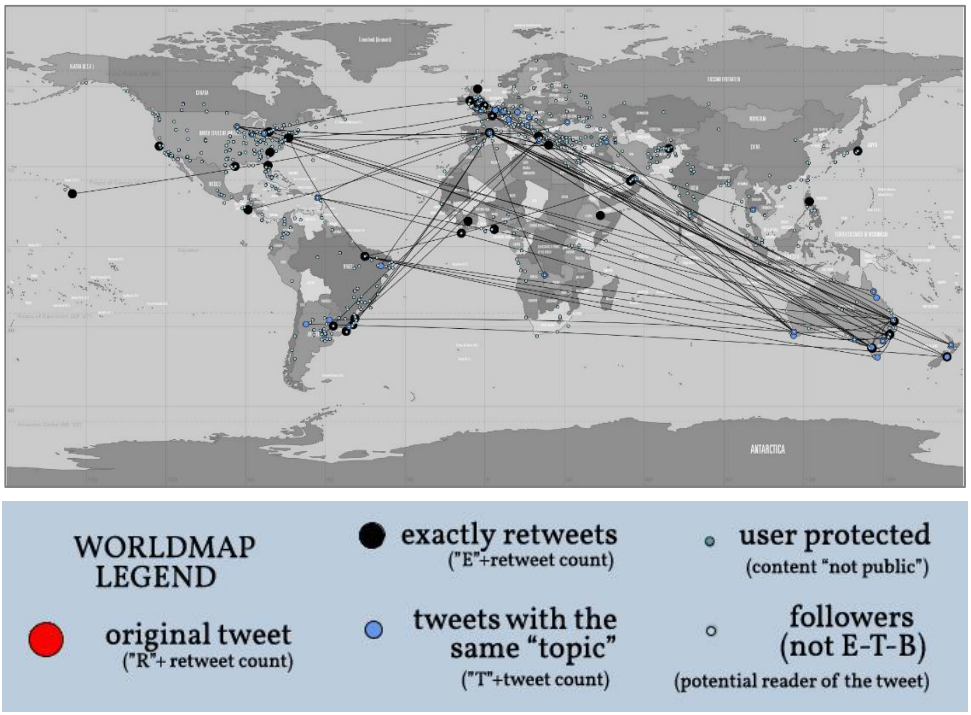


Figura 60: *WorldMap ottenuta analizzando gli utenti coinvolti nella condivisione.*

Nella *worldmap* sono stati rappresentati solamente gli *EDGE (archi)* relativi alle condivisioni di nodi (E) e (T), mentre sono stati rappresentati tutti i nodi con diverse colorazioni e dimensioni espresse in *LEGENDA*.

- Riepilogando sono stati rappresentati (**208** su 5795) archi.

OSSERVAZIONI:

- Anche ad una prima occhiata, è possibile notare come, a parire dalle zone limitrofe all'evento (in particolare quindi nelle coste Australiane) siano partiti numerosi *retweet*.
- Nonostante tali dati, fossero relativi alle prime ore, superato l'evento catastrofico, l'informazione si è presto diffusa nel continente Americano e in quello Europeo.
    - Sempre per tale motivo, essendo passate poche ore dall'evento, è possibile assumere che, il numero di *retweet* esatti (E) sia molto elevato, soprattutto se confrontato con altri casi di analisi.
    - E che rispetto ad altri casi confrontati, il numero di *retweet* di tipo (T) sia inferiore rispetto alla media.

5.3 Caso di studio (2): ISIS Threat Sphinx (Egypt)

Il secondo caso di studio riguarda una minaccia di attentato, che si è verificata proprio nelle ultime settimane, e che ha come soggetto l'organizzazione ISIS.

L'evento e le sue stime

La notizia riguarda una minaccia da parte di uno degli emissari dell'organizzazione, nota nel mondo come *ISIS*.

> *Un predicatore islamico del Kuwait ha esortato i sostenitori, minacciando la distruzione di antichi monumenti, quali la Sfinge e le Piramidi di Giza.*

Tale affermazione è stata mossa, non per l'importanza religiosa di questi monumenti, ma piuttosto per la loro importanza culturale e storica. Analogamente agli ultimi avvenimenti accaduti nel mondo, infatti le truppe *ISIS* hanno attaccato e distrutto numerosi centri di cultura e siti archeologici. Una delle motivazioni che spinge tale organizzazione terroristica è infatti quella di cercare di porre fine al "culto delle immagini".

Dopo l'attacco alla sede del giornale francese *"Charlie Hebdò"*, avvenuta nel Gennaio 2015, con la morte di numerose persone innocenti, e la forte rivendicazione di questa organizzazione terroristica, si è visto un incremento dell'utilizzo dei *social network*. In tale occasione è stato addirittura istituito un *hashtag* specifico (#JeSuisCharlie) che è diventato presto virale sulla rete.

Figura 61: Notizia da *RT.com* e da *ShujaRabbani* relativa alla suddetta notizia riguardante l'*ISIS*, prelevate da *Twitter*

Estrazione dei Tweet e Classificazione

Per questo particolare caso di studio, si è scelto di prendere un articolo proposto dal sito di informazione *www.rt.com*.

Da tale articolo, si è risaliti allo stesso *tweet* pubblicato dalla pagina *RT.com* su *Twitter*. Questa ha sicuramente ottenuto molte visualizzazioni, e condivisioni di tale *status*. Tra queste, si è poi deciso di selezionare uno dei principali *retweeter*, un giornalista e intervistatore famoso per le sue "provocazioni" relative al mondo Afghano. Egli infatti, ha sorprendetemente ottenuto un numero di condivisioni del suo status (prelevato direttamente dal sito internet rt.com e non dalla pagina twitter della stessa organizzazione), di molto maggiore rispetto alla prima.

Dal processo di estrazione dei *tweet* pubblicato dall'utente ***rt.com***, sono stati ottenuti **781 *status***, ottenuti tra il 4 Marzo e l'11 Marzo 2015.

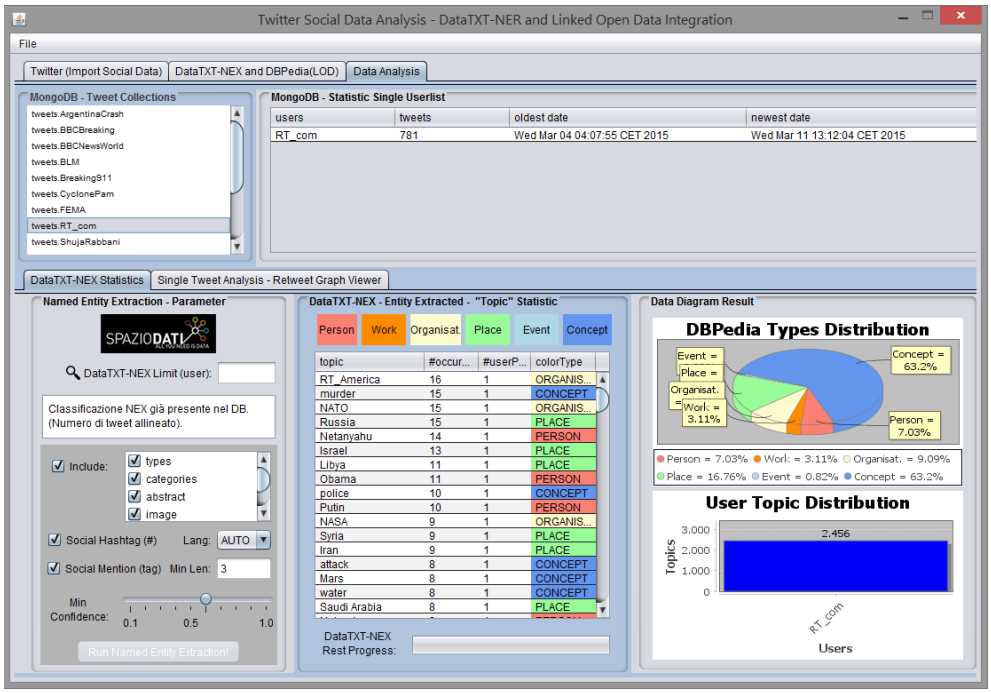


Figura 62: Classificazione dei *topic* estratti dalla totalità di *tweet* reperiti sulla pagina ***rt.com***

Dal secondo processo di estrazione dei *tweet* dalla pagina di **Shuja Rabbani**, sono stati ottenuti **221 *status***, tra il 4 Marzo e l'11 Marzo 2015.

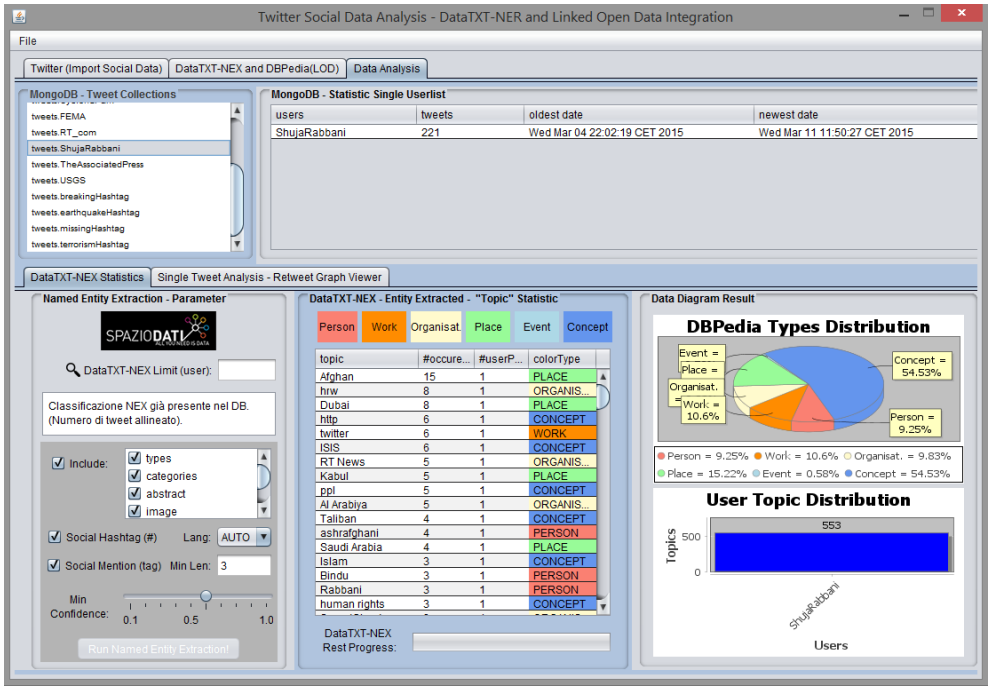


Figura 63: Classificazione dei topic estratti dalla totalità di tweet reperiti sulla pagina di Shuja Rabbani.

Confrontando le due figure [**Figura 62** e **Figura 63**] si possono evincere le seguenti affermazioni:

- Sono stati classificati **1281 *topic* differenti** per la pagina ***rt.com***, e **394 *topic*** per la pagina di **Shuja Rabbani**.
- Nel caso di ***rt.com*** nel **63% dei casi** i *topic* sono stati riferiti a ***Concept***, nel **17% a *Place***, e nel **9% a Organisation**.
- Nel caso di ***Shuja Rabbani*** nel **55% dei casi** i *topic* sono stati riferiti a ***Concept***, nel **15% a *Place***, e nel **10% a Organisation, Work e Person**.

Analisi della "Information Diffusion"

Per quanto riguarda lo studio dell'*Information Diffusion*, sono stati calcolati **l'*albero dei retweet*** (a partire dal nodo radice) e la ***mappa con gli utenti geo-localizzati*,** per **entrambe** le situazioni.

Per entrambe le computazioni, i parametri sono stati impostati come segue:

- *DATI DEL CALCOLO dell'Albero dei Retweet*:
  - *DEPTH (profondità):* ***5***
  - *BREATH (ampiezza):* ***60***
  - *Starts From (nodi di partenza):* ***60 retweeter***.
- *Twitter* al momento dell'estrazione, indicava come *retweet count di R:*
  - *Tweet di* ***rt.com****:* ***370***.
  - *Tweet di* ***Shuja Rabbani****:* ***1789***.

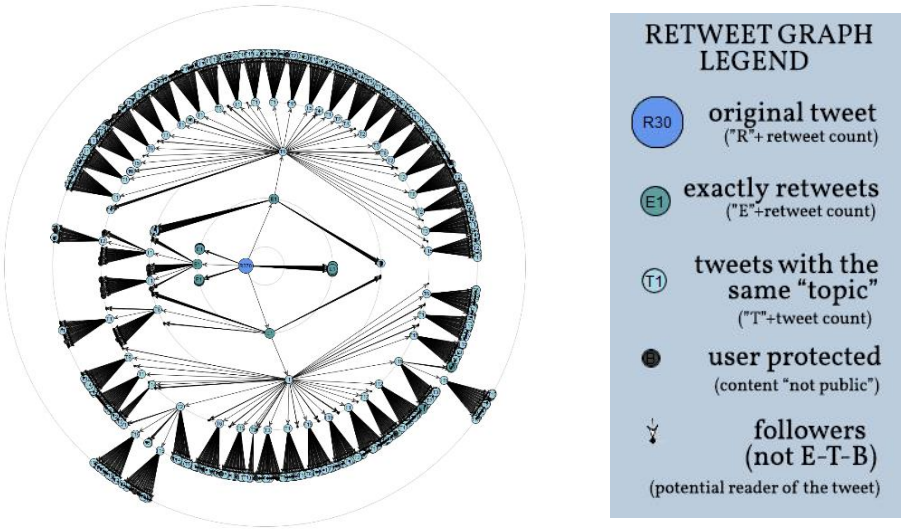


Figura 64: Albero di *Retweet* ottenuto partendo dal ***rt.com*** con *topic* ***"ISIS"***.

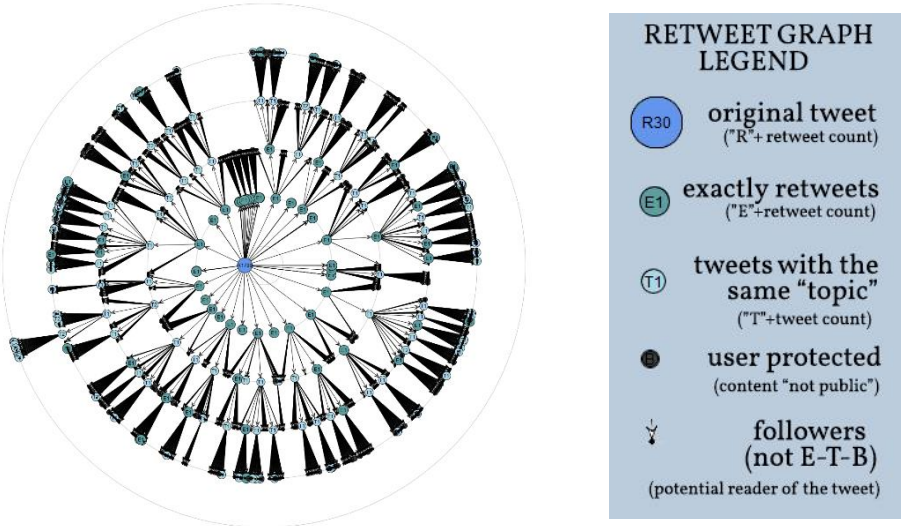


Figura 65: Albero di *Retweet* ottenuto da ***Shuja Rabbani*** con topic ***"ISIS"***.

	Tweet di ***rt.com***	*Tweet di* ***Shuja Rabbani***
Autenticazione Utilizzata	*User OAuth*	*Application OAuth*
Ora d'inizio	13/03/15, 15.30.11	13/03/2015, 15.29.10
Ora di fine	14/03/15, 00.21.55	14/03/15, 00.11.32
Ore di calcolo	8.51.44 (hh/mm/ss)	8.42.22 (hh/mm/ss)
Numero utenti potenziali	790.779.661	790.779.661
Numero utenti analizzati	3166	6122
Vertici (R)	1	1
Vertici (E)	62	100
Vertici (T)	790	191
Vertici (A)	2249	5709
Vertici (B)	64	121
Numero utenti salvati	3166	6122
Numero "location"	2877	5649
Numero "lat/long"	2596	5238

Tabella 5: Confronto tra i risultati di ***rt.com*** e da ***ShujaRabbani*** sul *topic* ***ISIS***.

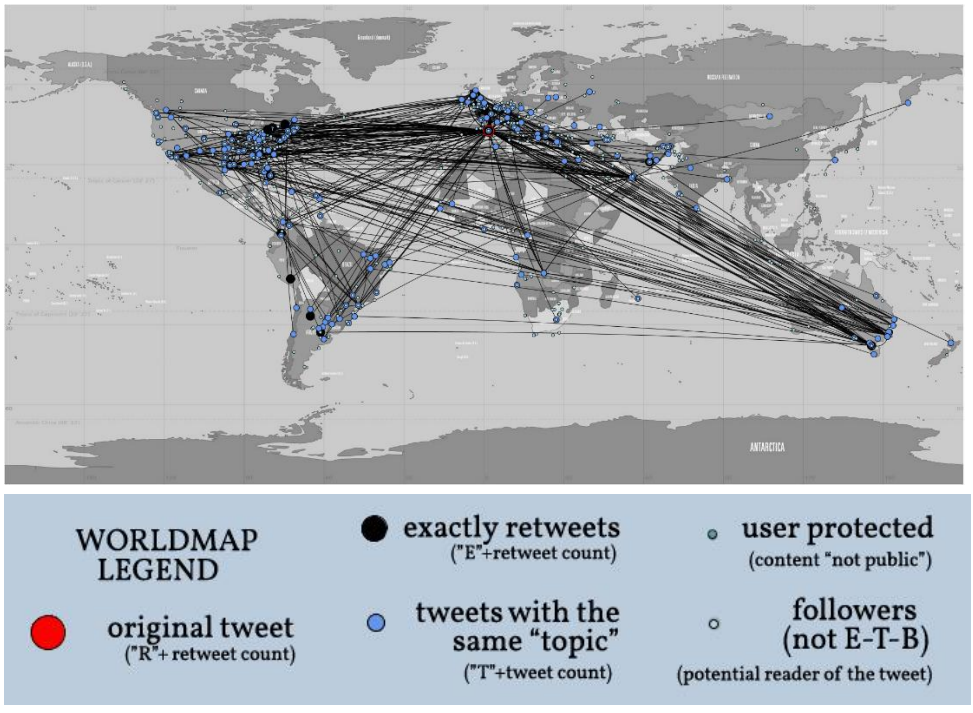


Figura 66: *WorldMap ottenuta posizionando gli utenti coinvolti in* ***rt.com***.

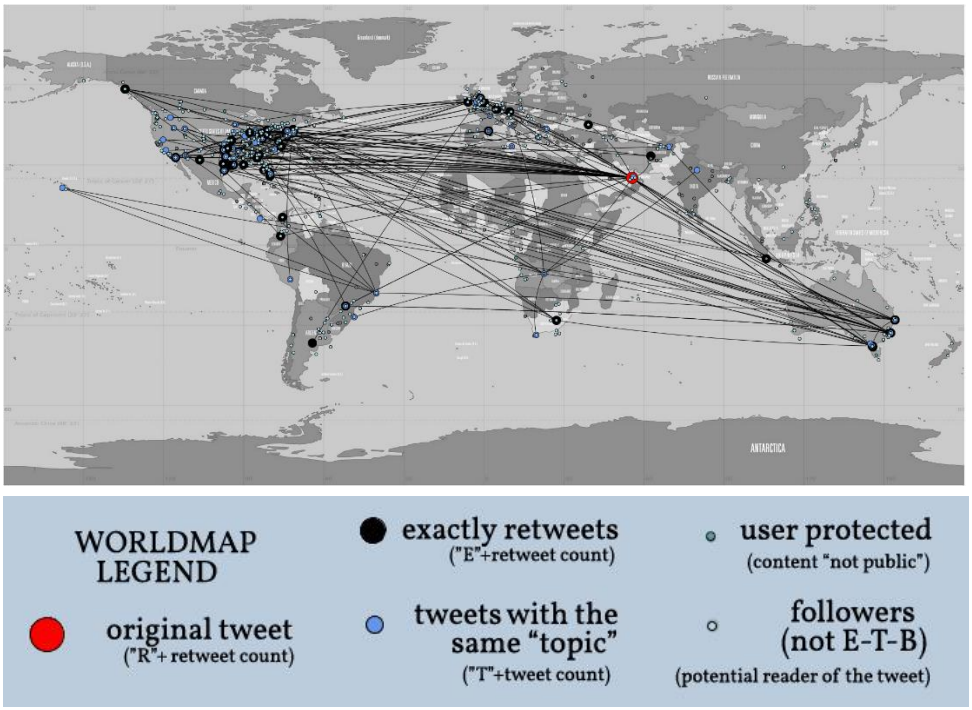


Figura 67: *WorldMap ottenuta posizionando gli utenti coinvolti in* ***ShujaRabbani****.*

Nelle *worldmap* sono stati rappresentati solamente gli *EDGE (archi)* relativi alle condivisioni di nodi (E) e (T), mentre sono stati rappresentati tutti i nodi con diverse colorazioni e dimensioni espresse in *LEGENDA*.

- Per la mappa in [**Figura 66**] sono stati inseriti (**852** su **3166**) archi.
- Per la mappa in [**Figura 67**] sono stati inseriti (**231** su **5462**) archi.

OSSERVAZIONI:

- Anche ad una prima occhiata, è possibile notare che nel caso di *Shuja Rabbani* sono presenti maggiori *retweet esatti (E)*, ma molti meno *retweet "topic" (T)* rispetto al caso *rt.com.*
- Nella *worldmap* di *S.R.* si nota come sulla totalità degli *edge* reperiti, quelli tra nodi (E) e (T), ovvero i soli rappresentati, siano in netta minoranza (con un rapporto di soli 231/5462).
- Il nodo *root* di *S.R.,* nonostante sia geolocalizzato in *Medio Oriente*, ha ricevuto pochissime condivisioni nei territori limitrofi (e stranamente, anche nel territorio minacciato, ovvero l'Egitto). Questo può indicare che la maggioranza dei *follower* di questo utente, provengano da altri stati o continenti (si vedano *USA e Europa).*

5.4 Caso di studio (4): Helicopter Crash (Argentina)

Il terzo caso di studio riguarda un avvenimento di cronaca nera, riguardante la morte di 10 persone all'interno di due velivoli che si sono scontrati, in merito alle riprese di un *"reality show"* denominato *"Dropped"*, girato nei pressi di ***Villa Castelli***, nella provincia di ***La Rioja***, a 1200km da ***Buenos Aires (Argentina)***.

L'evento e le sue stime

Nel tragico evento avvenuto in data 09 Marzo 2015, hanno perso la vita tra le altre persone, anche 3 campioni sportivi francesi:

- ***Camille Muffat*** *(campionessa nei 400m stile libero alle Olimpiadi di Londra 2012).*
- ***Alexis Vistine*** *(pugile medaglia di bronzo, Olimpiadi di Pechino 2008).*
- ***Florence Arthaud*** *(velista pluripremiata, vincitrice della "Route du Rhum" nel 1990).*

Figura 68: Notizia da *BBC Breaking News* relativa a*ll'incidente*, su *Twitter*

Estrazione dei Tweet e Classificazione

Dal processo di estrazione dei *tweets* con *query*: *"Argentina helicopter crash"* (senza aver specificato alcun utente), sono stati estratti **9585 tweet**.

Tutti questi *status* sono stato raccolti tra il **10 Marzo** e il **13 Marzo 2015**.

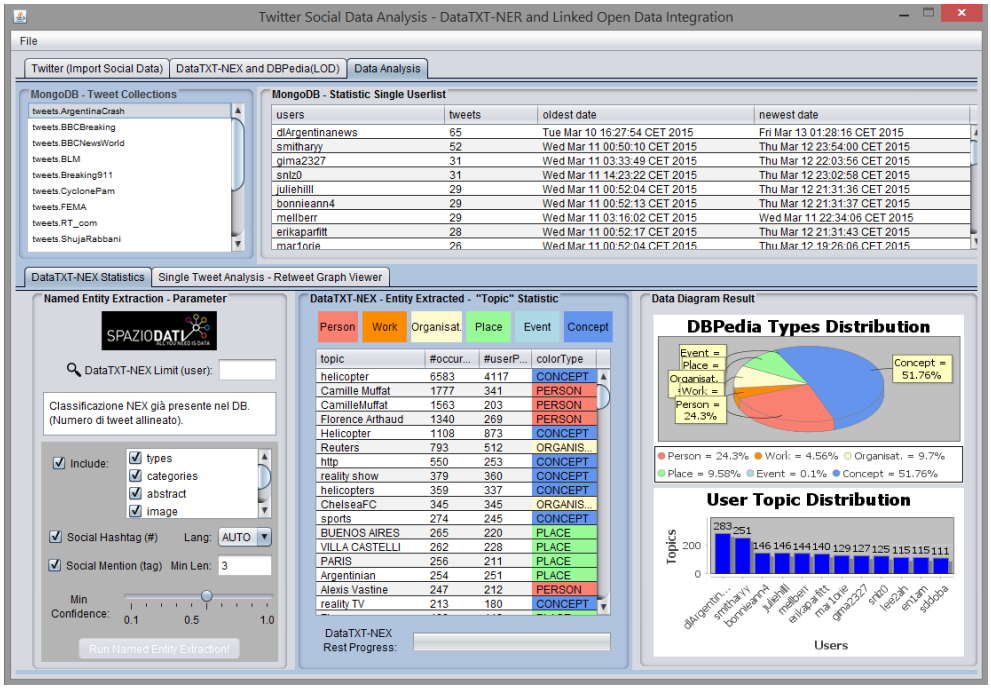


Figura 69: Classificazione dei *topic* estratti dalla totalità di *tweet* reperiti dalla *query "Argentina helicopter crash"*.

Osservando [**Figura 69**] si possono evincere le seguenti affermazioni:

- L'utente con maggior numero di *tweet* pubblicati (all'interno della finestra temporale analizzata) ha condiviso **65 tweet**.
- Sono presenti **15 utenti** con un numero di *tweet* superiore alla ventina, rispetto alla **totalità di 6112 utenti**.
- Osservando la tabella *"User Topic Distribution"*, si nota che rispetto agli utenti totali di cardinalità 6112, solamente **12 utenti** presentano un quantitativo di *topic* inerenti all'argomento (classificati con *confidence di almeno 0.6*), di almeno 110, all'interno delle loro pagine.
- Sono stati classificati **869 *topic* differenti**, di cui **22 *topic*** con un numero di occorrenze superiore ai 100.

- I *topic* appartengono nel **52% dei casi a *Concept***, nel **24% a *Person***, e nel **10% a Place e Organisation**.

Analisi della "Information Diffusion"

Per quanto riguarda lo studio dell'*Information Diffusion*, sono stati calcolati **l'*albero dei retweet*** (a partire dal nodo radice) e la ***mappa con gli utenti geo-localizzati*,** andando a prelevare uno dei tweet che ha riscontrato maggior ricondivisione su *Twitter*.

Partendo dal *tweet* in [**Figura 68**], al termine della *computazione*, sono stati ottenuti i seguenti risultati:

- *Twitter* al momento dell'estrazione dello *status* indicava come numero totale di ricondivisioni per il *tweet* "radice" un valore di **2245**.
- *DATI DEL CALCOLO dell'Albero dei Retweet*:
  - *DEPTH (profondità):* ***6***
  - *BREATH (ampiezza):* ***60***
  - *Starts From (nodi di partenza):* ***60 retweeter***.

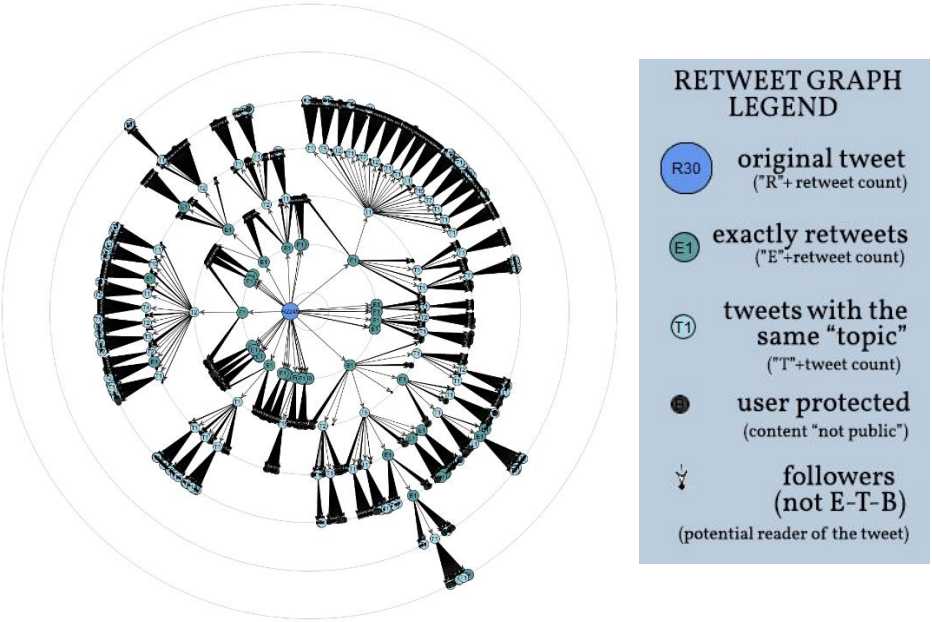


Figura 70: Albero di *Retweet* ottenuto partendo dal *tweet con topic* ***Argentina***.

Il processo di calcolo ha effettuato una computazione dell'ordine di:

- Ora d'inizio computazione: 14 Marzo 2015, ore 01.57.20 (hh/mm/ss).
- Ora di fine computazione: 14 Marzo 2015, ore 11.06.46 (hh/mm/ss).
- Ore di calcolo: **09.09.26 (hh/mm/ss).**
- Numero di utenti potenziali: **47.446.779.660** (calcolato secondo la formula presentata in [Paragrafo 4.5.3].
- Numero di utenti analizzati: **6388** (contando l'utente *root*).
  Di seguito espressi in dettaglio, secondo la *LEGENDA* in [**Figura 64**].
  - VERTICE (R): 1
  - VERTICI (E): 92
  - VERTICI (T): 304
  - VERTICI (A): 5810
  - VERTICI (B): 181
- Numero di utenti salvati nel *DB*: **6388**.
  - Numero di *"location"* trovate nel campo descrizione: **4715**.
  - Numero di "latitude/longitude" reperite ricercando tali *location* all'interno del *database GeoNames* locale: **3979**.

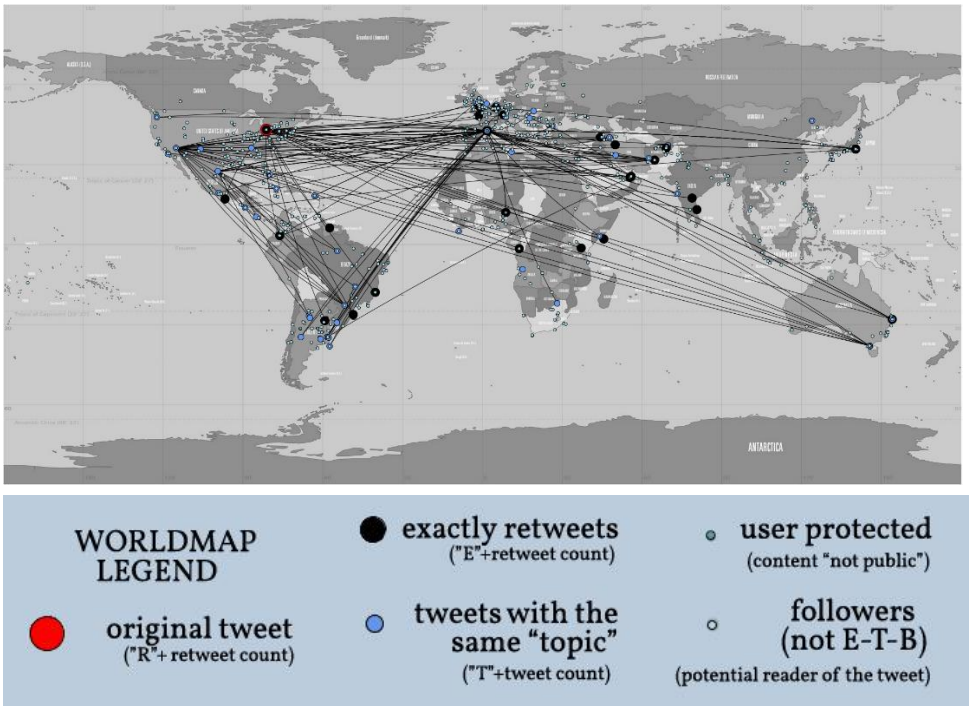


Figura 71: *WorldMap ottenuta analizzando gli utenti coinvolti nella condivisione.*

Nella *worldmap* sono stati rappresentati solamente gli *EDGE (archi)* relativi alle condivisioni di nodi (E) e (T), mentre sono stati rappresentati tutti i nodi con diverse colorazioni e dimensioni espresse in *LEGENDA*.

- Riepilogando sono stati rappresentati (**389** su 6208) archi.

OSSERVAZIONI:

- Anche ad una prima occhiata, è possibile notare che, anche in questo caso come nei casi precedenti, il numero di *edge* tra i nodi di tipo (E) e (T) è di molto inferiore rispetto al totale di quelli reperiti (in rapporto 389/6208).
- In questo caso, il nodo radice (in rosso sulla mappa) è situato in territorio Americano. I nodi con maggior numero di ricondivisioni (quindi gli utenti che hanno maggior influenza sugli altri, e che di conseguenza hanno avuto più *"retweet"* sono situati, uno in USA e uno in Francia.
- Il *tweet* è stato maggiormente visualizzato soprattutto in *Francia* (nazione d'origine dei tre campioni olimpici coinvolti) e in *Argentina* (nazione coinvolta perché sede dell'incidente).
- Da notare inoltre le potenziali ricondivisioni avvenute in territorio Giapponese e Australiano. Questo evento entra a far parte di una grande fetta di "incidenti" che, coinvolgendo figure dello spettacolo o personaggi noti a molti, riescono ad avere maggior possibilità di ricondivisione vista l'adesione popolare mondiale che riescono a riscuotere.

Conclusioni

Nello sviluppo della tesi proposta, sono state affrontate diverse problematiche relative all'integrazione dei differenti servizi coinvolti nella risoluzione dei processi richiesti. Inoltre si è dovuto realizzare una soluzione software robusta, affidabile ed estendibile, per eventuali progetti futuri. Parte dell'attenzione è stata posta sulla selezione dei parametri e delle configurazioni presenti nell'interfaccia, al fine di migliorare l'esperienza di utilizzo dell'utente; mantenendo invariata l'usabilità e la responsività dell'applicazione, anche durante l'esecuzione di attività computazionalmente complesse e onerose. Gran parte delle problematiche affrontate, ha riguardato l'ottimizzazione dei task e l'utilizzo di thread differenti per l'esecuzione di essi, andando a sfruttare al meglio le limitazioni e le temporizzazioni imposte dai servizi di API utilizzati.

I. Lavoro svolto

Il lavoro svolto in questo progetto di tesi, può essere suddiviso in cinque fasi:

Nella parte iniziale è stato effettuato uno studio accurato sull'attuale stato dell'arte relativo ai *Social Media*, e alle tecniche di analisi dei dati in essi pubblicati. Particolare enfasi è stata posta sullo studio dei *BigData*, e sulle tecniche di *Semantic Web Mining*. Nella parte conclusiva del capitolo, è stata

introdotta una breve trattazione sulle principali metodologie di *Social Data Analysis* e sullo studio della *Information Diffusion*.

Nella seconda fase è stato studiato il contesto del problema, e sono stati cercati pregi e difetti dei singoli *social network*. Questo studio ha portato alla scelta della piattaforma sociale *Twitter*, come ambiente di estrapolazione delle informazioni.

Nella terza fase è stata effettuata un'analisi logico-concettuale del progetto, tenendo conto dei vincoli e dei requisiti imposti dalle specifiche realizzative. Al termine di questa fase, si è progettata una struttura logica in grado di rispettare tutte le funzionalità richieste dagli obiettivi di progetto.

Nella quarta fase è stato realmente implementato l'applicativo *software*, in tutte le sue parti. È stato adottato un processo di sviluppo incrementale, alternato da una continua fase di test, che hanno portato allo sviluppo di una soluzione in grado di essere testata su dei casi reali di utilizzo.

Nella quinta e ultima fase, sono stati eseguiti dei test, andando a prelevare i dati da *Twitter*, secondo delle interrogazioni ben precise. Gli eventi selezionati per i casi di studio analizzati, sono stati scelti all'interno del panorama mondiale, per le loro caratteristiche peculiari e per la maggior attinenza con il contesto di sviluppo del progetto.

II. Sviluppi futuri

Al termine di questo progetto di tesi è giusto e doveroso ricordare che, il lavoro ultimato al termine del progetto risulta essere un buon punto di partenza per l'estrazione, la classificazione e l'analisi dei dati interessati.

Nello sviluppo della tesi proposta, sono state effettuate delle supposizioni e sono state adottate, in certi contesti, delle piccole semplificazioni per arrivare ad ottenere una soluzione *software* che potesse essere stabile ed applicabile a contesti di utilizzo reali.

Di seguito verranno elencati alcuni punti, che sicuramente potrebbero essere oggetto di studi e di sviluppi futuri:

- Innanzitutto l'applicazione è stata contestualizzata ai fenomeni catastrofici, per focalizzare il lavoro su eventi che richiedano interventi ed analisi immediatamente successive agli accaduti. Chiaramente questo applicativo può essere esteso, andando ad operare sui contesti più svariati. La presentazione di moduli aggiuntivi che permettano di visualizzare altre tipologie di informazioni, non richiederà alcuna modifica dei moduli precedenti, in quanto questi sono stati realizzati seguendo i principi della "buona programmazione a oggetti", e cercando di rispettare il più possibile la *modularità* del codice.
- Tra tutti i dati estratti dal *social network Twitter*, è stata utilizzata solamente una minima parte dei campi e delle informazioni reperite, al fine di produrre i risultati richiesti dagli obiettivi di progetto. L'espansione della soluzione attuale al fine di realizzare nuove indagini ed analisi dei dati, richiederebbe uno sforzo minimo, andando ad aumentare notevolmente il potere espressivo dell'applicazione, e dei risultati proposti. (Un esempio potrebbe essere quello di analizzare i dati reperiti sugli utenti coinvolti nel processo di ricondivisione dei *tweet*, per poi presentare delle analisi e delle statistiche mediate, relative ad essi. Tra le informazioni, già memorizzate all'interno del *database* locale, sono presenti tra le altre informazioni, anche quelle relative al profilo e alle informazioni biografiche, quindi età, sesso, occupazione, interessi, ...).
- Nell'applicazione realizzata, i *linked-open-data* sono stati reperiti solamente dal portale *DBPedia*, e sono stati impiegati ai soli scopi di integrazione e di classificazione. È ipotizzabile pensare ad un loro utilizzo, per aumentare ancora di più il potere espressivo delle indagini proposte, andando in un futuro, ad integrare nuovi servizi di *API*, al fine di utilizzare tali dati anche nei restanti processi.
- È auspicabile pensare ad un'estensione dell'applicazione, orientata ad un sviluppo distribuito ed eterogeneo, in modo da permettere la condivsione della conoscenza e dei risultati ottenuti, anche tra macchine ed utenti localizzati fisicamente in spazi differenti.

Bibliografia

[1] C. Di Natale e M. Luglio, «Smart city: un punto su infrastruttura e sensori.,» Consorzio NITEL, 18 Dicembre 2014. [Online]. Available: http://www.smartforcity.it/20141218359/opinioni/smart-city-un-punto-su-infrastruttura-e-sensori.html.

[2] O. Lassila e J. Hendler, «Embracing "Web 3.0",» *IEEE Internet Computing,* vol. 11, n. 3, pp. 90-93, May/June 2007.

[3] J. Gantz e D. Reinsel, «The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East,» *IDC iView (EMC),* p. Dicembre, 2012.

[4] T. O'Reilly, «What Is Web 2.0,» 30 Settembre 2005. [Online]. Available: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html.

[5] T. Berners-Lee, J. Yang e R. Hastings, «Web 3.0,» 2 Novembre 2013. [Online]. Available: http://it.wikipedia.org/wiki/Web_3.0.

[6] N. Spivack, «Web 3.0: The Third Generation Web is Coming,» Lifeboat Foundation Scientific Advisory Board, 16 Febbraio 2007. [Online]. Available: http://lifeboat.com/ex/web.3.0.

[7] T. Berners-Lee, «Man Who Invented the World Wide Web Gives it New Definition,» Riseforth Inc, 11 Febbraio 2011. [Online]. Available:

http://computemagazine.com/man-who-invented-world-wide-web-gives-new-definition/.

[8] D. Pfirrmann, «Web 3.0: "Semantic Web" – navigating in an ocean of data,» 07 Maggio 2014. [Online]. Available: http://www.alumniportal-deutschland.org/en/science-research/news-from-science/article/web-30-semantic-web-metadata.html.

[9] V. Eletti, «Big data per l'Agenda digitale in salsa Web 3.0,» 16 Novembre 2012. [Online]. Available: http://www.agendadigitale.eu/competenze-digitali/111_big-data-per-l-agenda-digitale-in-salsa-web-30.htm.

[10] USA.Gov, Office of Science and Technology Policy, «Obama Administration Unveils "Big Data" Initiative,» 29 Marzo 2012. [Online]. Available: http://www.whitehouse.gov/sites/default/files/microsites/ostp/big_data_press_release_final_2.pdf.

[11] E. U. S. F. Programme, «FuturICT Overview,» [Online]. Available: http://www.futurict.eu/the-project/overview.

[12] C. Gionti, «1997 - SixDegrees.com: Il primo brevetto nel mondo dei social network,» 5 Luglio 2010. [Online]. Available: http://socialnetworkhistory.blogspot.it/2010/07/sixdegreescom-il-primo-brevetto-nel.html.

[13] V. Cosenza, «World Map Of Social Networks,» Dicembre 2014. [Online]. Available: http://vincos.it/world-map-of-social-networks/.

[14] M. Davis, «Semantic Social Computing (Semantic Wave 2007: Industry roadmap to web 3.0),» 2007.

[15] Rajiv e M. Lal, «Web 3.0 in Education & Research,» *BIJIT - BVICAM's International Journal of Information Technology,* vol. 3, n. 2, pp. 335-340, 2011.

[16] A. Guille, H. Hacid, C. Favre e D. A. Zighed, «Information Diffusion in Online Social Networks: A Survey,» *SIGMOD Record,* vol. 42, n. 2, pp. 17-28, Giugno 2013.

[17] R. Dawkins, The Selfish Gene, II a cura di, Oxford University Press, 1989, p. 192.

[18] O. Solon, « Richard Dawkins on the internet's hijacking of the word 'meme',» *Wired UK,* 2013.

[19] J. Kleinberg, «The Convergence of Social and Technological Networks,» *Communications of the ACM,* vol. 51, n. 11, Novembre 2008.

[20] L. Palen, «Online Social Media in Crysis Events,» *Educause Quarterly,* n. 3, pp. 76-78, 2008.

[21] M. Brindicci, «#JeSuisCharlie: Social media reacts to Charlie Hebdo massacre,» 11 Gennaio 2015. [Online]. Available: http://rt.com/news/220711-charlie-hebdo-massacre-social-media/.

[22] G. Pass, «Building on Open Source,» 14 Gennaio 2009. [Online]. Available: https://blog.twitter.com/2009/building-open-source.

[23] Twitter, «Barack Obama - How can the incumbent U.S. President mobilize supporters and keep voters informed in real time to win the 2012 election?,» [Online]. Available: https://biz.twitter.com/success-stories/barack-obama.

[24] G. Pena, «Twitter Alerts: Critical information when you need it most,» 25 Settembre 2013. [Online]. Available: https://blog.twitter.com/2013/twitter-alerts-critical-information-when-you-need-it-most.

[25] I. Shklovski, L. Palen e J. Sutton, «Finding Community Through Information and Communication Technology During Disaster Events,» *In Proceeedings of the Conference on Computer Supported Cooperative Work (CSCW '08),* pp. 1-10, 2008.

[26] B. (. Chae, «Insights from hashtag #supplychain and Twitter Analytics: Considering Twitter and Twitter data for supply chain practice and research,» *International Journal of Communication,* p. 13, Gennaio 2015.

[27] M. Mendoza, B. Poblete e C. Castillo, «Twitter Under Crisis: Can we trust what we RT?,» *1st Workshop on Social Media Analytics (SOMA '10),* 25 Luglio 2010.

[28] Y. Qu, C. Huang e P. Zhang, «Microblogging after a Major Disaster in China: A Case Study of the 2010 Yushu Earthquake,» *Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW 2011),* pp. 25-34, 19-23 Marzo 2011.

[29] F. Giglietto e A. Lovari, «Amministrazioni pubbliche e gestione degli eventi critici attraverso i social media: il caso di #firenzeneve,» *Mediascapes Journal,* pp. 98-116, 2013.

[30] S. Harris Smith, K. J. Bennett e A. A. Livinski, «Evolution of a Search: The Use of Dynamic Twitter Searches During Superstorm Sandy,» *PLOS Currents Disasters,* pp. 1-18, 26 Settembre 2014.

NOTA: tutte le citazioni da testi in lingua straniera sono state effettuate traducendo o riassumendo nel modo semanticamente più fedele possibile.