From the Web of Documents to the Linked Data

Gabriel Képéklian[1](✉), Olivier Curé[2,3], and Laurent Bihanic[1]

[1] Atos France, Beauchamp, France
{gabriel.Kepeklian,laurent.Bihanic}@atos.net
[2] Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, 75005 Paris, France
olivier.cure@lip6.fr
[3] CNRS, UMR 7606, LIP6, 75005 Paris, France

Abstract. We start off this paper with a description of the evolution that led from the first version of the Web to the Web of data, sometimes referred to as the Semantic Web. Based on the "data > information > knowledge" hierarchy, we adopt a usage-based perspective. We then make explicit the structures of knowledge representation and the building blocks of the Web of data. In the next chapters, we show how RDF (*Resource Description Framework*) data are managed and queried. This allows us to describe an innovative data processing platform which produces linked data. We continue on ontologies, from their creation to their alignment that allows us developing the delicate question of reasoning. We can then conclude on research and development prospects offered by the linked data environment.

1 Some Historical Milestones to the Web of Data

As the main designer and visionary of the Web, Tim Berners-Lee had anticipated on the evolution from a Web of documents to a Web of data. In order to apprehend correctly the development and full dimension of this emerging Web and its promising research perspectives, it is important to clarify the notions of data, information and knowledge.

1.1 Data's Stance in the Early Web

As stated in [3], "Data are defined as symbols that represent properties of objects, events and their environment". Taken alone, data are not very useful. They have value when they are correctly interpreted.

In the first version of the Web, end-users were the only agents with the ability to provide such an interpretation. In fact, programs supporting the Web were manipulating data stored in complex data structures but were only delivering documents containing them. This motivated the Web of Documents designation. With the amount of Web pages rapidly increasing, it was necessary to organize things in order to help the search process. Along directories like *dmoz.org*, appeared search engines, e.g., *Altavista*, *Yahoo!*, *Google*. But the problem was still

E. Zimányi and R.-D. Kutsche (Eds.): eBISS 2014, LNBIP 205, pp. 60–87, 2015.
DOI: 10.1007/978-3-319-17551-5_3

the same: end-users were still being proposed documents out of search expressed in terms of few words. The search engines were not capable of interpreting the data contained in HTML documents.

We well know the confusion and errors that this entails. For instance, if one enters the word "Venus" in a search engine, what one can expect to get as a result? The planet, the Botticelli painting (i.e., The birth of Venus) or the tennis player (i.e. Venus Williams)? When the search possibilities remain lexical, it is not possible to take the context into account, and so to precise the field of knowledge, to know the freshness of the informations in the page, impossible to establish their reliability, to evaluate the sources, to compare with other available sources, etc.

Alone, data has no meaning. Data has a value only when placed in a precise context and when that context in itself gives sense or contributes to give one, that is to say, it provides access to meaning by building relations.

1.2 Information Emergence

The notion of information helps in decision-making or in problem solving. It is of a different order than the data. Information is a message that contains meaning, and whose inclusion will allow decision or action. Although well understood, this definition does not tell us how to achieve that goal.

It is important to understand that it is when we express something about a data that it becomes an information that we can share with people. The language allows locating the data in a statement; it places the data either as the subject of the sentence, or as the object. In "John eats an apple", we have two data: "John" and "apple". With the presence of the verb "eat", the sentence becomes information and the assembly makes sense. To move from data to information, we required a vocabulary.

At its origins, the Web was presented as a "global information space of linked documents". The fact that the links are between the documents is a main limitation especially due to the indescribable relationship between information and the document. How should we arrange the information to make a document? How to decompose a document? What is the nature of links between documents? As we know, these links between documents are purely technical, they have no semantic value. If two documents are linked, it is not possible to qualify the relationship between them.

It is well understood that there is information in the document. But the links are also rich in information. And in some cases, it can even be considered that it is these links that bring the most value and meaning.

1.3 The Advent of Knowledge

The information depends on two factors: the first deals with the data, these are facts emanating from sources, the second is communication between entities. For stakeholders to understand the communication, they must share the knowledge the information refers to.

Moreover, it is a fact often heard that we produce data in excess. And the fear that too much data kill information is not far. Certainly, the data deluge is real, the phenomenon of big data is not a fiction and mechanically causes an inflation of the same order of magnitude for the information. But if big data can lead to big information, big information doesn't lead to big knowledge.

Knowledge is a set of conceptual structures. This is the reason why the volume of information is uncorrelated with the knowledge that is expressed. For example, once we know a language, we are able to share all information with our neighbors.

Knowledge meets the requirement of a group to share information, understand it to take decisions or undertake actions, as we mentioned earlier in this section.

1.4 Summary

A. Liew noted that the usual definitions of data, information and knowledge had all one major flaw: "they are defined with each other, *i.e.* data in terms of information, information is defined in terms of data and/or knowledge, and knowledge is defined in terms of information. If we are just describing the interrelationships, that is all very well. However, with regard to definitions, this is a logical fallacy i.e. circular definitions or argumentations" [15].

It is precisely because there are relationships between these concepts, that a path leads from the data to the linked data. "Data are Elements of analysis (. . .) Information is data with context (. . .) Knowledge is information with Meaning" [4]. The knowledge is then used to understand, explain, decide and act.

2 The Structures of Knowledge Representation

At the heart of information architecture, classification techniques are particularly essential while growth of online data volume is accelerating. The ratio of what we are looking for to what we are not looking for is increasingly unfavorable in constant technology. The Semantic Web offers new approaches to tackle this problem. We will address them in a graduated manner.

2.1 Controlled Vocabulary

As the name suggests it, the purpose of controlled vocabularies is to control the set of terms that forms a vocabulary. The name of the days of the week is an example for understanding that this set is of finite dimension and can be exhaustively described. The list of species names is another example (see Fig. 1); but that will never be exhaustively defined because we keep discovering new species on a regular basis. These two examples show the character of referential lists of controlled vocabularies where the only relationship between the elements is that of belonging to the whole.

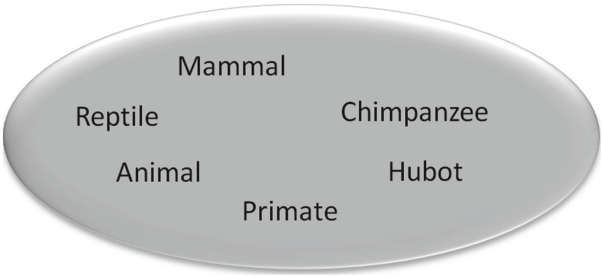

Fig. 1. Controlled vocabulary

In practice, controlled vocabularies often consider the following good practice: all words must be expressed in the same language. A set of names of the months of the year that would mix different languages, e.g., French and English, would serve no purpose. It should also be noted that the translation of a controlled vocabulary in another language can induce semantic variations that the structure cannot state. For example, the set of all the names given to the green color does not have the same cardinality in different languages. So, a good practice is to better characterize the controlled vocabulary.

The advantages of controlled vocabularies are the limitation of indexing disparity, of synonymy problems and of multiple or incorrect spellings.

2.2 Taxonomy

The need to build sub-groups in a controlled vocabulary, to organize terms into categories, leads to a tree-like structure which is quite natural, see Fig. 2. It becomes possible to place elements not only in respect to the whole, but also in relation to each other using hierarchical relations. These ordering relationships take into account classified realities like movie or musical genres, menus of an application, etc.

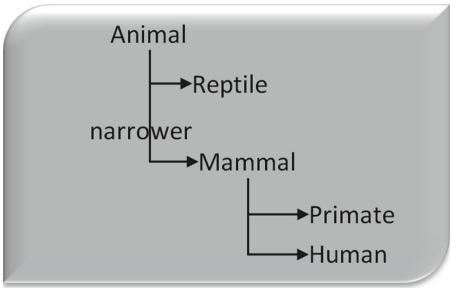

Fig. 2. Taxonomy

A taxonomy has several advantages. It is a classification system very simple to build, to understand and to exchange about. It provides a mapping of knowledge organized in levels of increasing details.

2.3 Thesaurus

The thesaurus is a structure which increases again the level of represented knowledge. It is often presented as a linguistic tool to overcome the ambiguities of natural language when defining shared concepts.

But unlike taxonomies, thesauri structure is rarely strictly hierarchical - otherwise we would be satisfied with taxonomies. Indeed, beyond the hierarchical relationships, a thesaurus allows using other properties to describe relationships between terms.

The most common properties are:

- BT = Broader Term (a symbol used in a thesaurus to identify the following terms as broader terms to the heading term);
- NT = Narrower Term (indicates one or more specific, or child descriptors)
- RT = Related Term ("See also")
- SN = Scope Note (if present, provides an explanation on using the given descriptor)
- TT = "Top Term" (identifies the "ancestor" term, the highest in the hierarchy, the highest using the relationship "BT")
- UF = Used For
- USE = "See" (refers the reader from a variant term to the vocabulary term).

Figure 3 provides a small example of a thesaurus using several of these properties. A thesaurus also allows stating a preference between several terms describing the same concept. Taxonomies do not permit such descriptions.

SKOS (*Simple Knowledge Organization System*) is an RDF vocabulary for modeling and defining thesaurus (and of course taxonomies). The UNESCO

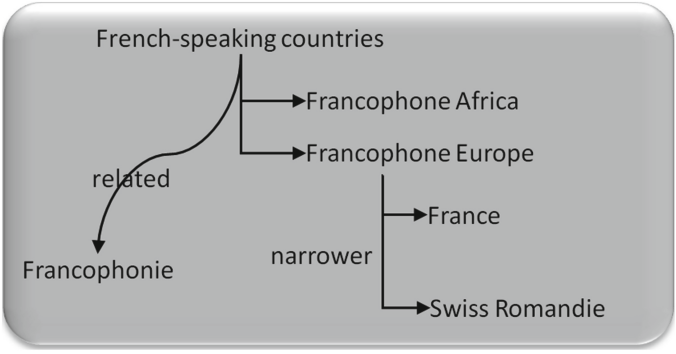

Fig. 3. Thesaurus

thesaurus[1] is a fine example that can be queried easily through a Web interface. It is very rich and continuously updated.

2.4 Ontology

Ontologies represent the higher degree of sophistication of knowledge structuring. Unlike previous approaches offering a limited number of relationships, it is possible to define new relationships in an ontology and thus to act on the language description of a domain of knowledge. An ontology is indeed a formal language, that is to say, a grammar that defines how the terms can be used together.

The aim consists in giving a unique meaning to concepts in order to avoid polysemy and to achieve consensus within a community of practice. Contrary to the approach of expert systems, an ontology describes a field of knowledge regardless of the uses that will be made.

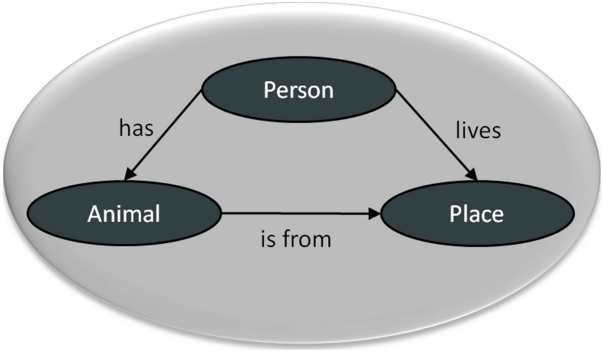

Fig. 4. Ontology

The ontology terms may be used to select the knowledge in which they are used. They can therefore be used as selection criteria, *i.e.*, viewpoints. The terms can be arranged, different perspectives on knowledge can be structured. In Fig. 4, concepts (Animal, Person, Place) are linked by relations (has, lives, is from). Note that the links are generally oriented and some cyclic graphs can be obtained. Many practical ontology languages are being represented using a logical formalism and will hence support some form of automatic reasoning (see Sect. 6 of this paper).

2.5 Summary

As we have seen, knowledge is mainly designed for sharing. And to share knowledge, they should be organized rationally. Besides the four structures presented here, there are other topics such as folksonomies - a set of tags organized as a classification system. The key is to find the right level of knowledge representation for the problem at hand. For example, if a thesaurus is sufficient, there

[1] http://databases.unesco.org/thesaurus/.

is no need to create an ontology since its construction is more complex and its manipulation is more involved.

3 Building Blocks for the Web of Data

We can consider that there are three building blocks to the Web of data: HTTP (*HyperText Transfer Protocol*), URI (*Uniform Resource Identifier*) and RDF. In this section, we present each of them and motivate the RDF data model against the popular relational one.

3.1 URI and HTTP

In RDF, URIs (actually IRIs: *Internationalized Resource Identifiers*, an extension of URIs that fully supports Unicode) identify resources. Hence, one of the key aspects when planning publishing information as linked data is to carefully define the URI naming policy.

Technically, URIs can be divided into two distinct sets depending on whether they can be dereferenced. Dereferenceable URIs use a scheme associated to a network protocol (`http`, `ftp`...). URIs using schemes not associated to a protocol (`urn`, `ark`...) can identify resources but require an additional component (a *resolver*) to support network access to the resources.

When publishing linked data, the protocol of choice is HTTP. This is one of the core building blocks of the World Wide Web (W3). It provides a simple and universal mechanism for accessing resources and retrieving representations (descriptions of a resource). That choice alone ensures that the data be widely accessible, from web browsers to applications.

One of HTTP core features is content negotiation. Upon access to the same URL, clients can request and obtain a view of the resource that best matches their needs or capacity to support a given output. For instance, one could retrieve a simple HTML page to be displayed in a browser or binary formats suitable for direct consumption by applications (picture, map, spreadsheet file, video...).

With RDF, it is also important to distinguish a resource from its representations. A resource is an abstract set of information that may (or may not) correspond to a real world object. Upon access (typically an HTTP request), the server will return a view of the resource, a representation. With HTTP, this representation is subject to content negotiation, *i.e.*, the client lists the types of representations (denoted as MIME (*Multi-Purpose Internet Mail Extensions*) types) it is ready to accept with, for each one a level of preference, and the server selects the best matching candidate among the representation the resource can offer.

The URI policy then dictates whether the server will directly return the selected representation in the response or instead redirect the client to a specific URL (Uniform Resource Locator). In this case, each representation of a resource has its own identity (URL). This allows for example, users to share or bookmark URLs that include both the reference of the resource and the type of representation to provide.

Several governmental organizations have started publishing guidelines to define and manage URIs in the context of Linked Data, for example INSEE in France [8] or Data.gov.uk in the UK [19]. These documents are good starting points for defining local URI policies.

3.2 Limits of the Relational Model

Before presenting the RDF data model, one can ask why we are not using an existing data model. As of today, most data are stored following the relational model [2]. Yet this model suffers from severe limitations that hinder consumption, sharing or reuse of these data. Although the relational model is not the only model for managing data, it is still the dominant one and most other existing models (hierarchical, key-value...) suffer from the same limitations.

The first limit comes from the separation between the structure of the data and the data themselves. A data corresponds to a column associated to a record in a table and can not be understood outside the context of the record. Moreover, the structure of the data is captured by the definition of the table, hence outside the data itself.

The structure of the data is also quite rigid and opaque:

- If a data is missing, a fictional `NULL` value shall be inserted;
- If multiple values occur for the same data, it is necessary to define a second column or use a separate table;
- There is no support for internationalized labels, these labels must be handled specifically, typically as a multi-valued field;
- Relations between records is not explicit but relies on usage of shared identifiers (foreign keys). The nature or intent of the relationship is never expressed, either in the date itself or in the structure (tables);
- The data structure is local. There is no standard way to convey this structure (and the relationships between data) along with the data themselves (exports).

The relational model does not provide any form of universal naming. Record identifiers are not normalized and are most of the time local to each table or, in the best cases, the local database.

As a consequence, database access and data extracts are specific to each product. There is no standard protocol for querying remote databases nor standard data formats to transfer data and their structure between databases from different providers.

So, there is a need for alternative data models that address these shortcomings. The RDF data model is the W3C response to these challenges in the context of the World Wide Web.

3.3 RDF

The objective of the W3 Consortium when it started designing RDF was to provide interoperability between applications that exchange machine-understandable

information on the Web. Originally RDF was aiming at expressing metadata, i.e. "data about data", specifically to describe web resources.

RDF is not a data format. It is a syntax-independent model for representing named properties of resources and their values. A resource is identified by a URI. Values can hold either a literal (string, integer, date...) or a URI, in which case the property models a relationship between two resources. In the RDF data model, resources and values are nodes in a directed labeled graph, the properties being the arcs connecting nodes.

In RDF, each information is expressed as a statement of three elements, a triple: the resource (subject), the property (predicate) and the value (object). Each statement is independent and self descriptive. Hence, it is possible to extract as little information as needed from an RDF document without loosing context information.

In order to be able to transfer RDF statements between applications, serialization formats were required. The W3C has proposed a standard syntax, RDF/XML[2], but other formats exist and are commonly used: Turtle[3], Notation 3 (N3)[4], N-Triples[5] and more recently JSON-LD[6]. As the usage of RDF stores (triple stores) increased, new syntaxes appeared to support exporting whole named graphs, for example: Trig[7] and N-Quads[8].

4 Managing and Querying RDF Data

The ability to manage large RDF data sets is a fundamental prerequisite to the fulfillment of the Semantic Web vision. The W3C Data Activity[9] regroups a set of standards enabling the design of Knowledge Base Management Systems (KBMS). In this section, we present the storage and indexing approaches adapted to the RDF data model. Then, we present the SPARQL (*SPARQL Protocol and RDF Query Language*) query language which is mainly used to express queries over these systems. Finally, we present two original important notions that are specific to RDF and SPARQL, namely SPARQL endpoints and federated queries.

4.1 Triple Stores

RDF is a logical data model and hence does not impose any constraint on the physical storage approach. This opens a wide range of possibilities to design RDF database management systems, henceforth RDF Stores. There exists more than fifty different RDF stores which can be categorized as native or non-native [9].

[2] http://www.w3.org/TR/rdf-syntax-grammar/.
[3] http://www.w3.org/TR/turtle/.
[4] http://www.w3.org/TeamSubmission/n3/.
[5] http://www.w3.org/2001/sw/RDFCore/ntriples/.
[6] http://www.w3.org/TR/json-ld/.
[7] http://www.w3.org/TR/trig/.
[8] http://www.w3.org/TR/n-quads/.
[9] http://www.w3.org/2013/data/.

The non-native stores are based on an existing data management system. Typically, this can be a Relational DataBase Management System (RDBMS) and in that case, the RDF Stores benefits from many functionalities that have been designed over more than thirty years of research and development. Of course the original RDF graph-based data model has to be adapted to the relational model. The most popular approach consists in storing all RDF triples in a single relation containing only three columns, respectively one for the subject, property and object. Figure 5 presents an extract of such a triple table that stores items of a library. This simple approach presents several drawbacks. In particular, on real-world queries, the amount of self-joins, i.e., when one attribute of a relation needs to be joined to an attribute of that same relation, can be quite important and is known to have poor performances especially on very large relations.

There exists several other approaches to represent RDF Triples in a set of relations, e.g., clustered property, property-class and vertical partitioning [1]. These systems generally accept SPARQL queries which are being translated to SQL queries that the RDBMS understands. The overall performance of query processing is considered not to suffer from these translation steps. Some of the NoSQL stores have also been used to store some RDF Triples. It is for instance the case of Trinity.RDF [20] that uses a key value store on Microsoft's Azure cloud, MarkLogic[10] with a document-based approach and Rya [16] which uses a column-oriented store. A main advantage of these last systems consists of their capacity to be distributed over a large number of machines which makes them particularly adapted to a cloud computing setting. In terms of query

Subject	Property	Object
id1	type	Music
id1	title	"Nefertiti"
id1	author	id5
id1	year	1968
id2	type	Book
id2	title	"The road"
id2	author	"Cormac McCarthy"
id2	year	2006
id2	language	"english"
id3	type	DVD
id3	title	"Into the Wild"
id3	year	2007
id4	type	Music
id4	title	"1984"
id5	type	Man
id5	fullName	"Miles Davis"
id5	yob	1926

Fig. 5. Triple table example

[10] http://www.marklogic.com/.

answering, queries are again expressed in SPARQL and are either translated to specific query language accepted by the underlying NoSQL store or to some API (Application Programming Interface) calls.

The systems belonging to the native category do not rely on an existing DBMS. Hence, they generally design their architecture from scratch. Although being time consuming to design and implement, it enables to define a solution that is particularly adapted to graph data model of RDF. In these systems, a peculiar attention is given to the indexing solution, *i.e.*, to ensure high query answering performances, as well as query processing optimization which are dedicated to SPARQL. Examples of native triple store implementations include OWLIM/GraphDB, Virtuoso, Mulgara.

The market currently proposes production-ready RDF stores which are almost evenly distributed over the native and non-native categories.

4.2 SPARQL

There are two versions of the SPARQL specifications: SPARQL 1.0 (2008) only supports information retrieval query facilities and the SPARQL 1.1 (2013) that, in addition to adding new query capabilities, supports data updates. In this subsection, we concentrate solely on the query aspect of SPARQL. Concerning the other functionalities supported by this technology, e.g. service description, one is invited to read the SPARQL 1.1 recommendations[11].

Introduction. The SPARQL query language operates over RDF graphs made of triples and processes queries by adopting a graph pattern matching approach. Intuitively, a SPARQL query specifies a set of so-called graph patterns (GPs) corresponding to triple patterns (TP) composed of a subject, a predicate and an object. A difference compared to the triples encountered in RDF documents is that, in a TP, any of the three positions can be a variable. In that situation, using the same variable over different TPs consists in creating some joins. Hence the principle of SPARQL query answering is to find bindings, i.e. values, for these variables, when executed over a given graph. This approach is denoted as graph pattern matching due to the fact that we are trying to find matches between the set of GPs of a query over an given graph. In the next subsection, we provide an example of graph pattern matching.

The 4 Basic Commands. The syntax of the SPARQL query language resembles the one of SQL. This has been motivated by the popularity of SQL which is present in all RDBMS. Hence the learning curve of SPARQL is fast for users mastering SQL. The main operators of SPARQL are `SELECT, ASK, CONSTRUCT, DESCRIBE`. In order to retrieve some information from a database, one writes a `SELECT` query which contains a `WHERE` clause and possibly a `FROM` clause, which identifies target graphs. In the `SELECT` clause, one specifies the variables (s)he

[11] http://www.w3.org/TR/sparql11-overview/.

wants to see in the result set. Such variables are denoted as distinguished. In the `WHERE` clause, we declare the GPs that are supporting the graph pattern matching.

Let us consider the query that retrieves the full name of males that have authored some music document. This is expressed in SPARQL as:

```
SELECT ?fn
WHERE
  {
      ?x type Music.
      ?x author ?y.
      ?y fullname ?fn.
      ?y type Man.
  }
```

This query can be represented as the graph of Fig. 6(a). Executing this query over the data set of Fig. 5 corresponds to finding a subgraph in Fig. 6(b), which represents that data set, and to associate values to the variables of the query. For that query, we have a match for `?x=id1`, `?y=id5` and `?fn=‘‘Miles Davis’’`. Note that only the ?fn will be displayed in the answer set since it is the only variable appearing in the `SELECT` clause of that query.

Typical `SELECT` queries can also contain operators such as `LIMIT, OFFSET, ORDER BY, DISTINCT` that SQL users already know. It also support aggregation with `GROUP BY, HAVING` and the standard `max, min, sum, avg, count` functions. Moreover, one can obtain the union of two queries, perform some regular expression operations using the `FILTER` keyword, execute some outer joins with `OPTIONAL`.

SPARQL possesses some other operators which are using the notion of GP. `ASK` queries are boolean queries, i.e., returning true or false, on the existence of

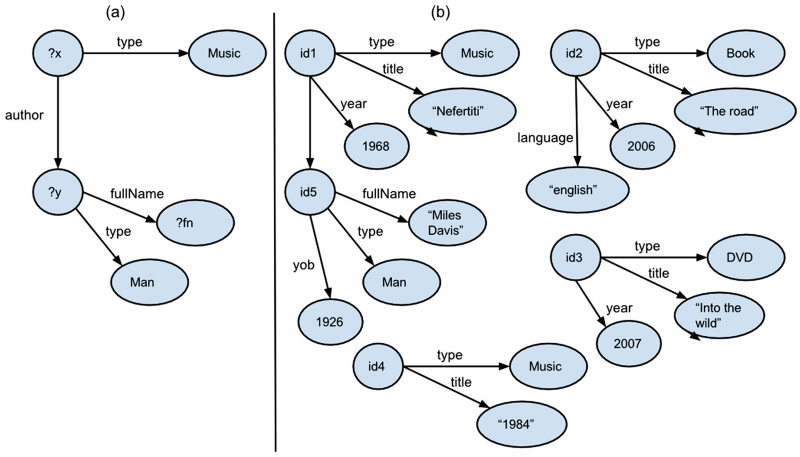

Fig. 6. Graph pattern matching example

a given graph pattern. The form of a `CONSTRUCT` query is quite similar to the one of `SELECT` query except that it does not specify any distinguished variable. In fact, the answer to such a query is a new graph. This is quite efficient if one wants to integrate RDF graphs from queries obtained from another graph. Finally, `DESCRIBE` provides for any RDF information related to some specific variables of the result template. This is a nice approach to learn about what a graph knows about some node.

The last SPARQL recommendation, denoted 1.1, also proposes support for update operations. Compared to SQL, it limits its patterns to `DELETE` and `INSERT`, so no `UPDATE` like in SQL. They respectively correspond to removing and adding some statements from (resp. into) a given RDF graph. We will see in Sect. 6 that in the context of reasoning, the operational aspect of these directives are quite involved.

Limits. RDF Stores correspond more to data management systems that support discovery than to systems supporting high transaction rates. Hence, we can consider that SPARQL is lacking some analytic features. Moreover, support for efficient management of update operations is poor in current RDF Stores.

4.3 SPARQL Endpoint and Federation

As its name implies, the SPARQL specifications define both a query language dedicated to RDF and a protocol for accessing RDF stores. This protocol[12] supports two standard bindings (HTTP and SOAP - a lightweight W3C protocol intended for exchanging structured information in a decentralized, distributed environment) and defines the request parameters as well as the supported response formats.

The SPARQL endpoint is a key feature of RDF stores as it ensures interoperability between all implementations. Hence a change in the RDF store back end will have no impact on the applications accessing the database (provided they do not make use of any provider-specific extensions, such a plain-text search that some products offer).

In addition to the standard bindings, most RDF stores also provide a basic HTML GUI (Graphical User Interface) to their SPARQL endpoint for the users to browse the store contents. Query results are usually displayed as simple HTML tables.

Although SPARQL 1.1 added support for modifying data (create, update and delete), most public endpoints, for obvious security reasons, block write access to the data. Although this can be achieve through access rights managements, some tools (Datalift for example) favor providing a read-only SPARQL endpoint that acts as a reverse proxy, shielding the underlying RDF store from direct access from the Web. In addition to blocking update queries, this front-end can include additional features such as support for the SPARQL 1.1 Graph Store HTTP Protocol[13] (when not provided by the RDF store endpoint) or

[12] http://www.w3.org/TR/sparql11-service-description/.
[13] http://www.w3.org/TR/sparql11-http-rdf-update/.

graph-based access control. They may also provide value-added user interfaces, such as the Flint SPARQL Editor,[14] to assist SPARQL query editing (syntax highlighting, auto-completion...) or results browsing.

Scalability of RDF stores is today limited to several billions of triples. As RDF stores are schema-less databases and the SPARQL language puts no restrictions to the complexity of the queries or the scope of data they may apply to, it is quite difficult to efficiently split RDF data into partitions (sharding) and then route a query to only a subset of the partitions.

The SPARQL Federated Query specification[15] helps addressing these shortcomings by defining an extension to the SPARQL query language that allows a query author to direct a portion of a query to a particular SPARQL endpoint. Results are returned to the federated query processor and are combined with results from the rest of the query.

Uses for SPARQL federations are:

- Spread one organization's datasets on several RDF stores if their combined sizes exceed the capacity of the single RDF store installation.
- Perform join queries on a set of RDF datasets hosted on independent RDF stores without having to copy these data locally.

The main drawback of SPARQL Federated Query is that it mandates the query author to know which data are available at each SPARQL endpoint and explicitly craft the query from this knowledge. In the following example, note the `SERVICE` clause that enables to retrieve some data from a given URI:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
  {
      <http://example.org/myfoaf/I> foaf:knows ?person .
      SERVICE <http://people.example.org/sparql>
      { ?person foaf:name ?name . }
  }
```

4.4 The Data-Lifting Process

During his talk at the "Gov2.0 Expo" in 2010, Tim Berners-Lee presented a scale to rate open data [7]:

- 1 star: Make your data available on the Web (whatever the format) under an open license
- 2 stars: Make it available as machine-readable structured data (e.g. Excel instead of image scan of a table)
- 3 stars: Use a non-proprietary format (e.g., CSV instead of Excel)

[14] http://openuplabs.tso.co.uk/demos/sparqleditor.
[15] http://www.w3.org/TR/sparql11-federated-query/.

– 4 stars: Use URIs to identify things, so that people can point at your stuff
– 5 stars: Link your data to other people's data to provide context.

The Datalift project[16] is one of the (many) initiatives that were launched to provide tool sets to help users publishing 5-star linked open data.

The challenge here is that of a virtuous circle where quantity calls for quality. The more data is linked, and the more it becomes a reference that can attract new data to be connected to. Efforts made by some to link their data will benefit others and have a positive feedback. In this game, every new link takes advantage of those already established, and all existing connections benefit from new contributions.

The raw data from which we have to start are heterogeneous, generally unstructured and stuck at the bottom of information systems. They are generally quite difficult to handle by users. It is possible to refer to them by file name or to extract them from a database but they often do not comply with standards. Their structure is at most syntactical.

As seen above, for the users to make the most out of the published data, it is necessary to explicit the data semantics, by using appropriate vocabularies, also called ontologies. A good practice is to avoid creating a new ontology for any new dataset, but to start with those that already exist. Ontologies durability and reuse rate can instruct us on their efficiency. By efficiency, we must understand the power to make the web less sparse. Indeed, each new usage increases the expressiveness of the data already described, making them more universal.

So, the platform should include features such as ontology management. Among past initiatives, we know that a simple catalog or a search engine is not enough. Ontologies describe knowledge, their management therefore requires discovery heuristics based on existing or deducted links.

The Datalift open-source project aims at providing a one-stop shopping solution to transform raw datasets into RDF structures according to a set of selected ontologies (see Fig. 7). As described above, RDF is a simple yet powerful model for representing knowledge on the web using a triple pattern resulting in a graph-based representation of the data. The transition to RDF makes the resulting data self-expressive. Usage of public and well-known, and widely used ontologies reinforce this expressiveness.

Thus the first step of the data-lifting process is the selection of a (set of) ontologies. Rather than letting users searching for such ontologies, the project maintains an ontology catalog: the Linked Open Vocabularies (LOV)[17]. This catalog only references open RDFS (RDF Schema) and OWL (Web Ontology Language) ontologies, classifies them into business-oriented categories (spaces) and provides a powerful search engine.

Once the ontologies applying the user raw dataset is selected, the data transformation process begins by first converting the data into RDF. This first RDF format derives directly from the raw data format, making use of every possible

[16] http://datalift.org/.
[17] http://lov.okfn.org/.

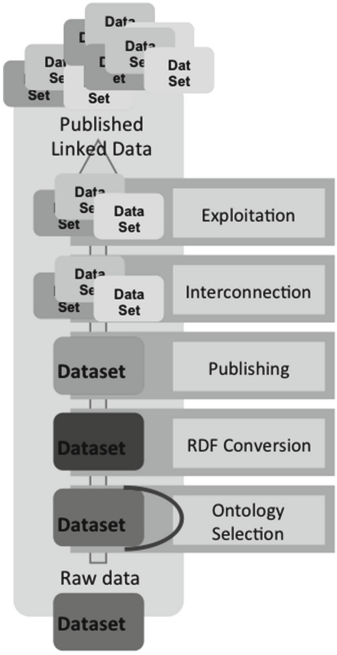

Fig. 7. The data-lifting process

metadata available (column names in CSV files, data types in SQL databases. . .). Datalift then proposes a set of modules to incrementally transform these basic RDF data into refined ones that match the structure defined by the selected ontologies. These transformation steps may be performed several times as, for complex datasets, each ontology may only model part of the business domain. Each transformation step produces a new RDF graph, a *Named Graph*, which is an administrative object managed by RDF stores to group related RDF triples, allowing to manipulate them as a whole. Until data are ready to be published, these graphs are stored in an internal RDF store.

Once the transformation steps are completed, data can be published, by copying the corresponding named graphs into a public RDF store. Yet, just as data alone has no value at the unit level, and that only the links matter, the same applies at graph level. By publishing the RDF graphs built in the previous steps into a publicly-accessible triple store, we open a new potential of linking: to intra-graph links, it becomes possible to add inter-graph links.

This is the interconnection step: it is possible to find relationships between data, *e.g.*, discovering identical resources (two URI identifying the same real world object) in different graphs, local or remote (*i.e.* hosted on distant SPARQL endpoints). Interconnection can occur either at the dataset level (by comparing RDF resources) or at the ontology level. Ontology alignment (mapping, matching) is an active research topic where new metrics are developed to perform data integration at the schema level rather than at the object level.

The final step is the exploitation of the published linked data. While Datalift directly exposes published RDF resources over HTTP, it also provides additional modules to ease the access and consumption of RDF data:

- A SPARQL endpoint supporting the SPARQL 1.1 syntax but limited to read-only access
- An RDF-based data access control module, S4AC (Social Semantic SPARQL Security for Access Control), that controls which data are accessible to each user: two users running the same SPARQL query will get different results, depending on the graphs they are each allowed to access
- Tools to enforce URI policies and tailor content negotiation. These tools allow, for example, to set up URI policies distinguishing representation URLs (one for each supported MIME type) from canonical RDF resource URIs or provide alternative representations, such as legacy GML or WKT representations for GeoSPARQL data
- Tools to help developing RDF-based web applications, *e.g.* generation of HTML pages or deployment of REST web services relying on SPARQL queries.

Figure 8 depicts the Datalift platform software architecture.

5 Ontologies

Ontologies are at the heart of many applications of knowledge engineering, especially the Semantic Web. They correspond to the expression of some knowledge and thus provide a vocabulary consisting of concepts together with their relationships. They support knowledge management and reasoning facilities and hence offer semantic interoperability between agents, either human or computerized ones. In this section, we begin by addressing the issue of creating ontologies in a mainly methodological angle. Then we will discuss the issues related to their management and we will be referring to many tools. We conclude with a very exciting topic: ontology alignment.

5.1 Ontology Creation

One has to understand that creating an ontology is a time consuming and difficult task. A first step generally consists in searching if an existing ontology can be reused and/or modified. This would make the creation process much easier.

Because an ontology expresses the view of its designer, it could be appropriate if it matches the right knowledge domain, but inappropriate due to a wrong point of view. We could also use portions of existing ontologies to cover part of the problem and create only some complements. The first step is therefore methodological and we can state some good practices.

- Meet those who know: Without domain experts and users of applications, it is not possible to formalize their knowledge and needs, to establish a common vocabulary and to build a consensus (even if it is very hard to read) on the concepts of their domain.

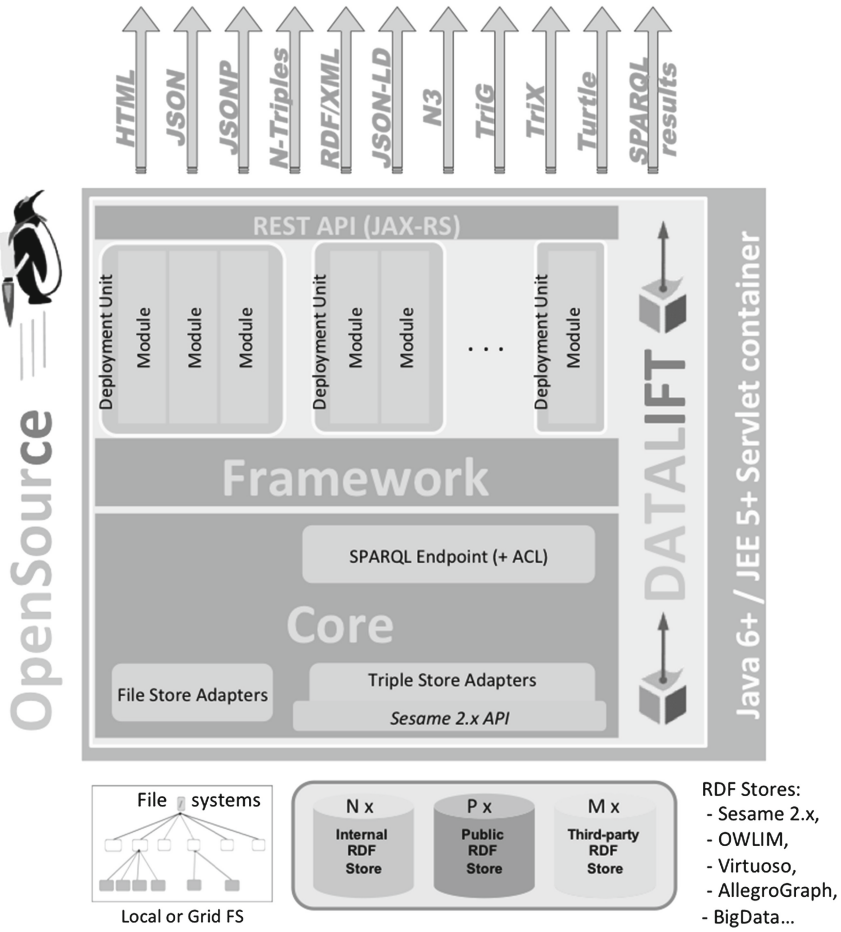

Fig. 8. The Datalift architecture

– Create an ontology according to usage: A good ontology is not the one that meets to the whole domain, but which is guided by usage. What are the questions that the ontology must answer? Although an ontology seems to be a particularly abstract object, it must be limited to the concrete needs we have.
– Do not repeat what has already been done by others: We will take this matter further (see Catalog).
– Do a work of quality and be responsible: When coding the ontology, we must provide care for its expression. The syntax of the language is not too demanding, so we have to impose ourself some rules (mention the language, place a cartridge, comment, etc.). Being responsible is perhaps the least recognized best practice. Yet, it is important to feel responsible for the ontology created according to those who use it. The ontology is a reference, it should evolve and

be maintained as long as it will be used. Other important questions concern who will use and maintain the ontology?

There is no miracle tool to create an ontology. We can adopt a bottom-up approach based on a corpus. It is very common to start by building lexicons from domain materials and semi-automated software that extract terms. We can also analyze an area with an expert and develop a top-down approach. A mind-map tool, *e.g.* FreeMind,[18] can be very useful here. It is also possible to start with weakest structures (taxonomy or thesaurus) to enrich them after. The literature abounds. In any case, this is an intellectual work that requires thought, a step back, assessments ... and time.

To express an ontology, different languages and possible syntaxes can be used, Table 1. The most common are OWL (OWL1 in Feb. 2004, OWL2 in Oct. 2009) and RDFS (in March 1999).

RDFS is a very basic ontology language. It is easy to use and get started with but it is less expressive than the OWL languages (more on this in Sect. 6).

5.2 Ontology Management

Ontologies are complex objects. Some may be very large, for example in the medical field, *e.g.*, the SNOMEDCT (Systematized Nomenclature of Medicine - Clinical Terms) has more than three hundred thousands concepts. So, tools are needed to edit, transform (modify, merge, split), view, query, validate, catalog, etc. We can not, in this pages, give an exhaustive list of tools. We limit ourselves to a short description of LOV and Protégé.

LOV provides a technical platform for search and quality assessment among the vocabularies ecosystem, but it also aims at promoting a sustainable social

Table 1. OWL2 syntaxes, (source : http://www.w3.org/TR/owl2-overview/)

Syntax name	Specification	Status	Purpose
RDF/XML	Mapping to RDF graphs, RDF/XML	Mandatory	Interchange (can be written and read by all compatible OWL 2 software)
OWL/XML	XML serialization	Optional	Easier to process using XML tools
Functional syntax	Structural specification	Optional	Easier to see the formal structure of ontologies
Manchester syntax	Manchester syntax	Optional	Easier to read/write DL ontologies
Turtle	Mapping to RDF graphs, turtle	Optional, not from OWL-WG	Easier to read/write RDF triples

[18] http://freemind.sourceforge.net/wiki/index.php/Main_Page.

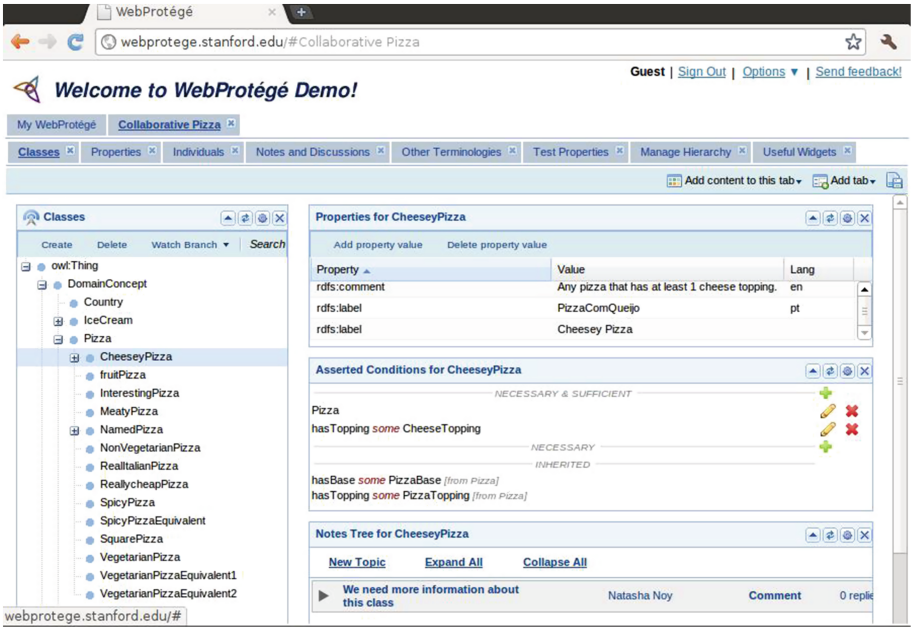

Fig. 9. WebProtégé

management of this ecosystem. Beyond the LOV, there is the vision of a future linked data Web supported by a living Vocabulary Alliance gathering as many as possible stakeholders in the long-term conservation of vocabularies.

Protégé is an ontology editor designed at the Stanford University, distributed as an open-source project and whose catalog of plug-ins[19] provides a good idea of the tools it is possible to find. These plug-ins tackle domains such as visualization, natural language processing, export, import, navigation, query and reasoning. WebProtégé is a Web-based lightweight ontology editor but its goal is not only to be another ontology editor. The purpose is to offer an easier tool and to fill in a significant gap in the landscape of ontology tools. First it is devoted to a large spectrum of users, from ontology experts to domain experts. The user interface (Fig. 9) is customizable by the users with different levels of expertise. It is built like a portal and is composed of predefined or user-defined tabs. The previous version, Protégé, was considered difficult to use.

WebProtégé supports collaboration, ontology reuse and interlinking. It creates an open, extensible and flexible infrastructure that can be easily adapted to the needs of different projects [13].

5.3 Ontology Alignment

Ontology matching [12] is the process used to establish semantic relations between concepts and relations of two ontologies. If the ontologies are designed to

[19] http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library.

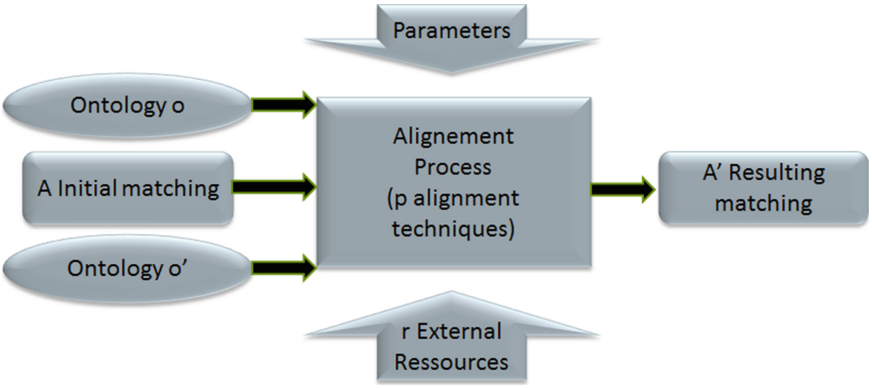

Fig. 10. The matching process, A' = f(o; o'; A; p; r)

neighboring areas, with analogies, similarities or any form of correlations, alignment will facilitate their joint manipulation within a single system or multiple systems using these ontologies to communicate.

Ontology matching is a function f represented in Fig. 10 and applied:

- to two ontologies o and o',
- to a set of original matching A,
- from a set of parameters applicable to p alignment techniques implemented in the process
- and a set of external resources r,
- and that produces a set of matching A' between the two ontologies.

Alignments express correspondences between entities belonging to different ontologies.

6 Reasoning in a Semantic Web Context

6.1 Introduction

In this section, we present the role played by ontologies in the process of deriving novel information from a set of RDF statements. We then introduce the different methods on which reasoners have based their implementation.

The reasoning aspect is reminiscent of the Knowledge Representation (KR) field of Artificial Intelligence (AI). Reasoning can be described as the ability to deduce implicit knowledge from explicitly one. Here, we consider that the concept of knowledge subsumes the previously introduced notions of information, data and obviously knowledge. In the context of the Semantic Web, this implies a Knowledge Base composed of a set of RDF facts together with a non empty set of ontologies expressed in RDFS and/or OWL. Note that such derivations can not be performed in the absence of at least one ontology since it is the component responsible for storing the knowledge. On the other hand, it is possible to reason over an ontology alone, i.e., without facts.

This is clarified with a simple example where the fact base is limited to the assertion `father(Camille, Pierre)` and the ontology comprises some concept descriptions, `Man` and `Woman` are disjoint subconcepts of `Person`, and two property descriptions, `father` and `parent` both have the `Person` concept as their range and the `Man`, resp. `Person`, concepts as domain. First, note that to represent this ontology, the expressive power of RDFS is not sufficient. This is due to the disjointness between the `Man` and `Woman` concepts which requires a negation. Since the Fact base does not provide any description on either concept and property, it is not possible to infer anything interesting from this Knowledge base component. With the ontology alone, we can deduce that a `Man` or `Woman` instance can be involved in parent relationship, either as the parent or the child. Finally, with both the ontology and fact base, we can infer that `Camille` and `Pierre` are respectively instances of the `Man` and `Person` concepts.

Thus, ontologies are supporting the ability to reason over the facts and depending on their expressive power different inferences can be obtained. The expressive power of an ontology is essentially associated to the constructors provided by the ontology language, i.e., concept negation, disjunction, quantifiers, cardinalities to name a few. Obviously, the more expressive the ontology language, the more inferences can be deduced but this also comes at a certain cost. Previous work in Description Logics (DL) [5] have emphasized a trade-off between the expressive power of an ontology and the computational complexity. To simplify, the more expressive the ontology language, the longer the duration, in favorable cases where the processing terminates, one has to wait to obtain the correct complete set of deductions. For very large Knowledge Bases, this duration may correspond to hours, days or more. By itself, this trade-off motivates the existence of the different ontology languages of the Semantic Web with RDFS being the less expressive and OWL2DL being the most expressive among the decidable ones, i.e., those ensuring that the deduction process will terminate. For instance, OWL2 Full is known to be undecidable. In between RDFS and OWL2DL lies some languages like the OWL Lite and the more recent OWL2QL, OWL2EL and OWL2RL which have precisely been designed to provide interesting expressiveness/computational complexity compromises. Figure 11 presents the expressiveness containment relationships of RDFS and OWL2 ontology languages.

So far, we have not said anything on the "how" these reasoning services are being performed over a KB. Conceptually, two different approaches can be used to address the reasoning task.

6.2 Procedural Approach

With a procedural approach, the inferences are hand coded in some programming language, e.g., Java programming language. Although it can reply to some special domain dependent needs, this approach is generally associated with the following drawbacks:

– adding some new reasoning functionalities for an existing system requires programming skills and a very good understanding of the programs. One can not expect a non-programmer to perform this task, *e.g.*, a biology expert will not learn the Java to add some new inference services.

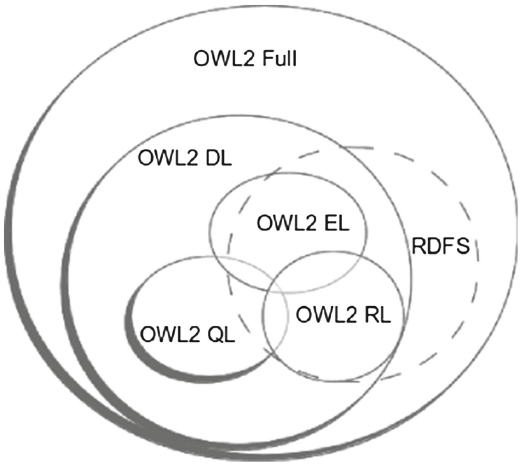

Fig. 11. Expressiveness of RDFS and OWL2 ontology languages

- reusing the reasoning services from one application domain to another involves too much effort from both the programmers and the domain experts. This would require the ability to identify and extract some reasoning patterns encoded in the syntax of a programming language and modify them for another application domain. This is definitely not a task that can be performed automatically.
- enabling the system to explain its inferences, a feature expected from a KB system, is very complex. This would again imply a form of understanding of the intention of the code lines of the program. A thing that we already seen in the two previous bullet points is not possible.

6.3 Declarative Approach

The second approach is characterized as declarative. Intuitively, an inference engine performs some tasks on a external component that is responsible for storing the facts and can communicate with an application. This separation between these three components is a differentiator to the previous procedural approach where the inference engine, a.k.a. a reasoner, and the application were kind of merged. This separation eases the maintenance of both the reasoner and the application due to a clear separation between the responsibilities of each component. The reasoner is a generic, *domain independent*, logic-based, i.e., adapted to a specific ontology language, and optimized implementation that is able to make some deductions. These two characteristics are supported by operating directly on the logic axioms of the knowledge base. Hence, the same reasoner can be used on any domain, e.g., biology, finance, culture, as long as a compliant KB is provided. It also enables any domain expert to maintain the set of axioms using a user-friendly, graphical interface such as the Protégé KB editor. Moreover, these reasoners are generally equipped with components that

enable to explain inferences, usually by going through the different reasoning steps. Due to its nice properties, the declarative approach is more frequently used than the procedural one.

6.4 Declarative Reasoning Methods

There exists several methods to perform inferences within the declarative approach. They can be classified into two categories depending on the representation of the knowledge, that is either rule or object based.

In the former, the inference is performed using the resolution principle which has been introduced by Robinson in [17]. It was used in the area of automated theorem-proving by trying to derive a theorem from a set of axioms and proceeds by construction a refutation proof. It is easily adapted to standard ontology languages such as RDFS, the RL and EL OWL2 fragments, to some extent to OWL2 QL as well as the non-standard RDFS+ and OWL Horst languages.

In the object-based category the main inference pattern is based on the notion of inheritance, i.e., subsumption relationships between concepts or properties. For instance, it is particularly adapted to compute the concept hierarchy in an OWL ontology. Depending on the expressiveness of the underlying ontology, it is most frequently computed with a structural subsumption algorithm or a tableau-based one (but automata-based approaches or translation to a First Order logic reasoner are possible). In the former, the inheritance is strictly computed at the syntax level of the concept descriptions. That is one searches if the concept, denoted as C, corresponding to a happy man whose parents are all doctors is a specialization, i.e., a subconcept, of the concept, denoted as D, described as a man whose parents are all doctors. The problem can be stated as "is $C \sqsubseteq D$?" and intuitively tries to find if all elements on the D definition has an equivalence in the C description. If this is the case then the subsumption between these two concepts holds otherwise it does not. For instance, the concept Man and *all parents are doctors* are on both sides of the $\sqsubseteq$ symbol. So we remove these concepts on both sides, thus nothing remains on the right side (D) and the concept $Happy$ remains on the left side C. So it is the case that $C \sqsubseteq D$. This method is adapted to ontologies with a low level of expressiveness, e.g., OWL2 EL, since for more expressive ones, it may not be able to perform the task appropriately. The tableau-based approach is a widely used approach and is the one we encounter in well-known systems such as Pellet, RacerPro, Fact++ and to some extent HermiT (which is now integrated within the Protégé editor). Just like in the resolution approach, the method relies on proof by contradiction using a set of so-called expansion rules. Additionally to the classification service, the tableau method can perform other standard services such as consistency checking, i.e., detect whether a KB is consistent or not, and so-called realization and retrieval which respectively find the most concept an object is an instance of and find the individuals that are instances of a given concept. Some reasoners even support some denoted non-standard services and provide some explanations on their inferences. This approach is particularly adapted to expressive ontology languages such as the decidable OWL DL and OWL2 DL.

It is important to stress that the SPARQL query language does not support any reasoning service. Hence, the kind of answer it returns for a query do not natively contain any inferred statements. In our running example, asking for instances of the `Man` concept would return an empty set.

In order to reply complete results to query requiring inferences, two approaches are used by RDF stores. In the first one, the complete set of facts that can inferred are stored in the database. This materialization is performed when the data set is first loaded in the RDF store or where the database is updated. Consider the KB of our previous example, one of the materialized statement would state that `Camille` is of type `Man`, represented as `Man(Camille)`. Thus, this approach is quite efficient in terms of query answering performance since no additional inferences are performed at query run-time. That is `Camille` would be part of the query asking for man instances. Nevertheless, it comes with the following limitations: the size of stored database can grow exponentially compared to the size of the original data set, the loading time of the data set can be quite important and handling updates is a complex task since one basically needs to know if a statement has been inferred or is part of the original data set. For instance, consider that we modify the KB of our running example by substituting `Camille` by `Joe` in the `father(Camille, Pierre)` assertion. Then the system would have to remove the `Man(Camille)` from the database since that fact does not hold anymore.

The second solution to support deductions in RDF stores is based on the introduction of a reasoning step at query run-time. This implies to rewrite the original query into a possibly large number of new queries that are all going to be executed. The final answer set then consists of the union of the result set of each query. In our running example, the query asking for all man occurrences would result in the execution of the original query plus the query asking for the domain of `father` property assertions. Obviously, the correct `Camille` result would be obtained from that last query. Of course, the integration of the reasoning step has the negative impact of slowing down query answering. Nevertheless, the size of the stored data is not expanded and updating the database is less demanding than in the materialization solution.

7 Research and Development Perspectives

7.1 Introduction

As emphasized in previous sections, available Web of data building blocks are covering expected functionalities and are of sufficient quality to be used in production. It is for instance the case for the RDF data model, the spectrum of ontology languages with a W3C recommendation status and their associated reasoners as well as the SPARQL query language. This permits to address data integration in an efficient way, an aspect that is particularly interesting in the context of Linked Open Data. During the last few years, we have witnessed an acceleration of the adoption of these technologies in the information technology industry. For instance, more and more web sites are annotating their content

with Schema.org. A blog entry at SemanticWeb.com[20] announced that in September 2014, 21 % of a sample of 12 billions web pages were using that solution for annotation purposes. Moreover it is said that "every major site in every major category, from news to e-commerce (with the exception of Amazon.com), uses it." As a last example, the Knowledge Vault project [11] developed at Google and which is believed to replace the Knowledge Graph for improving semantic search, claims to use RDF Triples to represent extract information.

We nevertheless consider that more is needed to speed up the adoption of these technologies at a large scale. In the remaining of this section, we present some directions that we believe are primordial to attain the vision of the Web of data and of the Semantic Web.

7.2 Big Data

We can first wonder what is the stance occupied but the technologies of the Web of data in the Big Data phenomenon. Although a large set of RDF Triple stores are being produced, we consider that a robust, equipped with efficient support for updates operations and reasoning services, is needed. The current state of RDF stores does not address these issues with sufficient guarantees. For instance, we know of several large projects that are using some commercial, state of the art RDF stores but which are still not using their update operations or reasoning functionalities due to poor performances. That is these companies prefer to use a bulk loading approach, rather than an incremental one, when new data are to be integrated in the RDF Store. Intuitively, this means that updates are kept somewhere, waiting to be sent to the store together with the previous data. The main drawback is the staleness of the database. Moreover they are mostly not using any forms of inferences although their use cases would probably benefit from it. The data and processing distribution aspects are relatively well managed by existing commercial and production ready RDF Stores. These database systems are mainly used for data integration and analytics purposes rather than to process high rates of transactions, due to their relatively poor update performances especially in the situation of a materialization approach. Thus these systems are more used as data warehouses, relevant to the OnLine Analytical Processing (OLAP), rather than the OnLine Transactional Processing (OLTP) movement. Within this context, we can argue that the SPARQL query language is missing some analytical operations toward the management of aggregations and dimensions, e.g., drilling down and rolling up, slicing and dicing. We believe that these tools should benefit from the concept and property hierarchies already represented in ontologies. Other operations such as graph navigation features are needed to compete with graph store of the NoSQL ecosystem. The integration of statistical languages, such as the R language, would also open some new perspectives to end-users in need to take some decisions. In that same data analytic aspect, libraries and tools for data visualization and user interfaces are needed by data scientists that are manipulating RDF data.

[20] http://semanticweb.com/schema-org-fires-lit_b44380.

7.3 Data Streams

Still related to the data deluge aspects, it seems that methods to handle streams in both the processing and reasoning aspects are highly expected from the community of developers and designers of Web of data applications. An important number of researchers in the field of database management systems is currently working on this subject which addresses the velocity characteristic of Big Data. In particular, [18] presents a set of requirements expected from such systems in the context of RDBMS. Considering streams in the context of the Web of data implies to tackle the generation or efficient transformations of data emanating from different kind of sensors, the capacity to analyze, filter, store, summarize them as well as reason over them or their summarizations. These are aspects that have started to be investigated for the RDF data format, for instance with the proposition of SPARQL extensions [6,14].

7.4 Linked Open Data

Other expectations concern Linked Open Data and in particular the quantity of linked data sets and the quality of the proposed links[21]. We consider that these two notions are related. It is clear that the more linked open data sets, the more mash-up application can be designed. Nevertheless, a notion of quality has to be considered and recent studies [10] highlighted that the `owl:sameAs` property is widely used with different meaning. Just like in recent automatic KB construction solutions [11] a probabilistic approach should be used to qualify the confidence on the link between entities present in different ontologies or KB.

7.5 Other Aspects

Finally and not least, it is obvious that security, privacy which is related to identity management in the virtual world of the Web will play an important role in fulfilling the vision of the Semantic Web.

References

1. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable semantic web data management using vertical partitioning. In: Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, 23–27 September 2007, pp. 411–422 (2007)
2. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level, 1st edn. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
3. Ackoff, R.: From data to wisdom–presidential address to ISGSR.J. Appl. Syst. Anal. **16**, 3–9 (1989)
4. Amidon, D.M.: Innovation Strategy for the Knowledge Economy: The Ken Awakening. Routledge, London (1997)

[21] http://lod-cloud.net/.

5. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York (2003)
6. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-sparql: sparql for continuous querying. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, pp. 1061–1062. ACM, New York (2009)
7. Berners-Lee, T.: Linked data-design issues (2006). http://www.w3.org/DesignIssues/LinkedData.html (2011)
8. Cotton, F.: Politique d'identification des ressources (2012). https://gforge.inria.fr/docman/view.php/2935/8286/D3.1+Politique+dpdf
9. Curé, O., Guillaume, B. (eds.): RDF Database Systems: Triples Storage and SPARQL Query Processing, 1st edn. Morgan Kaufmann, Boston (2015)
10. Ding, L., Shinavier, J., Shangguan, Z., McGuinness, D.L.: SameAs networks and beyond: analyzing deployment status and implications of owl:sameAs in linked data. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 145–160. Springer, Heidelberg (2010)
11. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2014. New York, NY, USA, pp. 601–610, 24–27 August 2014
12. Euzenat, J., Shvaiko, P.: Ontology Matching, 2nd edn. Springer, Heidelberg (2013)
13. Horridge, M., Tudorache, T., Vendetti, J., Nyulas, C.I., Musen, M.A., Noy, N.F.: Simplified OWL ontology editing for the web: Is webProtégé enough? In: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 200–215. Springer, Heidelberg (2013)
14. Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 370–388. Springer, Heidelberg (2011)
15. Liew, A.: Understanding data, information, knowledge and their inter-relationships. J. Knowl. Manage. Pract. **8**(2), (2007)
16. Punnoose, R., Crainiceanu, A., Rapp, D.: Rya: a scalable rdf triple store for the clouds. In: Proceedings of the 1st International Workshop on Cloud Intelligence, Cloud-I 2012, pp. 4:1–4:8. ACM, New York (2012)
17. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (1965)
18. Stonebraker, M., Çetintemel, U., Zdonik, S.B.: The 8 requirements of real-time stream processing. SIGMOD Record **34**(4), 42–47 (2005)
19. Tennison, J.: Creating uris (2010). http://data.gov.uk/resources/uris
20. Zeng, K., Yang, J., Wang, H., Shao, B., Wang, Z.: A distributed graph engine for web scale RDF data. In: Proceedings of the 39th International Conference on Very Large Data Bases, PVLDB 2013, pp. 265–276. VLDB Endowment (2013)