POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Polo Regionale Di COMO

Master of Science in Computer Enigneering

"Streaming Linked Data"

Supervised by: Professor Emanuele Della Valle

Master Thesis by: Nausherwan Adil

Student Id: 736665

Academic Year 2008/2010

POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Polo Regionale Di COMO

Corso di Laurea Specialistica in Ingegneria Informatica

"Streaming Linked Data"

Relatore: Prof. Emanuele Della Valle

Tesi di laurea di: Nausherwan Adil

Matricula: 736665

Anno Accademico 2008/2010

“The limits of my language mean the limits of my world.”
Ludwig Wittgenstein, Tractatus Logico-Philosophicus (1922)

Dedication

To my Grand Parents, Parents, Teachers and especially my guiders;

Syed Yaqoob Ali Shah Naqsbandi Mujadadi Azizi,
Syed Mahboob Ali Shah Naqsbandi Mujadadi Azizi and
Syed Anwar Shah Gelani Al-Hassani Wal-Hussaini Al-Gelani Al-Qadri Al-Baghdadi

Whose, continuous prayers brought me here up to this level of knowledge.

Sommario (in lingua italiana)

L'utilizzo di data stream sta diventando sempre più popolare su Internet, in particolare, nei siti Web che distribuiscono e presentare le informazioni in tempo reale. Il Resource Description Framework (RDF) permette di descrivere documenti Web e risorse provenienti del mondo reale -- persone, organizzazioni, oggetti -- in modo machine-processable. Questo lavoro di tesi risponde ad una necessità sempre crescente di fare mash-up di data stream e informazioni statiche utilizzando RDF. Questo lavoro estende il motore esistente C-SPARQL, con la possibilità di pubblicare data stream come Linked Data mediante un grafo RDF, chiamato S-Graph, per ogni data stream e numerosi grafi RDF istantanei chiamati I-Graphs. Il lavoro di tesi, inoltre, ha testato e analizzato questa estensione del motore C-SPARQL.

Abstract (in lingua inglese)

The use of data streams is becoming increasingly popular over the Internet, especially, in the Web sites that distribute and present information in real-time. The Resource Description Framework (RDF) allows to describe web documents and resources from the real world -- people, organizations, things -- in a computer-process able way. This work anticipates a rapidly growing need of mashing up the streaming information with more static information/data using RDF. This work extends the existing C-SPARQL engine by publishing RDF data streams as linked data. It does so using one RDF graph per each data stream named S-Graph and several instantaneous RDF graphs named I-Graphs. This enrichment of existing C-SPARQL engine has been tested and analyzed.

Acknowledgements

This work was partially supported by the European project LarKC (FP7-215535).

I owe most of my gratitude to my Professor Emanuele Della Valle, who gave me the opportunity to work with him for this thesis during my “Master of Science in Computer Engineering”. I very much appreciate his help and guidance during my whole duration of thesis.

I also want to say thanks to Students and Staff from the Department of Electronics and Informatics, Politechnico Di Milano for providing such a stimulating and friendly environment.

Finally, I am most grateful to my parents for supporting me throughout all these years and for letting me choose my own destiny.

Nausherwan Adil

Table of Contents

List of Figures

List of Drawings (Graphics created by Author)

List of Tables

Chapter 1: Introduction

1.1. Overview

This chapter contains the highlights of the thesis. Firstly, the motivation of the work is provided. Secondly, the objectives of this work have been discussed and lastly, the contributions made by this work are discussed.

1.2. Thesis Statement

Semantic Web is constantly growing and is built on the fundamental idea that data items are being published using dereference able URIs and related data items are connected using RDF links. These data items are termed as streams, appearing more and more on the web that distribute and present information in real-time systems. This work anticipates the growing need of mashing up streaming information with more static one. This work followed an approach to publish the data streams as linked data. As a proof of concept this work transformed the Twitter tweets into RDF stream, used the C-SPARQL for continuous querying over stream of RDF and extend C-SPARQL engine by publishing RDF data streams as linked data. The heart of this work deals with designing and implementing a client/server based architecture for publishing such streams as linked data.

1.3. Motivation

Motivation for this work is two-fold. A lot of work has been done in the area of data stream processing. The data streams are highly available in real-time systems on the Web but there is a lack of Web-based approaches to process them, brought the motivation for this work.

A data stream represents the data arriving from an online source. Streaming data is received by DSMS continuously and in real-time, either implicitly ordered by arrival time or explicitly, associated with timestamps for processing. The data format could be in different formats like XML/RDF or Turtle representation. Some sources in data stream applications typically produce time stamped information (e.g. financial applications uses streams of trading data, stock tickers, news feeds). We defined an RDF raw data stream with timestamps, too. Since data arrives in a time varying manner, data streams should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the fly by continuous queries. Continuous queries, after being registered, keep analyzing such streams, producing answers triggered by the streaming data and not by explicit invocation. Such a paradigmatic change has been largely investigated in the last decade by the database community (1). Traditional DBMS's are best equipped to run one time queries over finite stored data sets. However, many modern applications such as network monitoring, financial analysis, manufacturing, and sensor net works require long-running, or continuous, queries over continuous unbounded streams of data.

The STREAM: Data Stream Management System (2) supports a large class of declarative continuous queries over continuous streams and traditional stored data sets. Its prototype targets environments where streams may be rapid, stream characteristics and query loads may vary over time, and system resources may be limited. Declarative queries in this DSMS, translated into physical query plans, are flexible enough to support optimizations and fine grained scheduling decisions. The DSMS exploit possibilities for sharing state and computation within and across query plans. Since data, system characteristics, and query load may fluctuate over the lifetime of a single continuous query, an adaptive approach to query execution is essential for good performance. When incoming data rates exceed the DSMS's ability to provide exact results for the active queries, the system should perform load-shedding by introducing approximations that gracefully degrade accuracy. Due to the long-running nature of continuous queries, DSMS administrators and users require tools to monitor and manipulate query plans as they run. This functionality is supported by a graphical interface.

Aurora (3) is a new system to manage data streams for monitoring applications. It deals with large numbers of data streams. It supports the users built queries out of a small set of operators (boxes) and combination of (boxes) for better answers. Monitoring applications differ substantially from conventional business data processing. The fact that a software system must process and react to continual inputs from many sources (e.g., sensors) rather than from human operators requires one to rethink the fundamental architecture of a DBMS for this application area.

Stream Mill (DSMS) (4) is designed to support complex applications, along with the simpler applications serviced by other DSMS. In particular, Stream Mill is the first DSMS to support stream mining applications. In fact, Stream Mill provides tools that facilitate the process of migrating queries on tables to continuous queries on data streams. Stream Mill is based on client/server architecture. Thus, the server is always running, and managing the incoming data streams. Users can login through a client and register continuous queries on those streams.

In analogy to the application of database management systems (DBMS) for many standard applications like personnel or warehouse management, we believe that data stream management systems (DSMS) helps in managing data streams in many modern applications, such as network monitoring and traffic engineering, financial analysis, manufacturing, and sensor networks require long-running, or *continuous*, queries over continuous unbounded streams of data. In DSMS continuous queries are used to determine stream information.

The Semantic Web is defined as “the meaning of a Web”, was thought up by Tim Berners-Lee, inventor of the WWW, URIs, HTTP, and HTML. It is a mesh of information linked up in such a way as to be easily process able by machines, on a global scale. You can think of it as being an efficient way of representing data on the World Wide Web, or as a globally linked database. However, Semantic Web is growing up and is known as Web 3.0. The future of the semantic web is in general appears to be bright.

The Semantic Web foundation is laid on the use URIs for documents, Resource Description Framework (RDF), a standard language for representing information about resources in the World Wide Web. This is also certified by W3C. RDF has three species for writing into triples, N3, Turtle, N-triples.

RDF Schemas declare vocabularies and the XML namespace mechanism serves to identify RDF Schemas. An RDF schema may be accessed by de-referencing the schema URI. If the schema is machine-process able, it may be possible for an application to learn some of the semantics of the property-types named in the schema. However an RDFS has structure based on the RDF data model.

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called ontology. OWL facilitates greater machine interpretability of Web content than what is supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

SPARQL-Simple protocol and RDF query language can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs. SPARQL is the W3C Recommendation for a query language over RDF.

The tools for publishing relational documents on semantic web are getting mature as semantic is growing day by day. E.g. D2R, D2RQ, Pubby and Virtuoso Universal Server.

Linked Data builds on the classic architecture of the Web. The data in the web becomes part of a single global data space. The data is crawled by Semantic Web search engines and is used by various applications. People can use various data browsers to explore your data. People start setting links to your data, which might make more people find and use your data.

The data is published on standards like HTTP, URIs and RDF and is based on the following design principles:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
- Include links to other URIs. So that they can discover more things.

There are certain linked data browsers available for browsing the web.

- Tabulator Browser (MIT, USA)
- Marbles (FU Berlin, DE)
- OpenLink RDF Browser (OpenLink, UK)
- Zitgist RDF Browser (Zitgist, USA)
- Disco Hyperdata Browser (FU Berlin,DE)

There are two well known and common solutions for URI publishing which are:

1. 303 URIs
2. Hash URIs

303 URIs are good for large sets of data that are, or may grow, beyond the point where it is practical to serve all related resources in a single document. However 303 URI approach is more feasible.

Content negotiation is a mechanism defined in the HTTP specification that makes it possible to serve different versions of a document (or more generally, a resource) at the same URI, so that user agents can specify which version fits their capabilities the best. When a client sends a request to a server, the client informs the server what media types it understands with ratings of how well it understands them. However the client provides an Accept HTTP header that lists acceptable media types and associated quality factors. The server is then able to supply the version of the resource that best fits the client's needs. So, a resource may be available in several different representations. In our case it will be HTML and RDF.
C-SPARQL is an extension to SPARQL for continuous querying over virtual streams of RDF and static RDF graphs i.e., queries registered over both RDF repositories and RDF streams. C-SPARQL queries can be considered as inputs to specialized reasoners that use their knowledge about a domain to make real-time decisions. Our C-SPARQL Engine (5) treats non-RDF DSMSs as virtual RDF streams and graphs. However C-SPARQL engine process and publishes those RDF data streams as linked data.

Publishing an RDF stream or graph as Linked Data on the Web will follow the following three basic steps:

- Assign URIs to the entities described by the data set and provide for dereferencing these URIs over the HTTP protocol into RDF representations.
- Set RDF links to other data sources on the Web, so that clients can navigate the Web of Data as a whole by following RDF links.
- Provide metadata about published data, so that clients can assess the quality of published data and choose between different means of access.

DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data. We hope this will make it easier for the amazing amount of information in Wikipedia to be used in new and interesting ways, and that it might inspire new mechanisms for navigating, linking and improving the encyclopedia itself. The DBpedia data set uses a large multi-domain ontology which has been derived from Wikipedia. DBpedia provides three different classification schemata for things. Yago classification is one of them. However the YAGO Classification is derived from the Wikipedia category system using Word Net. Using the classifications within SPARQL queries allows you to select things of a certain type.

Micro blogging is a minor version of blogging that allows users to write entries of up to 140 characters. The messages can be viewed by any users. The difference between a traditional blog and micro blogging is, that micro blogs often are headlines of present activity, thoughts and news from a given source whereas blogging mostly is larger stories, with more background information, research including links and pictures since weblogs allow more content. There are several web services that provide microblog functionality such as:

- Twitter
- Jaiku
- Tumblr
- Yammer

Micro blog feature i.e status updates:

- Facebook
- LinkedIn
- Myspace

Twitter is a social networking website which offers micro blogging service. Tweets are text-based posts of up to 140 characters displayed on the user's profile page. Tweets are the messages which are publicly visible by default; however senders can restrict message delivery to their followers. Twitter has given the user's following and being followed concept.

People describe their status in the form of tweets. They can group posts together by topic or type by use of *hashtags* — words or phrases prefixed with a #. Similarly, the @ sign followed by a username is used for mentioning or replying to other users.

However we transformed all the tweets, including user name, topic name, external link, and the user being followed, to the RDF stream.

Developed a client server architecture based prototype server named as Streaming Linked data server which is responsible for publishing the RDF streams in response to HTML and RDF client requests. However the system involves the SPARQL query end point.

The twitter tweets when transformed to an RDF stream we represent this RDF stream in RDF using named graphs. However we distinguish between two kind of named graphs: the Stream Graphs (shortly s-graphs) and the Instantaneous Graphs (shortly i-graphs). The s-graph contains the RDF triples that refer to the i-graphs using rdfs:seeAlso. These graphs are made query able by a SPARQL end point.

As the number of available streams is very large, so we introduced the windowed data stream model. This is useful to limit the extraction of the data stream elements upon continuous queries. The windowing model is expressed in C-SPARQL. A window extracts from the stream the last data stream elements, which are considered by the query. Such extraction can be physical (a given number of triples) or logical (all the triples which occur during a given time interval, the number of which is variable over time). Logical windows are further divided in to sliding and tumbling window. With tumbling windows every triple of the stream is included exactly into one window, whereas with sliding windows some triples can be included into several windows. We believe that consumers of Streaming Linked Data would largely benefit from controlling the window of a running CSPARQL query.

1.4. Objectives

The objective of my work is to provide architecture for conceptual understanding on how to publish a stream as linked data on the web. The architecture consists of three main parts:

1. Streaming linked data Server (Only a prototype server for testing purpose)
2. C-Cparql Engine
3. HTML ,RDF and SPARQL query end point

1.5. Visualization

The visualization of this work will be a web interface through which user can access HTML and RDF template presenting a data stream transformed from twitter. Other than this the user directly put a request to the Streaming linked data server through a web browser and the server will respond accordingly.

1.6. Contribution

Micro blogging is a very social method of communication that appeals to our inherently inquisitive human nature. We're social beings who like to know what other people are doing, reading, writing, watching, listening to etc.! It's open, informal and spontaneous, all of which encourages interaction. The 'posts' or 'tweets' if using Twitter, can be about any topic and are available to be read by anyone. People freely reply to tweets, entering the conversation; responses are made, ideas are swapped and suggestions are offered. However, posts are most likely to be seen by the author's 'followers' – people who have indicated that they're interested and want to receive updates from them.

However we used these tweets as our source of data and we transformed these to RDF streams, linked with DBpedia , queried on sindice and developed a Web-based approach to process them.

Chapter 2: Background and State of the Art

2.1. Overview

To build a semantic web application, for publishing streams as linked data, is the driving force behind this work. Therefore, to bring clarity into this topic, this chapter is being included. The principles behind the concept of publishing data streams will be presented in this chapter. We will also go through different existing systems using DSMS and see their advantages and shortcomings as well.

2.2. The Data Stream

As a data stream is defined as the real-time, continuous, ordered sequence of items. Therefore Data streams differ from the conventional stored relational model in several ways:

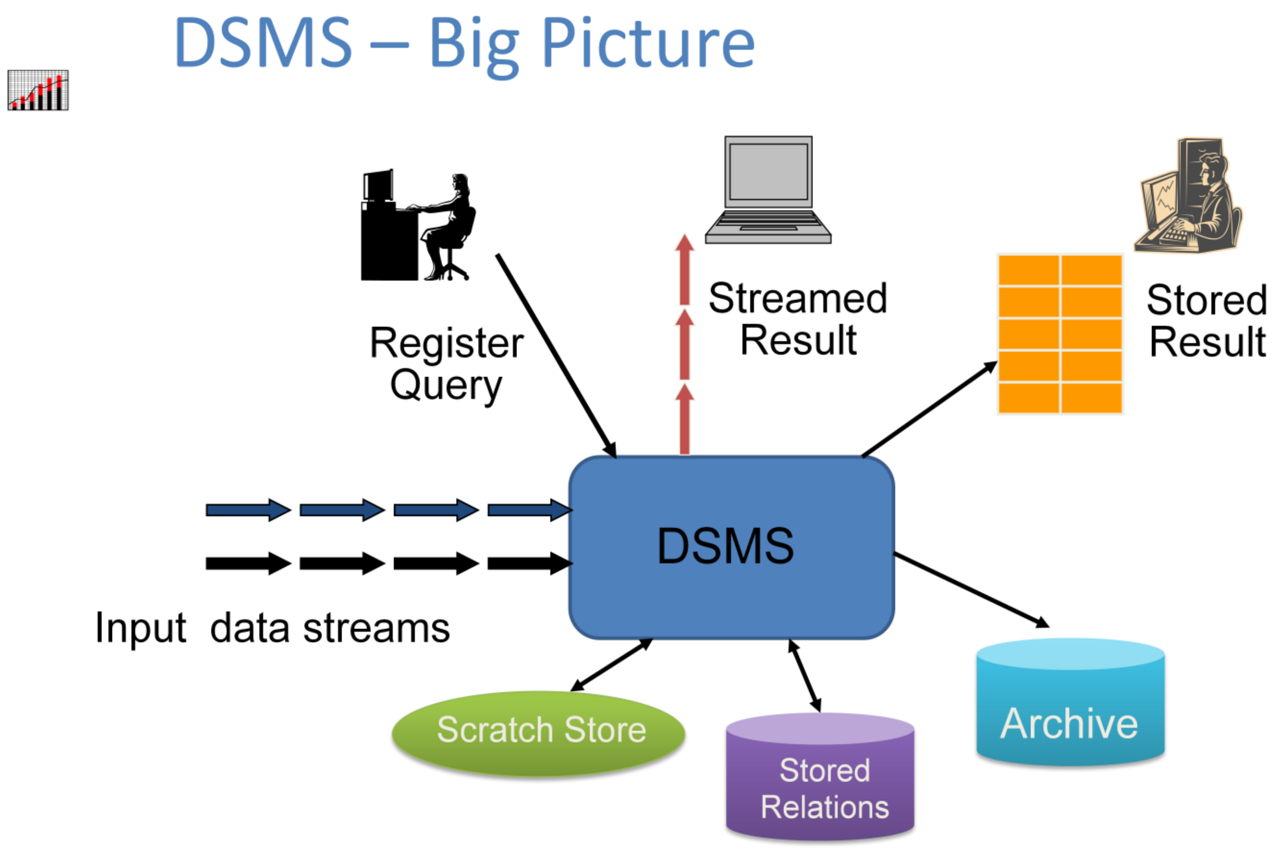
- The data elements in the stream arrive online.
- The system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams.
- Data streams are normally unbounded in size.
- Once an element from a data stream has been processed it is discarded or archived this means that it cannot be retrieved easily unless it is explicitly stored in memory, which typically is small relative to the size of the data streams.

Data Streams, being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the fly by continuous queries.

Figure 2.2.1 Data Streams

2.3. Data Stream Management System

Traditional database management systems are best equipped to run *one-time* queries over finite stored data sets. However, many modern applications such as network monitoring, financial analysis, manufacturing, and sensor net- works require long-running, or *continuous*, queries over continuous unbounded streams of data. We can see the Big picture of data stream management system.



Drawing 2.3.1 Data Stream Management System In A Big Picture

2.3.1. Characteristics

The data stream management system must hold the following characteristics:

- Manage traditional stored data (relations)
- Handle multiple continuous, unbounded, possibly rapid and time-varying data streams
- Supports long-running continuous queries, and produce answers in a continuous and timely fashion

Data stream model creates the fundamental difference between a classical DBMS and a DSMS. DBMS process a query over a persistent set of data that is stored in advance on disk however in DSMS queries are performed over a data stream. In a data stream, data elements arrive on-line and stay only for a limited time period in memory. Consequently, the DSMS has to handle the data elements before the buffer is overwritten by new incoming data elements. The order in which the data elements arrive cannot be controlled by the system. Once a data element has been processed it cannot be retrieved again without storing it explicitly. The size of data streams is potentially unbounded and can be thought of as an open-ended relation. In DSMSs, *continuous queries* evaluate continuously the arriving data elements. Standard operator types that are supported by most existing DSMSs are filtering, mapping, aggregates, and joins. Since continuous streams may not end, intermediate results of continuous queries are often generated over a predefined *window* and then either stored, updated, or used to generate a new data stream of intermediate results. Window techniques are especially important for aggregation and join queries. The interested reader can find an extensive overview on DSMSs in (6; 7)

2.4. STREAM

In the ***STREAM:*** The Stanford Data Stream Management System (8) project at Stanford investigated on the applications which involve data management and query processing.

As a part of the project general-purpose prototype Data Stream Management System (DSMS) has been built, STREAM: Data Stream Management System supports a large class of declarative continuous queries over continuous streams and traditional stored data sets. Its prototype targets the environments where streams may be rapid, stream characteristics and query loads may vary over time, and system resources may be limited.

The STREAM: Data Stream Management System supports a continuous query language (CQL) for registering continuous query. When a CQL query is registered, STREAM constructs a query plan. Query plans are composed of three types of components.

- Query operators (similar to traditional DBMS)
- Inter-operator queues (similar to some traditional DBMS)
- Synopses

CQL is defined by instantiating the operators of abstract semantics defined for STREAM. Syntactically, CQL is a relatively minor extension to SQL. However the abstract semantics is based on two data types, streams and relations

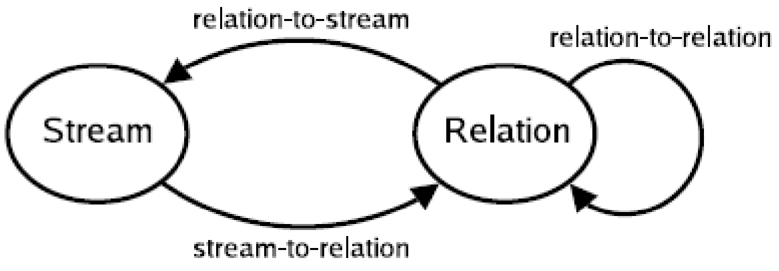


Figure 2.4.1 Data types and operator classes in abstract semantics

- A relation-to-relation operator takes one or more relations as input and produces a relation as output.
- A stream-to-relation operator takes a stream as input and produces a relation as output.
- A relation-to-stream operator takes a relation as input and produces a stream as output.

The stream-to-relation operators in CQL is based on the concept of a sliding window (9) over a stream, and are expressed using a window specification language derived from SQL-99:

Sliding window specification:

- Partitioning clause (grouping)
- Window size (ROWS or RANGE)
    - e.g. “ROWS 50 PRECEDING”
    - e.g. “RANGE 15 MINUTES PRECEDING”
- Filtering predicate (WHERE)
- Sampling clause
- specifies that a random sample of the data elements should be used for query processing
    - (e.g. “1 % SAMPLE” means each data element in the stream should be retained with probability 0.01 and discarded with probability 0.99)

A queue in a query plan connects its “producing" plan operator *OP* to its “consuming" operator *OC*. At any time a queue contains null collection of elements representing a portion of a stream or relation. The elements that *OP* produces are inserted into the queue and buffered there until they are processed by *OC*.

Mechanisms for buffering tuples and generating *heartbeats* to ensure non decreasing timestamps, without sacrificing correctness or completeness, are discussed in detail in (10)**.**

The most common use of a synopsis in this system is to materialize the current state of a (derived) relation, such as the contents of a sliding window or the relation produced by a sub query.

- used to maintain state associated with operators
- summarization technique (sliding windows) used to limit their size (produce approximate results)

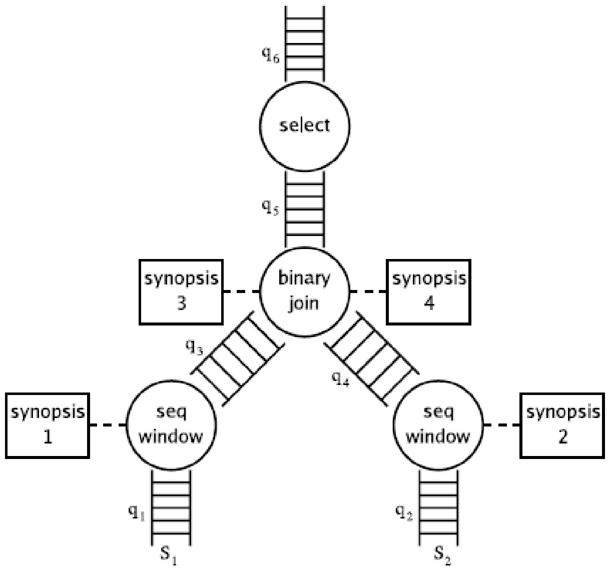


Figure 2.4.2 Simple query plan include operators, queues, synopses

When a query plan is executed, a *scheduler* selects operators in the plan to execute in turn. The semantics of each operator depends only on the times-tamps of the elements it processes, not on system or “wall-clock" time. Thus, the order of execution has no effect on the data in the query result, although it can affect other properties such as latency and resource utilization. *Global scheduler* for plan execution (calls run methods)

- uses round-robin scheme
- Focus on minimizing intermediate (inter-operator) queue sizes
- Parallelism not considered
- Greedily schedule the operator that “consumes” the largest number of tuples per time unit and is the most selective (i.e. “produces” the fewest tuples)
- Example:
    - a query plan with two unary operators:
    - *O*1 operates on input queue q1, writing results to queue q2 which is input to operator *O*2
    - *O*1 takes one time unit to operate on a batch of n tuples from q1, and has 20% selectivity (produces *n*/5 tuples in q2)
    - operator *O*2 takes one time unit to operate on *n*/5 tuples, produces no tuples on its output queue
    - assume the average arrival rate of tuples on q1 is no more than *n* tuples per two time units, so all tuples can be processed and queues will not grow without bound

- Approximation technique goal is to maximize the precision of query answers based on the available resources; there are Static and Dynamic Approximations.
- *In Static* Approximation Queries are modified when they are submitted to the system (use less resources)
- Two techniques:
  - Window Reduction (reduce memory and computation)
    - Decrease the window size or introduce a window where none was specified originally (band joins)
    - This can have a ripple effect that propagates up the operator tree
  - Sampling Rate Reduction (reduce output rate)
    - Reduce the sampling rate of the SAMPLE clause or introduce one where none was specified originally Can take an existing sample operator and push it down the query plan
- In Dynamic Approximation Queries are unchanged
- System may not always provide precise query answer
- Three techniques:
  - Synopsis Compression (analogous to window reduction)
    - Reduce synopsis sizes at one or more operators
    - Incorporating a sliding window into a synopsis or shrinking the existing window
    - Maintaining a sample of the intended synopsis content
  - Sampling (reduce queue size)
    - Introduce one or more *sample* operators into the query plan, or to reduce the sampling rate at existing operators
  - Load Shedding (reduce queue size)
    - Simply drop tuples from queues when they grow too large

In a system for continuous queries, it is important for users, system administrators, and system developers to have the ability to inspect the system while it is running and to experiment with adjustments. To meet these needs, we have developed a graphical *query and system visualizer* for the STREAM system.

The visualizer allows the user to view the structure of query plans and their component *entities* (operators, queues, and synopses). Users can view the path of data flow through each query plan as well as the sharing of computation and state within the plan.

View the detailed properties of each entity. For example, the user can inspect the amount of memory being used (for queue and synopsis entities), the current throughput (for queue and operator entities), selectivity of predicates (for operator entities), and other properties.

Dynamically adjust entity properties. These changes are reflected in the system in real time. For example, an administrator may choose to increase the size of a queue to better handle bursty arrival patterns.

View *monitoring graphs* that display time-varying entity properties such as queue sizes, throughput, overall memory usage, and join selectivity, plotted dynamically against time. The technique for implementing the monitoring graphs is based on *introspection queries* on a special system stream called 18 Arasu et al.

2.5. Aurora

The Aurora system (3) is a data stream management system with fully functional prototype which includes both graphical development environment, and runtime system. Such a data stream management system is used for monitoring applications. Monitoring applications are developed in consultation with defense, financial, and natural science communities. These applications use the data streams which are continuous data from the sources such as sensors, satellites and stock feeds. These applications track the data from numerous streams, filtering them for signs of abnormal activity and processing them for purposes of aggregation, reduction and correlation. Also the management requirements for these monitoring applications are different from the traditional DBMS. However the management requirements for DSMS are

- Monitoring and alerting humans for abnormal activities
- Processing of data that is bounded by finite window of values and not over unbounded past
- Respond to real-time deadlines and provide reasonable approximations to queries
- Benefits from Application specific optimization criteria (QoS)
- The norm is push-based data processing

The Aurora system consists of the following components:

- A Java-based GUI development environment, where tuple structures and Aurora flow networks are defined.
- A server that executes an Aurora network. The inputs and outputs of the Aurora server are streams of tuples, delivered over TCP/IP sockets.

- A Java-based GUI performance monitor that shows the quality of service being provided by the server at a given moment.

However the overall goal of the Aurora system is to maximize overall quality of service from all applications. There are some Novel Features which includes Stream-oriented query operators for stream processing and window based operators. The window operations have optional timeout and slack parameters that enable them to deal with slow and out-of-order arrivals, respectively.

Aurora application defines one or more Quality of Service (QoS) functions/graphs. QoS specifications serve as specifications of desired system behavior. The three main functions graph of QoS are:

- A latency graph
- A loss-tolerance graph
- A value-based graph

The latency graph indicates how QoS drops as the results are delayed. The loss-tolerance graph is a simple way to describe how averse the application is to approximate or incomplete answers. The value-based graph defines the relative importance of the output values.

QoS specifications also serve to drive policies for scheduling, storage management and load shedding. The scheduler decides which operators to execute and in which order to execute them, pays special attention to reducing operator scheduling and invocation overheads, batches (i.e., groups) multiple tuples and operators and executes each batch at once. The storage manager is designed for storing ordered queues of tuples instead of sets of tuples (relations). It also combines the storage of push-based queues with pull-based access to historical data stored at connection points.

The load shedder is responsible for detecting and handling overload situations. The latter is accomplished by shedding tuples by temporarily adding “drop” operators to the Aurora processing network. The goal of a drop is to filter messages, either based on the value of the tuple or in a randomized fashion, in order to rectify the overload situation and provide better overall QoS at the expense of reduced answer quality.

The Aurora System Model:

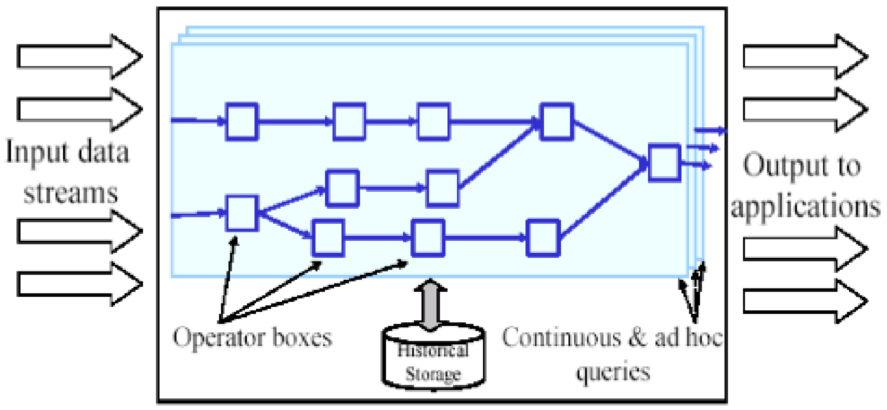


Figure 2.5.1 Aurora System Model

The basic job of Aurora is to process incoming streams in the way defined by an application Administrator

- Data Stream Flow
- Input from external Stream
- Data flow through a loop-free, directed graph of processing operations (ie. boxes)
- Output streams are presented to applications
- Maintain historical storage (support ad-hoc query) Operators
- Eight Primitive Operators
- Windowed Operators
    - Slide
    - Tumble
    - Latch
    - Resample
- Non-Windowed Operators
    - Filter – Drop
    - Map
    - GroupBy
    - Join

Query Model

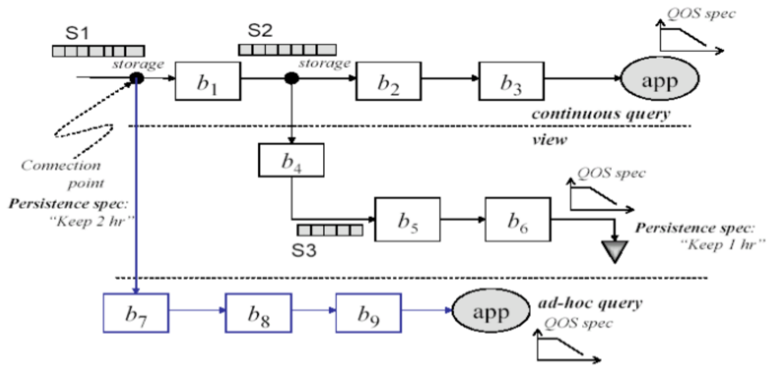


Figure 2.5.2 Aurora Query Model

There are three types of queries

- Continual queries (real-time processing)
- Views
- Ad-hoc queries

Continuous queries do not need to store the data once they are processed. The QoS specification at the end of the path controls how resources are allocated to the processing elements along the path. Application – programmed to deal with asynchronous tuples.

Views are a path is defined with no connected application. It is allowed to have a QoS specification as an indication of the importance of the view. Applications can connect to the end of this path whenever there is a need. Moreover, it can store these partial results at any point along a view path.

Adhoc queries has a Connection Point .A connection point is an arc that will support dynamic modification to the network. An ad-hoc query can be attached to a connection point at any time. Data stored in the connection point is delivered to ad-hoc query .Thus, the semantics for an Aurora ad-hoc query is the same as a continuous query that starts executing at t_{now}-T and continues until explicit termination.

The example of Aurora based application are financial Applications. This application monitors streams of stock quotes. As with the military application, the user can both view a digest of the information, as well as be alerted when interesting events occur. In particular, we have been working with Fidelity on several problems that they have ranging from intelligent routing of trades to fraud detection on transaction data.

Aurora's performance monitoring tools are useful for showing how much data is in various parts of the system and its flow rate, as well as the QoS level that Aurora is able to achieve.

We'll show how varying the rate of input data flow affects the Aurora network's internal queue contents as well as the overall delivered QoS.

Aurora, whose design was presented at VLDB 2002 (11), has been the focus of positive attention room the industrial, scientific, and academic sectors. We've developed a prototype Aurora application for a defense company that easily solved a problem that the company considered difficult. We're working with partners in the financial and natural science fields to develop prototype Aurora applications for those sectors as well.

2.6. Stream Mill

Stream Mill (4) is a Data Stream Management System (DSMS) designed to support complex applications. In particular, Stream Mill is the first DSMS to support stream mining applications; this is achieved by the Stream Mill Miner (SMM) workbench. Stream Mill provides a user-friendly introduction to the system and its powerful query language Expressive Stream Language (ESL), which extends SQL into a Turing-complete language by means of User Defined Aggregates (UDAs).

Most of the ESL constructs are applicable to both database tables and data streams. This simplifies the task of ESL programmers who are likely to develop and test their ESL queries on tables before applying them to streams. It is recommended that new users learning the system follow this procedure; in fact, Stream Mill provides tools that facilitate the process of migrating queries on tables to continuous queries on data streams.

Stream Mill is based on client/server architecture. Thus, the server is always running, and managing the incoming data streams. Users can login through a client and register continuous queries on those streams. In writing queries, the ESL programs can use the following declared objects:

1. Imported data streams,
2. Data streams derived from other streams,
3. Database tables, residing on disk or main memory,
4. Tables derived from data streams,
5. User-Defined Aggregates, with, and without windows,
6. Table Functions.

Multiple clients can connect to a single server. Users, using the client, can create and save their entities such as streams and queries on the server for future retrieval.

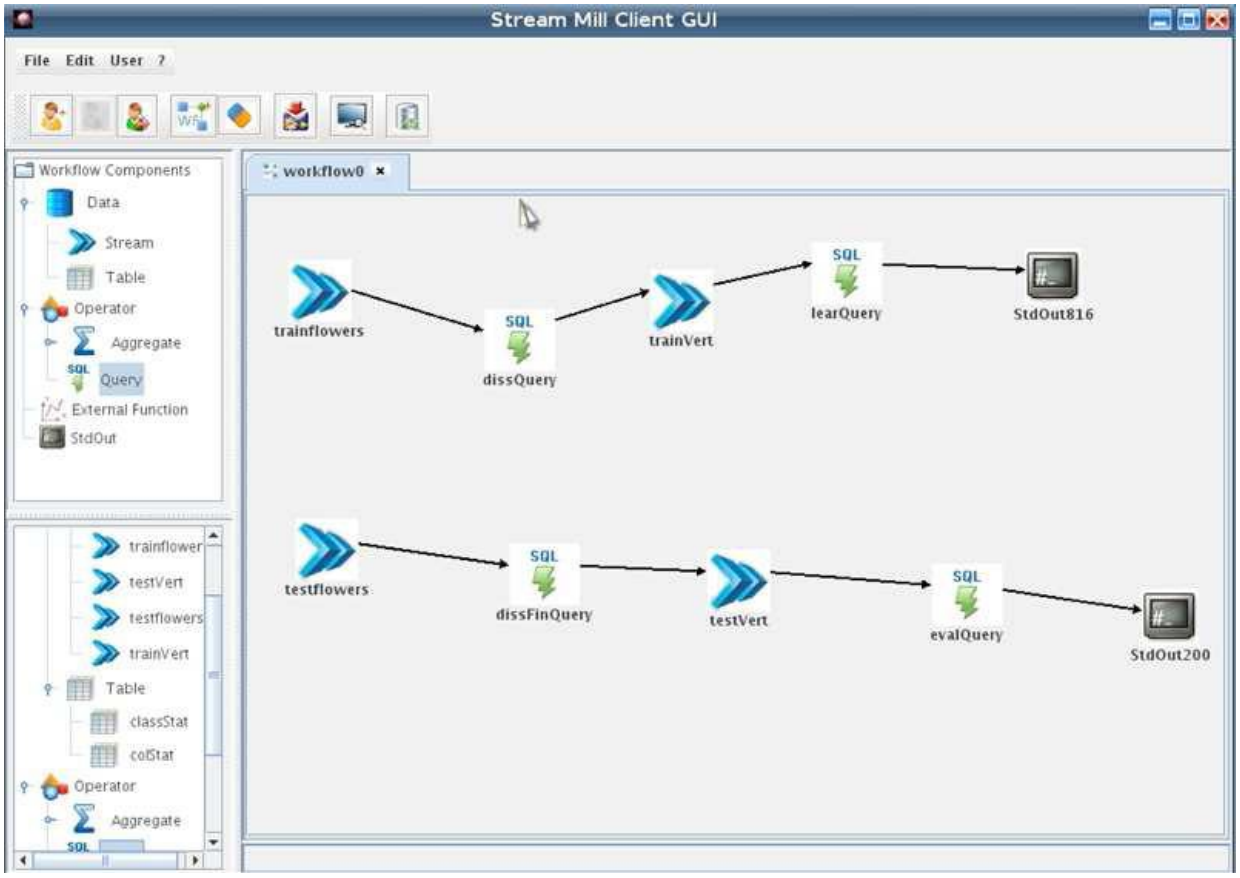


Figure 2.6.1 Stream Mill DSMS Client

In the figure 2.6.1 the main window is used for building workflows. The smaller rectangular window on the bottom left is used to view components which are available on the server for a specific user. The smaller rectangular window on the top left is used to create new components on the server (such as streams etc). The client allows the user to create a workflow where he can specify all steps w.r.t the data mining process he is building. Basically he can create his own entities (i.e. streams, tables, queries, aggregates etc.) on the server, or just use ones created previously. User can use created entities as building blocks for his workflow. Note that entities created cannot be modified, thus if modification is desired entity must first be deleted and re-created with appropriate adjustments. The client has 3 modes, namely, Not-Logged-in mode, Logged-in mode and View Library. In a Not Logged-in mode, only a few functions are available like log in, create new user; while in the View Library mode previous functions and also object viewing/monitoring are available. Functions of all types are available in the Logged-in mode where user can create new tables, streams, queries etc on the server and build workflow related to data mining process.

2.7. Semantic Web Publishing

Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C (World Wide Web Consortium).

- "Classical" Web: computers deliver documents (text, multimedia…)
- "Semantic" Web: let computers process (interpret, combine, select, judge) and deliver information

As the word "Semantic" is the study of meaning. So Semantic Web means a Web with meaning.

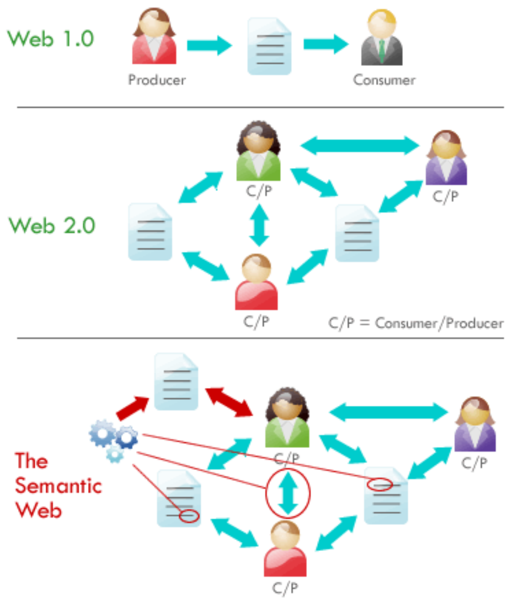


Figure 2.7.1 Web and Web Of Semantics

The Web 1.0 has the scope of web pages that are able to just give out the information to the viewers as they can see in TVs. Similarly in Web 2.0 the web pages became dynamic in behavior; they provide information and also accept data from the viewers and produce results based on that data. The current version of Web which is Web 2.0 allows the search engines use the keywords in the web pages to show the results of the search query. This means that means the contents are index by means of keywords or tags, but not by the meaning.

However the semantic Web provides a better efficient searching of the web using the semantics of the content in the pages. This means that the search engines work not with the keywords we use to search, but with the meaning of the search query we gave. So this new feature in the development of the web Semantic Web and now being called as Web 3.0.

Lets take an example that I am going to Milan. And I want to know about the hotels, tourists spots etc in Milan. In the present condition if I search the term '***I am going to Milan***', I get the results page showing the websites containing the keywords I searched, but not useful for me. But in Semantic Web the search engine understands the semantics, that is the meaning of the search query and give results based on the semantics of the query. This will helps the users more than ever before.

Therefore we can say that the Semantic Web is a mesh of information linked up in such a way as to be easily process able by machines, on a global scale. So the Semantic web publishing means to publish the information on the web as data objects using the semantic web language or as documents having semantic markups. It is a way for computers to understand the structure and even the meaning of the published information, making information search and data integration more efficient. Semantic publishing is driven by the increase of semantic web. In the semantic web, published information is accompanied by metadata describing the information, providing a "semantic" context.

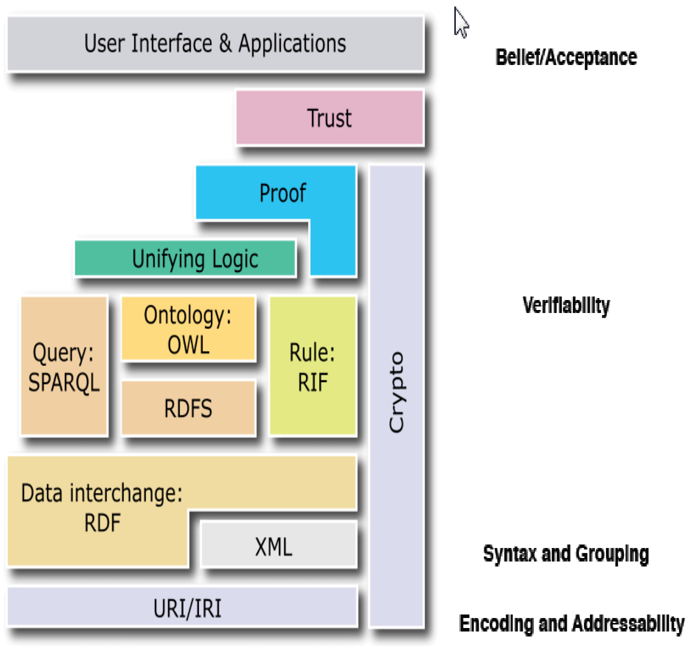


Figure 2.7.2 Semantic Web Stack

However Web applications are being developed using the semantic web standards such standards involve W3C which is developing semantic web infrastructures and standards through its many semantic web activities.

The Semantic Web is generally built on syntaxes which use URIs to represent data, usually in triples based structures: i.e. many triples of URI data that can be held in databases, or interchanged on the World Wide Web using a set of particular syntaxes developed especially for the task. These syntaxes are called "Resource Description Framework" syntaxes.

2.7.1. URI - Uniform Resource Identifier

A URI is simply a string starting with "http:" or "ftp:" that you often find on the World Wide Web, used to identify a name or a resource. Such identification enables interaction with representations of the resource over a network (typically the World Wide Web) using specific protocols.

Here we are discussing the URIs for Web Documents. Taking an example of a company named as ABC. Assume that ABC Inc., the fictional company from many Williams Brothers, has a web site at http://www.abc.com/.

Imagine, part of the site is a white-pages service listing the names and contact details of the employees. Alice and Bob both work at ABC. The structure of ABC's web site might thus be:

http://www.abc.com/ – the homepage of ABC, Inc.
http://www.abc.com/people/alice – the homepage of Alice
http://www.abc.com/people/bob – the homepage of Bob

Like everything on the traditional web, these are web documents. Every web document has its own URI. Note that a web document is not the same as a file: A single web document can be available in many different formats and languages, and a single file, for example a PHP script, may be responsible for generating a large number of web documents with different URIs. On the traditional web, URI's are used only for web documents – to link to them, and to access them in a browser.

2.7.2. RDF - Resource Description Framework

Resource Description Framework (RDF) is a semantic web standard as well as a language for representing information about resources in the World Wide Web. This is also certified by W3C.

The RDF data model (12) is similar to classic conceptual modeling approaches such as Entity-Relationship or Class diagrams, as it is based upon the idea of making statements about web resources in the form of subject-predicate-object expressions. These expressions are known as *triples* in RDF terminology. The subject denotes the resource, and the predicate denotes aspects of the resource and expresses a relationship between the subject and the object.

Let’s take an example that someone says that “the grass is green”. We will represent the notion in RDF as a triple: a subject denoting "the grass", a predicate denoting "has the color", and an object denoting "green". However RDF is an abstract model with several serialization formats. It’s simple data model and ability to model disparate, abstract concepts has also led to its increasing use in knowledge management applications unrelated to Semantic Web activity. In practice, RDF data is often persisted in relational database or native representations also called Triple stores.

However the goals of RDF are extensive and the potential opportunities are enormous. So a discussion of the functionality of RDF and an overview of the model, schema and syntactic considerations of this framework follow.

A triple can simply be described as three URIs. Since RDF is a generic format, which already has many parsers. XML RDF is quite a verbose specification, and it can take some getting used to (for example, to learn XML RDF properly, you need to understand a little about XML and namespaces beforehand...), but let's take a quick look at an example of XML RDF right now:-

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:foaf="http://xmlns.com/0.1/foaf/" >
    <rdf:Description rdf:about="">
        <dc:creator rdf:parseType="Resource">
            <foaf:name>Sean B. Palmer</foaf:name>
        </dc:creator>
        <dc:title>The Semantic Web: An Introduction</dc:title>
    </rdf:Description>
</rdf:RDF>
```

This piece of RDF basically says that this article has the title "The Semantic Web: An Introduction", and was written by someone whose name is "Sean B. Palmer". RDF has three notions for writing into triples.

2.7.2.1. Notation3

Notation3, or N3 as it is more commonly known, is a serialization of Resource Description Framework models, designed with human-readability in mind: N3 is much more compact and readable than XML RDF notation. The format is being developed by Tim Berners-Lee and others from the Semantic Web community.

N3 has several features that go beyond a serialization for RDF models, such as support for RDF-based rules. Here are the triples that XML/RDF produces:-

```
@prefix : <http://www.w3.org/2000/10/swap/log#> .
     @prefix dc: <http://purl.org/dc/elements/1.1/> .
     @prefix foaf: <http://xmlns.com/0.1/foaf/> .
     @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

    <>     dc:creator  [
             foaf:name "Sean B. Palmer" ];
         dc:title "The Semantic Web: An Introduction" .
```

2.7.2.2. Turtle

Turtle is a simplified, RDF-only subset of N3. Turtle (Terse RDF Triple Language) is a serialization format for RDF graphs. Turtle doesn't go beyond RDF's graph model. The SPARQL Protocol and RDF Query Language uses a similar N3 subset to Turtle for its graph patterns, but using N3's braces syntax for delimiting sub graphs.

Turtle has no official status with any standards organization, but has become popular amongst Semantic Web developers as a human-friendly alternative to RDF/XML.

2.7.2.3. N-triples

N-Triples, is a format for storing and transmitting data. It is a line-based, plain text serialization format for RDF graphs. N-Triples was designed to be a simpler format than Notation 3 and Turtle, and therefore easier for software to parse and generate. However, because it lacks some of the shortcuts provided by other RDF serializations (such as CURIEs and nested resources, which are provided by both RDF/XML and Turtle) it can be difficult to type out large amounts of data by hand, and difficult to read.

```
<> <http://purl.org/dc/elements/1.1/creator> _:ag0 .
    <http://purl.org/dc/elements/1.1/title> "The Semantic Web: An
Introduction".
     _:ag0     <http://xmlns.com/0.1/foaf/name> "Sean B. Palmer" .
```

2.7.2.4. Why RDF?

When people are confronted with XML RDF for the first time, Firstly, the benefit that one gets from drafting a language in RDF is that the information maps *directly* and *unambiguously* to a model, a model which is decentralized, and for which there are many generic parsers already available. This means that when you have an RDF application, you know which bits of data are the semantics of the application, and which bits are just syntactic fluffs. And not only do you know that, *everyone* knows that, often implicitly without even reading a specification because RDF is so well known.

The second part of the twofold answer is that we hope that RDF data will become a part of the Semantic Web, so the benefits of drafting your data in RDF now draws parallels with drafting your information in HTML in the early days of the Web.

The answer to "do we use XML Schema in conjunction with RDF?" is almost as brief. XML Schema is a language for restricting the *syntax* of XML applications.

2.7.3. RDF Schema

RDF Schemas are used to declare vocabularies, the sets of semantics property-types defined by a particular community. RDF schemas define the valid properties in a given RDF description, as well as any characteristics or restrictions of the property-type values themselves. The XML namespace mechanism serves to identify RDF Schemas.

A human and machine-process able description of an RDF schema may be accessed by de-referencing the schema URI. If the schema is machine-process able, it may be possible for an application to learn some of the semantics of the property-types named in the schema. To understand a particular RDF schema is to understand the semantics of each of the properties in that description. RDF schemas are structured based on the RDF data model. Therefore, an application that has no understanding of a particular schema will still be able to parse the description into the property-type and corresponding values and will be able to transport the description intact (e.g., to a cache or to another application).

RDF Schema was designed to be a simple data typing model for RDF. Using RDF Schema, we can say that "Fido" is a type of "Dog", and that "Dog" is a sub class of animal. We can also create properties and classes, as well as doing some slightly more "advanced" stuff such as creating ranges and domains for properties.

RDF Schema give us are the "Resource" (rdfs:Resource), the "Class" (rdfs:Class), and the "Property" (rdf:Property).

For example, all terms in RDF are types of resource. To declare that something is a "type" of something else, we just use the rdf:type property:-

```
rdfs:Resource rdf:type rdfs:Class .
rdfs:Class rdf:type rdfs:Class .
rdf:Property rdf:type rdfs:Class .
rdf:type rdf:type rdf:Property .
```

This simply says that "Resource is a type of Class, Class is a type of Class, Property is a type of Class, and type is a type of Property". These are all true statements.

It is quite easy to make up your own classes. For example, let's create a class called "Dog", which contains all of the dogs in the world:-

```
:Dog rdf:type rdfs:Class .
```

Now we can say that "Fido is a type of Dog":-

```
:Fido rdf:type :Dog .
```

We can also create properties quite easily by saying that a term is a type of rdf:Property, and then use those properties in our RDF:-

```
:name rdf:type rdf:Property .
:Fido :name "Fido" .
```

Why have we said that Fido's name is "Fido"? Because the term ":Fido" is a URI, and we could quite easily have chosen any URI for Fido. However, we still have to tell machines that his name is Fido, because although people can guess that from the URI (even though they probably shouldn't), machines can't.

RDF Schema also has a few more properties that we can make use of: rdfs:subClassOf and rdfs:subPropertyOf. These allow us to say that one class or property is a sub class or sub property of another. For example, we might want to say that the class "Dog" is a sub class of the class "Animal". To do that, we simply say:-

```
:Dog rdfs:subClassOf :Animal .
```

Hence, when we say that Fido is a Dog, we are also saying that Fido is an Animal. We can also say that there are other sub classes of Animal:-

```
:Human rdfs:subClassOf :Animal .
:Duck rdfs:subClassOf :Animal .
```

You can see that RDF Schema is very simple, and yet allows one to build up knowledge bases of data in RDF very quickly.

The next concepts which RDF Schema provides us, which are important to mention, are ranges and domains. Ranges and domains let us say what classes the subject and object of each property must belong to.

2.7.4. The Web Ontology Language

The Web Ontology Language (OWL) facilitates greater machine interoperability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics.

The Web Ontology Language (OWL) is a component of the *Semantic Web* activity. The Web Ontology Language (OWL) is a family of knowledge representation languages for defining web ontology. The language extends by web semantic standards RDF and RDFS. It is designed to support different levels of expression with different computational requirements. Its primary aim is to bring the expressive and reasoning power of description logic to the semantic web. It is certified by World Wide Web Consortium (W3C).

The W3C has played an important role in an increasing number and range of applications built by using OWL, and is the focus of research into tools, reasoning techniques, formal foundations and language extensions. This level of experience with OWL means that the community is now in a good position to discuss how OWL to be applied, adapted and extended to fulfill current and future application demands.

The OWL Web Ontology Language is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications. Thus it provides us the way to:

- Formalize a domain by defining classes and properties of those classes,
- Define individuals and assert properties about them, and
- Reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language.

Unfortunately, DL does not describe everything from RDF. For example, the classes of classes are not permitted in the DL, and some of the triple expressions would have no sense in DL. That is why OWL can be only syntactic extension of RDF/RDFS (note that RDFS is both syntactic and semantic extension of RDF). To partially overcome this problem, and also to allow layering within OWL, three species of OWL are defined.

2.7.4.1. OWL Lite

OWL Lite was originally intended to support users whose primary need is a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It was hoped that it would be simpler to provide tool support for OWL Lite than its more expressive relatives, allowing quick migration path for systems utilizing some taxonomies.

However, in practice the most of the expressiveness constraints placed on OWL Lite amount to little more than syntactic inconveniences: most of the constructs available in OWL DL can be built using complex combinations of OWL Lite features. Development of OWL Lite tools has thus proven almost as difficult as development of tools for OWL DL, and OWL Lite is not widely used.

2.7.4.2. OWL DL

OWL DL was designed to provide the maximum expressiveness possible while retaining computational completeness (either ϕ or ¬ϕ belong), decidability (there is an effective procedure to determine whether ϕ is derivable or not), and the availability of practical reasoning algorithms.

OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, number restrictions may not be placed upon properties which are declared to be transitive). OWL DL is so named due to its correspondence with description logic, a field of research that has studied the logics that form the formal foundation of OWL.

2.7.4.3. OWL Full

OWL Full has no expressiveness constraints, but also does not guarantee any computational properties. It is formed by the full OWL vocabulary, but does not no impose any syntactic constrains, so that the full syntactic freedom of RDF can be used.

The OWL is defined in these three species because as

- every legal OWL Lite ontology is a legal OWL DL ontology
- every legal OWL DL ontology is a legal OWL Full ontology
- every valid OWL Lite conclusion is a valid OWL DL conclusion
- and every valid OWL DL conclusion a valid OWL Full conclusion.

The inverses of these relations generally do not hold. However every OWL ontology is a valid RDF document (i.e., DL expressions are mapped to triples), but not all RDF documents are valid OWL Lite or OWL DL documents.

Below is an example of OWL ontology expressed in triples and serialized in N3. "Pizza has PizzaBase as its base; Pizza is disjoint with PizzaBase; NonVegetarianPizza is exactly Pizza that is not VegetarianPizza; isIngredientOf is a transitive property; isIngredientOf is inverse of hasIngredient". The example expressed in follows:

Syntax:

```
@prefix :     <http://example.com/pizzas.owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .

:Pizza rdfs:subClassOf
        [ a owl:Restriction ;
          owl:onProperty :hasBase ;
          owl:someValuesFrom :PizzaBase ] ;
       owl:disjointWith :PizzaBase .

:NonVegetarianPizza  owl:equivalentClass
        [  owl:intersectionOf
            (  [owl:complementOf :VegetarianPizza]
               :Pizza  )  ] .

:isIngredientOf
      a owl:TransitiveProperty , owl:ObjectProperty ;
      owl:inverseOf :hasIngredient .
```

2.7.5. SPARQL

SPARQL- simple protocol and RDF query language is proposed by W3C for querying RDF data published on the web, either stored natively or vied via middle ware. SPARQL offers a syntactically SQL-like language for querying RDF graphs via pattern matching, as well as simple communication protocol that can be used by the clients for issuing SPARQL queries against endpoints.

SPARQL, which is both a query language and a data access protocol, has the ability to become a key component in Web 2.0 applications: as a standard backed by a flexible data model, it can provide a common query mechanism for all Web 2.0 applications. SPARQL should be of interest to developers exploring the available options for publishing open data on the Web.

Work on RDF query languages has been progressing for a number of years. Several different approaches have been tried similar to SQL-style syntaxes. SPARQL actually consists of three separate specifications. The query language specification makes up the core. But alongside it sits the query results XML format which, as you might guess, describes an XML format for serializing the results of a SPARQL SELECT (and ASK) query. This simple format is easily process able with common XML tools such as XSLT. The third specification is the data access protocol which uses WSDL 2.0 to define simple HTTP and SOAP protocols for remotely querying RDF databases.

However there are a number of issues that SPARQL does not address yet; most notably, SPARQL is read-only and cannot modify an RDF dataset.
Happily the SPARQL specifications don't exist in isolation. There are several tools and APIs that already provide SPARQL functionality, and most of them are up to date with the latest specifications. A brief list includes:

- ARQ, a SPARQL processor for Jena
- Rasqal, the RDF query library
- Twinql, a SPARQL processor for Lisp
- Pellet, an open source OWL DL reasoner in Java, that has partial SPARQL query support
- KAON2, another OWL DL reasoner that has partial SPARQL support.

Most forms of SPARQL query contain a set of RDF triple patterns called a *basic graph pattern*. The example below shows a SPARQL query to find the title of a book from the given data graph. The query consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The basic graph pattern in this example consists of a single triple pattern with a single variable (?title) in the object position.

Data
```
 <http://example.org/book/book1>      <http://purl.org/dc/elements/1.1/title>
"SPARQL Tutorial" .
```

Query
```
SELECT ?title
WHERE
{    <http://example.org/book/book1>  <http://purl.org/dc/elements/1.1/title>
?title . }
```

Query Result
```
title "SPARQL Tutorial"
```

2.7.6. The Power Of Semantic Web Languages

The power of Semantic Web languages is that anyone can create application for publishing some RDF that describes a set of URIs. As we use URIs for each of the terms in our semantic web standard languages, we can publish the languages easily without fear that they might get misinterpreted or stolen, and with the knowledge that anyone in the world has a generic RDF processor can use them.

We can measure the power on a principle of least power: the lesser the rules, the better. This means that the Semantic Web is essentially very unconstraining in what it lets one say, and hence it follows that anyone can say anything about anything. When you look at what the Semantic Web is trying to do, it becomes very obvious why this level of power is necessary… if we started constraining people, they wouldn't be able to build a full range of applications, and the Semantic Web would therefore become useless to some people.

2.7.7. Relational DB to RDF framework

Real-world applications store data in relational databases. Such RDBs are faster and reliable. With the growth of semantic web we need to express relational databases in a form and language *that may be machine process able.* To make the data process able by the machines is the aim of semantic web.

The purpose of using relational databases is to store, manage and retrieve data. Then some mythology will be used to express these RDBs to RDF which is the standard semantic web language. Below shown is the complete mapping process of relational data to RDF.

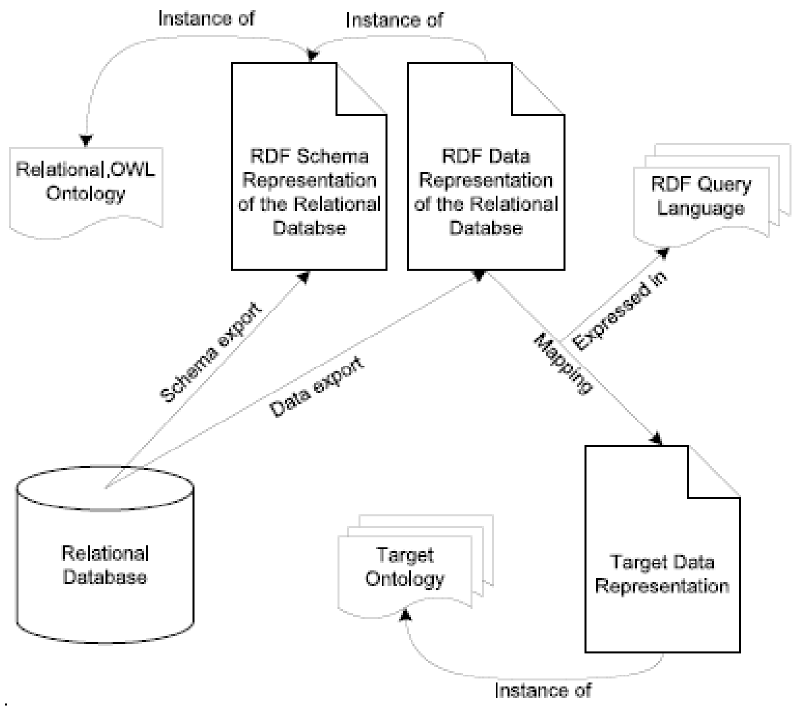


Figure 2.7.3 Mapping Process

The above mapping process has two main steps. First, the Relational.OWL, which represents the schema of RDB. The reference OWL classes describe the tables in a database. Ontology will be constructed by specifying the possible relationships among OWL classes. The schema will then be an instance of the Relational.OWL ontology. In turn, the data items converted become instances of the schema ontology just created. The schema and data components are translated statically in a using a virtual representation of that RDF model, e.g. with RDQuery (13).

When Relational.OWL schema has been created which represents the relational database, the second step including the actual mapping can be performed. The RDF model can be queried with an RDF query language e.g. SPARQL. As long as the query language is closed, the resulting query response is again within the Semantic Web, i.e. it is a valid RDF model or graph and may then be processed by other Semantic Web applications using their own built-in functionality for reasoning tasks.

2.8. Publishing Relational Artifacts on Semantic Web

As Semantic Web technologies are getting mature, there is a growing need for RDF applications to access the content of huge, live, non- RDF, legacy databases without having to replicate the whole database into RDF.

2.8.1. D2RQ

D2RQ is a declarative language to describe mappings between relational database schemata and OWL/RDFS ontologies. It takes a relational database schema as input and presents an RDF interface of the same as output using Jena API's. A D2RQ graph wraps one or more local relational databases into a virtual, read only RDF graph. D2RQ rewrites RDQL queries and Jena API calls into application data model-specific SQL queries. The result sets of these SQL queries are transformed into RDF triples which are passed up to the higher layers of the Jena framework.

The D2RQ mapping language builds on experience gained with D2R. The central object within D2RQ is the ClassMap. A ClassMap represents a class or a group of similar classes from the ontology. It specifies whether instances are identified by using URI column values from the database, by using an URI pattern together with the primary key values or by using blank nodes.

The complete D2RQ application architecture is shown below.

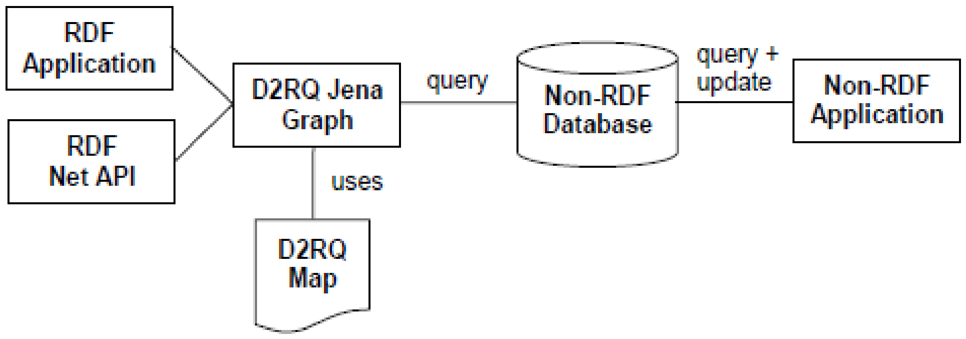


Figure 2.8.1 D2RQ Application Architecture

The D2RQ, Semantic Web applications can:

- Query a non-RDF database using RDQL,
- Publish the content of a non-RDF database on the Semantic Web using the RDF Net API,
- Do RDFS and OWL inferencing over the content of a non-RDF database using the Jena ontology API,
- Access information in a non-RDF database using the Jena model API.

2.8.2. D2R Server

D2R Server is a tool for publishing relational databases on the Semantic Web. It enables RDF and HTML browsers to navigate the content of the database, and allows applications to query the database using the SPARQL query language. Database content is mapped to RDF by a declarative mapping which specifies how resources are identified and how property values are generated from database content. Based on this mapping, D2R Server allows Web agents to retrieve RDF and XHTML representations of resources and to query non-RDF databases using the SPARQL query language over the SPARQL protocol.

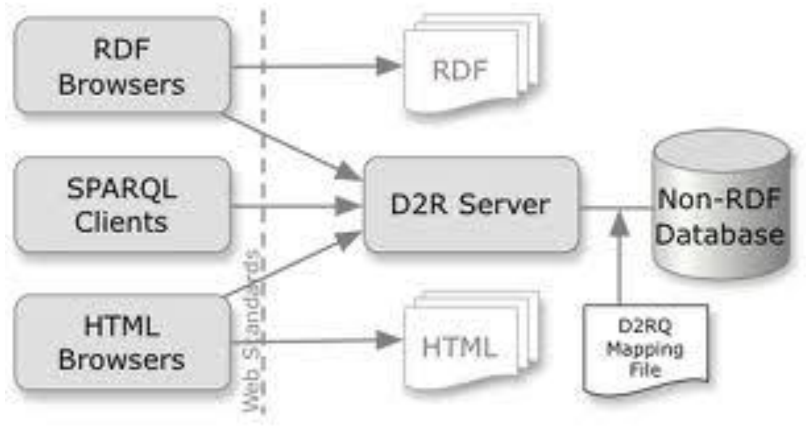


Figure 2.8.2 D2R Server

D2R Server uses the D2RQ mapping language to capture mappings between application specific database schemas and RDFS schemas or OWL ontologies. A D2RQ mapping specifies how resources are identified and how property values are generated from database content. D2R Server includes a tool that automatically generates a D2RQ mapping from the table structure of a database. The tool generates a new RDF vocabulary for each database, using table names as class names and column names as property names. The mapping can be customized afterwards by substituting auto-generated terms with terms from well-known RDF vocabularies.

2.8.3. Virtuoso Universal Server

Virtuoso Universal Server is the OpenLink Virtuoso server for serving RDF data via a Linked Data interface and a SPARQL endpoint. RDF data can either be stored directly in Virtuoso or can be created on the fly from non-RDF relational databases based on a mapping.

2.8.4. Pubby

The Pubby server can be used as an extension to any RDF store that supports SPARQL. Pubby rewrites URI requests into SPARQL and describe queries against the underlying RDF store. Besides RDF, Pubby also provides a simple HTML view over the data store and takes care of handling 303 redirects and content negotiation between the two representations. Pubby can be used to add Linked Data interfaces to SPARQL endpoints.

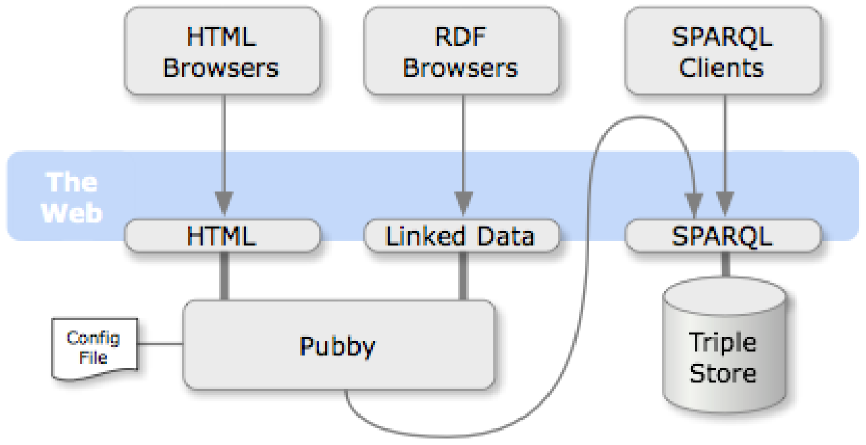


Figure 2.8.3 Pubby Server

2.9. Linked data

Linked Data is about using the Web to connect the related data that was not previously linked. More specifically we can say that sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF. Linked Data embodies an important goal of the Semantic Web, to create a web of data which can be queried like a global database as well as traversed by the links connecting the individual data elements.

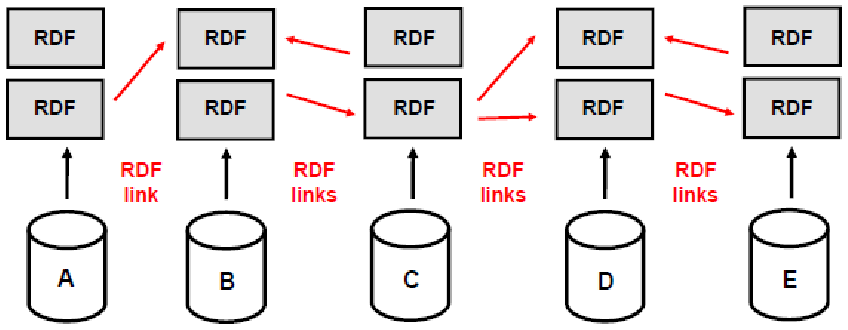


Figure 2.9.1 Linked Data

This web of data is published on standards like HTTP, URIs and RDF and is based on the following design principles:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs. So that they can discover more things.

"By publishing data according to Web standards and Linked Data principles others are empowered to create the shiny interfaces that are meaningful and useful to them (14).

The first rule is to identify things with URIs. It is understandable by most of the people involved in semantic web technology. If we do not use the universal URI set of symbols, we don't call it Semantic Web.

The second rule is to use HTTP URIs. This is the only deviating thing since the web started, a constant tendency for people to invent new URI schemes (and sub-schemes within the urn: scheme) such as LSIDs and handles and XRIs and DOIs and so on, for various reasons.

The third rule is that one should serve information on the web against a URI, is, in 2006, well followed for most ontologies, but, for some reason, not for some major datasets. One can, in general, look up the properties and classes one finds in data, and get information from the RDF, RDFS, and OWL ontologies including the relationships between the terms in the ontology.

The basic format here for RDF/XML, with its popular alternative serialization N3 (or Turtle). Large datasets provide a SPARQL query service, but the basic linked data should be provided as well.

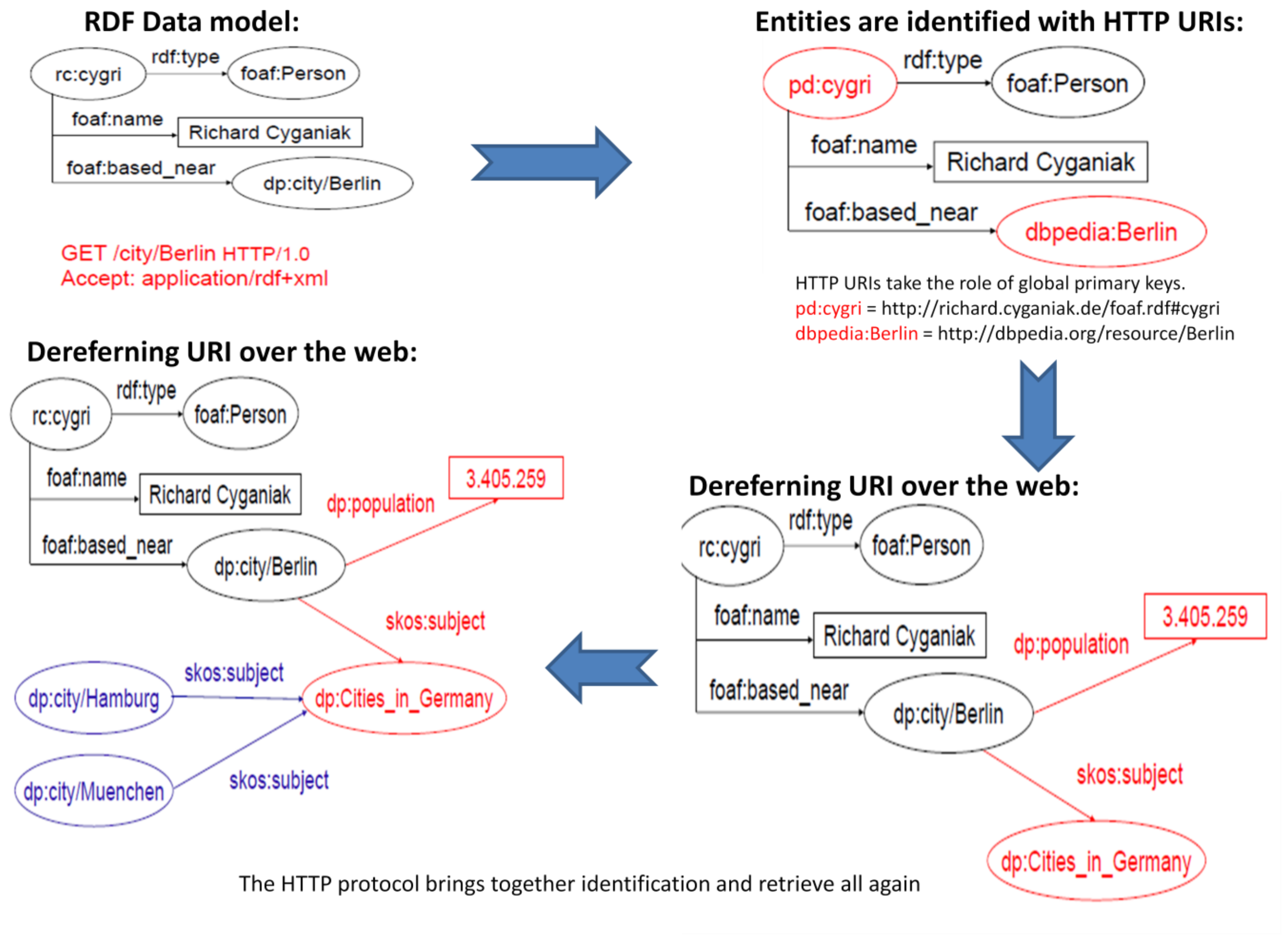
The fourth rule is to make links elsewhere. This is necessary to connect the data we have into a web, a serious, unbounded web in which one can find all kinds of things that just as on the hypertext web we have managed to build. So let's look at the ways of linking data, starting with the simplest way of making a link.

For publishing Linked Data on the Web, the information about resources is represented by using the RDF. RDF provides a data model that is extremely simple but strictly tailored towards Web architecture.

Below is an example that uses RDF browser to display information about Richard from his FOAF profile. Richard has identified himself with the URI.

http://richard.cyganiak.de/foaf.rdf#cygri

When the user types the above URI into the navigation bar, the browser dereferences this URI over the Web, asking for content type application/rdf+xml and displays the retrieved information. In his profile, Richard has stated that he is based near Berlin, using the DBpedia URI http://dbpedia.org/resource/Berlin as URI alias for the non-information resource Berlin. As the user is interested in Berlin, he instructs the browser to dereference this URI by clicking on it. The browser now dereferences this URI asking for application/rdf+xml.



RDF Data model:
rc:cygri
rdf:type
foaf:Person
foaf:name
Richard Cyganiak
foaf:based_near
dp:city/Berlin
GET /city/Berlin HTTP/1.0
Accept: application/rdf+xml
Entities are identified with HTTP URIs:
pd:cygri
dbpedia:Berlin
HTTP URIs take the role of global primary keys.
pd:cygri = http://richard.cyganiak.de/foaf.rdf#cygri
dbpedia:Berlin = http://dbpedia.org/resource/Berlin
Derefering URI over the web:
dp:population
3.405.259
skos:subject
dp:Cities_in_Germany
dp:city/Hamburg
dp:city/Muenchen
The HTTP protocol brings together identification and retrieve all again

Figure 2.9.2 Linked data being created

After being redirected with a HTTP 303 response code, the browser retrieves an RDF graph describing Berlin in more detail. A part of this graph is shown in the third step of the figure above. The graph contains a literal triple stating that Berlin has 3.405.259 inhabitants and another RDF link to a resource representing a list of German cities. Therefore in the same step, as both RDF graphs share the same URI http://dbpedia.org/resource/Berlin, they are naturally merged together.

The user might also be interested in other German cities. Therefore he lets the browser dereference the URI identifying the list. The retrieved RDF graph contains more RDF links to German cities, for instance, Hamburg and München as shown in the fourth step of the figure above. So this is how RDF graphs being presented by using the RDF browsers.

2.9.1. Linked data browsers

2.9.1.1. Tabulator Browser (MIT, USA)

This is one possible form of a semantic web browser. It works with Firefox as add on. The Tabulator project is a generic data browser and editor. Using outline and table modes, it provides a way to browse RDF data on the web. RDF is the standard for inter-application data exchange.

2.9.1.2. Zitgist's RDF Browser

This is new tool from Zitgist for browsing information available on the Semantic Web. Like other semantic web browsers this browser also depends on the data available for a given thing (a resource), its user interface is pity simple so that the data is best displayed for a better understanding of its semantic.

The core of this new RDF browser is its templating system. Users only have to put the URI of a resource (it can be a URL where the browser can find RDF information about this Thing), then pressing the "browse" button. Depending on the information available about this Thing, the RDF browser will shape its interface to optimize users' browsing experience with the data.

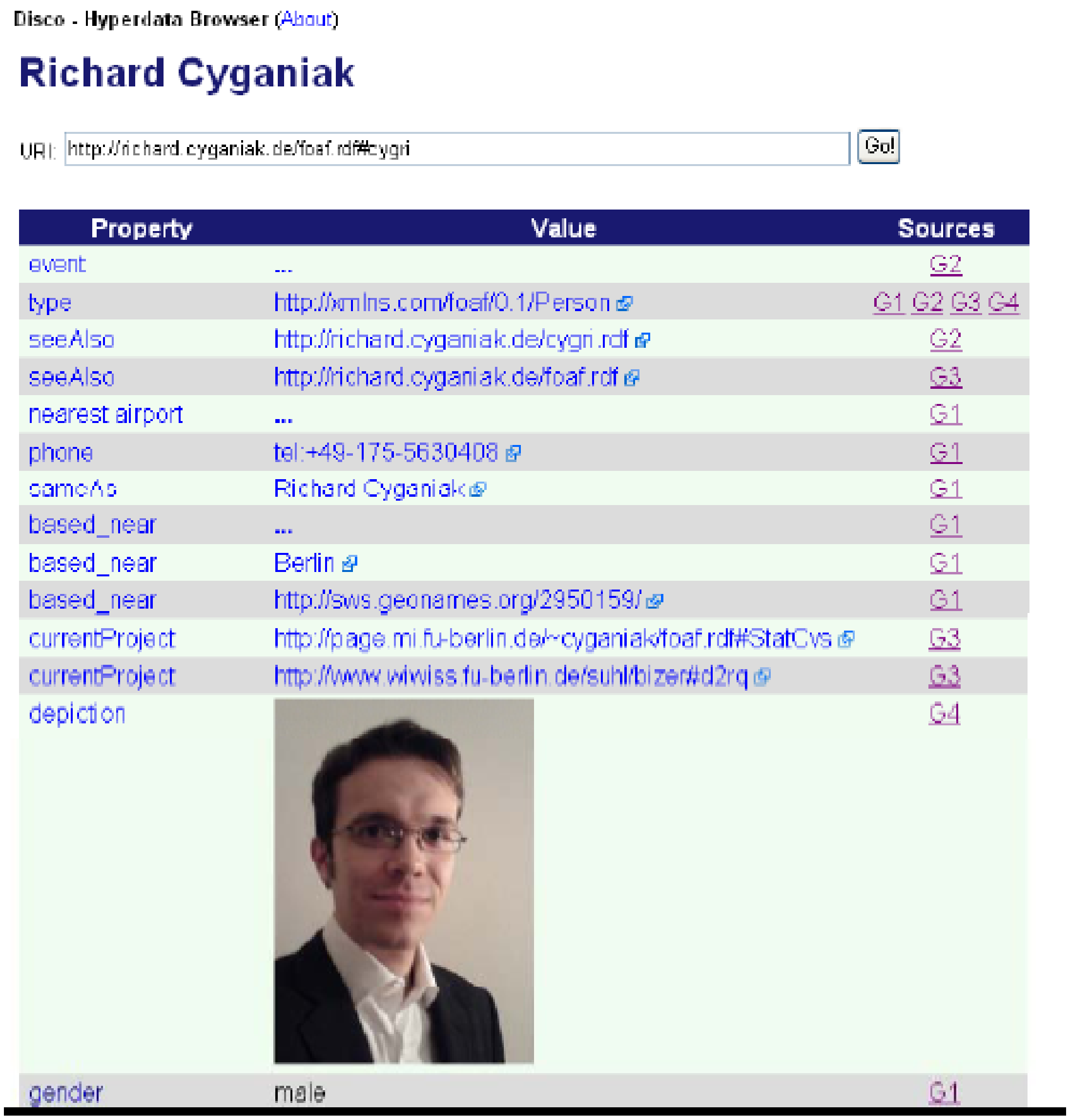
Data displayed in the Zitgist RDF Browser can come from many different data sources:

- Zitgist's internal RDF datastore
- URI dereferencing
- On-the-fly conversation of data sources such as:
    - Microformats
    - RDFa
    - eRDF
    - HTML meta tags
    - API data source such as: Amazon.com, Google Base, etc.

So, depending on what information is available for a given URI, the browser will mesh-up these data sources and displays the information to the user.

2.9.1.3. Disco Hyperdata Browser (FU Berlin, DE)

The Disco - Hyperdata Browser is a simple browser for navigating the Semantic Web. The browser renders all information that it can find on the Semantic Web about a specific resource, as an HTML page. This resource description contains hyperlinks that allow you to navigate between resources. While you move from resource to resource, the browser dynamically retrieves information by dereferencing HTTP URIs and by following rdfs:seeAlso links.



Disco - Hyperdata Browser (About)

Richard Cyganiak

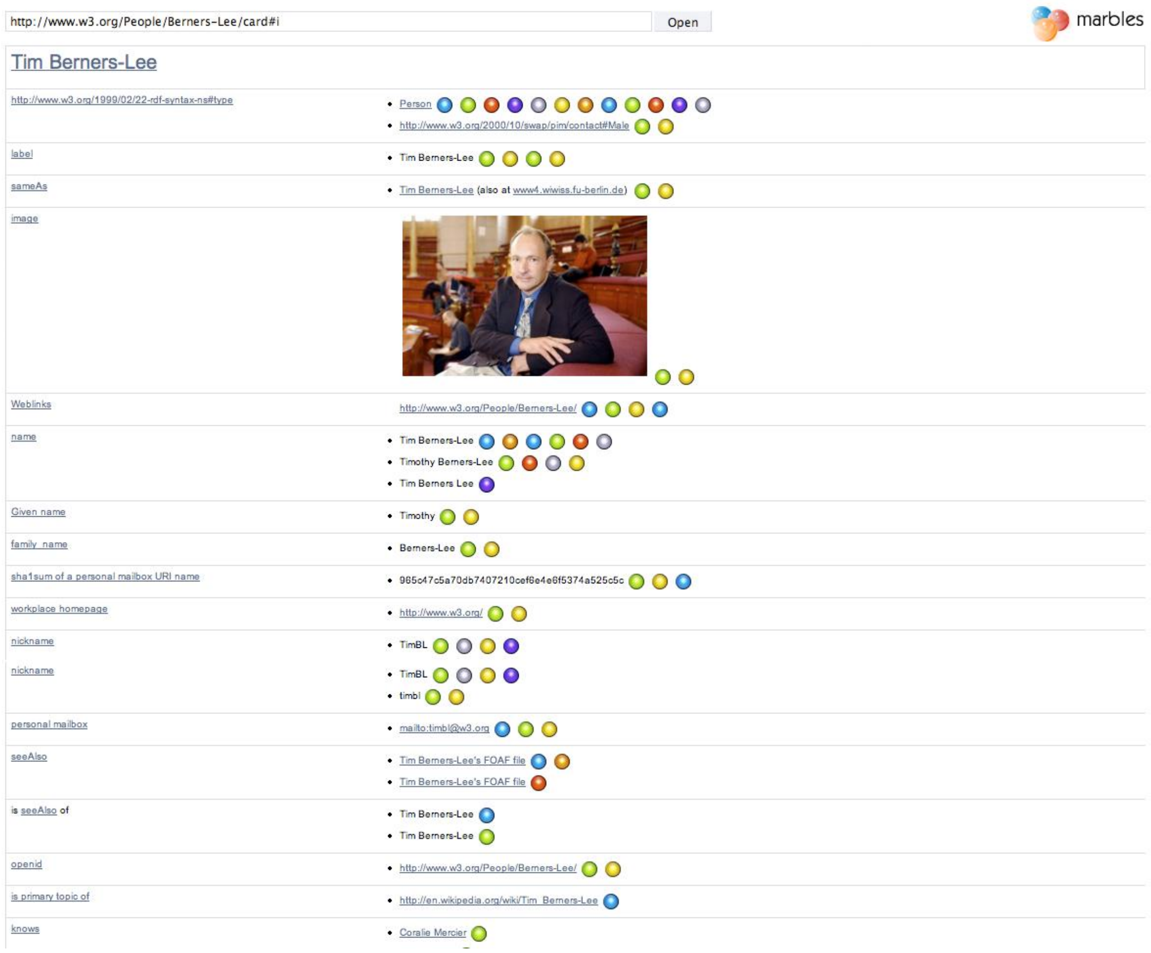
URI: http://richard.cyganiak.de/foaf.rdf#cygri Go!

Property	Value	Sources
event	...	G2
type	http://xmlns.com/foaf/0.1/Person	G1 G2 G3 G4
seeAlso	http://richard.cyganiak.de/cygri.rdf	G2
seeAlso	http://richard.cyganiak.de/foaf.rdf	G3
nearest airport	...	G1
phone	tel:+49-175-5630408	G1
sameAs	Richard Cyganiak	G1
based_near	...	G1
based_near	Berlin	G1
based_near	http://sws.geonames.org/2950159/	G1
currentProject	http://page.mi.fu-berlin.de/~cyganiak/foaf.rdf#StatCvs	G3
currentProject	http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rq	G3
depiction		G4
gender	male	G1

Figure 2.9.3 Disco Hyperdata Browser

2.9.1.4. Marbles (FU Berlin, DE)

Similarly like other Web browsers marbles allow users to navigate between HTML pages by hypertext links and like Linked Data browsers allow users to navigate between data sources by links expressed as RDF triples.



Tim Berners-Lee

http://www.w3.org/1999/02/22-rdf-syntax-ns#type	• Person ● ● ● ● ● ● ● ● ● ● ● ● • http://www.w3.org/2000/10/swap/pim/contact#Male ● ●
label	• Tim Berners-Lee ● ● ● ●
sameAs	• Tim Berners-Lee (also at www4.wiwiss.fu-berlin.de) ● ●
image	● ●
Weblinks	http://www.w3.org/People/Berners-Lee/ ● ● ● ●
name	• Tim Berners-Lee ● ● ● ● ● ● • Timothy Berners-Lee ● ● ● ● • Tim Berners Lee ●
Given name	• Timothy ● ●
family_name	• Berners-Lee ● ●
sha1sum of a personal mailbox URI name	• 965c47c5a70db7407210cef6e4e6f5374a525c5c ● ● ●
workplace homepage	• http://www.w3.org/ ● ●
nickname	• TimBL ● ● ● ●
nickname	• TimBL ● ● ● ● • timbl ● ●
personal mailbox	• mailto:timbl@w3.org ● ● ●
seeAlso	• Tim Berners-Lee's FOAF file ● ● • Tim Berners-Lee's FOAF file ●
is seeAlso of	• Tim Berners-Lee ● • Tim Berners-Lee ●
openid	• http://www.w3.org/People/Berners-Lee/ ● ●
is primary topic of	• http://en.wikipedia.org/wiki/Tim_Berners-Lee ●
knows	• Coralie Mercier ●

Figure 2.9.4 Marble Linked Data browser

Figure: The Marbles Linked Data browser displaying data about Tim Berners-Lee.The colored dots indicate the data sources from which data was merged.

2.9.1.5. Other browsers

- OpenLink RDF Browser (OpenLink, UK)
- Humboldt (HP Labs, UK)
- Fenfire (DERI, Irland)

2.9.2. Linking

As discussed earlier that for linked data we will URIs related to a single real-world object – resource identifier, RDF document URL, HTML document URL and these should be explicitly linked with each other to help information that consumers understand their relation.

Let’s assume ABC Inc. wants to publish contact data of their employees on the semantic web.

This is the company of two brothers. As described in section of URI for documents. We will have the following URIs.

```
http://www.abc.com/ – the homepage of ABC, Inc.
http://www.abc.com/people/alice – the homepage of Alice
http://www.abc.com/people/bob – the homepage of Bob
```

Now in case of linked data there are three URIs related to Alice:

```
http://www.abc.com/id/alice - Identifier for Alice, the person
http://www.abc.com/people/alice - Alice’s homepage
http://www.abc.com/data/alice - RDF document with description of Alice
```

As an example the RDF document located at http://www.abc.com/data/alice might contain these foaf statements (expressed in N3):

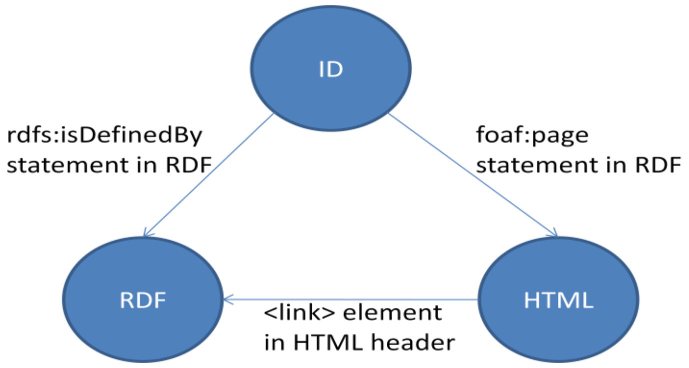
```
<http://www.abc.com/id/alice>
foaf:page <http://www.abc.com/people/alice>;
rdfs:isDefinedBy <http://www.abc.com/data/alice>;
a foaf:Person;
foaf:name "Alice";
foaf:mbox <mailto:alice@acme.com>;
```

The document makes statements about Alice, the person, using the resource identifier. The first two properties relate the resource identifier to the two document URLs. The foaf:page statement links it to the HTML document. This allows RDF-aware clients to find a human-readable version of the resource, and at the same time, by linking the page to its topic, defines useful metadata about that HTML document. The rdfs:isDefinedBy statement links the person to the document containing its RDF description and allows RDF browsers to distinguish this main resource from other auxiliary resources that just happen to be mentioned in the document. We use rdfs:isDefinedBy instead of its weaker superproperty rdfs:seeAlso because the content at /data/alice is authoritative.

The remaining statements are the actual white pages data. The HTML document at http://www.acme.com/people/alice should contain in its header a <link /> element that points to the corresponding RDF document:

```
<html lang="en">
<head>
<title>Alice's Homepage</title>
<link rel="alternate" type="application/rdf+xml"
title="RDF Version"
href="http://www.acme.com/data/alice" />
</head>
```

This allows RDF-aware web clients to discover the RDF information. The approach is recommended in Section 9 of the RDF/XML specification (15). If the information on the web page differs significantly from the RDF version, then we recommend using rel="meta" instead of rel="alternate". The three desired links for each resource are shown in drawing 2.9.2.


ID
rdfs:isDefinedBy statement in RDF
foaf:page statement in RDF
RDF
HTML
<link> element in HTML header

Drawing 2.9.1 The RDF and HTML documents should relate the URIs to each other

2.9.3. Best practices

For publishing the URIs we have two solutions.

- 303 URIs
- Hash URIs

Both of these solutions have their own advantages and disadvantages.

2.9.3.1. 303 URIs

The first solution is to use a special HTTP status code, "303 See Other", to distinguish non-document resources from regular web documents. Since 303 is a redirect status code, the server can also give the location of a document that describes the resource. If, on the other hand, a request is answered with one of the usual status codes in the 2XX range, like 200 OK, then the client knows that the URI identifies a web document. This practice has been embraced by the W3C's Technical Architecture Group in its httpRange-14 ruling (16). If ABC adopts this solution, they could use these URIs to represent the company, Alice and Bob:

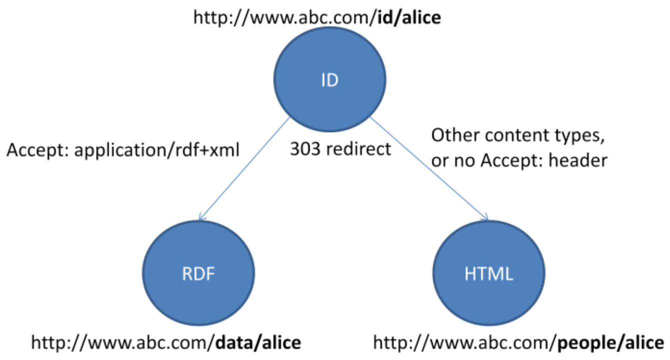
```
http://www.abc.com/id/acme - ACME, the company
http://www.abc.com/id/bob - Bob, the person
http://www.abc.com/id/alice - Alice, the person
```

The web server would be configured to answer requests to all these URIs with a 303 status code and a Location HTTP header that provides the URL of a document that describes the resource, as seen in drawing 2.9.3. The server could employ content negotiation to send either the URL of an HTML description or RDF. Requests for HTML would be redirected to the web page URLs and requests for RDF data would be redirected to RDF documents, such as:

```
http://www.abc.com/data/acme – RDF document describing ABC, the company
http://www.abc.com/data/bob – RDF document describing Bob, the person
http://www.abc.com/data/alice – RDF document describing Alice, the person
```

```
http://www.abc.com/id/alice
http://www.abc.com/data/alice
http://www.abc.com/people/alice
```

303 redirect
Accept: application/rdf+xml Other content types,
or no Accept: header


http://www.abc.com/id/alice
ID
Accept: application/rdf+xml
303 redirect
Other content types, or no Accept: header
RDF
HTML
http://www.abc.com/data/alice
http://www.abc.com/people/alice

Drawing 2.9.2 The 303 URI solution

Each of the RDF documents would contain statements about the appropriate resource, using the original URI, e.g. `http://www.abc.com/id/alice`, to identify the described resource.

2.9.3.2. Hash URIs

The second solution is to use “hash URIs” for non-document resources. URIs can contain a fragment, a special part that is separated from the rest of the URI by a hash symbol (“#”).
When a client wants to retrieve a hash URI, then it is required to strip off the fragment part before requesting the URI from the server. This means a URI that includes a hash cannot be retrieved directly and therefore cannot identify a web document. We can use them to identify other, non-document resources, without creating ambiguity. If ABC adopts this solution, then they could use these URIs to represent the company, Alice, and Bob:

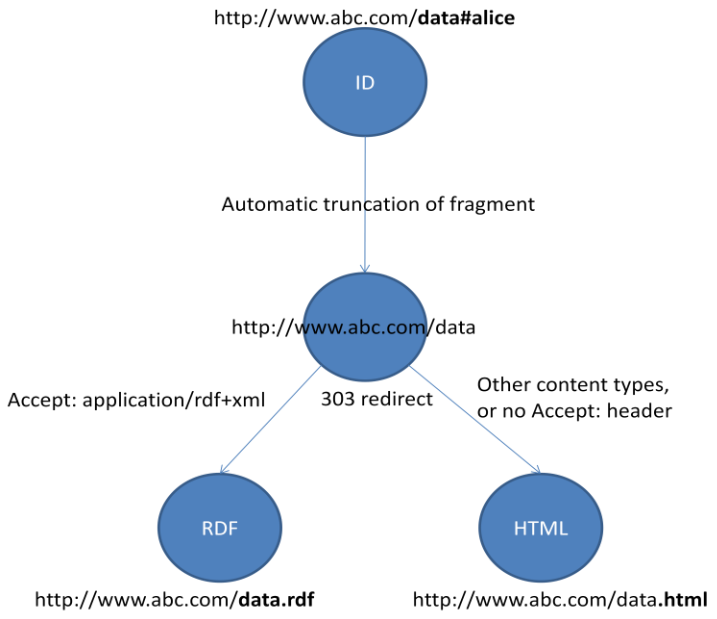
```
http://www.abc.com/data#acme – ABC, the company
http://www.abc.com/data#bob – Bob, the person
http://www.abc.com/data#alice – Alice, the person
```

Clients will always strip off the fragment part before requesting any of these

URIs, resulting in a request to this URI:

```
http://www.abc.com/data – RDF document describing ABC, Bob, and Alice
```

At this URI, ACME could serve an RDF document that contains descriptions of all three resources, using the original hash URIs to identify the resources. Alternatively, content negotiation (see Section 3.16) could be employed to redirect from the data URI to separate HTML and RDF documents. This is a somewhat intricate affair because of the way HTML interacts with fragments:



http://www.abc.com/data#alice
ID
Automatic truncation of fragment
http://www.abc.com/data
Accept: application/rdf+xml
303 redirect
Other content types, or no Accept: header
RDF
HTML
http://www.abc.com/data.rdf
http://www.abc.com/data.html

Drawing 2.9.3 The hash URI solution with content negotiation

The redirect to the HTML document must send a 303 See Other status code (otherwise a client could conclude that the hash URI represents a part of the HTML document). For the redirect to the RDF, 302 Found, 303 See Other, and 301 Temporary Redirect are fine; a recent W3C Best Practices document uses 303 (17). The hash URI solution, with content negotiation, is shown in drawing 2.9.4.

2.9.4. Ontologies

There is no common definition of the term "ontology" itself. However to define an ontology we can say that ontologies share many structural similarities, regardless of the language in which they are expressed. So,

1. Ontology is a term in philosophy and its meaning is “theory of existence''.
2. Ontology is an explicit specification of conceptualization.
3. Ontology is a body of knowledge describing some domain, typically common sense knowledge domain.

Ontology describes individuals (instances), classes (concepts), attributes, and relations. Therefore the common components of ontology include:

- Individuals: instances or objects (the basic or "ground level" objects)
- Classes: sets, collections, concepts, classes in programming, types of objects, or kinds of things
- Attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have
- Relations: ways in which classes and individuals can be related to one another
- Function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement
- Restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as input
- Rules: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form.

In Semantic Web and Linked Data the models are based on the use of RDF, RDFS and OWL which are used to describe the types, properties, and relationship types. So ontologies are commonly known as RDF "vocabularies".

The following is a table of commonly used ontologies, together with their namespaces and preferred prefixes.

Prefix	Namespace	Description
Dc	http://purl.org/dc/elements/1.1/	DCMI Namespace for the Dublin Core Metadata Element Set, Version 1.1
dcterms	http://purl.org/dc/terms/	DCMI Namespace for metadata terms in the http://purl.org/dc/terms/ namespace
event	http://purl.org/NET/c4dm/event.owl#	
foaf	http://xmlns.com/foaf/0.1/	Friend of a Friend (FOAF) vocabulary
geo	http://www.w3.org/2003/01/geo/wgs84_pos#	geo
owl	http://www.w3.org/2002/07/owl#	
Po	http://purl.org/ontology/po/	Programmes ontology
Rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	The RDF Vocabulary (RDF)
rdfs	http://www.w3.org/2000/01/rdf-schema#	The RDF Schema vocabulary (RDFS)
skos	http://www.w3.org/2004/02/skos/core#	
Xsd	http://www.w3.org/2001/XMLSchema#	

Figure 2.9.5 Commonly Used Ontologies and their Namespaces

2.9.4.1. *Dbpedia Ontologies*

DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data. We hope this will make it easier for the amazing amount of information in Wikipedia to be used in new and interesting ways, and that it might inspire new mechanisms for navigating, linking and improving the encyclopaedia itself.

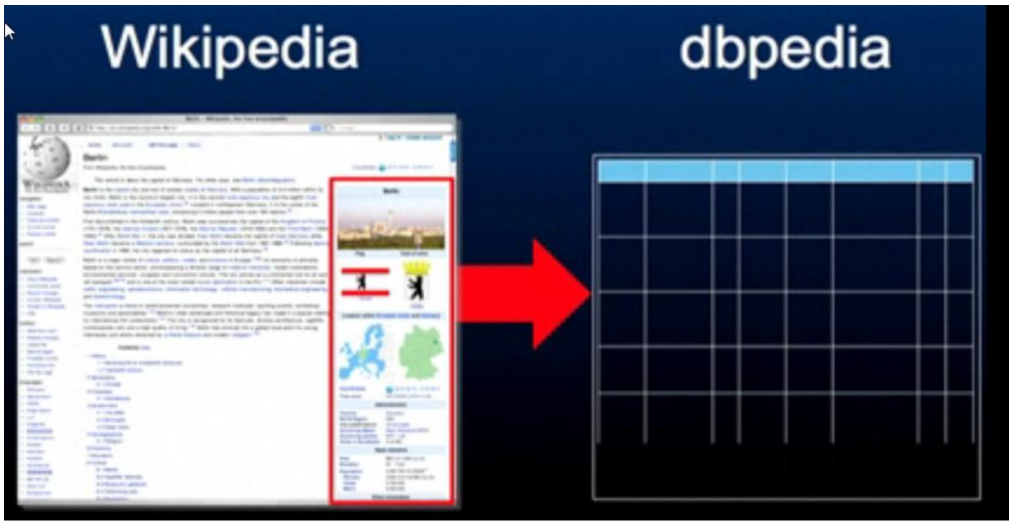


Figure 2.9.6 DBpedia

The DBpedia dataset describes more than 3.5 million things, including at least 213,000 persons, 328,000 places, 57,000 music albums, 36,000 films, 20,000 companies. The dataset features labels and short abstracts for these things in 30 different languages; 609,000 links to images and 3,150,000 links to external web pages; 4,878,100 external links into other RDF datasets, 415,000 Wikipedia categories, and 75,000 YAGO categories.

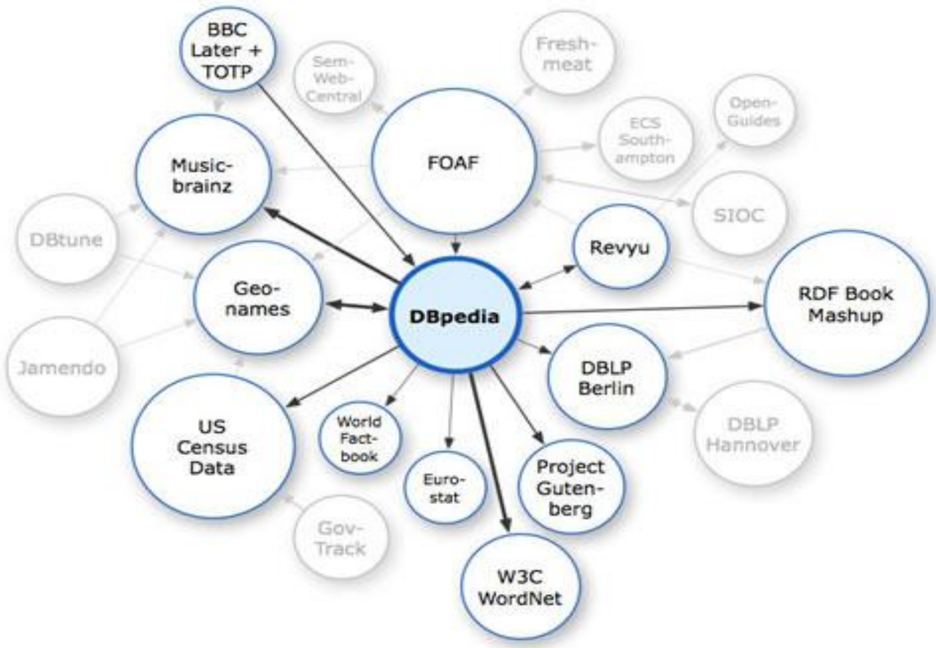


Figure 2.9.7 DBpedia Linking Relationship With Web Datasets

DBpedia Ontology http://dbpedia.org/Ontology "is a shallow, cross-domain ontology, which has been manually created based on the most commonly used infoboxes within Wikipedia. The ontology currently covers over 170 classes which form a subsumption hierarchy and has 940 properties".

There is an overview of the class hierarchy at:

```
http://www4.wiwiss.fu-berlin.de/dbpedia/dev/ontology.htm
```

Top level classes i.e. Subclasses of owl:Thing can be viewed by running in following query in the Dbpedia SPARQL Web form:

```
http://dbpedia.org/snorql/

SELECT * WHERE {
?s rdfs:subClassOf owl:Thing
}
E.g. the Artist class is a sub class of the Person class.
<http://dbpedia.org/ontology/Artist> rdfs:subClassOf
<http://dbpedia.org/ontology/Person> .
```

2.9.4.2. *Yago*

YAGO is a large ontology with high coverage and precision. YAGO has been automatically derived from Wikipedia and WordNet. It comprises entities and relations, and currently contains more than 1.7 million entities and 15 million facts. These include the taxonomic Is-A hierarchy as well as semantic relations between entities. The facts for YAGO have been extracted from the category system and the infoboxes of Wikipedia and have been combined with taxonomic relations from WordNet.

2.10. Managing data streams at semantic level

2.10.1. C-sparql

It is termed as Continuous SPARQL which is a new language for continuous queries over streams of RDF data. The support of streams in RDF format guarantees interoperability and opens up important applications, in which reasoners can deal with knowledge evolving over time. Such modern applications deals with are network monitoring, traffic engineering, sensor networks and financial sector. However processing of data streams has been largely investigated in the last decade (18), specialized data streams management systems has been developed, and their features are becoming supported by major database products , such as ORACLE and DB2. Thus the extension of SPARQL for dealing with streaming data is currently investigated.

Streaming SPARQL (19), C-SPARQL (20) and TA-SPARQL (21) are two proposals for extending SPARQL to stream data management. Both of them introduce the notion of RDF streams as the natural extension of the RDF data model to this scenario, and then extend SPARQL to query RDF-streams. All queries over RDF data streams are denoted as continuous queries, because they continuously produce output in the form of tables of variable bindings or RDF graphs. Here in the start we address the registration of queries that do not produce a stream, but a result that is periodically updated. C-SPARQL queries are registered through the following statement:

```
Registration ‘REGISTER QUERY’ QueryName

[‘COMPUTED EVERY’ Number TimeUnit] ‘AS’ Query
```

The COMPUTED EVERY clause is the same as the one for stream registration. As a very simple example of a registered query that does not generate a stream, consider the following query. For each known user, the query counts the overall number of interactions performed in the last 30 minutes and the number of distinct topics to which the documents refer.

```
REGISTER QUERY GlobalCountOfInteractions COMPUTED EVERY 5m AS
SELECT ?user
COUNT(?document) as ?numberOfInteractions
COUNT(DISTINCT ?topic) as ?numDifferentTopics
FROM  STREAM  <http://streamingsocialdata.org/interact.trdf>  [RANGE  30m  STEP
5m]
WHERE { ?user sd:accesses ?document .
?document sd:describes ?topic . }
GROUP BY { ?user }
```

The query is executed by matching all interactions in the window, grouping them by ?user, and computing the aggregates. The result has the form f a table of bindings that is updated every 5 minutes.

```
REGISTER QUERY GlobalCountOfInteractions COMPUTED EVERY 5m AS
SELECT ?user COUNT(?document) as ?numberOfMovies
FROM    STREAM    <http://streamingsocialdata.org/MoviesJohnsFriendsLike.trdf>
[RANGE 30m STEP 5m]
WHERE { ?user sd:likes ?document }
GROUP BY { ?user }
```

The query above counts among the friends of John, the number of movies that each friend has liked in the last 30 minutes.

The following example shows the construction of a new RDF data stream by selecting all interactions that are of the "likes" type, that are performed by a friend of John, and that concern movies.

```
REGISTER STREAM MoviesJohnsFriendsLike COMPUTED EVERY 5m AS
CONSTRUCT {?user sd:likes ?document}
FROM STREAM <http://streamingsocialdata.org/
interact.trdf> [RANGE 30m STEP 5m]
WHERE { ?user sd:likes ?document .
?user foaf:knows ?john .
?john foat:name "John" .
?document sd:describes ?topic .
?topic skos:subject yago:Movies . }
```

This query uses the same logical conditions as the previous one on static data, but only matches the sd:likes predicate. The output is constructed in the format of a stream of RDF triples. Every query execution may produce from a minimum of zero triples to a maximum of an entire graph. The timestamp is always dependent on the query execution time only, and is not taken from the triples that match the patters in the WHERE clause. Thus, even though in the example the output stream is a restriction of the input stream, a new timestamp is assigned to every triple. Also note that, if the window contains more than one matching triple with a sd:likes predicate, then also the result contains more than one triple, that are returned as a graph. In this case the same timestamp is assigned to all the triples of the graph. In all cases, however, timestamps are system-generated in monotonic non-decreasing order. Results of two evaluations of the previous query are presented in the table below, generating two graphs (one at = 100 and one at = 101).

triple	Timestamp
c:Usr1 sd:likes c:Movie1	t_{100}
c:Usr2 sd:likes c:Movie2	t_{100}
c:Usr1 sd:likes c:Movie2	t_{101}
c:Usr2 sd:likes c:Movie1	t_{101}
c:Usr3 sd:likes c:Movie3	t_{101}

Figure 2.10.1 Triples with Timestamp

The following is another example of C-SPARQL query that given a static description of brokers and a stream of financial transactions for all Swiss brokers, computes the amount of their transactions with in last one hour.

```
1 REGISTER STREAM TotalAmountPerBroker COMPUTE EVERY 10m AS
2 PREFIX ex: <http :// example />
3 CONSTRUCT {? broker ex: hasTotalAmount ? total .}
4 FROM <http :// brokerscentral . org / brokers .rdf >
5 FROM STREAM <http :// stockex .org/ market .trdf >
6 [ RANGE 1h STEP 10m]
7 WHERE {
8 ? broker ex: from ? country .
9 ? broker ex: does ?tx .
10 ?tx ex: with ? amount .
11 FILTER (? country = "CH" )
12 }
13 AGGREGATE { (? total , SUM (? amount ), ? broker ) }
```

At line 1, the REGISTER clause is use to tell the C-SPARQL engine that it should register a continuous query, i.e. a query that will continuously compute answers to the query. In particular, we are registering a query that generates an RDF stream. The COMPUTE EVERY clause states the frequency of every new computation, in the example every 10 minutes. At line 5, the clause FROM STREAM defines the RDF stream of financial transactions, used within the query. Next, line 6 defines the window of observation of the RDF stream. Streams, for their very nature, are volatile and for this reason should be consumed on the y; thus, they are observed through a window, including the last elements of the stream, which changes over time. In the example, the window comprises RDF triples produced in the last 1 hour, and the window slides every 10 minutes. The WHERE clause is standard; it includes a set of matching patterns and FILTER clauses as in standard SPARQL. Finally, at line 13, the AGGREGATE function asks the C-SPARQL engine to include in the result set a new variable ?total which is bound to the sum of the amount of the transaction of each broker.

2.11. Publishing data streams as linked data

We describe the publishing of data on the Web according to the Linked Data principles, data providers add their data to a global data space, which allows data to be discovered and used by various applications. Publishing a data set or data Stream as Linked Data on the Web involves the following three basic steps:

- Assign URIs to the entities described by the data set and provide for dereferencing these URIs over the HTTP protocol into RDF representations.
- Set RDF links to other data sources on the Web, so that clients can navigate the Web of Data as a whole by following RDF links.
- Provide metadata about published data, so that clients can assess the quality of published data and choose between different means of access.

Chapter 3: Problem / Solution

3.1. Overview

The objective of this chapter is to review the problem on our hand and propose a solution for this. If we consider Waterfall model as a standard, then this chapter will cover the Requirement Analysis and Specifications.

3.2. Micro blogging

Micro blogging is a way of communication channel with some considerable potential to improve intra-firm transparency and knowledge sharing. However, the adoption of such social software presents certain challenges to enterprises.

Users and organizations can set up their own micro blogging services. Over the last few years communication patterns have been changed. They are shifted primarily from face-to-face to more online communication like email, IM, text messaging, and other tools. The idea of sharing short messages using multiple access points (e.g. mobile phone, web, instant messaging) seems to be appealing to people worldwide. Currently, more than 1 million users generate a total of 3 million messages per day (TechCrunch, 2008) updating each other about their daily occurrences.However, some argue that email is now a slow and inefficient way to communicate.

For instance, time-consuming 'email chains' can develop, whereby two or more people are involved in lengthy communications for simple matters, such as arranging a meeting. The 'one-to-many' broadcasting offered by micro blogs is thought to increase productivity by circumventing this.

There is an implication of remote collaboration that there are fewer opportunities for face-to-face informal conversations. However, micro blogging has the potential to support informal communication among coworkers. Many individuals like sharing their whereabouts and status updates with micro blogging.

Another reason is that the information shared via micro blogging remains available to a definable population outside the sender-recipient sphere facilitating knowledge storage and exchange as well as providing potential for data mining. Allowing to conveniently sharing daily updates micro blogging helps people to keep in touch. This is of particular value given one of the most consistent findings in the social science literature saying that you know is highly correlated with what you come to know.

Micro blogging is therefore expected to improve the social and emotional welfare of the workforce, as well as streamline the information flow within an organization. It can increase opportunities to share information, help to realize and utilize expertise within the workforce, and help to build and maintain common ground between coworkers. As microblogging use continues to grow every year, it is quickly becoming a core component of Enterprise Social Software.

Micro blogging service is provided my several online platforms like twitter[1], jaiku[2] and most recent is pownce[3].

Recognizing the tremendous success of Twitter in the personal communication domain, companies have started to investigate a possible transfer of microblogging concepts into the commercial domain. This raises the question for the business value. Indeed, facilitating the work of distributed teams, improving transparency and collaboration, and dealing with the "expert finding problem" are often-mentioned advantages of microblogging. Twitter's success in an enterprise context by offering an extended set of functions accounting for business needs.

3.3. Twitter

Twitter is a website, owned and operated by Twitter Inc., which offers a social networking and micro blogging service, enabling its users to send and read messages called *tweets*. Tweets are text based posts of up to 140 characters displayed on the user's profile page. Tweets are publicly visible by default; however senders can restrict message delivery to their followers. Users may subscribe to other users' tweets—this is known as *following* and subscribers are known as *followers*.

Twitter users follow others or are followed. The relationship of following and being followed requires no reciprocation. A user can follow any other user, and the user being followed need not follow back. Being a follower on Twitter means that the user receives all the messages (called tweets) from those the user follows.

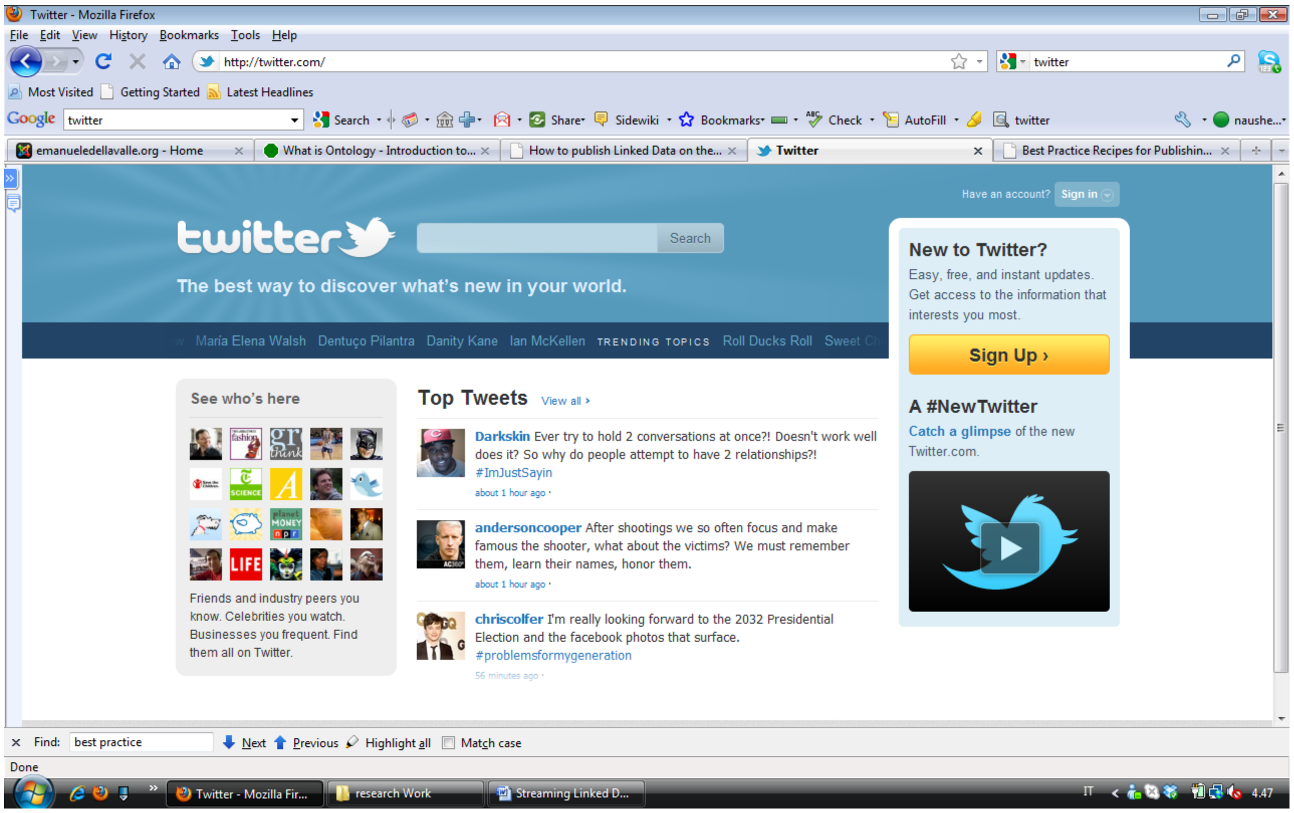
Twitter Annotations represent structured metadata about the tweet. What that metadata contains is up to you. The feature simply provides a structure for how to specify the annotations and retrieve them along with the tweet. Common practice of responding to a tweet has evolved into well-defined markup culture: RT stands for retweet, '@' followed by a user identifier address the user, and '#' followed by a word represents a hashtag. This well-defined markup vocabulary combined with a strict limit of 140 characters per posting conveniences users with brevity in expression. The retweet mechanism empowers users to spread information of their choice beyond the reach of the original tweet's followers.

[1]http://www.twitter.com, [2]http://www.jaiku.com, [3]http://www.pownce.com

All users can send and receive tweets via the Twitter website, compatible external applications such as for smartphones. Since its creation in March 2006 it has gained popularity worldwide and currently has more than 175 million users.

For many reasons Micro blogging platforms like Twitter have become increasingly popular. It has received public attention due to its ability to rapidly spread news. For example, before reaching any major news station, Twitter users have broadcasted about the Hudson River plane crash landing—one of them even being on the ferry boat heading out to rescue the passengers.

The length of messages is one of the key aspects differentiating Twitter from other social software such as online social networks, forums, wikis or blogs. Due to its “pull”-nature microblogging is less disruptive than instant messages, text messages, or the phone.


Twitter - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://twitter.com/
twitter
Most Visited Getting Started Latest Headlines
Google twitter Search Share Sidewiki Bookmarks Check AutoFill twitter naushe...
emanueledellavalle.org - Home
What is Ontology - Introduction to...
How to publish Linked Data on the...
Twitter
Best Practice Recipes for Publishin...
Have an account? Sign in
twitter
Search
María Elena Walsh Dentuço Pilantra Danity Kane Ian McKellen TRENDING TOPICS Roll Ducks Roll Sweet Ch
See who's here
Top Tweets View all ›
Sign Up ›
Find: best practice Next Previous Highlight all Match case
Done
Twitter - Mozilla Fir... research Work Streaming Linked D...
4.47

The best way to discover what's new in your world.

New to Twitter?

Easy, free, and instant updates. Get access to the information that interests you most.

A #NewTwitter

Catch a glimpse of the new Twitter.com.

Friends and industry peers you know. Celebrities you watch. Businesses you frequent. Find them all on Twitter.

Darkskin Ever try to hold 2 conversations at once?! Doesn't work well does it? So why do people attempt to have 2 relationships?! #ImJustSayin
about 1 hour ago

andersoncooper After shootings we so often focus and make famous the shooter, what about the victims? We must remember them, learn their names, honor them.
about 1 hour ago

chriscolfer I'm really looking forward to the 2032 Presidential Election and the facebook photos that surface. #problemsformygeneration
56 minutes ago

Figure 3.3.1 Twitter

3.4. The DBpedia Data Set

The DBpedia data set uses a large multi-domain ontology which has been derived from Wikipedia. The DBpedia data set currently describes 3.4 million “things” with over 1 billion “facts” (March 2010).

DBpedia uses the Resource Description Framework (RDF) as a flexible data model for representing extracted information and for publishing it on the Web. The DBpedia data set uses a large multi-domain ontology which has been derived from Wikipedia. The ontology was manually created from the most commonly used infobox templates within the English edition of Wikipedia. The ontology is used as target schema by the mapping-based infobox extraction.

The table below contains links to some example “things” from the data set:

Class	**Examples**
City	Cambridge, Berlin, Mnchester
Country	Spain, Iceland, South Korea
Politician	George W. Bush, Nicolas Sarkozy, Angela Merkel
Musician	AC/DC, Diana Ross, Röyksopp
Music album	Led Zeppelin III, Like a Virgin, Thriller
Director	Woody Allen, Oliver Stone, Takashi Miike
Film	Pulp Fiction, Hysterical Blindness, Breakfast at Tiffany's
Book	The Lord of the Rings, The Adventures of Tom Sawyer, The Holy Bible
Computer Game	Tetris, World of Warcraft, Sam & Max hit the Road
Technical Standard	HTML, RDF, URI

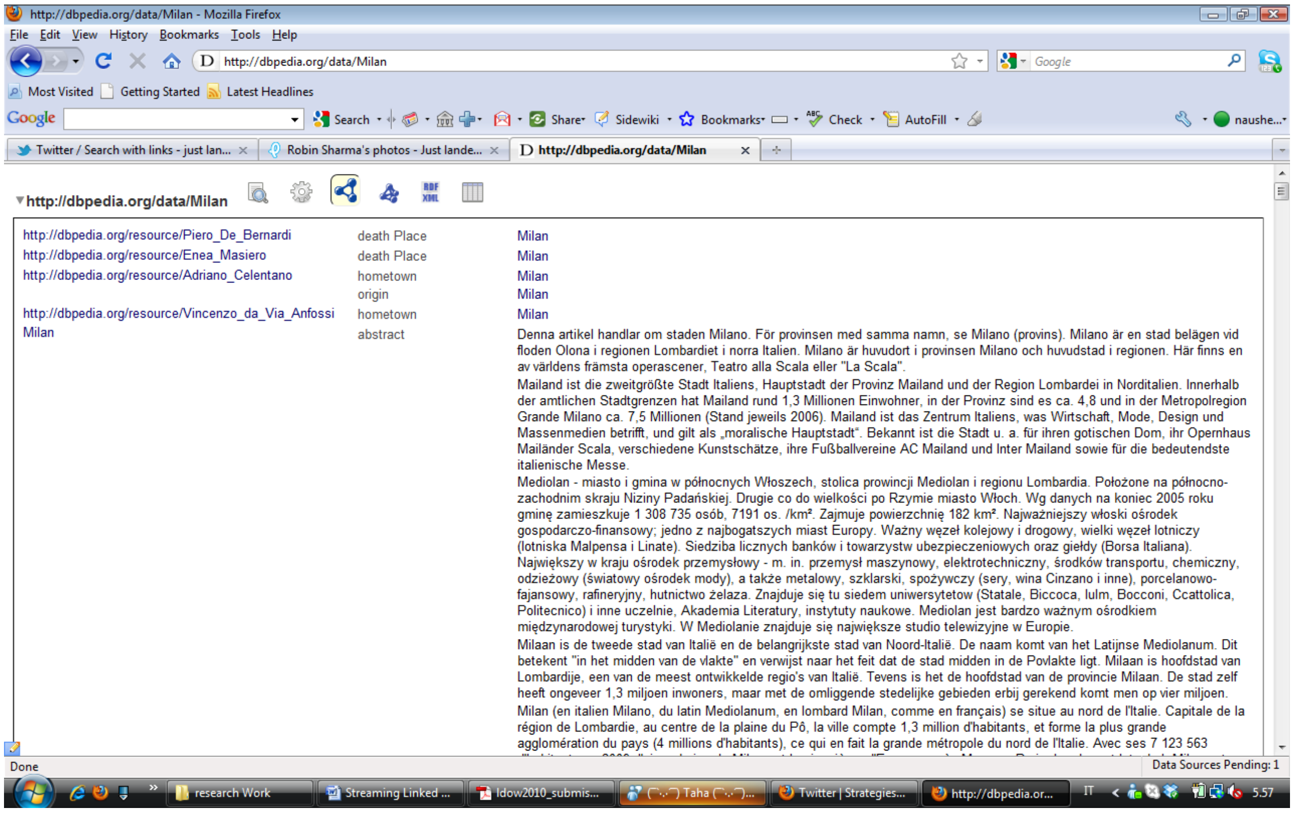
Table 3.4.1 Example Things from DBpedia DataSet

Each thing in the DBpedia data set is identified by a URI reference of the form http://dbpedia.org/resource/Name, where Name is taken from the URL of the source

Wikipedia article, which has the form http://en.wikipedia.org/wiki/Name. Thus, each resource is tied directly to an English-language Wikipedia article.

Each DBpedia resource is described by various properties. Below, we give an overview about the most important types of properties.Every DBpedia resource is described by a label, a short and long English abstract, a link to the corresponding Wikipedia page, and a link to an image depicting the thing (if available). RDF Document View

Below is the snapshot of RDF view of the tabulator semantic web browser presenting the RDF resource information of the city Milan. The information about Milan, presented in RDF format is actually taken from the DBpedia data sets.

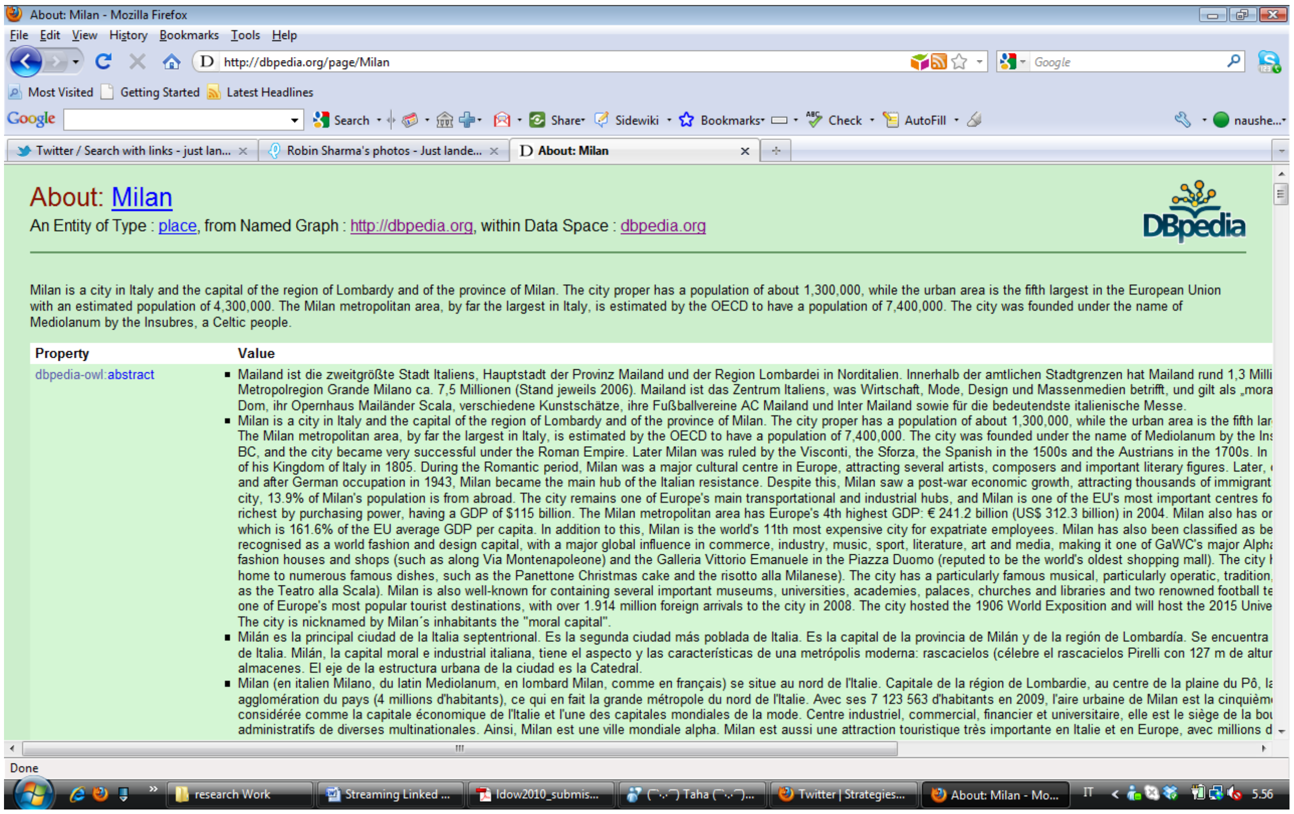

http://dbpedia.org/data/Milan - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dbpedia.org/data/Milan
Most Visited Getting Started Latest Headlines
Twitter / Search with links - just lan...
Robin Sharma's photos - Just lande...
http://dbpedia.org/data/Milan
Done
Data Sources Pending: 1

▾http://dbpedia.org/data/Milan

http://dbpedia.org/resource/Piero_De_Bernardi	death Place	Milan
http://dbpedia.org/resource/Enea_Masiero	death Place	Milan
http://dbpedia.org/resource/Adriano_Celentano	hometown	Milan
	origin	Milan
http://dbpedia.org/resource/Vincenzo_da_Via_Anfossi	hometown	Milan
Milan	abstract	Denna artikel handlar om staden Milano. För provinsen med samma namn, se Milano (provins). Milano är en stad belägen vid floden Olona i regionen Lombardiet i norra Italien. Milano är huvudort i provinsen Milano och huvudstad i regionen. Här finns en av världens främsta operascener, Teatro alla Scala eller "La Scala".
		Mailand ist die zweitgrößte Stadt Italiens, Hauptstadt der Provinz Mailand und der Region Lombardei in Norditalien. Innerhalb der amtlichen Stadtgrenzen hat Mailand rund 1,3 Millionen Einwohner, in der Provinz sind es ca. 4,8 und in der Metropolregion Grande Milano ca. 7,5 Millionen (Stand jeweils 2006). Mailand ist das Zentrum Italiens, was Wirtschaft, Mode, Design und Massenmedien betrifft, und gilt als „moralische Hauptstadt". Bekannt ist die Stadt u. a. für ihren gotischen Dom, ihr Opernhaus Mailänder Scala, verschiedene Kunstschätze, ihre Fußballvereine AC Mailand und Inter Mailand sowie für die bedeutendste italienische Messe.
		Mediolan - miasto i gmina w północnych Włoszech, stolica prowincji Mediolan i regionu Lombardia. Położone na północno-zachodnim skraju Niziny Padańskiej. Drugie co do wielkości po Rzymie miasto Włoch. Wg danych na koniec 2005 roku gminę zamieszkuje 1 308 735 osób, 7191 os. /km². Zajmuje powierzchnię 182 km². Najważniejszy włoski ośrodek gospodarczo-finansowy; jedno z najbogatszych miast Europy. Ważny węzeł kolejowy i drogowy, wielki węzeł lotniczy (lotniska Malpensa i Linate). Siedziba licznych banków i towarzystw ubezpieczeniowych oraz giełdy (Borsa Italiana). Największy w kraju ośrodek przemysłowy - m. in. przemysł maszynowy, elektrotechniczny, środków transportu, chemiczny, odzieżowy (światowy ośrodek mody), a także metalowy, szklarski, spożywczy (sery, wina Cinzano i inne), porcelanowo-fajansowy, rafineryjny, hutnictwo żelaza. Znajduje się tu siedem uniwersytetow (Statale, Biccoca, Iulm, Bocconi, Ccattolica, Politecnico) i inne uczelnie, Akademia Literatury, instytuty naukowe. Mediolan jest bardzo ważnym ośrodkiem międzynarodowej turystyki. W Mediolanie znajduje się największe studio telewizyjne w Europie.
		Milaan is de tweede stad van Italië en de belangrijkste stad van Noord-Italië. De naam komt van het Latijnse Mediolanum. Dit betekent "in het midden van de vlakte" en verwijst naar het feit dat de stad midden in de Povlakte ligt. Milaan is hoofdstad van Lombardije, een van de meest ontwikkelde regio's van Italië. Tevens is het de hoofdstad van de provincie Milaan. De stad zelf heeft ongeveer 1,3 miljoen inwoners, maar met de omliggende stedelijke gebieden erbij gerekend komt men op vier miljoen.
		Milan (en italien Milano, du latin Mediolanum, en lombard Milan, comme en français) se situe au nord de l'Italie. Capitale de la région de Lombardie, au centre de la plaine du Pô, la ville compte 1,3 million d'habitants, et forme la plus grande agglomération du pays (4 millions d'habitants), ce qui en fait la grande métropole du nord de l'Italie. Avec ses 7 123 563

Figure 3.4.1 RDF Document View

Below is the snapshot of the Firefox internet browser which is displaying the information about the city, Milan, in HTML format. However the information about Milan is actually taken from the DBpedia data sets.



About: Milan

An Entity of Type : place, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org

Milan is a city in Italy and the capital of the region of Lombardy and of the province of Milan. The city proper has a population of about 1,300,000, while the urban area is the fifth largest in the European Union with an estimated population of 4,300,000. The Milan metropolitan area, by far the largest in Italy, is estimated by the OECD to have a population of 7,400,000. The city was founded under the name of Mediolanum by the Insubres, a Celtic people.

Property	Value
dbpedia-owl:abstract	▪ Mailand ist die zweitgrößte Stadt Italiens, Hauptstadt der Provinz Mailand und der Region Lombardei in Norditalien. Innerhalb der amtlichen Stadtgrenzen hat Mailand rund 1,3 Milli Metropolregion Grande Milano ca. 7,5 Millionen (Stand jeweils 2006). Mailand ist das Zentrum Italiens, was Wirtschaft, Mode, Design und Massenmedien betrifft, und gilt als „mora Dom, ihr Opernhaus Mailänder Scala, verschiedene Kunstschätze, ihre Fußballvereine AC Mailand und Inter Mailand sowie für die bedeutendste italienische Messe. ▪ Milan is a city in Italy and the capital of the region of Lombardy and of the province of Milan. The city proper has a population of about 1,300,000, while the urban area is the fifth lar The Milan metropolitan area, by far the largest in Italy, is estimated by the OECD to have a population of 7,400,000. The city was founded under the name of Mediolanum by the Ins BC, and the city became very successful under the Roman Empire. Later Milan was ruled by the Visconti, the Sforza, the Spanish in the 1500s and the Austrians in the 1700s. In of his Kingdom of Italy in 1805. During the Romantic period, Milan was a major cultural centre in Europe, attracting several artists, composers and important literary figures. Later, and after German occupation in 1943, Milan became the main hub of the Italian resistance. Despite this, Milan saw a post-war economic growth, attracting thousands of immigrant city, 13.9% of Milan's population is from abroad. The city remains one of Europe's main transportational and industrial hubs, and Milan is one of the EU's most important centres fo richest by purchasing power, having a GDP of $115 billion. The Milan metropolitan area has Europe's 4th highest GDP: € 241.2 billion (US$ 312.3 billion) in 2004. Milan also has or which is 161.6% of the EU average GDP per capita. In addition to this, Milan is the world's 11th most expensive city for expatriate employees. Milan has also been classified as be recognised as a world fashion and design capital, with a major global influence in commerce, industry, music, sport, literature, art and media, making it one of GaWC's major Alpha fashion houses and shops (such as along Via Montenapoleone) and the Galleria Vittorio Emanuele in the Piazza Duomo (reputed to be the world's oldest shopping mall). The city I home to numerous famous dishes, such as the Panettone Christmas cake and the risotto alla Milanese). The city has a particularly famous musical, particularly operatic, tradition, as the Teatro alla Scala). Milan is also well-known for containing several important museums, universities, academies, palaces, churches and libraries and two renowned football te one of Europe's most popular tourist destinations, with over 1.914 million foreign arrivals to the city in 2008. The city hosted the 1906 World Exposition and will host the 2015 Unive The city is nicknamed by Milan´s inhabitants the "moral capital". ▪ Milán es la principal ciudad de la Italia septentrional. Es la segunda ciudad más poblada de Italia. Es la capital de la provincia de Milán y de la región de Lombardía. Se encuentra de Italia. Milán, la capital moral e industrial italiana, tiene el aspecto y las características de una metrópolis moderna: rascacielos (célebre el rascacielos Pirelli con 127 m de altur almacenes. El eje de la estructura urbana de la ciudad es la Catedral. ▪ Milan (en italien Milano, du latin Mediolanum, en lombard Milan, comme en français) se situe au nord de l'Italie. Capitale de la région de Lombardie, au centre de la plaine du Pô, la agglomération du pays (4 millions d'habitants), ce qui en fait la grande métropole du nord de l'Italie. Avec ses 7 123 563 d'habitants en 2009, l'aire urbaine de Milan est la cinquième considérée comme la capitale économique de l'Italie et l'une des capitales mondiales de la mode. Centre industriel, commercial, financier et universitaire, elle est le siège de la bou administratifs de diverses multinationales. Ainsi, Milan est une ville mondiale alpha. Milan est aussi une attraction touristique très importante en Italie et en Europe, avec millions d

Figure 3.4.2 HTML Document View

3.5. DBpedia Classificaiton

DBpedia provides three different classification schemata for things.

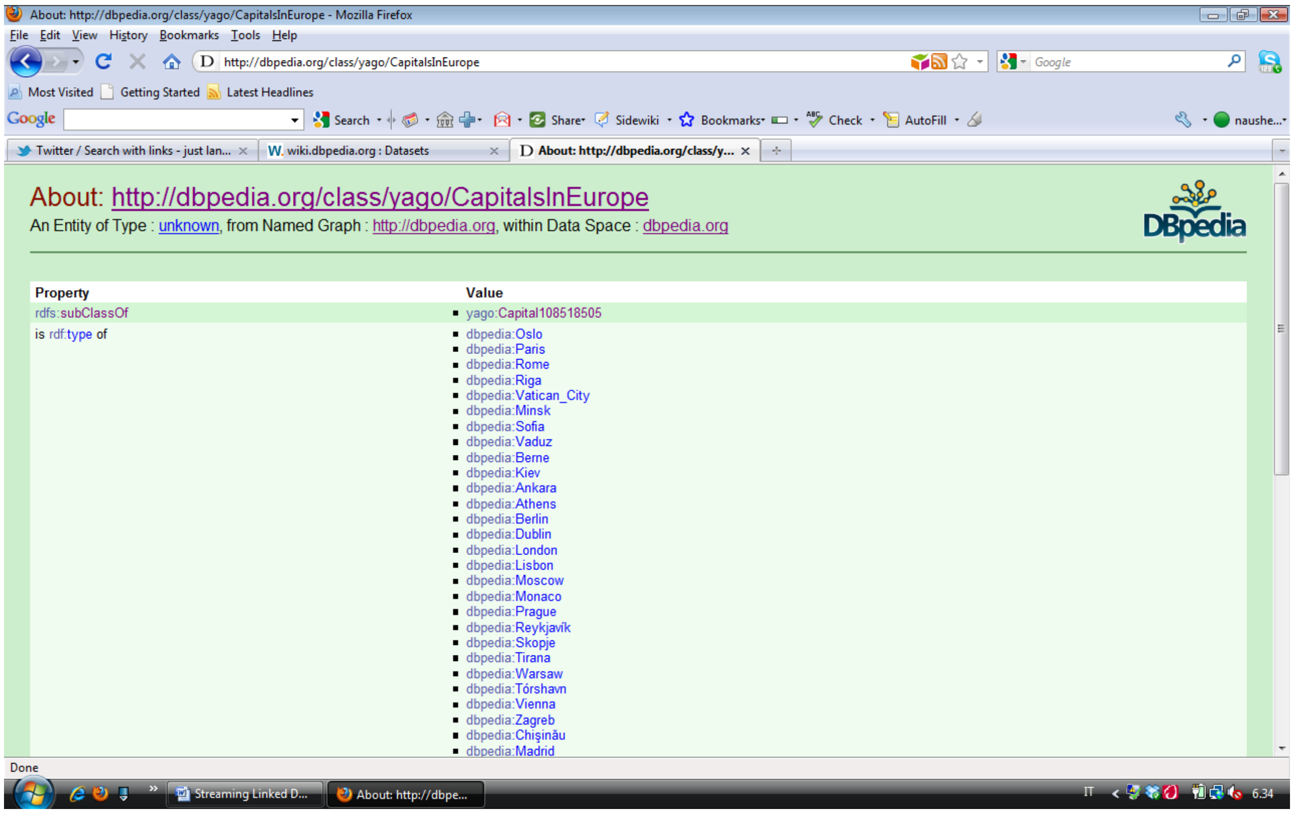
1. Wikipedia Categories are represented using the SKOS vocabulary.
2. The YAGO Classification is derived from the Wikipedia category system using Word Net.
3. Word Net Synset Links were generated by manually relating Wikipedia infobox templates and Word Net synsets, and adding a corresponding link to each thing that uses a specific template. In theory, this classification should be more precise then the Wikipedia category system.

Using these classifications within SPARQL queries allows you to select things of a certain type.

3.5.1. Yago

YAGO ontology contains 1 million entities and 5 million facts. The schema was created by mapping Wikipedia leaf categories, i.e. those categories which has no subcategories. Characteristics of the YAGO hierarchy are its deepness and the encoding of much information in one class (e.g. the class \MultinationalCompaniesHeadquarteredInTheNetherlands"). While YAGO achieves a high accuracy in general, there are a few errors and omissions (e.g. the mentioned class is not a subclass of \MultinationalCompanies") due to its automatic generation. However the Yago classes consist of two main categories.

1. European Capitals
2. Birds of the United States


About: http://dbpedia.org/class/yago/CapitalsInEurope - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://dbpedia.org/class/yago/CapitalsInEurope
Most Visited Getting Started Latest Headlines
Twitter / Search with links - just lan...
wiki.dbpedia.org : Datasets
About: http://dbpedia.org/class/y...
DBpedia
Done
Streaming Linked D...
About: http://dbpe...

About: http://dbpedia.org/class/yago/CapitalsInEurope

An Entity of Type : unknown, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org

Property	Value
rdfs:subClassOf	▪ yago:Capital108518505
is rdf:type of	▪ dbpedia:Oslo ▪ dbpedia:Paris ▪ dbpedia:Rome ▪ dbpedia:Riga ▪ dbpedia:Vatican_City ▪ dbpedia:Minsk ▪ dbpedia:Sofia ▪ dbpedia:Vaduz ▪ dbpedia:Berne ▪ dbpedia:Kiev ▪ dbpedia:Ankara ▪ dbpedia:Athens ▪ dbpedia:Berlin ▪ dbpedia:Dublin ▪ dbpedia:London ▪ dbpedia:Lisbon ▪ dbpedia:Moscow ▪ dbpedia:Monaco ▪ dbpedia:Prague ▪ dbpedia:Reykjavík ▪ dbpedia:Skopje ▪ dbpedia:Tirana ▪ dbpedia:Warsaw ▪ dbpedia:Tórshavn ▪ dbpedia:Vienna ▪ dbpedia:Zagreb ▪ dbpedia:Chişinău ▪ dbpedia:Madrid

Figure 3.5.1 Yago European Capitals

3.6. Examples

As described above that people state that their status for example students say that they are doing assignments, they mention their topic of an assignment, and reading books. Researchers update their status as they are working on some projects. Similarly someone one is travelling he will mention that he is travelling to some place e.g "I am travelling to Milano".

However there is common practice of responding to such tweets has evolved into well-defined markup culture: RT stands for retweet, '@' followed by a user identifier address the user, and

’#’ followed by a word represents a hashtag. This well-defined markup vocabulary combined with a strict limit of 140 characters per posting conveniences users with brevity in expression. The retweet mechanism empowers users to spread information of their choice beyond the reach of the original tweet’s followers.

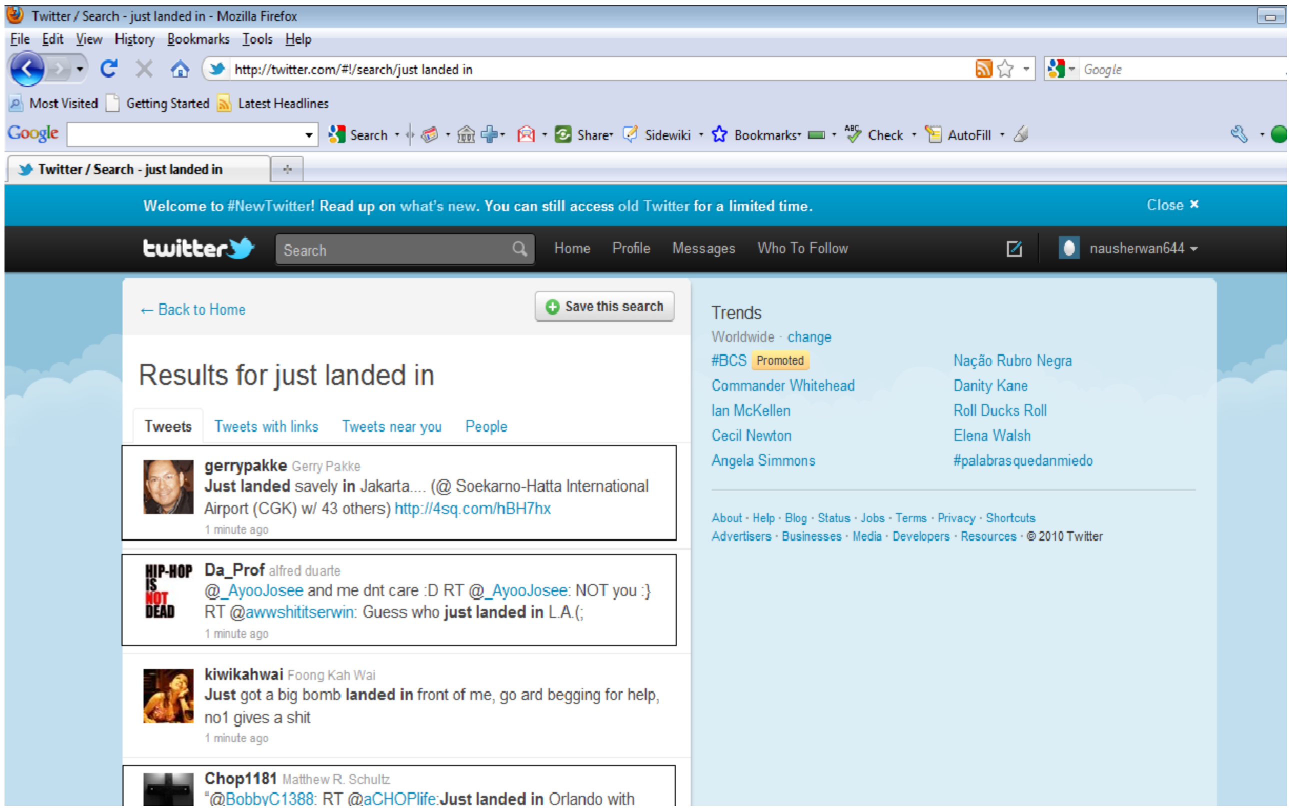
However our example could elaborate the status of the people, presented in the tweets of the twitter, and we can find answers of some relevant questions and explain their semantics. E.g.

- Person(s) is just landed in place
- Someone is just landed in NYC
- Who is just landed in NYC
- Who just seen an accident
- Who just read Left Neglected

These are the questions that have semantics in them and are the complete sentences as well. All of these can be divided in the subject its predicate and its value, which is the simple RDF triple to represent the knowledge

3.6.1. Just Landed In

Below is result of an example of string searched on the twitter. The string is “Just landed in".


Twitter / Search - just landed in - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://twitter.com/#!/search/just landed in
Most Visited Getting Started Latest Headlines
Google Search Share Sidewiki Bookmarks Check AutoFill
Twitter / Search - just landed in
Close
Home Profile Messages Who To Follow
naushervan644
← Back to Home
Save this search
Tweets Tweets with links Tweets near you People
About · Help · Blog · Status · Jobs · Terms · Privacy · Shortcuts
Advertisers · Businesses · Media · Developers · Resources · © 2010 Twitter

Welcome to #NewTwitter! Read up on what's new. You can still access old Twitter for a limited time.

Results for just landed in

gerrypakke Gerry Pakke
Just landed savely in Jakarta.... (@ Soekarno-Hatta International Airport (CGK) w/ 43 others) http://4sq.com/hBH7hx
1 minute ago

Da_Prof alfred duarte
@_AyooJosee and me dnt care :D RT @_AyooJosee: NOT you :} RT @awwshititserwin: Guess who just landed in L.A.(;
1 minute ago

kiwikahwai Foong Kah Wai
Just got a big bomb landed in front of me, go ard begging for help, no1 gives a shit
1 minute ago

Chop1181 Matthew R. Schultz
"@BobbyC1388: RT @aCHOPlife:Just landed in Orlando with

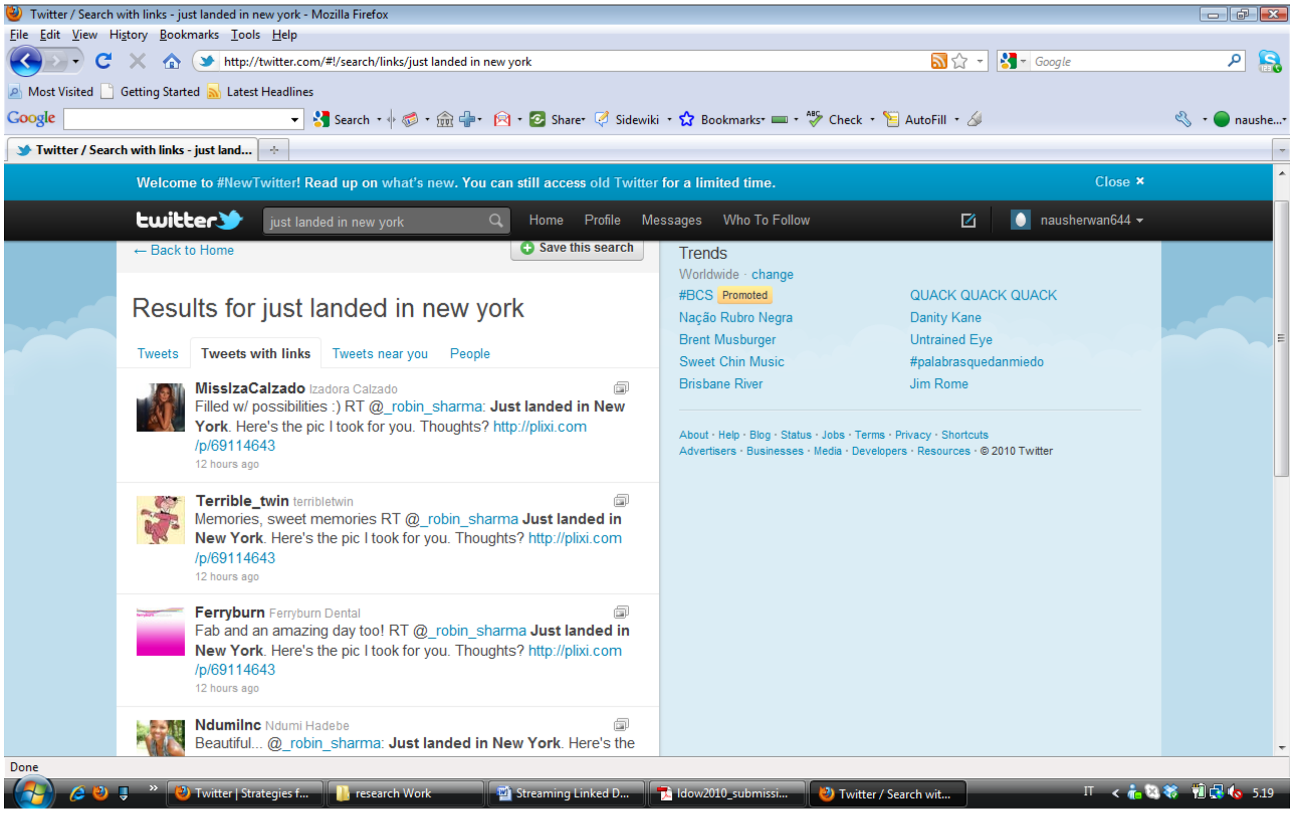
Trends
Worldwide · change

#BCS Promoted	Nação Rubro Negra
Commander Whitehead	Danity Kane
Ian McKellen	Roll Ducks Roll
Cecil Newton	Elena Walsh
Angela Simmons	#palabrasquedanmiedo

Figure 3.6.1 Results for “JustlandedIn” Using Twitter

3.6.2. Just Landed in New york

Similarly the above searched string “just landed in” is then modified to get more specific results. The string is modified as “just landed in new york”. The result is all the tweets including use who mentioned that he is just landed in New York.



Twitter / Search with links - just landed in new york - Mozilla Firefox

http://twitter.com/#!/search/links/just landed in new york

Welcome to #NewTwitter! Read up on what's new. You can still access old Twitter for a limited time.

Results for just landed in new york

Tweets | Tweets with links | Tweets near you | People

MissIzaCalzado Izadora Calzado
Filled w/ possibilities :) RT @_robin_sharma: **Just landed in New York**. Here's the pic I took for you. Thoughts? http://plixi.com/p/69114643
12 hours ago

Terrible_twin terribletwin
Memories, sweet memories RT @_robin_sharma **Just landed in New York**. Here's the pic I took for you. Thoughts? http://plixi.com/p/69114643
12 hours ago

Ferryburn Ferryburn Dental
Fab and an amazing day too! RT @_robin_sharma **Just landed in New York**. Here's the pic I took for you. Thoughts? http://plixi.com/p/69114643
12 hours ago

NdumiInc Ndumi Hadebe
Beautiful... @_robin_sharma: **Just landed in New York**. Here's the

Trends
Worldwide · change

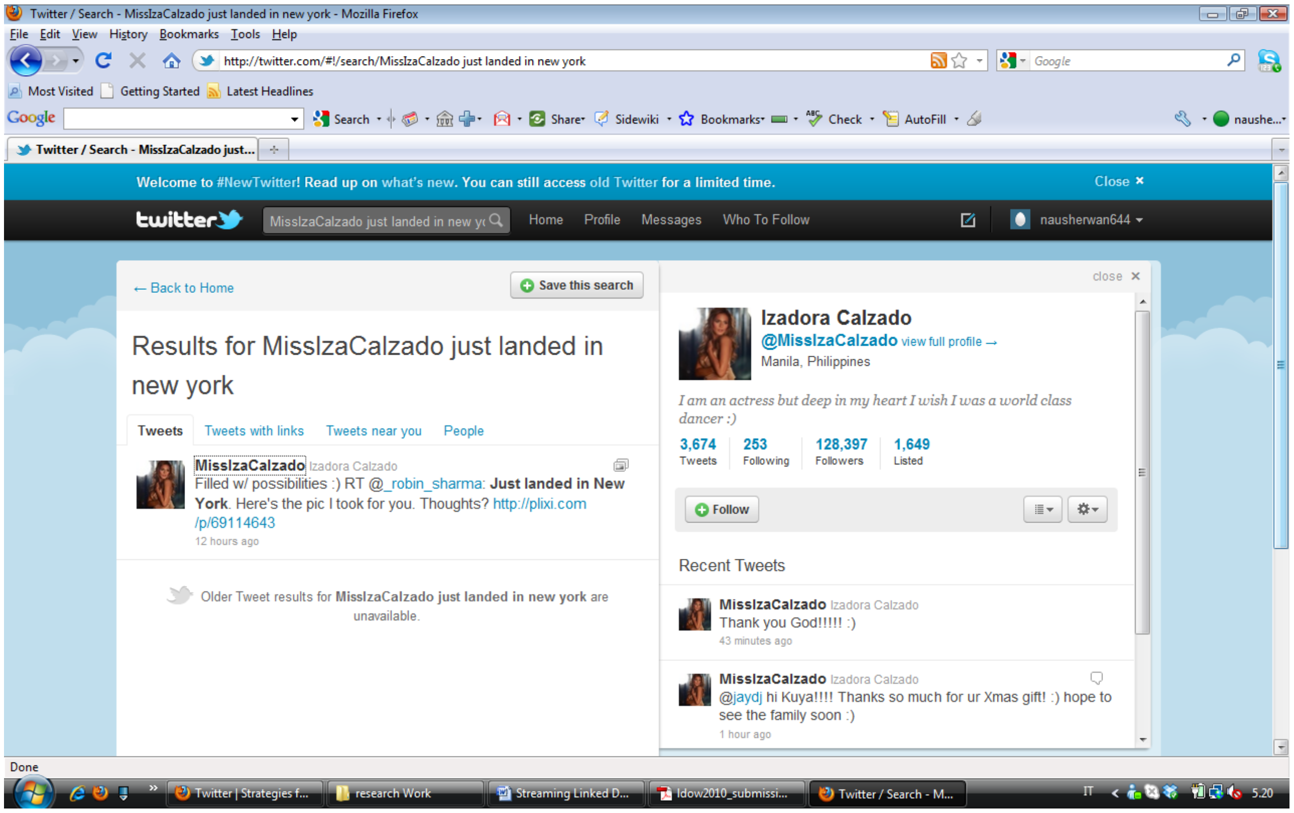
#BCS Promoted	QUACK QUACK QUACK
Nação Rubro Negra	Danity Kane
Brent Musburger	Untrained Eye
Sweet Chin Music	#palabrasquedanmiedo
Brisbane River	Jim Rome

File Edit View History Bookmarks Tools Help
Home Profile Messages Who To Follow
nausherwan644
Back to Home
Save this search
Close
Done

Figure 3.6.2 Results for "JustlandedIn new York" Using Twitter

3.6.3. MisslzaCalzado just Landed In

If we would like to search a specific user and we searched a string that where a specific user is just landed in we get a single result as an example, the string is “MissIzaCalzado landed in New York”. The figure given below shows the results of the searched string against the particular user.


Twitter / Search - MissIzaCalzado just landed in new york - Mozilla Firefox

http://twitter.com/#!/search/MissIzaCalzado just landed in new york

Welcome to #NewTwitter! Read up on what's new. You can still access old Twitter for a limited time.

Results for MisslzaCalzado just landed in new york

MisslzaCalzado Izadora Calzado
Filled w/ possibilities :) RT @_robin_sharma: Just landed in New York. Here's the pic I took for you. Thoughts? http://plixi.com/p/69114643
12 hours ago

Older Tweet results for MisslzaCalzado just landed in new york are unavailable.

Izadora Calzado
@MisslzaCalzado
Manila, Philippines
I am an actress but deep in my heart I wish I was a world class dancer :)
3,674 Tweets | 253 Following | 128,397 Followers | 1,649 Listed

Recent Tweets

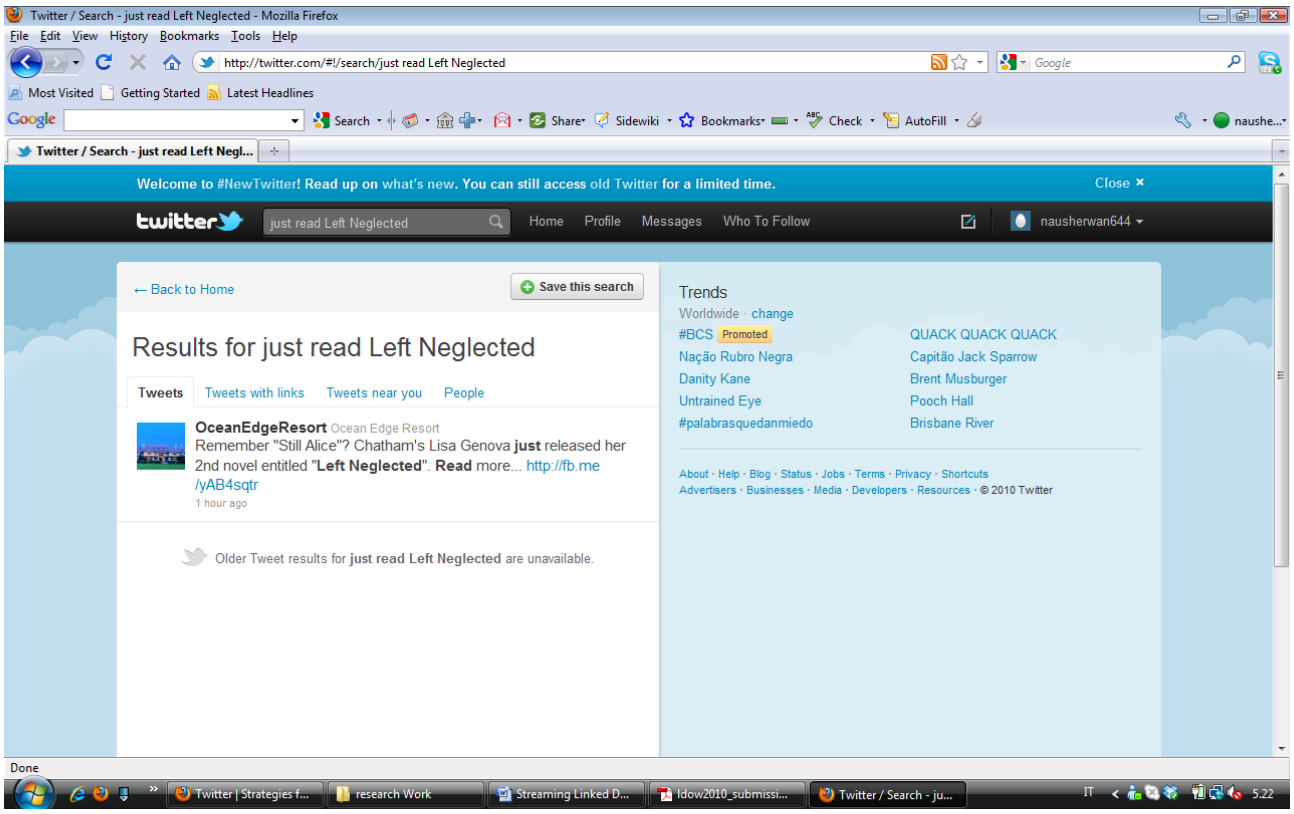
MisslzaCalzado Izadora Calzado
Thank you God!!!!! :)
43 minutes ago

MisslzaCalzado Izadora Calzado
@jaydj hi Kuya!!!! Thanks so much for ur Xmas gift! :) hope to see the family soon :)
1 hour ago

Figure 3.6.3 Result for "MisslzaCalzado just Landed In New york" Using Twitter

3.6.4. Just Read Left Neglected

Another example of a search string is for the user who Just read a book named “left neglected” we got the following result.


Twitter / Search - just read Left Neglected - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://twitter.com/#!/search/just read Left Neglected
Most Visited Getting Started Latest Headlines
Welcome to #NewTwitter! Read up on what's new. You can still access old Twitter for a limited time.
Close
just read Left Neglected
Home Profile Messages Who To Follow
nausherwan644
← Back to Home
Save this search
Tweets Tweets with links Tweets near you People
Trends
Worldwide · change
#BCS Promoted
QUACK QUACK QUACK
Nação Rubro Negra
Capitão Jack Sparrow
Danity Kane
Brent Musburger
Untrained Eye
Pooch Hall
#palabrasquedanmiedo
Brisbane River
About · Help · Blog · Status · Jobs · Terms · Privacy · Shortcuts
Advertisers · Businesses · Media · Developers · Resources · © 2010 Twitter
Done

Results for just read Left Neglected

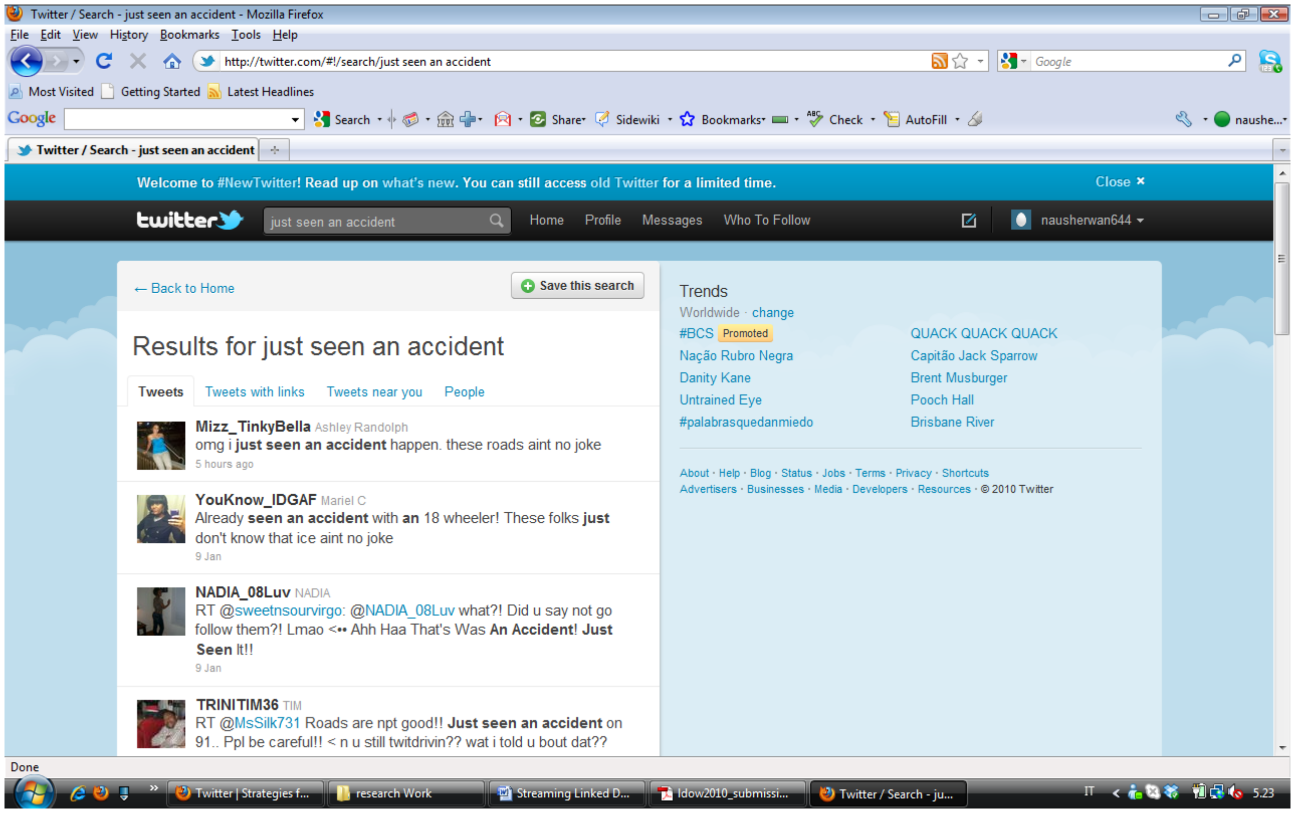
OceanEdgeResort Ocean Edge Resort
Remember "Still Alice"? Chatham's Lisa Genova **just** released her 2nd novel entitled "**Left Neglected**". **Read** more... http://fb.me/yAB4sqtr
1 hour ago

Older Tweet results for **just read Left Neglected** are unavailable.

Figure 3.6.4 Result for "just read Left Neglected" USing Twitter

3.6.5. Just Seen An Accident

Similarly one can search the user who has Just seen an accident the number of users will be listed down who have mentioned in their tweet message that they just have seen an accident.


Twitter / Search - just seen an accident - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://twitter.com/#!/search/just seen an accident
Welcome to #NewTwitter! Read up on what's new. You can still access old Twitter for a limited time.
Close
just seen an accident
Home Profile Messages Who To Follow
nausherwan644
← Back to Home
Save this search
Tweets Tweets with links Tweets near you People
Trends
Worldwide · change
#BCS Promoted
Nação Rubro Negra
Danity Kane
Untrained Eye
#palabrasquedanmiedo
QUACK QUACK QUACK
Capitão Jack Sparrow
Brent Musburger
Pooch Hall
Brisbane River
About · Help · Blog · Status · Jobs · Terms · Privacy · Shortcuts
Advertisers · Businesses · Media · Developers · Resources · © 2010 Twitter
Done

Results for just seen an accident

Mizz_TinkyBella Ashley Randolph
omg i **just seen an accident** happen. these roads aint no joke
5 hours ago

YouKnow_IDGAF Mariel C
Already **seen an accident** with **an** 18 wheeler! These folks **just** don't know that ice aint no joke
9 Jan

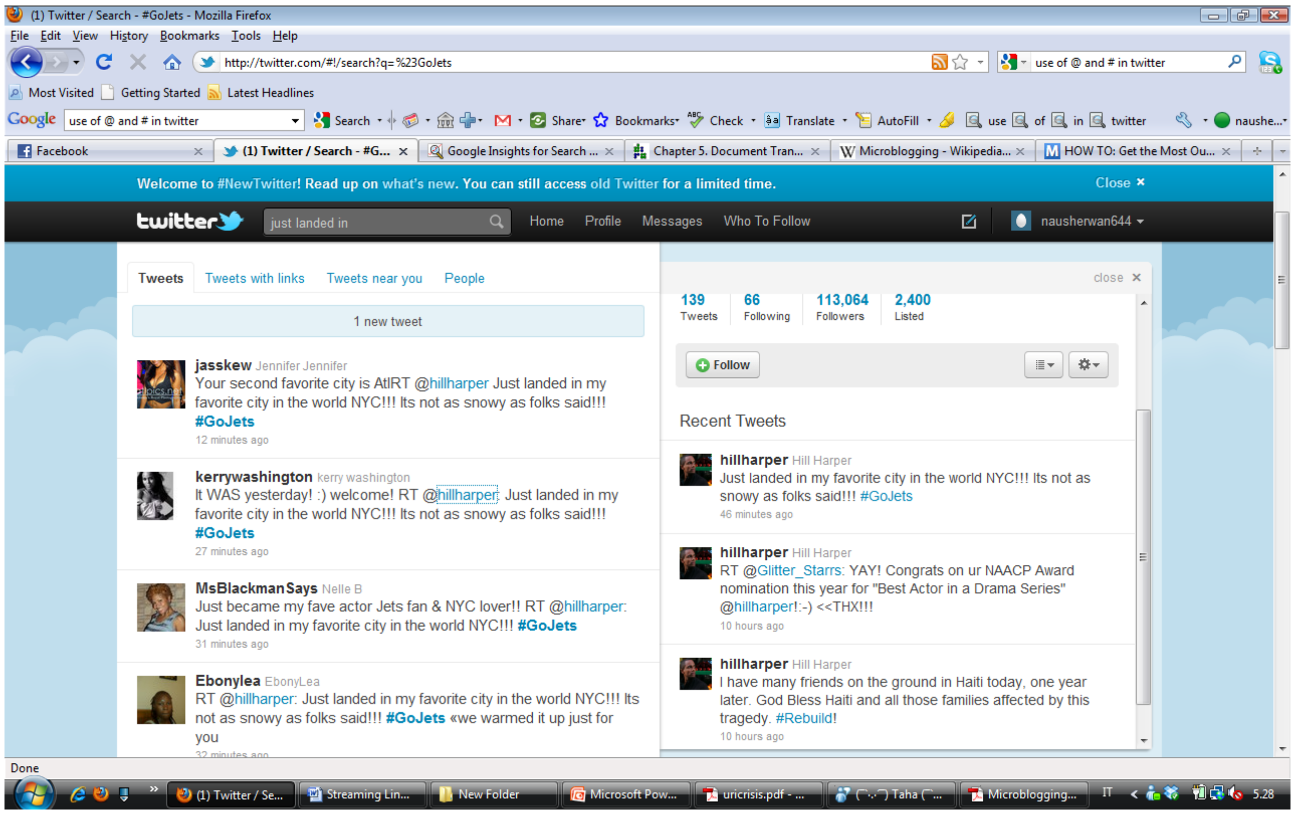
NADIA_08Luv NADIA
RT @sweetnsourvirgo: @NADIA_08Luv what?! Did u say not go follow them?! Lmao <•• Ahh Haa That's Was **An Accident**! **Just Seen** It!!
9 Jan

TRINITIM36 TIM
RT @MsSilk731 Roads are npt good!! **Just seen an accident** on 91.. Ppl be carefull!! < n u still twitdrivin?? wat i told u bout dat??

Figure 3.6.5 Results for "just seen an accident" Using Twitter

So come back to an example of "just landed in" we can find multiple information in a single stream of twitter. For example @user is the user who is following the tweet and replied to the post of the first user to whom he is following. Information we can find is from the user who is following. He uses the hashtags to organize an event or initiating a new topic. This is one of the most complex features of Twitter for new users to understand it; a topic with a hash symbol ("#") at the start is for its identification. Twitter hashtags like #followfriday help spread information on Twitter while also helping to organize it.

However in our example we can see that “hillharper” is the follower. He is stating that he is landed in his favorite city in the world NYC and he has travelled through “#GoJets”. His purpose it to welcoming his followers and those to whom he is following to let them know about “Gojets”.


(1) Twitter / Search - #GoJets - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://twitter.com/#!/search?q=%23GoJets
use of @ and # in twitter
Most Visited
Getting Started
Latest Headlines
Facebook
(1) Twitter / Search - #G...
Google Insights for Search ...
Chapter 5. Document Tran...
Microblogging - Wikipedia...
HOW TO: Get the Most Ou...
Welcome to #NewTwitter! Read up on what's new. You can still access old Twitter for a limited time.
Close
just landed in
Home
Profile
Messages
Who To Follow
nausherwan644
Tweets
Tweets with links
Tweets near you
People
1 new tweet
Follow
Recent Tweets
close
Done

jasskew Jennifer Jennifer
Your second favorite city is AtlRT @hillharper Just landed in my favorite city in the world NYC!!! Its not as snowy as folks said!!! #GoJets
12 minutes ago

kerrywashington kerry washington
It WAS yesterday! :) welcome! RT @hillharper: Just landed in my favorite city in the world NYC!!! Its not as snowy as folks said!!! #GoJets
27 minutes ago

MsBlackmanSays Nelle B
Just became my fave actor Jets fan & NYC lover!! RT @hillharper: Just landed in my favorite city in the world NYC!!! #GoJets
31 minutes ago

Ebonylea EbonyLea
RT @hillharper: Just landed in my favorite city in the world NYC!!! Its not as snowy as folks said!!! #GoJets «we warmed it up just for you

139	66	113,064	2,400
Tweets	Following	Followers	Listed

hillharper Hill Harper
Just landed in my favorite city in the world NYC!!! Its not as snowy as folks said!!! #GoJets
46 minutes ago

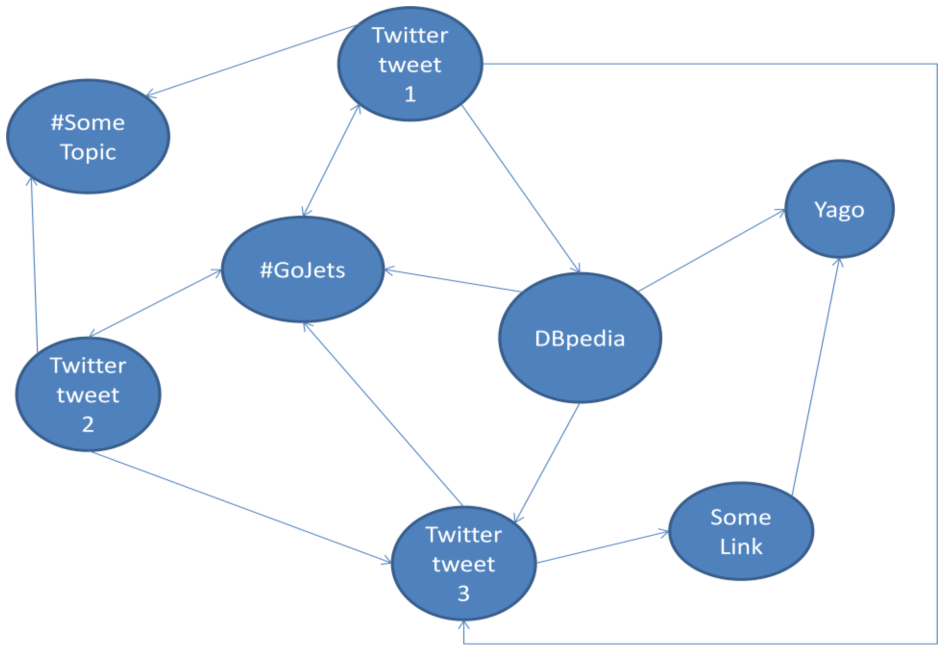
hillharper Hill Harper
RT @Glitter_Starrs: YAY! Congrats on ur NAACP Award nomination this year for "Best Actor in a Drama Series" @hillharper!:-) <<THX!!!
10 hours ago

hillharper Hill Harper
I have many friends on the ground in Haiti today, one year later. God Bless Haiti and all those families affected by this tragedy. #Rebuild!
10 hours ago

Figure 3.6.6 Result for "just landed in" Using Twitter

3.7. Data becoming available as linked data

The above example of “Just landed In” includes tweets and those tweets can be drown and shown as linked data.



Drawing 3.7.1 Linked Data from Twitter Tweets

From all of the above examples we can find some useful information and modifying to our own proof of concept, need and implementation.

We can write the instances in an engineering way. We have instances NYC, Left Neglected etc. exploring the instance of NYC by using the keywords “just landed in New York”. We can find a tweet/stream say 1 which includes:

- Some link: http://plixi.com/p/69114643
- Author : hill harper
- Time and date :
- Tweet text : Just landed in my favorite city in the world NYC!!! Its not as snowy as folks said!!! #GoJets
- just landed in : NYC

Similarly tweet 2 includes

- Some link: http:// 4sq.com/hBH7hx
- Author : gerypakke
- Time and date :

- Tweet text : safely reached in jakarta
- just landed in : jakarta

3.8. Transforming twitter tweets into an RDF Stream

A data stream is defined as an ordered sequence of pairs, where each pair is made of a RDF tuple and its timestamp. We can write the above tweets as a RDF stream.

Triples	**Timestamp**
:User1 :justlandedin :"http://dbpedia.org/resource/Milan"	T100
:Traveller2 :justlandedin :"http://dbpedia.org/resource/NYC"	T101
:Traveller1 :tweettext : "Message"	T101
:Traveller3 :justlandedin :http://dbpedia.org/resource/Como	T101
:Traveller2 :tweettext : "message"	T101

Table 3.8.1 Creating Triples from twitter Using RDF

In a similar way, we define an RDF stream (4) as an ordered sequence of pairs, where each pair is made of an RDF triple and its timestamp. By mapping the data stream above in RDF using D2RQ mapping language (22), we obtain the following RDF stream:

```
<http://localhost:8080/testproj/resource/Traveller1>
<http://localhost:8080/tweeter/TweeterText> "message" .
<http://localhost:8080/testproj/resource/Traveller1>
<http://purl.org/dc/elements/1.1/Date> "2010-10-
10T03:25:55Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://localhost:8080/testproj/resource/Traveller1>
<http://www.w3.org/2000/01/rdf-schema#justlandedin>
<http://dbpedia.org/resource/Milan> .
```

```
<http://localhost:8080/testproj/resource/Traveller2>
<http://localhost:8080/tweeter/TweeterText> "message" .
<http://localhost:8080/testproj/resource/Traveller2>
<http://purl.org/dc/elements/1.1/Date> "2010-10-
10T03:25:55Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://localhost:8080/testproj/resource/Traveller2>
<http://www.w3.org/2000/01/rdf-schema#justlandedin>
<http://dbpedia.org/resource/NYC> .
```

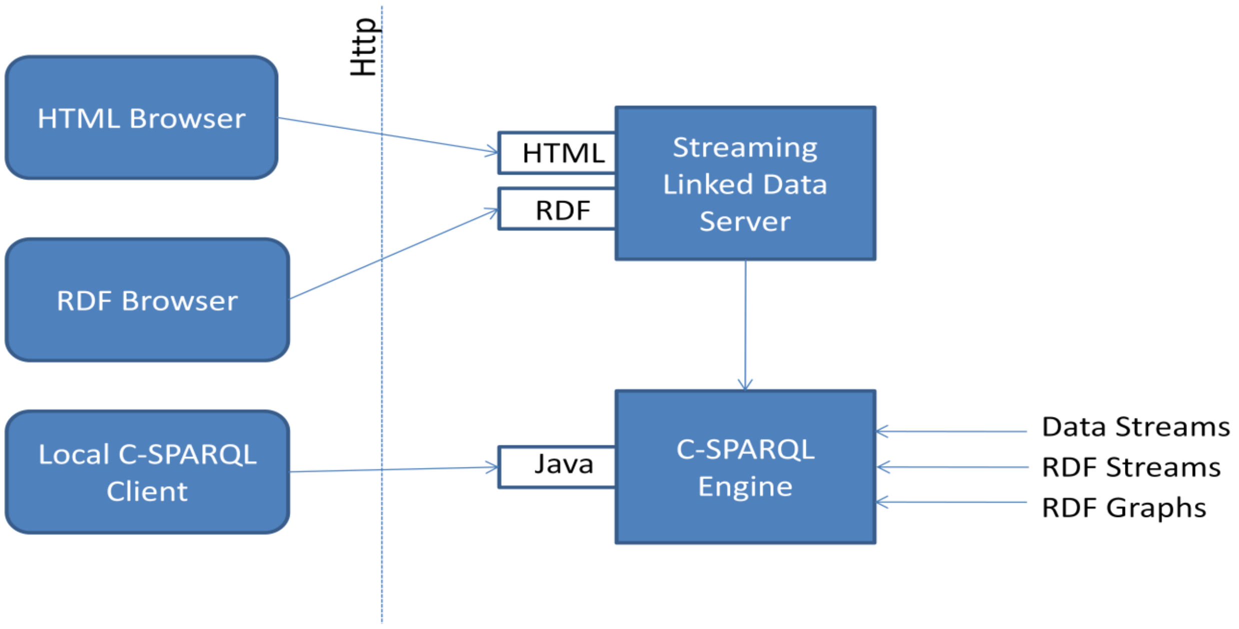
```
<http://localhost:8080/testproj/resource/Traveller3>
<http://localhost:8080/tweeter/TweeterText> "message" .
<http://localhost:8080/testproj/resource/Traveller3>
<http://purl.org/dc/elements/1.1/Date> "2010-10-
10T03:25:55Z"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
<http://localhost:8080/testproj/resource/Traveller3>
<http://www.w3.org/2000/01/rdf-schema#justlandedin>
<http://dbpedia.org/resource/Como> .
```

Table 3.8.2 Creating Triples from twitter Using RDF

3.9. Proposed solution

We are proposing a novel technique to publish the data elements of such streams. Below is the detail for the architectural design for a data stream publishing system as well the technique through which the system is publishing these streams.

3.10. Architectural design



Drawing 3.10.1 Architectural Design

3.11. Publish a stream

We propose to represent RDF streams in RDF using named graphs (23). We distinguish between two kind of named graphs: the Stream Graphs (shortly s-graphs) and the Instantaneous Graphs (shortly i-graphs). In our proposal, an RDF Stream can be represented using one s-graph and several i-graphs, one for each timestamp. A s-graphs is a metadata graph that describes the current content of the window over the RDF Stream. The most important part of an s-graph are the triples that refer to the i-graphs using **rdfs:seeAlso** and those that describe when each i-graph was received using the property **receivedAt**. Fewer other metadata complete the description of an graph. The property **lastUpdate** describes the last time the graph was updated. The property **expires** allows to indicate a Linked Data Client that the information in the graph will expire in a given moment in future. These graphs will be made query able by a SPARQL end point. The properties **sld:windowType** and **sld:windowSize** describe the window through which the steam is observed (see chapter 4 for more information).

For instance, if the data stream exemplified above were the current content of a window over the stream of twitter tweets transactions, it can be represented using the s-graph in Listing 1 and the two i-graphs in Listing 2 and 3.

Listing with notion of window:

```
@prefix rdfs : <http :// www.w3. org /2000/01/ rdf - schema #> .
@prefix sld : <http :// www . streaminglinkeddata .org/ schema #> .
@prefix : <http :// localhost:8080/SLD/resourse /> .
:sgraph1 sld : lastUpdate "ti+1"^^ xsd : dataTime ;
sld : expires "ti+2"^^ xsd : dataTime ;
sld : windowType sld : logicalTumbling ;
sld : windowSize " PT1H "^^ xsd: duration .
:sgraph1 rdfs : seeAlso : igraph1 .
:igraph1 sld : receivedAt "ti "^^ xsd : dataTime .
:sgraph1 rdfs : seeAlso : igraph2 .
:igraph2 sld : receivedAt "_i+1"^^ xsd : dataTime .
```

Listing 1: Example of Stream Graph linking two Instantaneous Graphs

```
@prefix rdfs : <http :// www.w3. org /2000/01/ rdf - schema #> .
@prefix sld : <http :// www . streaminglinkeddata .org/ schema #> .
@prefix : < http :// localhost:8080/SLD/resourse /> .
: igraph1 sld : receivedAt "ti "^^ xsd : dataTime ;
rdfs : seeAlso : sgraph1 .
```

Listing 2: The Instantaneous Graph timestamped with ti. We choose to link s-graphs to i-graphs using the property rdfs:seeAlso, because it has been largely adopted to link named graphs (see for instance the usage of rdfs:seeAlso in sindice (24) and in the Semantic Web Client (25))

```
@prefix rdfs : <http :// www .w3. org /2000/01/ rdf - schema #> .
@prefix sld : <http :// www . streaminglinkeddata . org / schema #> .
@prefix : <http :// localhost:8080/SLD/resourse > .
: igraph2 sld : receivedAt "ti+1"^^ xsd : dataTime ;
rdfs : seeAlso : sgraph1 .
```

Listing 3: The Instantaneous Graph timestamped with ti+1

```
@prefix rdfs : <http :// www .w3. org /2000/01/ rdf - schema #> .
@prefix sld : <http :// www . streaminglinkeddata . org / schema #> .
@prefix : < http :// localhost:8080/SLD/resourse /> .
:igraph2 sld : receivedAt "_i+1"^^ xsd : dataTime ;
rdfs : seeAlso : sgraph1 .
```

Following the guidelines on cool URIs (26) , we propose to give to s-graphs and i-graphs an IRI using the following schemata:

```
s- graph : http :// ex. org /\% stream - name \%
e.g., http :// http :// localhost:8080/SLD/resourse/justlandedin

i- graph : http :// ex. org /\% stream - name \%/ URLeconde (\%timestamp\%)
e.g.,http://localhost:8080/SLD/resourse/2010-02-12T13\%3A34\%3
```

Moreover, following the best practice on how to publish Linked Data on the Web (27) in terms of content negotiation, when IRIs, which follow the schemata shown above are dereferenced, the Streaming Linked Data Server sends a 303 redirect to an information resource appropriate for the Clients.

Linked Data Clients are redirected to:

```
http :// localhost:8080/SLD/data/justlandedIn.rdf
http :// localhost:8080/SLD/data/2011-02-15T071550Z.rdf (time-stamp)
```

HTML Clients are redirected to:

```
http :// localhost:8080/SLD/ page/justlandedIn.html
http :// localhost:8080/SLD/page/ 2011-02-15T071550Z.html (time-stamp)
```

3.12. Controlling the window

As a data stream is a real-time, continuous, ordered sequence of items. Processing queries over data streams, which are expected to run continuously and return new answers as new data arrive, introduces novel challenges such as adapting query plans in response to changing stream arrival rates, sharing resources among similar queries, and generating approximate answers in limited space (28). Data streams are more common today in most of the web applications, these streams are now fundamental to many data processing applications. For example in computer networks switches and routers periodically generate records of their network traffic, these records are actually the streams (29). Atmospheric observations, weather measurements, lighting stroke data and satellite imagery also produce multiple data streams (30). Emerging sensor networks produce many streams of observation, e.g. highway traffic conditions (31). Sources of data streams -large scale transactions, financial transactions of stock brokers are very common in our daily life. Two observations hold in practice.

1. In most of the scenarios, data analysis often involves questions about the most recent, just an hour back. This is because of the need to take decisions. For example a query is that, to compute the financial transactions of Swiss brokers within last one hour.
2. In most of the applications, despite of the fact that the capacity of storage devices is growing exponentially, still it is not common for the streams to be stored on the disk. However they are being processed in real time as they are produced. This means that we received the data stream and we consumed it on the "fly".

Considering such type of the problem in mind to monitor thousands of data streams in an online fashion and making decisions based on them. The windowed data stream model is introduced. This is useful to limit the extraction of the data stream elements upon continuous queries. However we need to store each item in the window to compute the minimum value at each step, which is again impossible for the worst case scenario. We get the overhead of $O(1/e \log^2 N)$ bits of memory. Please refer to (32) for further elaboration of a mathematically version of the same argument.

As we have explained earlier that streams are intrinsically infinite. In C-SPARQL, we introduce the notion of windows over streams. In Section 3.11, we focus on the general approach to publish a data stream rather than on the notion of window. However, we foresee the need for a
Consumer of Streaming Linked Data to be able to control the behavior of the window through which the stream is observed.

Types and characteristics of windows in C-SPARQL are inspired by those of the windows defined in continuous query languages for relational streaming data, such as CQL (33). Windows are expressed in C-SPARQL within the FROM STREAM clause, whose syntax is as follows:

```
FromStrClause →`FROM' [`NAMED'] `STREAM' StreamIRI`[ RANGE' Window `]'
Window → LogicalWindow | PhysicalWindow
LogicalWindow → Number TimeUnit WindowOverlap
TimeUnit → `ms' | `s'| `m' | `h' | `d'
WindowOverlap → `STEP' Number TimeUnit | `TUMBLING'
PhysicalWindow → `TRIPLES' Number
```

This text is elaborating the concept of windowing data stream model in an efficient way. This windowing data stream model is capable of presenting the limited set of elements of a data stream. We have divided the window in to two types; physical and logical where as logical window is further viewed as the sliding and the tumbling window.

In order to observe the working or results of both the physical and logical windowing model we consider the CSPARQL Engine (5) which is extracting the triples from an RDF Stream in a continuous fashion. It allows registering queries that continuously combine (virtual) RDF streams and RDF graphs. Under this respect, our C-SPARQL Engine is similar to D2RQ (31) that treat non-RDF databases as virtual RDF graphs. This means that the query environment is continuous so continuous queries need continuous reevaluation whenever new triples arrive, and secondly, queries are interested only in a specific part (window-of-interest) of the received data. The physical and logical-window query models are introduced to answer continuous queries that are interested only on the most recent stream triples. In a physical and logical-window query over "N" input streams, **"S1"** to "**Sn"**, a window of size "W**i**" is defined over the input stream "**Si**". As the window moves, the query answer is updated to reflect both the new triples entering the physical window and the old triples expiring from the

physical and logical-window. Triples enter and expire from the physical-window in a First-In-First-Expire (FIFE) fashion.
The basic syntax for the continuous query is given below:

```
REGISTER STREAM transactions COMPUTE EVERY 10m AS
PREFIX : < http://www.streaminglinkeddata.org />
CONSTRUCT *
FROM STREAM < http://localhost:8080/testproj/resource/justlandedin .rdf >
[RANGE 1h step 10m]
WHERE {?s ?p ?o }.
```

Here in the query, the REGISTER STREAM clause is use to tell the C-SPARQL engine that it should register a continuous query, i.e. a query that will continuously compute answers for the query. In particular, we are registering a query that generates RDF stream. The COMPUTE EVERY clause states the frequency of every new computation, in the example every 10 minutes. The clause FROM STREAM defines the RDF stream of justlandedin which is queried. [RANGE] defines the window of observation over the RDF stream. WHERE clause is common in all query languages and is used to specify selection criteria.

3.12.1. Physical window

The physical window extracts the given number of triples from an RDF stream, which are considered by the query. In this model, data streams arrive continually and only the most recent "N" data stream elements are used when answering queries. We present a novel technique for presenting these RDF data stream triples using physical windowing model. The physical window query model is useful for many RDF data stream applications.

For example we registered the following query in our CSPARQL Engine:

```
REGISTER STREAM transactions COMPUTE AS
PREFIX : < http://www.streaminglinkeddata.org />
CONSTRUCT *
FROM STREAM < http://localhost:8080/testproj/resource/justlandedin .rdf >
[RANGE 10triples]
WHERE { ?s ?p ?o }.
```

Now utilizing the physical-window query model, the query semantics reads as follows: We are interested in the last 10 triples from our Justlandedin RDF stream graph. So our window of interest includes the 10 triples in a way that, for example our CSPARQL Engine starts extracting triples, at time "t1" it received 2 triple then at time "t2" it received 4 triples then at time "t3" it received 5 triples. Now at the three time instances we received three **S-graphs** the total number of triples received are 11 but our window of interest needs to present 10 triples so last 10 triples will be presented and the first **S-graph** will be discarded. As the physical windows moves and at time instance "t4" two triples received now the total number of triples received are twelve however our window of interest is capable and of presenting 10 triples so the last 10 triples will be presented and the first two triples of **S-graph** arrived at time instance "t2"will be expired and upon the arrival of more two triples on next time instance, the **S-graph** at the time instance "t2" will completely be expired and discarded.

S-graph

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf- schema #> .
@prefix sld:<http://www.streaminglinkeddata.org/schema #> .

:sgraph1 sld:lastUpdate "i+1"^^ xsd:dataTime ;
sld:expires "i+10"^^ xsd:dataTime ;
sld:windowType sld:physicalwindow ;
sld:windowSize " PT10 "^^ xsd:integer .

:sgraph1 rdfs:seeAlso :igraph1 .
:igraph1 sld:receivedAt "i "^^ xsd:dataTime .

:sgraph1 rdfs:seeAlso :igraph2 .
:igraph2 sld:receivedAt "i "^^ xsd:dataTime .
```

Time:	T1		T2				T3					T4	
S-graph1	1	2											
S-graph1	1	2	3	4	5	6							
S-graph1	1	2	3	4	5	6	7	8	9	10	11		
S-graph1	1	2	3	4	5	6	7	8	9	10	11	12	13

Table 3.12.1 Physical Window S-Graph

I-graph 1

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:igraph1 sld:receivedAt "i "^^ xsd:dataTime ;
rdfs:seeAlso :sgraph1 .
```

I-graph 2

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:igraph2 sld:receivedAt "i+1 "^^ xsd:dataTime ;
rdfs:seeAlso :sgraph1 .
```

3.12.2. Logical window

The logical window extracts data stream elements from the Stream, which are considered by the query. Such extraction can be logical because they are continuously advanced by a given step and time period. In this model, the triples of RDF data stream arrive continually and only the most recent "N" RDF data streams are used when answering queries. We present a novel technique for solving two important and related problems in the logical window model. On the basis of such problems we divided the logical window into two; sliding window and tumbling window. One stream triple includes once in a tumbling window while in case of sliding window one stream triple can be included in several windows.

3.12.2.1. Sliding window

The sliding window query model is useful for discounting stale data in data stream applications. As the window slides, the query answer is updated to reflect both the new triples entering the sliding window and the old triples expiring from the sliding-window. Triples enter and expire from the sliding-window in a First-In-First-Expire (FIFE) fashion. Now utilizing the sliding-window query model, the query semantics reads as follows:

```
REGISTER STREAM transactions COMPUTEEVERY 10m  AS
PREFIX : < http://www.streaminglinkeddata.org />
CONSTRUCT *
FROM STREAM < http://localhost:8080/testproj/resource/justlandedin .rdf >
[RANGE 1h step 10m]
WHERE { ?s ?p ?o }.
```

In the example, the window comprises of RDF triples produced in the last 1 hour, and the window slides every 10 minutes. The WHERE clause is standard; it includes a set of matching patterns, so one step of sliding is of 10minutes. In one hour the windows slides up to 6 times. However in hour we have received 1000 triples then our sliding window behaves in practice:

1. The possible scenario for system is that we received 2 triple in 10 minutes so after the 10 minutes when the window slides, the triple arrived earlier in previous 10 minutes will be expired.
2. We received 100 triples in next 10 minutes so after the 10 minutes when the window slides, the triple arrived earlier will be expired as well.

However all the triples arrived in last hour will be presentable using the sliding window query model and here in our case the window size "**Wi**" is 10m. This means that our window of interest includes all the triples arrived in 10minutes and these all triples will be expired soon after the window slides in next 10minutes over the data streams. However with sliding windows some triples can be included into several windows.

The practical behavior of s-graph and i-graphs is shown below. As our CSPARQL engine starts' extracting the first triple arrives at "ti" the s-graph constructed like this.

Case1:

S-graph

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:sgraph1 sld:lastUpdate "i+1"^^ xsd:dataTime ;
sld:expires "i+2"^^ xsd:dataTime ;
sld:windowType sld:slidingwindow ;
sld:windowSize " PT1H "^^ xsd:duration .
sld:windowSize " PT10m "^^ xsd:duration .

:sgraph1 rdfs:seeAlso :igraph1 .
:igraph1 sld:receivedAt "i "^^ xsd:dataTime .
```

I-graph 1

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:igraph1 sld:receivedAt "i "^^ xsd:dataTime ;
rdfs:seeAlso :sgraph1 .
```

I-graph 2

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:igraph2 sld:receivedAt "i+1 "^^ xsd:dataTime ;
rdfs:seeAlso :sgraph1 .
```

Case2:

S-graph

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .

:sgraph1 sld:lastUpdate "i+1"^^ xsd:dataTime ;
sld:expires "i+2"^^ xsd:dataTime ;
sld:windowType sld:slidingwindow ;
sld:windowSize " PT1H "^^ xsd:duration .
sld:windowSize " PT10m "^^ xsd:duration .

:sgraph1 rdfs:seeAlso :igraph1 .
:igraph1 sld:receivedAt "i "^^ xsd:dataTime .
:sgraph1 rdfs:seeAlso :igraph2 .
:igraph2 sld:receivedAt "i+1 "^^ xsd:dataTime .

...
:sgraph1 rdfs:seeAlso :igraph100 .
:igraph sld:receivedAt "i+100 "^^ xsd:dataTime .
```

I-graph 1

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:igraph1 sld:receivedAt "i "^^ xsd:dataTime ;
rdfs:seeAlso :sgraph1 .
```

I-graph 100

```
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix sld:<http://www.streaminglinkeddata.org/schema#> .
:igraph2 sld:receivedAt "i+100 "^^ xsd:dataTime ;
      rdfs:seeAlso :sgraph1 .
```

3.12.2.2. Tumbling window

The tumbling window query model is useful for limited time data in data stream applications. Queries are interested in all part (window-of-interest) of the received data in a given time. The tumbling-window query model is introduced to answer continuous queries that are interested on the most recent stream triples. In a tumbling-window query over "N" input streams, **"S1**" to "**Sn"** given that a window of size "W**i**" is defined over the input stream "**Si**". As the window tumbles, the query answer is updated to reflect the new triples entering the tumbling window and all the old triples are discarded. Triples enter and then discarded from the tumbling-window in a produce and consumed fashion but in a limited time.
Now utilizing the tumbling-window query model, the query semantics reads as follows:

REGISTER STREAM transactions **COMPUTEEVERY 10m AS**
PREFIX : < http://www.streaminglinkeddata.org />
CONSTRUCT *
FROM STREAM < http://localhost:8080/testproj/resource/justlandedin .rdf >
[**RANGE** 10m step 10m]
WHERE { ?s ?p ?o }.

We have been given a range of 10minutes, the triples arrives in 10m should be extracted. The range and the step is same in case of tumbling window. The window tumbles over the data streams every 10 minutes. So the number of triples continues to be extracted in every 10 minutes. The possible scenario for system is that either we have received 1 triple or 100 triples in 10 minutes the window tumbles, the triple arrived earlier in previous 10 minutes of the tumbling window will be discarded.

This means that our window of interest includes all the triples arrived in 10minutes and these all triples will be discarded soon after the window tumbles in next 10minutes over the data streams. However with tumbling windows every triple of the stream is included once into the window.

Data Streams are usually the record measurements from sensors which are installed at distributed locations, e.g weather forecast. Such streams appearing more and more often on the Web, Similarly, users interested in the streams are often distributed as well. These streams are normally geographically distributes so we anticipate this rapid growing need of mashing up this streaming information with more static one. While best practices for linking static data on the Web were published and facilitate the mash up of static information published on the Web, streams were neglected. Now A basic question is thus arises how to publish mashed up streaming information and how to query it that allows interested users and to find them? We propose a web based approach to process the information behind these streams by publishing and querying them by an extension to SPARQL (32) for continuous querying over streams of RDF and static RDF graphs.

3.13. Resulting RDF/XML documents, Triples and RDF-Graphs

The prototype of this work results S-Graph and I-Graphs, these graphs with their RDF/XML document and triples are given below.

S-Graph RDF/XML document:

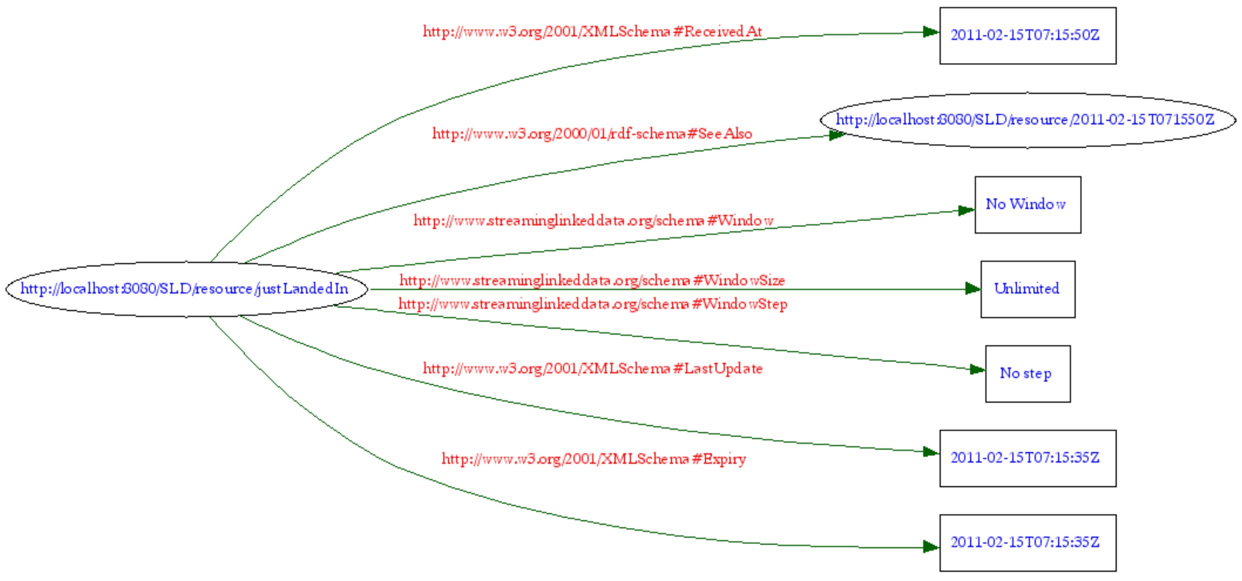
```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://www.streaminglinkeddata.org/schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
    <rdf:Description rdf:about="http://localhost:8080/SLD/resource/
            justLandedIn">
    <xsd:ReceivedAt
            rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
            2011-02-5T07:15:50Z
            </xsd:ReceivedAt>
    <rdfs:SeeAlso rdf:resource="http://localhost:8080/SLD/resource/
            2011-02-15T071550Z"/>
    <j.0:Window>No Window</j.0:Window>
    <j.0:WindowSize
            rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">
            Unlimited
            </j.0:WindowSize>
    <j.0:WindowStep
            rdf:datatype="http://www.w3.org/2001/XMLSchema#duration">
            No step
            </j.0:WindowStep>
    <xsd:LastUpdate
            rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
            2011-02-15T07:15:35Z
            </xsd:LastUpdate>
    <xsd:Expiry rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
            2011-02-15T07:15:35Z
            </xsd:Expiry>
    </rdf:Description>
</rdf:RDF>
```

S-Graph Triples:

Num	Subject	Predicate	Object
1	http://localhost:8080/SLD/resource/justLandedIn	http://www.w3.org/2001/XMLSchema#ReceivedAt	http://www.w3.org/2001/XMLSchema#LastUpdate
2	http://localhost:8080/SLD/resource/justLandedIn	http://www.w3.org/2000/01/rdf-schema#SeeAlso	http://localhost:8080/SLD/resource/2011-02-15T071550Z
3	http://localhost:8080/SLD/resource/justLandedIn	http://www.streaminglinkeddata.org/schema#Window	"No Window"
4	http://localhost:8080/SLD/resource/justLandedIn	http://www.streaminglinkeddata.org/schema#WindowSize	"Unlimited"^^http://www.w3.org/2001/XMLSchema#decimal
5	http://localhost:8080/SLD/resource/justLandedIn	http://www.streaminglinkeddata.org/schema#WindowStep	"No step"^^http://www.w3.org/2001/XMLSchema#duration
6	http://localhost:8080/SLD/resource/justLandedIn	http://www.w3.org/2001/XMLSchema#LastUpdate	"2011-02-15T07:15:35Z"^^http://www.w3.org/2001/XMLSchema#dateTime
7	http://localhost:8080/SLD/resource/justLandedIn	http://www.w3.org/2001/XMLSchema#Expiry	"2011-02-15T07:15:35Z"^^http://www.w3.org/2001/XMLSchema#dateTime

Table 3.13.1 Resulting S-Graph Triples

S-Graph:



http://localhost:8080/SLD/resource/justLandedIn
http://www.w3.org/2001/XMLSchema#ReceivedAt
2011-02-15T07:15:50Z
http://www.w3.org/2000/01/rdf-schema#SeeAlso
http://localhost:8080/SLD/resource/2011-02-15T071550Z
http://www.streaminglinkeddata.org/schema#Window
No Window
http://www.streaminglinkeddata.org/schema#WindowSize
Unlimited
http://www.streaminglinkeddata.org/schema#WindowStep
No step
http://www.w3.org/2001/XMLSchema#LastUpdate
2011-02-15T07:15:35Z
http://www.w3.org/2001/XMLSchema#Expiry
2011-02-15T07:15:35Z

Figure 3.13.1 Resulting S-Graph

I-Graph RDF/XML document:

```
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:j.0="http://www.streaminglinkeddata.org/schema#"
     xmlns:j.1="http://localhost:8080/tweeter/"
     xmlns:dc="http://purl.org/dc/elements/1.1/"
     xmlns:j.2="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
     <rdf:Descriptionrdf:about="http://localhost:8080/SLD/resource/
          2011-02-15T071550Z">
    <j.2:Date rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
          2011-02-15T07:12:58Z</j.2:Date>
    <dc:Author rdf:resource="http://localhost:8080/SLD/resource/
          189466385"/>
    <j.1:TweeterText>Just landed in Cali, I'm tired!!#ThankGodWeMadeIt
          </j.1:TweeterText>
    <j.0:justlandedin rdf:resource="http://dbpedia.org/resource/Cali"/>
    <j.2:ReceivedAt
          rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
          2011-02-15T07:15:50Z</j.2:ReceivedAt>
    <rdfs:SeeAlso
         rdf:resource="http://localhost:8080/SLD/resource/justLandedIn"/>
    </rdf:Description>
</rdf:RDF>
```

I-Graph Triples:

Num	Subject	Predicate	Object
1	http://localhost:8080/SLD/resource/2011-02-15T071550Z	http://www.w3.org/2001/XMLSchema#Date	"2011-02-15T07:12:58Z"^^http://www.w3.org/2001/XMLSchema#dateTime
2	http://localhost:8080/SLD/resource/2011-02-15T071550Z	http://purl.org/dc/elements/1.1/Author	http://localhost:8080/SLD/resource/189466385
3	http://localhost:8080/SLD/resource/2011-02-15T071550Z	http://localhost:8080/tweeter/TweeterText	"Just landed in Cali, I'm tired!! #ThankGodWeMadeIt"
4	http://localhost:8080/SLD/resource/2011-02-15T071550Z	http://www.streaminglinkeddata.org/schema#justlandedin	http://dbpedia.org/resource/Cali
5	http://localhost:8080/SLD/resource/2011-02-15T071550Z	http://www.w3.org/2001/XMLSchema#ReceivedAt	"2011-02-15T07:15:50Z"^^http://www.w3.org/2001/XMLSchema#dateTime
6	http://localhost:8080/SLD/resource/2011-02-15T071550Z	http://www.w3.org/2000/01/rdf-schema#SeeAlso	http://localhost:8080/SLD/resource/justLandedIn

Table 3.13.2 Resulting I-Graph Triples

I-Graph:

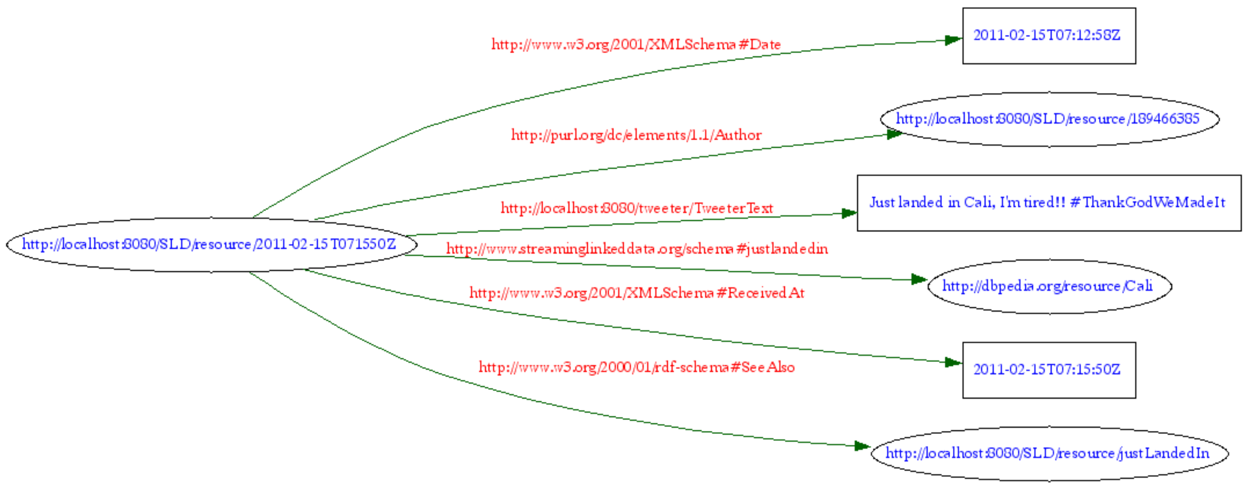

http://localhost:8080/SLD/resource/2011-02-15T071550Z
http://www.w3.org/2001/XMLSchema#Date
2011-02-15T07:12:58Z
http://purl.org/dc/elements/1.1/Author
http://localhost:8080/SLD/resource/189466385
http://localhost:8080/tweeter/TweeterText
Just landed in Cali, I'm tired!! #ThankGodWeMadeIt
http://www.streaminglinkeddata.org/schema#justlandedin
http://dbpedia.org/resource/Cali
http://www.w3.org/2001/XMLSchema#ReceivedAt
2011-02-15T07:15:50Z
http://www.w3.org/2000/01/rdf-schema#SeeAlso
http://localhost:8080/SLD/resource/justLandedIn

Figure 3.13.2 Resulting I-Graph

3.14. Streaming linked data Server prototype

In order to test our system, we need a prototype server. Server should be implemented to the extent that it would be able to process client request properly and respond. We will discuss implementation in detail in the coming chapter but for the proof of concept we will describe an example of a data stream taken from a twitter, we mapped this stream to RDF stream and make this stream available to the linked data client and is served as Linked Data by the Streaming Linked Data Server.

3.15. Client/Server Architecture

Client/Server architecture forms the basis of our solution. From our problem description, Client in our case is a broad term used for two types of clients. Client/Server architecture is based upon Request and Response. Request refers to the process of submitting your URL. Response is that server interprets the URL, locates the corresponding page, and ends back the page to the requesting browser. The browser is referred to as Client in this interaction.

3.16. URIs for Web Documents

The Resource Description Framework RDF allows you to describe web documents and resources from the real world—people, organizations, things—in a computer-process able way. Publishing such descriptions on the web creates the semantic web. URIs are very important as the link between RDF and the web.

3.17. HTTP and Content Negotiation

Web clients and servers use the HTTP protocol (37) to request web documents and send back the responses. HTTP has a powerful mechanism for offering different formats and language versions of the same web document:

When a user agent e. g. a web browser requests a URL, it sends along some HTTP headers to indicate what data formats and language it prefers. The server then selects the best match from its hard disk or generates the desired content on demand, and sends it back to the client. For example, a browser 2 could send this HTTP request to indicate that it wants an HTML or RDF version of an application.

```
GET /page/justlandedin HTTP/1.1
Host: http://localhost:8080/SLD/resource/justlandedin
Accept: text/html, application/xhtml+xml
```

The server could answer:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
```

Content negotiation is often implemented with a twist: Instead of a direct answer, the server redirects to another URL where the appropriate version is found:

```
HTTP/1.1 302 Found
Location: http://localhost:8080/SLD/page/justlandedin.html
```

The redirect is indicated by a special status code, here 302 Found. The client would now send another HTTP request to the new URL. By having separate URLs for all versions, this approach allows web authors to link directly to a specific version. RDF/XML, the standard serialisation format of RDF, has its own content type too, application/rdf+xml.

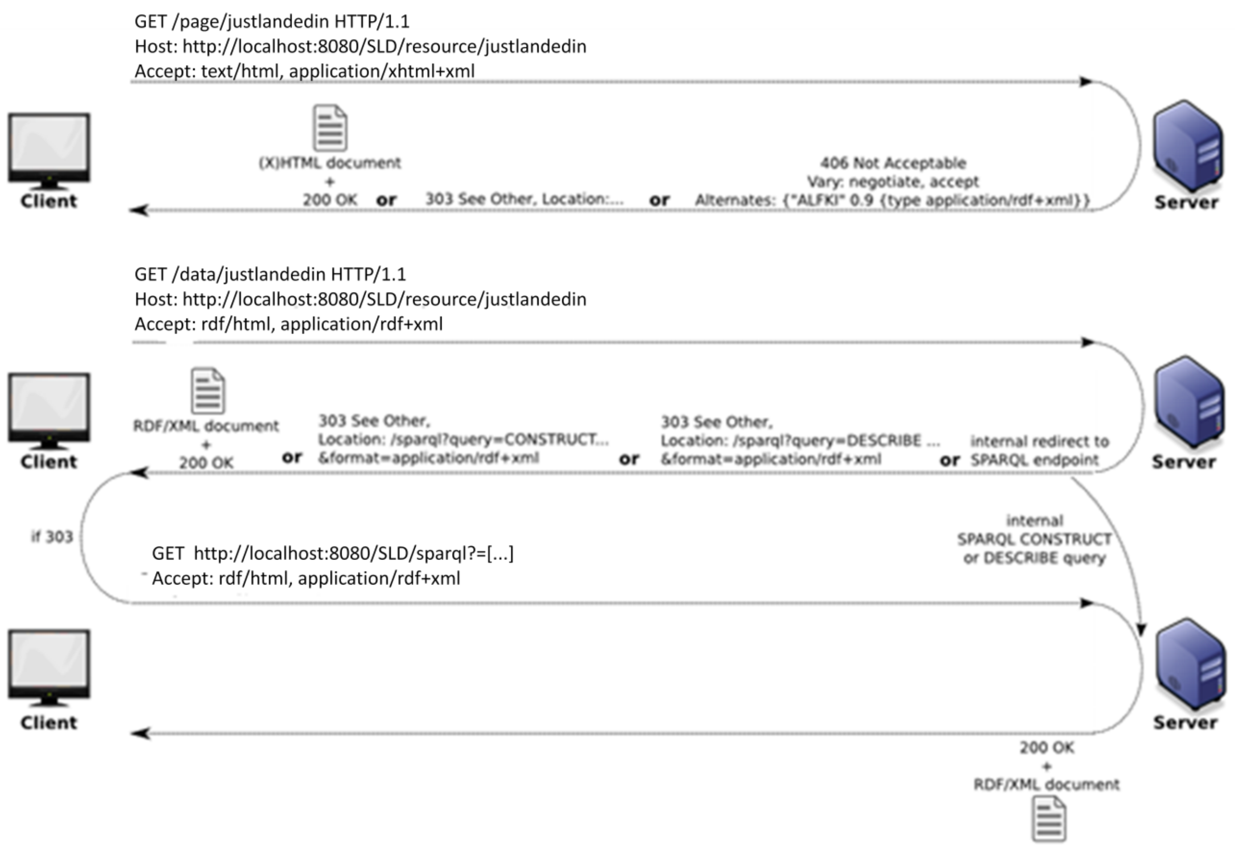

GET /page/justlandedin HTTP/1.1
Host: http://localhost:8080/SLD/resource/justlandedin
Accept: text/html, application/xhtml+xml
Client
(X)HTML document + 200 OK
or
303 See Other, Location:...
or
406 Not Acceptable
Vary: negotiate, accept
Alternates: {"ALFKI" 0.9 {type application/rdf+xml}}
Server
GET /data/justlandedin HTTP/1.1
Host: http://localhost:8080/SLD/resource/justlandedin
Accept: rdf/html, application/rdf+xml
Client
RDF/XML document + 200 OK
or
303 See Other, Location: /sparql?query=CONSTRUCT... &format=application/rdf+xml
or
303 See Other, Location: /sparql?query=DESCRIBE ... &format=application/rdf+xml
or
internal redirect to SPARQL endpoint
Server
if 303
GET http://localhost:8080/SLD/sparql?=[...]
Accept: rdf/html, application/rdf+xml
internal SPARQL CONSTRUCT or DESCRIBE query
Client
Server
200 OK + RDF/XML document

Figure 3.17.1 Http Content Negotiation

3.18. 303 URIs Selection

The first solution is to use a special HTTP status code, “303 See Other”, to distinguish non-document resources from regular web documents. Since 303 is a redirect status code, the server can also give the location of a document that describes the resource. If, on the other hand, a request is answered with one of the usual status codes in the 2XX range, like 200 OK, then the client knows that the URI identifies a web document.

303 URIs, are very flexible because the redirection target can be configured separately for each resource. There could be one describing document for each resource or one large document for all of them, or any combination in between. It is also possible to change the policy later on. But the large number of redirects may cause higher latency.

303 URIs should be used for large sets of data that are, or may grow, beyond the point where it is practical to serve all related resources in a single document. If in doubt, it’s better to use the more flexible 303 URI approach.

3.19. Internet Protocols

Two protocols are essential to be discussed here, as we will see later the importance of them.

1. TCP/IP
2. HTTP

3.19.1. TCP/IP

TCP/IP is a suite of networking protocol. A networking protocol is simply a method of describing information packets so that they can be sent down your telephone-, cable-, or T1-line from node to node, until they reach their intended destination.

When the user tells the browser to fetch a web page, the browser parcels up this instruction using a protocol called the Transmission Control Protocol (or TCP). TCP is a transport protocol, which provides a reliable transmission format for the instruction. It ensures that the entire message is packaged up correctly for transmission (and also that it is correctly unpacked and put back together after it reaches its destination).

3.19.2. HTTP

HTTP (Hyper Text Transport Protocol) is used to address the data which is parceled up by TCP. HTTP is the protocol used by the World Wide Web in the transfer of information from one machine to another – when you see a URL prefixed with http://, you know that the internet protocol being used is HTTP. You can think of TCP/IP as the postal service that does the routing and transfer, while HTTP is the stamp and address on the letter (data) to ensure it gets there.

3.20. Addressable Resources

Addressability is the idea that every object and resource in your system is reachable through a unique identifier. By addressable we mean resources that can be accessed and transferred between clients and servers. Subsequently, a resource is a logical, temporal mapping to a concept in a problem domain for which we are implementing a solution.

In the case of this work, a search result of "just landed in" on twitter is a resource. Even though a resource's mapping is unique, different requests for a resource can return the same underlying representation stored in the server. However HTTP enables us to communicate any kind of information between client and server. It could either be text, a picture, a flash movie and so on. The data is streamed in all cases over TCP/IP and our browser knows how to interpret the streams because of the HTTP protocol's response header Content-Type.

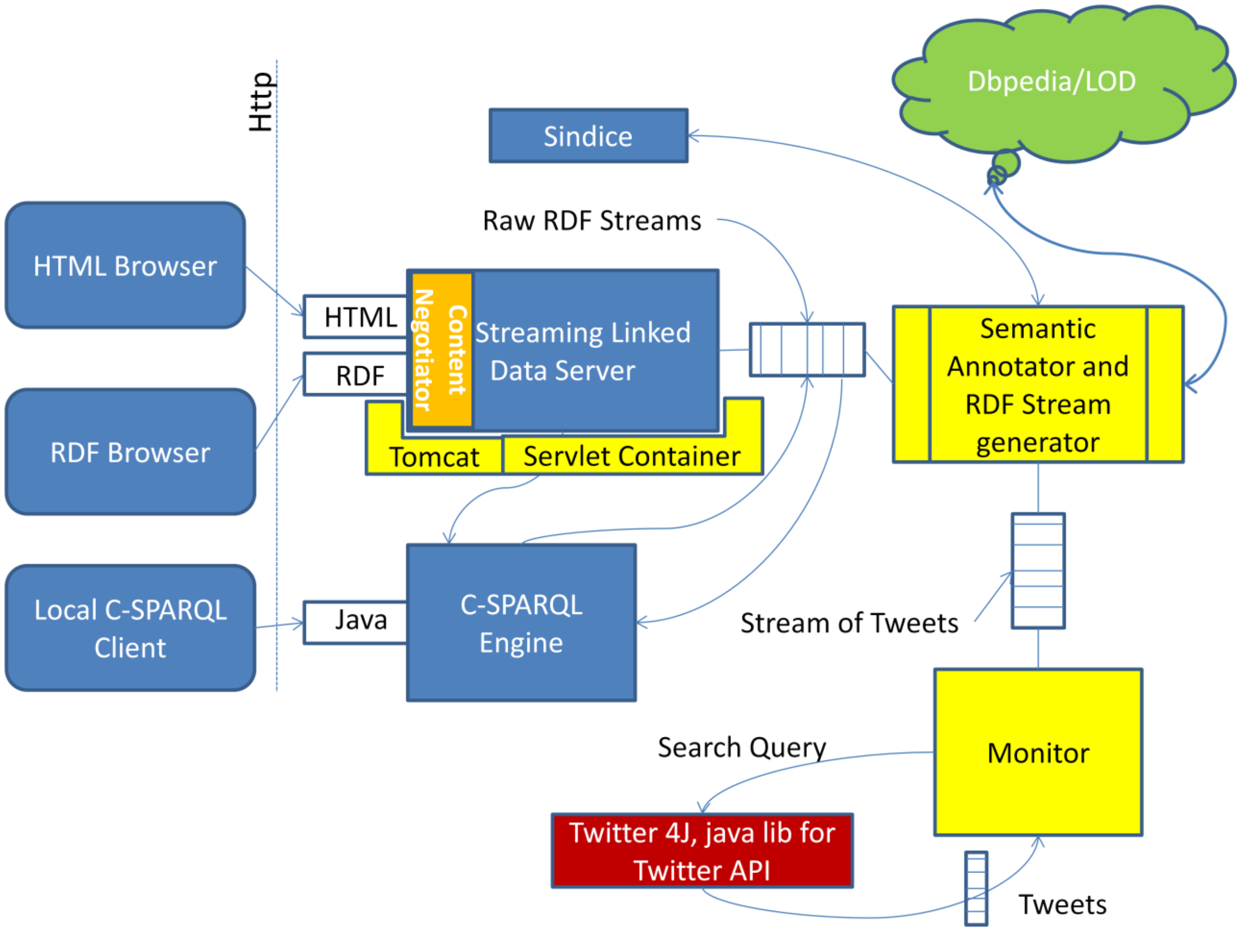
Chapter 4: Implementation

4.1. Overview

The objective of this chapter is to implement the proposed solution. However we elaborate all the major components used in the implementation for this web application.

4.2. Design and Functionality

The objective of this chapter is to implement the proposed solution. However we elaborate all the major components used in the implementation for this web application.



Drawing 4.2.1 Implementation Design

As described earlier about this application is based on client server approach. However we have three possible clients which are the internet web browsers supporting HTML and RDF, whereas, the third client is the SPARQL query endpoint. The compatible browsers are IE6.0, IE7.0, Firefox3.4, firefox3.5 for HTML requests while RDF request can be treated using the tabulator generic data browser or with its extension for Firefox 3.x web browser. So, the user can send a URI as request e.g. http://localhost:8080/SLD/resource/justlandedin, this request will be processed by server, when a web browser has sent this request with this URI, it sent the request along with HTTP header to indicate what data formats and language it prefers.

The user's has sent its request from an HTML browser or RDF browser (see content negotiation section 3.16) the steaming linked data server interprets the URL, locates the corresponding page, and sends back the page to the requesting browser.

Streaming linked data server is among the major components of the application. The server is based on the Java servlet technology. Servlets are the Java platform technology of popular choice for building interactive Web applications. Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. Servlets can also access a library of HTTP-specific calls. This is the component which is actually responsible for the publishing of the data streams gernatated.

JSP technology is an extension of the servlet technology created to support of HTML pages. It makes it easier to combine fixed or static template data with dynamic content. Web container manages the execution of JSP page and servlet components for Java applications. Web components and their container run on the java application server.

Semantic Annotator is the software component built using java, querying the keywords on sindice and DBpedia. These keywords are queried through SPARQL query language. RDF Stream Generator generates the raw RDF Streams by taking the data from sindice and DBpedia. Stream gernerator uses the Jena API for RDF graph creation.

Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Program.

Jena is a Java API which can be used to create and manipulate RDF graphs. Jena has object classes to represent graphs, resources, properties and literals. The interfaces representing resources, properties and literals are called Resource, Property and Literal respectively. In Jena, a graph is called a model and is represented by the Model interface.

Monitor is another software component which actually searches the tweets fulfilling the user search keyword. Monitor generates an output as the list of tweets.

The Twitter API consists of three parts: two REST APIs and a Streaming API. These APIs provide search capability for Twitter data. The Streaming API supports long-lived connections on a different architecture. However the Twitter REST API methods allow developers to access core Twitter data. This includes update timelines, status data, and user information. The Search API methods give developers methods to interact with Twitter Search and trends data. The Streaming API provides near real-time high-volume access to Tweets in sampled and filtered form.

Twitter4J is an unofficial Java library for the Twitter API. With Twitter4J, you can easily integrate your Java application with the Twitter service. Twitter4J is Pure Java and works on any Java Platform version 1.4.2 or later.

4.3. Prototype Application Deployment

Web applications are packaged into a WAR (web application archive). There are many different ways to create a WAR file. We can use jar command, ant or an IDE like Eclipse.

Now take the WAR file of this web application "sld.war"and install it into your container of choice. This war file is compatible and tested on tomcat 5.5 & 6.0

Tomcat deployment is trivial and requires copying the WAR file into the TOMCAT_HOME/webapps folder and restarting the container. However below are the steps to follow.

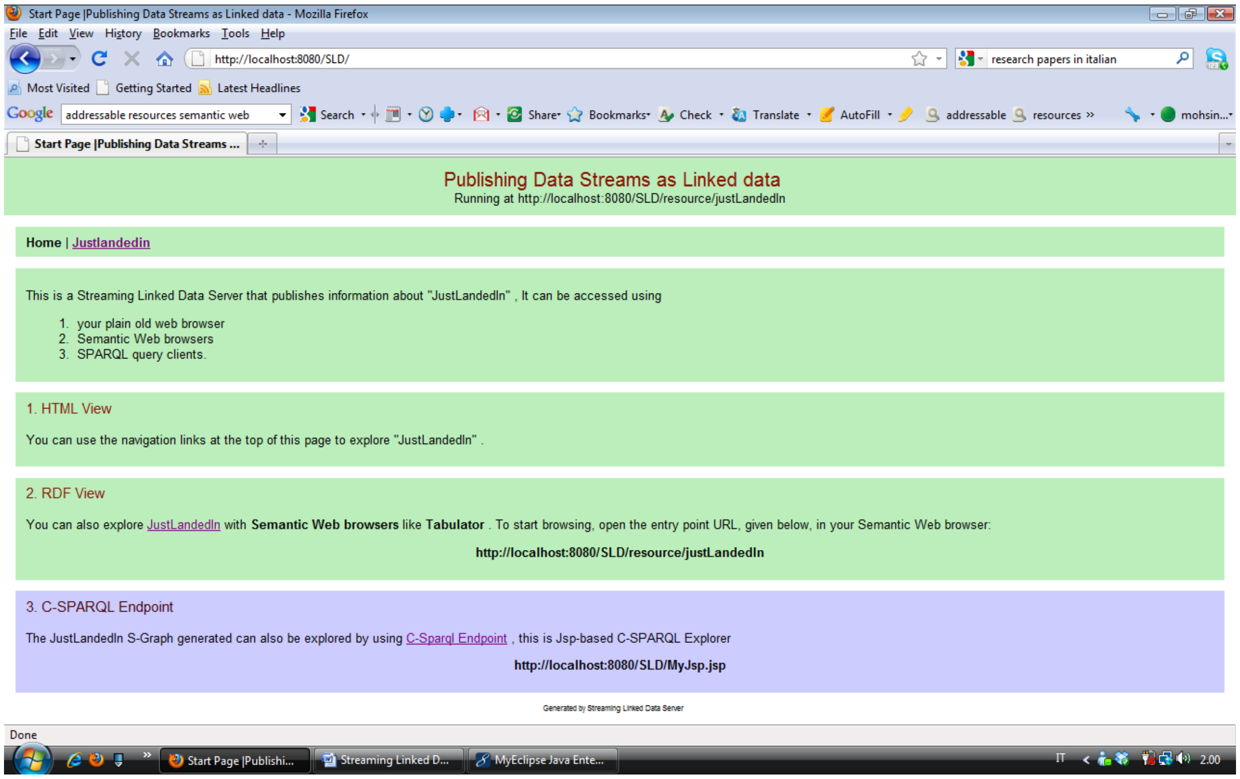
Copy "sld.war" to <TOMCAT_HOME>/webapp directory. That's it! You have successfully deployed the application to tomcat web server.

Once you start the server, tomcat will extract the war file into the directory with the same name as the war file.

To start the server, open the command prompt and change the directory to <TOMCAT_HOME/bin> directory and run the startup.bat file.

4.3.1. How the Application Works?

When you access the application by navigating to URL http://localhost:8080/SLD/ the web server serves the index.html file. Index.html file is specified as welcome file in web.xml file so web server serves this file by default. Below is the startup page of streaming linked data application prototype.


Start Page |Publishing Data Streams as Linked data - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/SLD/
research papers in italian
Most Visited Getting Started Latest Headlines
Google addressable resources semantic web Search Share Bookmarks Check Translate AutoFill addressable resources mohsin...
Start Page |Publishing Data Streams ...
Done
Start Page |Publishi... Streaming Linked D... MyEclipse Java Ente...

Publishing Data Streams as Linked data
Running at http://localhost:8080/SLD/resource/justLandedIn

Home | Justlandedin

This is a Streaming Linked Data Server that publishes information about "JustLandedIn" , It can be accessed using

1. your plain old web browser
2. Semantic Web browsers
3. SPARQL query clients.

1. HTML View

You can use the navigation links at the top of this page to explore "JustLandedIn" .

2. RDF View

You can also explore JustLandedIn with **Semantic Web browsers** like **Tabulator** . To start browsing, open the entry point URL, given below, in your Semantic Web browser:

http://localhost:8080/SLD/resource/justLandedIn

3. C-SPARQL Endpoint

The JustLandedIn S-Graph generated can also be explored by using C-Sparql Endpoint , this is Jsp-based C-SPARQL Explorer

http://localhost:8080/SLD/MyJsp.jsp

Generated by Streaming Linked Data Server

Figure 4.3.1 Streaming Linked Data Server First Page

If you would like to run any of the client's results you can simply click on the links, Click on Justlanded in for HTML request, for RDF use the JustLandedIn link for Tabulator Browser request and you can use the Sparql endpoint for querying the S-graph.

However you can send request directly from your browser using the URI, http://localhost:8080/sld/resource/justlandedin, your browser request will let to know about the request to the Streaming linked data application to generate the specific response.

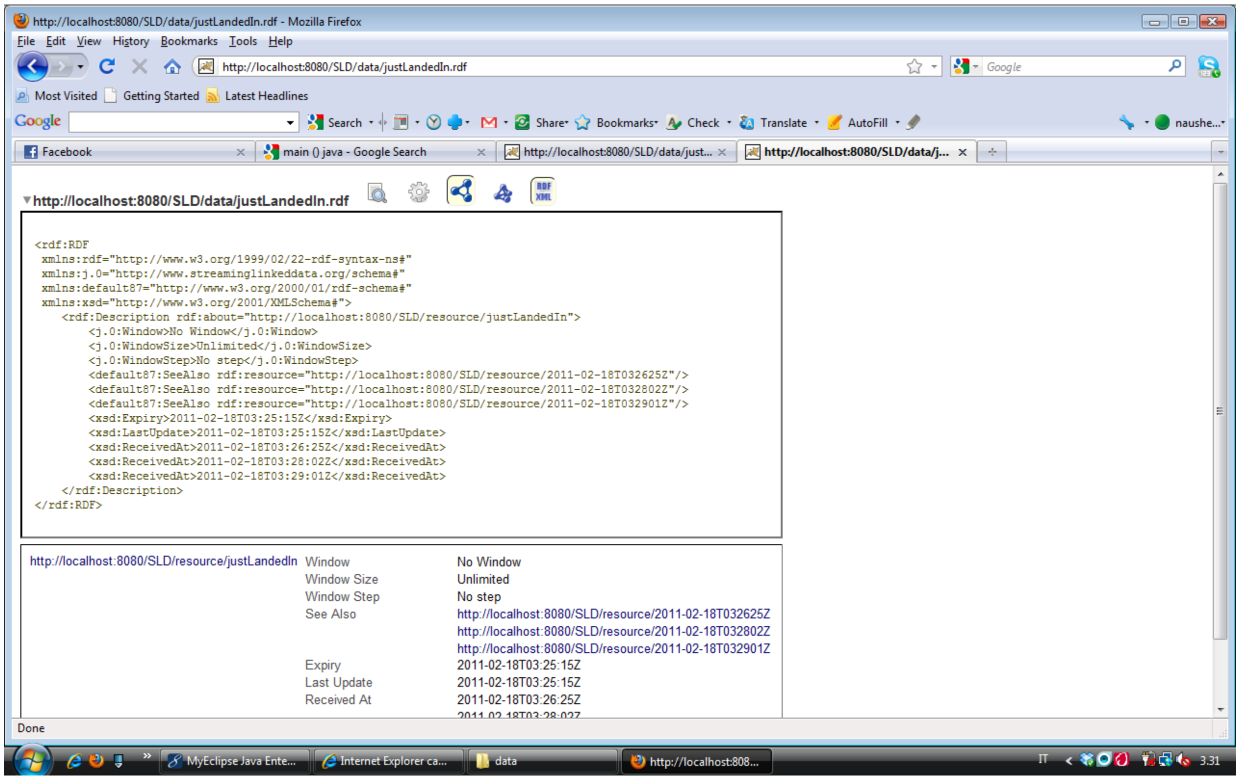
When you hit the URI http://localhost:8080/SLD/resource/justlandedin, browser sends the HTTP GET request. Based on the servlet mapping in web.xml, the web container delegates the request to servlet class.

When the request is received by PublishingServlet it performs following tasks.

- Extract the name parameter from HttpServletRequest object.
- Generate the contents for the request.
- Generate the HTML or RDF document and write the response to HttpServletResponse object.

Browser receives the HTML or RDF document as response and presents that document in its window. Below are given the screen shots of S-Graph presented in RDF and HTML formats.

Resulting S-Graph presented using tabulator add on for Firefox browser.



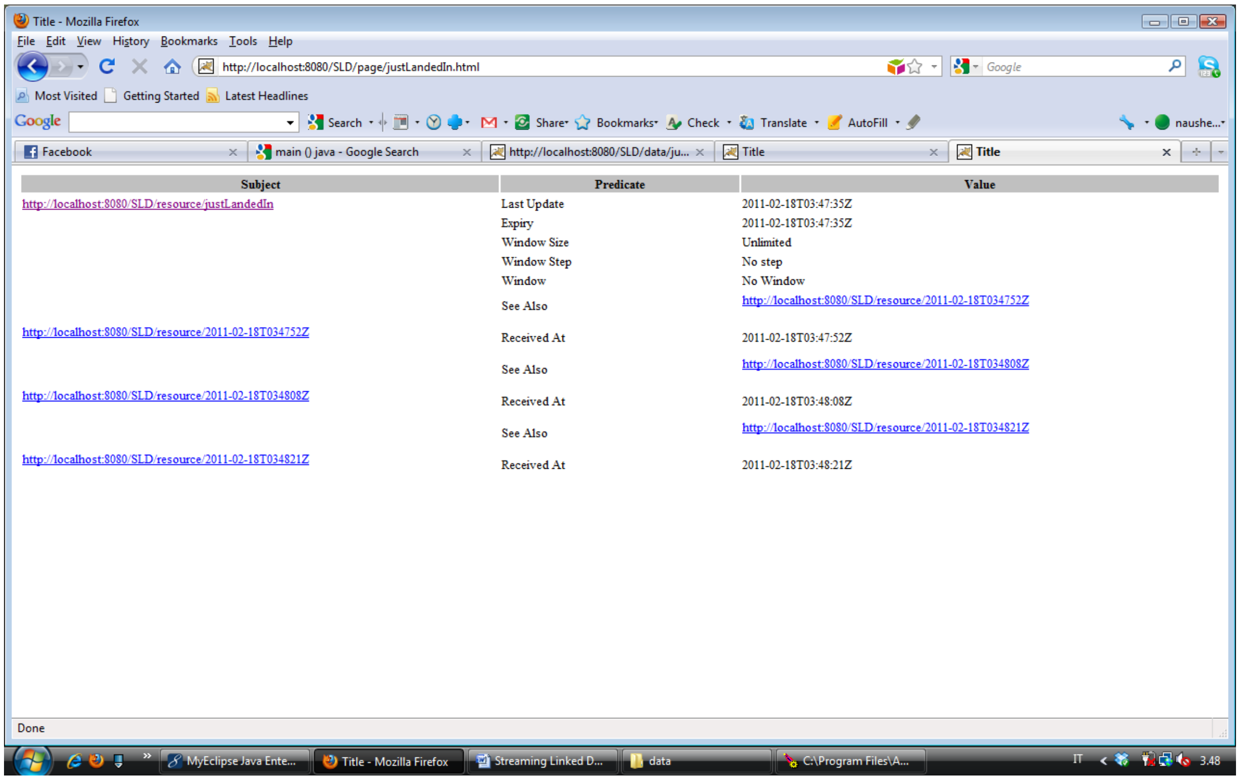
http://localhost:8080/SLD/data/justLandedIn.rdf

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:j.0="http://www.streaminglinkeddata.org/schema#"
 xmlns:default87="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
    <rdf:Description rdf:about="http://localhost:8080/SLD/resource/justLandedIn">
        <j.0:Window>No Window</j.0:Window>
        <j.0:WindowSize>Unlimited</j.0:WindowSize>
        <j.0:WindowStep>No step</j.0:WindowStep>
        <default87:SeeAlso rdf:resource="http://localhost:8080/SLD/resource/2011-02-18T032625Z"/>
        <default87:SeeAlso rdf:resource="http://localhost:8080/SLD/resource/2011-02-18T032802Z"/>
        <default87:SeeAlso rdf:resource="http://localhost:8080/SLD/resource/2011-02-18T032901Z"/>
        <xsd:Expiry>2011-02-18T03:25:15Z</xsd:Expiry>
        <xsd:LastUpdate>2011-02-18T03:25:15Z</xsd:LastUpdate>
        <xsd:ReceivedAt>2011-02-18T03:26:25Z</xsd:ReceivedAt>
        <xsd:ReceivedAt>2011-02-18T03:28:02Z</xsd:ReceivedAt>
        <xsd:ReceivedAt>2011-02-18T03:29:01Z</xsd:ReceivedAt>
    </rdf:Description>
</rdf:RDF>
```

http://localhost:8080/SLD/resource/justLandedIn	Window	No Window
	Window Size	Unlimited
	Window Step	No step
	See Also	http://localhost:8080/SLD/resource/2011-02-18T032625Z
		http://localhost:8080/SLD/resource/2011-02-18T032802Z
		http://localhost:8080/SLD/resource/2011-02-18T032901Z
	Expiry	2011-02-18T03:25:15Z
	Last Update	2011-02-18T03:25:15Z
	Received At	2011-02-18T03:26:25Z

Figure 4.3.2 S-Graph presentation using tabulator add on for Firefox internet browser

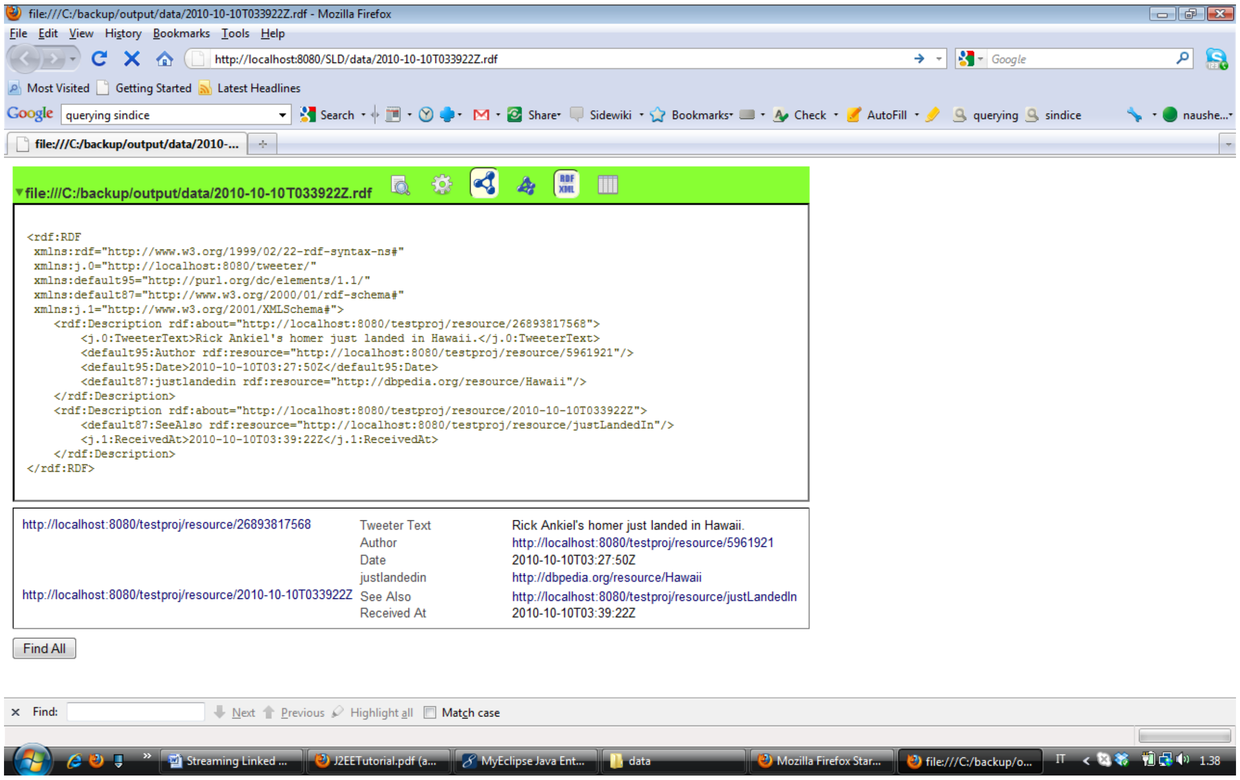
Below is the HTML presentation of S-Graph using Firefox internet browser.



Subject	Predicate	Value
http://localhost:8080/SLD/resource/justLandedIn	Last Update	2011-02-18T03:47:35Z
	Expiry	2011-02-18T03:47:35Z
	Window Size	Unlimited
	Window Step	No step
	Window	No Window
	See Also	http://localhost:8080/SLD/resource/2011-02-18T034752Z
http://localhost:8080/SLD/resource/2011-02-18T034752Z	Received At	2011-02-18T03:47:52Z
	See Also	http://localhost:8080/SLD/resource/2011-02-18T034808Z
http://localhost:8080/SLD/resource/2011-02-18T034808Z	Received At	2011-02-18T03:48:08Z
	See Also	http://localhost:8080/SLD/resource/2011-02-18T034821Z
http://localhost:8080/SLD/resource/2011-02-18T034821Z	Received At	2011-02-18T03:48:21Z

Figure 4.3.3 S-Graph presentation in HTML format using Firefox

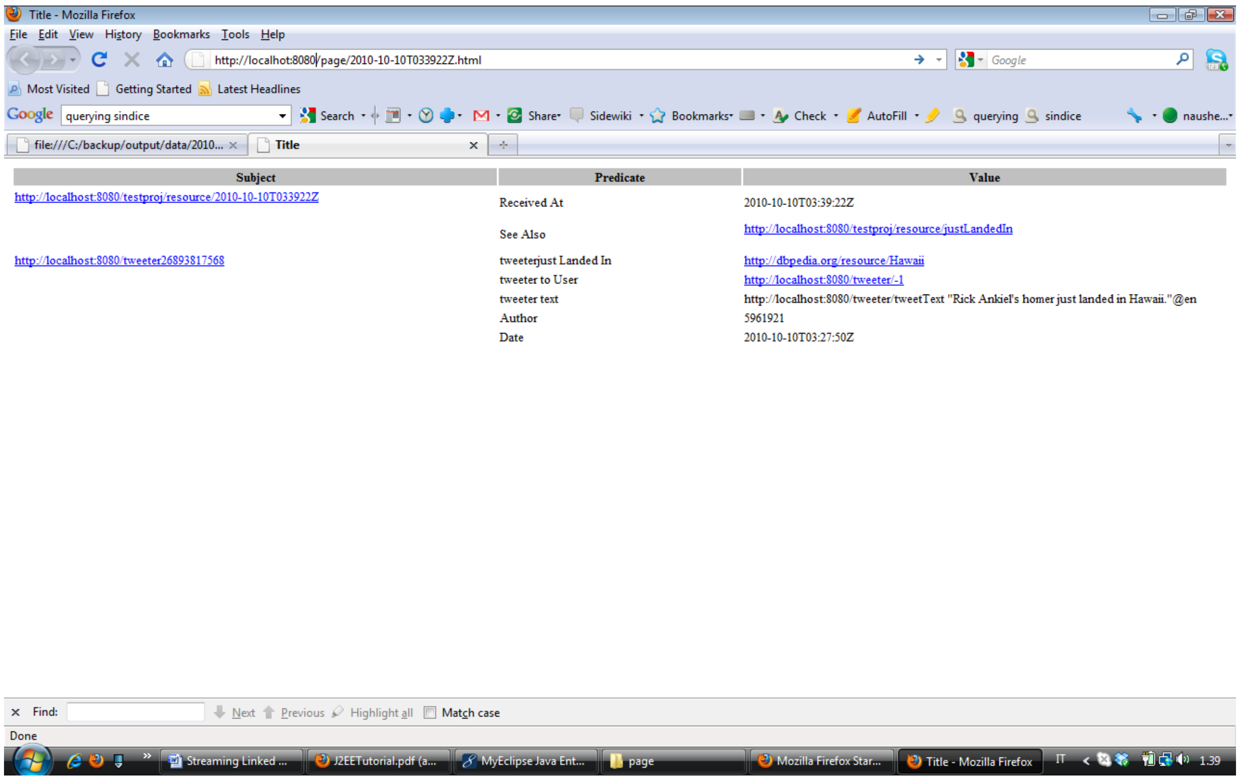
Below is the HTML presentation of I-Graph using Firefox internet browser.


file:///C:/backup/output/data/2010-10-10T033922Z.rdf - Mozilla Firefox
http://localhost:8080/SLD/data/2010-10-10T033922Z.rdf
▼file:///C:/backup/output/data/2010-10-10T033922Z.rdf

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:j.0="http://localhost:8080/tweeter/"
 xmlns:default95="http://purl.org/dc/elements/1.1/"
 xmlns:default87="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:j.1="http://www.w3.org/2001/XMLSchema#">
    <rdf:Description rdf:about="http://localhost:8080/testproj/resource/26893817568">
        <j.0:TweeterText>Rick Ankiel's homer just landed in Hawaii.</j.0:TweeterText>
        <default95:Author rdf:resource="http://localhost:8080/testproj/resource/5961921"/>
        <default95:Date>2010-10-10T03:27:50Z</default95:Date>
        <default87:justlandedin rdf:resource="http://dbpedia.org/resource/Hawaii"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://localhost:8080/testproj/resource/2010-10-10T033922Z">
        <default87:SeeAlso rdf:resource="http://localhost:8080/testproj/resource/justLandedIn"/>
        <j.1:ReceivedAt>2010-10-10T03:39:22Z</j.1:ReceivedAt>
    </rdf:Description>
</rdf:RDF>
```

http://localhost:8080/testproj/resource/26893817568	Tweeter Text	Rick Ankiel's homer just landed in Hawaii.
	Author	http://localhost:8080/testproj/resource/5961921
	Date	2010-10-10T03:27:50Z
	justlandedin	http://dbpedia.org/resource/Hawaii
http://localhost:8080/testproj/resource/2010-10-10T033922Z	See Also	http://localhost:8080/testproj/resource/justLandedIn
	Received At	2010-10-10T03:39:22Z

Figure 4.3.4 I-Graph presentation using tabulator add on for Firefox internet browser



http://localhot:8080/page/2010-10-10T033922Z.html

Subject	Predicate	Value
http://localhost:8080/testproj/resource/2010-10-10T033922Z	Received At	2010-10-10T03:39:22Z
	See Also	http://localhost:8080/testproj/resource/justLandedIn
http://localhost:8080/tweeter26893817568	tweeterjust Landed In	http://dbpedia.org/resource/Hawaii
	tweeter to User	http://localhost:8080/tweeter/-1
	tweeter text	http://localhost:8080/tweeter/tweetText "Rick Ankiel's homer just landed in Hawaii."@en
	Author	5961921
	Date	2010-10-10T03:27:50Z

Figure 4.3.5 I-Graph presentation in HTML format using Firefox

Given below is the quries done on DBpedia and sindice. The next step is to understand the code.

Queries performed on DBpedia:

```
dbpedia += "WHERE { " +
                        "{GRAPH ?graph {?graph a 
<http://dbpedia.org/ontology/PopulatedPlace> }} " +
                        "UNION " +
                        // a country
                        "{GRAPH ?graph {?graph a 
<http://sw.opencyc.org/2008/06/10/concept/Mx4rvViACZwpEbGdrcN5Y29ycA> }} " +
                        "UNION " +
                        // a state capital
                        "{GRAPH ?graph {?graph a 
<http://sw.opencyc.org/2008/06/10/concept/Mx4rvVi3vpwpEbGdrcN5Y29ycA> }} " +

                        "}";
            System.out.println(dbpedia);
            com.hp.hpl.jena.query.Query q2 = QueryFactory.create(dbpedia);
            QueryExecution innerqexec2 = QueryExecutionFactory.create(q2);
            ResultSet innerresultSet2 = innerqexec2.execSelect();
```

Queries performed on Sindice

```
String sindice = "SELECT ?link "
                  + "FROM
<http://api.sindice.com/v2/search?q="+encodedPlace+"&qt=term&page=1> "
                  + "WHERE { ?x <http://sindice.com/vocab/search#link> ?link
. "
                  + "   FILTER (regex(str(?link), \"dbpedia\" )) " + " }";
//          System.out.println(sindice);
            com.hp.hpl.jena.query.Query q = QueryFactory.create(sindice);
      QueryExecution innerqexec = QueryExecutionFactory.create(q);
      ResultSet innerresultSet = innerqexec.execSelect();
```

Queries performed for windowing using C-SPARQL

```
final String queryCountLogicalWindow = "REGISTER QUERY PIPPO AS "
                  + "SELECT ?t (count(?t) AS ?conto)" + " FROM STREAM
<http://localhost:8080/sld/resource/justlandedin> [RANGE
“+”justlandedin.size”+”justlandedin.step”+”] WHERE { ?s ?p ?o } "
                  + "GROUP BY ?t ";

final String queryphysicalwindow = "REGISTER QUERY PIPPO AS "
                  + "SELECT ?s (COUNT(?s) AS ?conto) FROM STREAM
<http://localhost:8080/sld/resource/justlandedin> [RANGE
TRIPLES”+”justlandedin.size”] WHERE { ?s ?p ?o } GROUP BY ?s";


 final justlandedinGenerator tg = new justlandedinGenerator();
```

Registering C-SPARQL queries

```
engine.registerStream(tg);

final Thread t = new Thread(tg);
```

4.3.2. Configurations

The deployment descriptor (web.xml) file

Copy the following code into web.xml file and save it directly under servlet-example/WEB-INF directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

<listener>
      <listener-class>eu.larkc.csparql.plugins.streaminglinkeddata.MainClass
      </listener-class>
      </listener>

<listener>
      <listener-class>eu.larkc.csparql.plugins.streaminglinkeddata.ForTest
      </listener-class>
      </listener>

<listener>
      <listener-
      class>eu.larkc.csparql.plugins.streaminglinkeddata.ForjspmainThread
      </listener-class>
      </listener>

 <Servlet>
  <display-name>PublishingServlet</display-name>
  <servlet-name>PublishingServlet</servlet-name>
      <servlet-
      class>eu.larkc.csparql.plugins.streaminglinkeddata.PublishingServlet
      </servlet-class>
 </servlet>

  <servlet-mapping>
        <servlet-name>PublishingServlet</servlet-name>
        <url-pattern>/resource/*</url-pattern>
        <url-pattern>/data/</url-pattern>
        <url-pattern>/page/</url-pattern>
        <url-pattern>/data/physical/</url-pattern>
        <url-pattern>/page/physical/</url-pattern>
        <url-pattern>/data/logical/</url-pattern>
        <url-pattern>/page/logical/</url-pattern>
        </servlet-mapping>
<welcome-file-list>
      <welcome-file>index.html</welcome-file>
      </welcome-file-list>
<login-config>
      <auth-method>BASIC</auth-method>
      </login-config>

</web-app>
```

The deployment descriptor (config.xml) file, copy the following lines of code into config.xml file and save it directly under servlet-example/WEB-INF directory

```
<SLD>
      <logininfo>
            <username>nausherwan644@hotmail.com</username>
            <password>amirkhalil12</password>
            <placetoannotate>""</placetoannotate>
  </logininfo>

</SLD>
```

Note: see war deployment for Dbpedia and Sindice queries

Congratulations! If you have reviewed the deployment procedure and both web.xml and config.xml configuration files then see how it works.

Chapter 5: Future and Related Work

5.1. Overview

In this chapter, we will discuss our implementation experience, analyzing the further enhancements in the current research, proposing the future work for the same work domain and then we will conclude our research work.

5.2. Controlling C-SPARQL Quries

We describe the RESTful (34) services which allow controlling each C-SPARQL query that continuously computes each RDF stream published with our approach. As we did that C-SPARQL queries have to be registered in the C-SPARQL Engine. As soon as a query is registered, the C-SPARQL engine starts to compute it. An explicit stop command is required to stop the processing of a registered query. Similarly in order to unregister the query we need an explicit delete command for deleting a C-SPARQL query.

5.3. Future Work

A RESTful service to control the C-SPARQL queries that generates the RDF streams is also detailed. This service could help us to complete another prototypical implementation of our Streaming Linked Data Server and evaluate it against several use cases.we need to extend our work with the design of a RESTful interface. A RESTful uses the HTTP methods to control the C-SPARQL queries:

- PUT, with a C-SPARQL query as parameter, allows registering a query that generates a certain RDF stream,
- POST, with start or stop command as parameters, is used to start or stop a registered query, and
- DELETE can be used to unregister a query.

Similarly this work is extendable and presentable for the social network service websites as we did elaborate the tweets of the twitter.
We intend to focus on the optimal deployment of multiple continuous queries over streams in distributed and heterogeneous environments, where RDF repositories and data streams will be managed by different systems, and stream managers may exhibit limited data management capabilities. We believe that generalizing some of the results presented in this paper in a multi-query, heterogeneous, and distributed context is possible, although far from trivial.

CSPARQL should solve the challenge of providing efficient access protocols for heterogeneous streams.

We should measure the performance of the data streams processed on the fly and by storing these streams first using Db4 or some other database system.

We can extend this work to measure the performance of the C-SPARQL engine and existing DSMS systems in order to answer the questions that which system exhibit better or ideal performance to process data streams on the fly.

5.4. Related Work

Two previous works (35) (36) address the need for publishing data streams as Linked Data. In (35), Corcho introduce the concept of Linked Stream Data, a way in which the Linked Data principles can be applied to stream data and be part of the Web of Linked Data. At a first glance, his proposal could appear similar to our one. Both his and our proposal use named graphs and define IRI schemata. However, his approach does take into account the nature of streams, that being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored (forever) and queried on demand, but rather as transient data to be consumed on the y by continuous queries. His proposal allows for opening window of any dimension and any position in time (see listing below):

This is incompatible with principle to consume data on the fly. It requires the Linked Stream Data server to store the stream for an indefinite time period.

```
http :// www. domain . org / sensor / name /\% start time \% ,\% end time \%
```

In (36), Rodrfiguez et al. introduce the notion of Time-Annotated RDF (TA-RDF) that allows for representing timeseries data, especially streaming data, using the Semantic Web approach. (TA-RDF) is an extension of the RDF model where resources are optionally annotated with a time
value, i.e, a time-annotated resource is a pair of the form resource[time] (see listing below for an example).

```
<urn:OHARE > <urn : hasRainSensor > <urn : sensor1 > .
<urn:sensor1 >["2009 -01 -01Z -06:00"^^ xsd : date ] <urn: hasReading > "0".
<urn:sensor1 >["2009 -01 -01Z -06:05"^^ xsd : date ] <urn: hasReading > "5".
...
<urn:sensor1  >["2009  -01  -31Z  -10:00"^^  xsd  :  date  ]  <urn:  hasReading  >
"15".
```

A TA-RDF graph can be represented as a set of RDF graphs using two special properties: belongsTo, which indicates a data element in a stream, and hasTimestamp, which points toward the timestamp of the data element. As for the previous related work, TA-RDF proposal looks very similar to our one, but still it lacks the paradigmatic change from persistent data to transient data. In TA-RDF streams are supposed to be stored indefinitely. Finally, the two proposals do not consider the rich types of windows proposed in DSMS. They do not propose a vocabulary to describe the window type (i.e., lsd:physical vs. lsd:logical) and the size of the window (i.e., the equivalent of our property windowSize). The properties lastUpdate and expires, which in our vocabulary allows to indicate a Linked Data Client when the graph was updated and when it will expire, are not present.

The TA-RDF and TA-SPARQL have been implemented in a prototype service, integrated with the NCSA Tupelo semantic middleware. This has enabled integration of stream data with other data and metadata, including documents, imagery, workflows, provenance, and annotations.

We believe that providing the ability to reason about streaming data to cope with the increasing amount of dynamic data on the web is the next big step in semantic technologies.

5.5. Conclusions

The proposition of this work provides an extension of C-SPARQL Engine by publishing data streams as Linked Data. C-SPARQL and their corresponding infrastructures provide an excellent starting point for further investigating expressive reasoning over data streams. Our proposition to publish RDF streams continuously generated by C-SPARQL queries is successfully using our concepts of Stream Graph (or s-graph) and Instantaneous Graph (or igraph) as well as a small vocabulary that allows to describe which part of the stream has been published and when the information will expire.

Windows data model is much more flexible and is very natural for query processing. Our technique of using the C-SPARQL query worked very well, however still there is a need to control the continuous queries and this could be better by using A RESTful service as we proposed it in section of future work.

6. References

1. *Rastogi. Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications).* **M. Garofalakis, J. Gehrke, and R.** Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2007.

2. *STREAM: The Stanford Stream Data Manager (Demonstration Description).* **A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom.** 2003. In Proc. ACM Intl. Conf. on Management of Data (SIGMOD 2003). p. page 665.

3. *The Design of the Borealis Stream Processing Engine.* **D. J. Abadi, Y. Ahmad, M. Balazinska, U. C_ etintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik.** In Proc. Intl. Conf. on Innovative Da.

4. *Introducing Stream Mill.* **Hetal Thakkar, Nikolay Laptev, Vincenzo Russo and Carlo Zaniolo.** Los Angeles : s.n., April 2009.

5. *An Execution Environment for C-SPARQL Queries. .* **D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus.** 2010. In Proc. Intl. Conf. on Extending Database Technology (EDBT).

6. *Issues in Data Stream Management, ACM SIGMOD Record vol 3.* **Golab, L., Özsu, M. T.** June 2003. p. 5-14 .

7. *Querying RDF Streams with C-SPARQL.* **Davide Francesco Barbieri, Daniele Braga, Stefano Ceri,Emanuele Della Valle, Michael Grossniklaus.**

8. *STREAM: The Stanford Data Stream Management System.* **Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom.**

9. *Models and issues in data stream systems. In Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems.* **B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom.** June 2002. p. 1,16.

10. *Flexible time management in data stream systems. In Proc. of the 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems.* **Widom., U. Srivastava and J.** June 2004.

11. *Monitoring Streams – A New Class of Data Management Applications.* **Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul and Stan Zdonik,.** honkong : s.n., 2002.

12. *"Resource Description Framework (RDF) Model and Syntax Specification".* **http://www.w3.org/TR/PR-rdf-syntax/.**

13. **http://sourceforge.net/projects/rdquery/, RDQuery.** 2006.

14. **http://www.w3.org/DesignIssues/LinkedData.html.**

15. *Rdf/xml syntax specification (revised). W3c recommendation, W3C, Oct 2004. http://www.w3.org/TR/rdf-syntax-grammar/.Tim Berners-Lee.* **Beckett., Dave.**

16. *Mailing list message to www-tag@w3.org, http://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html.* **Fielding, Roy T.** june 2005.

17. *Best practice recipes for publishing rdf vocabularies.* **Alistair Miles, Thomas Baker, and Ralph Swick.** March 2006.

18. *Data Stream Management Processing High-Speed Data Streams.* **Garofalakis, Minos, Gehrke, Johannes e Rastogi, Rajeev (Eds.).**

19. *Semantic Management of Streaming Data. Alejandro Rodriguez, Robert McGrath, Yong Liu and James Myers. Semantic Sensor Networks Workshop at ISWC.* **Andre Bolles Marco Grawunder Jonas Jacobi Davide F. Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Michael Grossniklaus Semantic Management of Streaming Data. Alejandro Rodriguez, Robert McGrath, Yong Liu and James Myers,Emanuele Della Valle.** 2009.

20. *The CQL Continuous Query Language: Semantic Foundations and Query Execution.* **A. Arasu, S. Babu, and J. Widom.** 2006. The VLDB Journal. p. 5(2):121,142.

21. *Sindice.com: a document-oriented lookup index for open linked data.* **E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello.** 2008. p. 3(1):37,52.

22. *C-SPARQL: SPARQL for Continuous Querying.* **D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus.** s.l. : In Proc. Intl. Conf. on World Wide Web (WWW), pages 1061,1062, 2009.

23. *D2r map - a database to rdf mapping language.* **Bizer, C.** 2003.

24. *Querying Sliding Windows over On-Line Data Streams.* **Golab, Lukasz.** School of Computer Science, University of Waterloo Waterloo, Ontario, Canada N2L 3G1 : s.n.

25. *How to publish linked data on the web.* **C. Bizer, R. Cyganiak, and T. Heath.** 2007, 2008, 2009.

26. *Surfing wavelets on streams: One-pass summaries for approximate aggregate queries.* **A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss.** 2001. In Proceedings of VLDB.

27. *Cool uris for the semantic web. World Wide Web Consortium.* **Vlkel, D. Ayers and M.** december 2008.

28. **service.http://www.nws.noaa.gov/, NOAA. U.S. national weather.**

29. *the stream: architecture for queries over streaming sensor data.* **Fjording, S. Madden and M. J. Franklin.** In Proceedings of ICDE.

30. *Maintaining Stream Statistics over Sliding Windows. to appear in Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms.* **M. Datar, A. Gionis, P. Indyk, and R. Motwani.**

31. *An Execution Environment for C-SPARQL Queries. .* **D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus.** In Proc. Intl. Conf. on Extending Database Technology (EDBT), 2010.

32. *D2rq - treating non-rdf databases as virtual rdf graphs.* **Seaborne., C. Bizer and A.** nov 2004. In ISWC2004 (posters).

33. *SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/.* **Seaborne., E. Prud'hommeaux and A.**

34. *The CQl Continuous Query Language: Semantic Foundations and Query Execution.* **A. Arasu, S. Babu, and J. Widom.** s.l. : The VLDB Journal, 2006.

35. *Hypertext transfer protocol.* **al, R. Fielding et.** June 1999.
36. *RESTful Web Services.* **Ruby., Richardson and S.** O'Reilly, Beijing : s.n., 2007.

37. *Linked stream data: A position paper.* **Corcho., O.** 2009. In The 2nd International Workshop on Semantic sensor Networks.
38. *Semantic management of Streaming Data.* **A. Rodriguez, R. McGrath, Y. Liu, and J. Myers.** 2009. In Proc. Intl. Workshop on Semantic Sensor Networks (SSN).

39. *A Data Stream Language and System Designed for Power and Extensibility. In Proc.* **Y. Bai, H. Thakkar, H. Wang, C. Luo, and C. Zaniolo.** 2006. Intl. Conf. on Information and Knowledge Management (CIKM 2006). p. pages 337,346.
40. *International Organization for Standardization. Data elements and interchange formats | information interchange | representation of dates and times. ISO 8601,Available on line at: http://xml.coverpages.org/ISO-FDIS-8601.pdf.* Dec 2004.
41. **R. Fielding et al. Hypertext transfer protocol – http/1.1. RFC 2616, Jun 1999. http://www.ietf.org/rfc/rfc2616.txt.**