Supervised tracking and correction of inconsistencies in RDF data

Youen Péron, Frédéric Raimbault, Gildas Ménier, Pierre-François Marteau

Supervised tracking and correction of inconsistencies in RDF data

Youen Péron, Frédéric Raimbault, Gildas Ménier, and Pierre-François Marteau

Valoria, Université de Bretagne Sud, Université Européene de Bretagne, Vannes, France
E-mail: {firstname.lastname}@univ-ubs.fr

Abstract. The rapid development of the Linked Data is hampered by the increase of errors in published data, mainly related to inconsistencies with domain ontologies. This problem alter reliability of application when analysis of independent and heterogeneous RDF data is involved. Therefore, it is important to improve the quality of the published data to promote the development of Semantic Web. We introduce a process designed to detect and identify specific inconsistencies in published data, and to fix them at an ontological level. We propose a method to quantitatively evaluate domain and range issues of a property. A diagnosis is then established and used to drive a supervised process for the correction (or possibly enhancement) of the ontology. We show the interest of this method on a case study involving DBpedia. The results of experiments carried on very large data sets generated with rdf sp2bench benchmark validate the applicability and scalability of our method.

1 Introduction

Launched in 2006 by Tim Berners-Lee, the *Linked Data*[1] movement and its recommendations is playing a leading role for publishing, sharing, and interconnecting data on the Semantic Web [3]. Many companies (eg. the BBC and The New York Times), organizations (eg. Wikipedia Geonames, FreeBase, UN) and governments (eg. U.S. and U.K.) have adopted the principles of *Linked Data* to publish their data using RDF[2] standard and their ontologies[3] using RDFS[4] and OWL[5]: the size and variety of available data sets keep growing. April 2011, the *Linked Data* giant global graph data has been estimated over 200 large data sets (more than 1000 triplets each), and totaling 29 billion RDF triples and 400 million links[6]. Unfortunately, this uncontrolled growth is accompanied by a proliferation of errors in published data associated with ontological inconsistencies. This problem alters application reliability when analysis of independent and heterogeneous RDF data is involved. Therefore, it is important to improve the quality of the published data to promote the development of the Semantic Web. Our method quantitatively evaluates domain and property misuses and helps fix these problems.

Most of the research –dealing with data quality in Semantic Web– focus either on ontology (class and properties definition) or raw data (class and literal instances).

Ontology validation has been extensively studied in [4, 2, 10, 8]. These works verify the consistency and completeness of an ontology within the assumptions of the open world and non-unique identification characterizing the semantic web : The tools are targeted towards ontology designers, irrespectively of the use related to the data publication.

Surprisingly, validation of raw data has been much less studied, even if it represents the main part of the published data. Some tools have been designed for syntactic validation, such as VRP [14] a tool from ICS-FORTHRFDFSUITE[7]. This kind of tools checks the data compliance with the

[1] http://www.w3.org/DesignIssues/LinkedData.html
[2] http://www.w3.org/TR/rdf-concepts/
[3] http://www.w3.org/TR/webont-req/#onto-def
[4] http://www.w3.org/TR/rdf-schema/
[5] http://www.w3.org/TR/owl-semantics/rdf-concepts/
[6] http://www4.wiwiss.fu-berlin.de/lodcloud/
[7] http://139.91.183.30:9090/rdf

RDF syntax and semantic constraints (also called consistency logic) in ontologies - under the assumption that the used ontologies are errors free. The origin of the error in raw data is searched a syntactic level. Error reporting is performed individually for each erroneous instance.

The Pedantic Web's project[8] proposes diagnosis of ontology errors and (most important) recommendation to avoid them. In [6] the initiators of this project perform an analysis of errors found in a sample set obtained by crawling: they identify four symptoms of recurring errors and propose recommendations to manage them. The authors also introduce an online tool, RDF:ALERTS[9] designed to detect these errors. Our contribution takes part to the Pedantic Web main works: we propose to enhance the semantic analysis by adding range and domain verification for properties on non literal data (unlike [6] where the analysis is strictly restricted to range checking on literal data). Our analysis is performed on class instances, being subject or object of property. We also propose a way to measure the importance of errors detected so that the priority of the correction can be evaluated: if a correction has to be done, we propose a diagnosis and an adapted fix.

In [13], the authors also address the problem of data compliance to ontologies. They propose a generic evaluation method of data, based on systematic search of some error patterns: this search is performed by SPARQL queries applied to the deductive closure of the graph of all inference rules. Their formulation is complex because it requires negations (which is nont trivial for SPARQL). Since we want our method to be scalable such to be tested on very large set of RDF data, we propose a one-rule inference (hierarchy) process to speed up the detection of possible errors. We performed our test using the parallel environment Hadoop and the request language PIG. Their method isn't providing any assistance to correct the ontology. If necessary, our approach aims also to fill this gap. The rest of the paper is organized as follows: In part 2, we recall some basic concepts of the semantic web and we define the notations used through out this article. The third part is the description of our processing method. The evaluation and tests are introduced in the fourth section. Then we conclude with a discussion in the last section.

2 Background and notations

In this part, we recall some concepts of the RDF data model, RDFS and OWL vocabularies and we define the notations we used in this paper.

2.1 rdf model

According to the RDF data model, all information is expressed as a triplet $(subject, predicate, object)$. The subject and predicate of a triple are resources identified by URIs[10]. In this article, we use the prefixed URI form to simplifies the examples. A triplet object can be either a ressource or a literal. Let $\mathcal{U}$ be the set of resources and $\mathcal{L}$ the set or literals. A triplet t is defined as follows :

$$t = (s, p, o) \in \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$$

Since a same resource can play the role of a subject or an object into a triplet, a set of triplets can be represented as a graph. We address the ABox/TBox separation problem (see [1]) by distributing the triplets in the terminology data (TBox) and assertion data (ABox).

2.2 Meta Data Vocabulary

Terminology information is described using specific RDF, RDFS or OWL vocabulary as defined by W3C. We introduce here in often the terminology we use in this paper :

[8] site : `http://pedantic-web.org`
[9] `http://swse.deri.org/RDFAlerts/`
[10] URI : Uniform Resource Identifier

`rdf:type` is a resource that provides an elementary system of belonging to a class. A triplet $(r, rdf : type, c)$ translates as 'r belongs to the class c'. Let $resources(c)$ be the set of resources belonging to the class c and let $\mathcal{C}$ be the set of classes. The number of resources in a class c is $|c| = |resoureces(c)|$. A same resource can belong to several classes. All resources belong to the class `owl:Thing`.

`rdf:Property` is a class whose instances are used as predicate in a triplet. These resources are called properties. Let the set of properties be $\mathcal{P} = resources($`rdf:Property`$)$. A property's domains and TBox ranges are denoted respectively by `rdfs:range` and `rdfs:domain`. Domain and range can be classes or even set of classes. Let $domain(p)$ be the set of graph resources that belong to the object class of `rdfs:domain` and let $range(p)$ be the set of resources belonging to the object class of `rdfs:range`. By default, the domain and the TBox range of a property p are `owl:Thing` .

`owl:ObjectProperty` is the class of properties which range cannot be a literal, but only a class instance (also called property-object in the text below).

`rdfs:subClassOf` is a property describing a subsumption relationship between a class c and a parent class f ($c \prec f$). This is a transitive relationship :

$$\forall c_1, c_2, c_3 \in \mathcal{C}\ (c_1 \prec c_2) \wedge (c_2 \prec c_3) \implies (c_1 \prec c_3)$$

Each class can subsume itself :

$$\forall c \in \mathcal{C},\ c \prec c$$

Let $\downarrow (c)$ be the set of classes that subsume a class c :

$$\downarrow (c) = \{j \in \mathcal{C}, j \prec c\}$$

The generated hierarchy is defined as $\mathcal{H}$. Let assume that, for each graph, the closure of this relationship has already been performed (as preprocessing) on the graph so that the following formula is true :

$$(\forall u \in \mathcal{U}), (\forall c, \forall f \in \mathcal{C}),\ (u \in c) \wedge (c \prec f) \implies (u \in f)$$

We used the method of reasoning described in articles [15] and [7], which enable the implementation of inference rules on large (distributed) RDF data sets.

2.3 Effective domain and range

Let $domain_e(p)$ be the set of resource subjects for a property p. Let $range_e(p)$ be the set of resource objects for a property p.

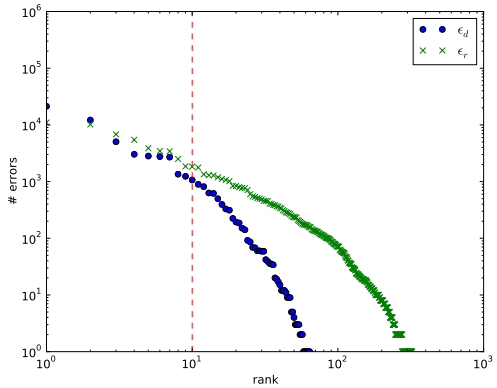

Fig. 1. Number of errors per property p as a function of the rank of p.

We check in paragraph 3 that the effective domain (resp. effective range) is included in the domain defined in the TBox (resp. in the range defined in the TBox). Let a domain error be defined as the occurrence of a subject resource of an object-property p that does not belong to $domain(p)$. Similarity, let a range error be defined as one occurrence of an object resource of an object-property p that does not belong to $range(p)$.

As a preliminary test, we have evaluated the domain and range error $\varepsilon_d(p)$ and $\varepsilon_r(p)$ in DBpedia for each object-property : 13.9 % of the properties has at least one domain error. These errors are depicted in figure 1: in wich x-axis represents the properties are sorted by descending number of errors and the y-axis represents the number of error per property. Each axis is in a log scale.

The shape of the curve reveals that the errors are concentrated on a small number of object-properties - located left to the vertical dotted line. The fixing of a small number of object-properties can enhance greatly the general consistency of the overall set of data. The method described above has two drawbacks :

(i) it is time and space expensive (especially for large data sets) because it relies on a dictionary for the classes of resources.
(ii) the lack of indication of which instances has caused the error, prevents a detailed tracking of the problem and therefore the design of a solution.

In the following parts of this paper, we describe another evaluation method that overcome these problems.

3 Our method

We propose a three-stage processing approach to compute the upper bound of the error number per object-property. Then we show how to use it to diagnose the source of the errors.

To illustrate this method, we will describe each step of application on the DBpedia's object-property `dbpedia:hometown`. Its definition's domain is `dbpedia:Person`, but in reality it is used with 21287 resources that do not belong to this class. Figure 2 shows the result of our lines processing method: in the left part, the domain of definition of `dbpedia:hometown` appears in dotted - its effective domain is hatched shaded - lines (note that the range is not depicted here). In the right part, we can see the hierarchy of the classes with the number of resources using the property.

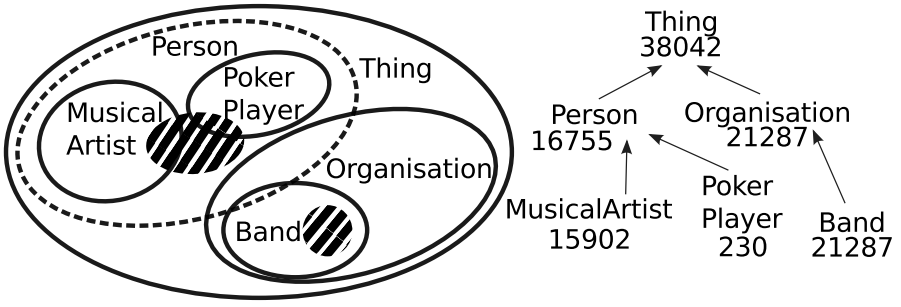

Fig. 2. Class `dbpedia:Person` is the domain of definition of `dbpedia:hometown`. The effective domain, hatched shaded, is divided between classes: `dbpedia:Person` and `dbpedia:Band`. The values appearing in the class hierarchy are those of the histogram of classes for property.

3.1 Computation of a histogram for a property

In the first stage, we calculate the distribution of the resources in the effective $domain_e(p)$ and the effective range $range_e(p)$ of an object-property p in the set of classes $\mathcal{C}$. Let us define two

histograms of size $|\mathcal{C}|$: p_d for the resources of the domain of p and p_r for the resources involved in the range of p :

$$\boldsymbol{p}_d[c] = |ressources(c) \cap domain_e(p)|$$
$$\boldsymbol{p}_r[c] = |ressources(c) \cap range_e(p)|$$

In the case of object-property `dbpedia:hometown`, the values of the histogram are indicated on the tree of Figure 2 for the relevant classes and are 0 for the other classes.

Generally, the maximum coordinate of these histogram (domain and range) is for the class `owl:Thing` because it includes all of the instances member of the domain and of the effective range.

$$\boldsymbol{p}_d[\texttt{owl:Thing}] = \max_{c \in \mathcal{C}} \{\boldsymbol{p}_d[c]\}$$
$$\boldsymbol{p}_r[\texttt{owl:Thing}] = \max_{c \in \mathcal{C}} \{\boldsymbol{p}_r[c]\}$$

In our example (DBpedia), the object-property `dbpedia:hometown` is involved by 38042 resources (distributed between people and organizations).

3.2 Searching for the most specific class

In the second stage, using the histogram for an object-property p, we select the most specific class $\texttt{class}_d(p)$ that includes the effective domain of p. It is obtained by finding the lowest class in the hierarchy $\mathcal{H}$ among the classes containing the effective domain of the property. By design, $\texttt{class}_d(p)$ features both the following relations :

$$(i)\ \ \boldsymbol{p}_d[\texttt{class}_d(p)] = \boldsymbol{p}_d[\texttt{owl:Thing}]$$
$$(ii)\ (\forall c \in \mathcal{C})(c \neq \texttt{class}_d(p))\ (\boldsymbol{p}_d[c] < \boldsymbol{p}_d[\texttt{owl:Thing}]) \vee (\texttt{class}_d(p) \prec c)$$

(i) ensures that resources($\texttt{class}_d(p)$) includes $domain_e(p)$ and (ii) ensures that $\texttt{class}_d(p)$ is the most specific class containing $domain_e(p)$. Since we assume the assumption that the classes of instances meet a hierarchy :

$$c_1 \prec c_2 \implies (\boldsymbol{p_d}[c_1] \leq \boldsymbol{p_d}[c_2])$$

This proves the existence and unicity of $\texttt{class}_d(p)$.

In the above example (see figure 2), `owl:Thing` is the most specific class for `dbpedia:hometown` that includes the effective domain.

In the same way as for the domain, lets define $\texttt{class}_r(p)$ the more specific class that includes the real range of p :

$$(i)\ \ \boldsymbol{p}_r[\texttt{class}_r(p)] = \boldsymbol{p}_r[\texttt{owl:Thing}]$$
$$(ii)\ (\forall c \in \mathcal{C})(c \neq \texttt{class}_r(p)\ (\boldsymbol{p}_r[c] < \boldsymbol{p}_r[\texttt{owl:Thing}]) \vee (\texttt{class}_r(p) \prec c)$$

3.3 Computing the upper bound of the number of errors

The last stage, involves the computation of the upper bound of the number of errors $\varepsilon_d^{sup}(p)$ - for the domain of p - and $\varepsilon_r^{sup}(p)$ for the range of the object-property p.

$$\varepsilon_d^{sup}(p) = \boldsymbol{p}_d[\texttt{class}_d(p)] - |domain(p)|$$
$$\varepsilon_r^{sup}(p) = \boldsymbol{p}_r[\texttt{class}_r(p)] - |range(p)|$$

The deviation between the number of errors and the upper bound is related to the resource (or the effective range) belonging to more than one class : these resources may be counted several times using the hierarchy of classes they belong to.

In the figure 2 example, the domain of definition is the class `dbpedia:Person`. The upper bound of error is :

$$\varepsilon_d^{sup}(\texttt{dbpedia:homeTown}) = \boldsymbol{p}_d[\texttt{owl:Thing}] - \boldsymbol{p}_d[\texttt{dbpedia:Person}] = 21287$$

This means that 21287 resources are subjects of `dbpedia:hometown` even if they do not belong to its domain of definition (here `dbpedia:Person`). We show in 4.1 that this upper bound matches perfectly the number of errors computed using the naive counting process.

3.4 Diagnostics and error correction

After having detect potential problems, we will now describe how it is possible to find and fix errors.

In the current example of figure 2, there is for instance a problem with `dbpedia:hometown`. The study of histogram class for `dbpedia:hometown` shows that most of the errors comes from the instance of `dbpedia:Band`. These instances use the property `dbpedia:hometown` but do not belong to the definition domain of `dbpedia:Person`.

A first idea involves the application of the following assumption : each resource used as a subject of a property belongs to the definition domain of the class of this property.

$$(3)\ r \in domain_e(p) \implies r \in domain(p)$$

In the case of `dbpedia:hometown`, this would imply that each music group (instance of `dbpedia:Band`) belongs also to people (instance of `dbpedia:Person`). For instance, the resource `dbpedia:The_Beach_Boys` use the property `dbpedia:hometown` but it is not an instance of `dbpedia:Person`. Changing this and letting `dbpedia:The_Beach_Boys` belong to `dbpedia:Person` may introduce errors on search related to `dbpedia:Person`.

A second solution would rely on the use of two different properties for the music group and for people. This is not acceptable since it would duplicate properties that have the same meaning - some ontology level problem may then arise.

A third idea is to relax the definition domain of `dbpedia:hometown`. In the DBpedia hierarchy, `owl:Thing` is the only less specific class than `dbpedia:Person`. The problem is that `dbpedia:hometown` is not compatible with every DBpedia's classes.

At least, we think that the best solution in this case, is to add the `dbpedia:Band` class to the definition domain of `dbpedia:hometown`.

In general, we propose a diagnostic on erroneous definition domain for property using the following scheme :

IF it seems that most of the instances of effective domain of the property belong to the definition domain's class THEN the inference (3) applies
ELSE IF the property may have different senses for the different resources THEN as much distinct properties as needed are created (one for each semantic)
ELSE IF it is possible to find a class that includes a part of the effective domain and complies with the semantic of the property and for which the number of errors is acceptable THEN the definition domain of this class is widened
ELSE the definition of the property is enhanced by merging it to the set of classes from the effective domain.

4 Evaluation

In this section, we analyze the quality of the results obtained by our approach applied on DBpedia (version 3.6), then we validate the scalability of our method using the benchmark SP2BENCH [12].

4.1 DBpedia analysis

DBpedia is built upon an ontology[11] of 272 classes, 629 object-properties and 706 litteral-properties. Range and domain of each property are defined by a class or a set of classes. The data in $843,000$ articles from Wikipedia (English version) are extracted and projected to a 3.5 million resources graph that complies to the ontology.

We performed our analysis on this set of data - effective domain and range evaluation and determination of the upper bound number of errors. An online demo that present the results of our system is available[12].

[11] `http://dbpedia.org/Downloads36`
[12] `http://cluster-valoria.univ-ubs.fr/swoct/swoct/`

Comparison with the exact number of error The table 1 shows a selection of the properties having the most errors and compares the upper bound error $\varepsilon_d^{sup}(p)$ to the exact number of errors $\varepsilon_d(p)$ - see paragraph 2.3 -

Comparison of domain	$\varepsilon_d(p)$	$\varepsilon_d^{sup}(p)$
dbpedia:class	166,004	166,004
dbpedia:hometown	21,287	21,287
dbpedia:network	12,171	12,171
dbpedia:sisterStation	5,044	5,044
dbpedia:designer	3,028	3,028
Comparison of range	$\varepsilon_r(p)$	$\varepsilon_r^{sup}(p)$
dbpedia:city	12,168	12,168
dbpedia:associatedMusicalArtist	11,027	11,027
dbpedia:artist	10,166	10,166
dbpedia:associatedBand	6,809	6,809
dbpedia:sisterStation	5,464	5,464

Table 1. Comparison of the upper bound error and the excat number of errors on the first top 5 erroneous properties

This table shows that the upper bound error $\varepsilon_d^{sup}(p)$ perfectly matches $\varepsilon_d(p)$: this is not surprising since every resource used in these properties belongs to one and only one class.

Case study. We discuss below some cases of errors encountered in DBpedia using the diagnostic process presented in paragraph 3.4.

`dbpedia:architect` is dedicated to the instance of buildings (`dbpedia:Buildings` class) but is in fact used in conjunction with `dbpedia:Place` - which is, into the hierarchy, parent to `dbpedia:buildings`. It seems acceptable to assume that every place having an architect is a building. For instance, the Mongomery place in New York is an instance of `dbpedia:Place`. Even if this place has an architect, it does not belong to `dbpedia:Building`. We propose to add the Montgomery place to `dbpedia:Building`.

`dbpedia:class` is the property causing the greatest number of errors. The notion of class is used in two different contexts. According to the ontology, this property should be used with transportation items. The histogram show that this property is mainly used by sub classes of `dbpedia:Species` - which has no relationship with transportation.

Species are in no way means of transportation (with exceptions such as horses). In this case, the inference rule (3) see in 3.4 should not apply.

It seems that DBpedia can have different semantics related to `dbpedia:Species` or `dbpedia:MeansOfTransportation`. We propose to fix this ambiguous property by creating - for each context - new properties in the TBox (and use the correct property in the ABox)

`dbpedia:city` has the definition range `dbpedia:City`. Resources from its effective range are mainly location instances (dbpedia:Place is a parent class of dbpedia:City). For instance, the triplet (`Cincinnati_bengals`, `dbpedia:city`, `paul_brown_stadium`) means that the American football team Cincinnati Bengals is located in the Paul Brown Stadium. It is therefore delicate to assume that all the places used as objects for `dbpedia:city` are cities. The property seems to have one and only one semantic for all resources in the effective range so it doesn't seem necessary to duplicate it. The most specific class that includes the effective range is `owl:thing`. This class is much too general to be used as range. The `dbpedia:place` is the best trade-off between the number of errors and the semantic consistency (only 37 resources among 17707 of the effective range of `dbpedia:city` are not instances of the `dbpedia:place` class).

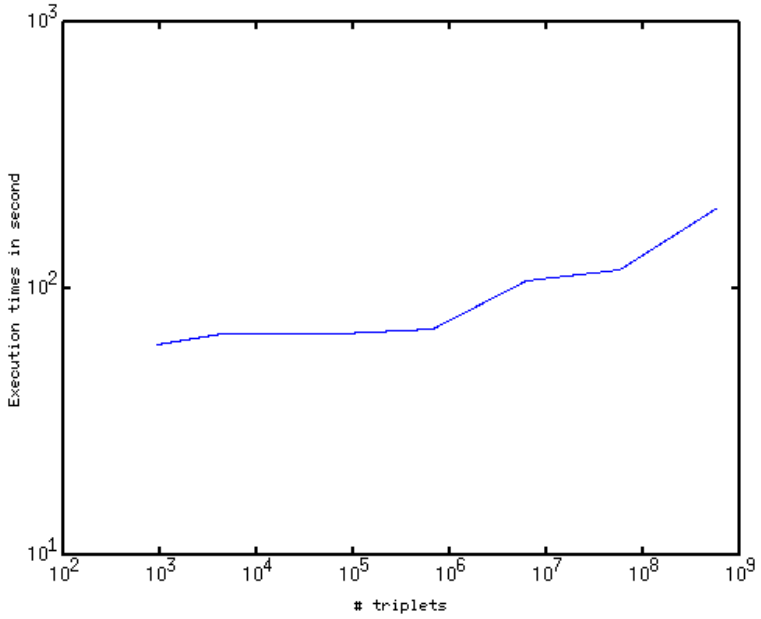

Fig. 3. Execution times required for the computation of the histograms as a function of the number of triplets generated by SP2BENCH

4.2 Scalability

We have used the PIG system [9] on a cluster of 16 computers which can process up to 250 tasks in parallel.

Since we want to study the processing time based on the graph size, we have tested our method on series of test sets of increasing size generated by SP2BENCH [12]. SP2BENCH is mainly designed to evaluate SPARQL engines but it is also useful for scalable data sets generation. We have introduces errors in ABOX by changing the reference ontology (TBOX).

The errors are detected and counted following a two step process :

1. the computation of distribution histograms executed in a data-parallel way. (see paragraph 3.1)
2. the upper bound error is sequentially computed given the histograms and the TBOX. This last step duration is not comparable to the histogram's computation so we ignored it in our results.

Figure 3 shows the results. We used a log scale for the two axis. The execution time is multiplied by 3 when the size of data increases by a factor of 10^7. This results show that our method scales relatively well acceptably scalable and demonstarte it efficiency on very large RDF datasets.

5 Conclusion

We have described and evaluated a method that analysis a RDF graph and computes upper bounds for the domain/range errors number for a given property. In the case of DBpedia, this error matches exactly the number of errors.

We explained how to perform an error diagnosis and proposed ways to fix the ontology accordingly. As an example, we applied this process to DBpedia and suggest dedicated fixing procedures..

The experimentations carried on large set of data show promising results and demonstrate the relative scalability and usability of our method for quantitative study of the evolution of Semantic Web Data.

Our approach has nevertheless some limitations: for instance, we assumed that the class hierarchy is error free so that it is possible to infer new relations. In some situations, the hierarchy relations should be checked or perhaps rebuilt - some systems [11] can compute a new hierarchy based on hierarchy clustering: anyway, this would be made under the assumption of instances correctness. These two problems are closely related so the scanning of the data base should involve jointly these processes that must be run in intricate ways.

We are currently expanding our method to the processing of other OWL constraints such as range restriction (`owl:allValueFrom`), disjonctions (`owl:disjoint`), cardinality (`owl:cardinality`) etc.. The diagnosis could also be enhanced using a semi-supervised process involving the selection of the most relevant example - for instance using page rank like algorithm [5].

References

1. Baader, F.: The description logic handbook: theory, implementation, and applications. Cambridge Univ Pr (2003)
2. Baclawski, K., Matheus, C., Kokar, M., Letkowski, J., Kogut, P.: Towards a symptom ontology for semantic web applications. In: ISWC. p. 650–667 (2004)
3. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. 5(3), 1–22 (2009)
4. Gomez-Perez, A.: Evaluation of ontologies. International Journal of Intelligent Systems 16(3), 391–409 (2001)
5. Hogan, A., Harth, A., Decker, S.: Reconrank: A scalable ranking method for semantic web data with context. In: 2nd Workshop on Scalable Semantic Web Knowledge Base Systems. Citeseer (2006)
6. Hogan, A., Harth, A., Passant, A., Decker, S., Polleres, A.: Weaving the pedantic web. In: 3rd International Workshop on Linked Data on the Web (LDOW2010), in conjunction with 19th International World Wide Web Conference, CEUR (2010)
7. Hogan, A., Harth, A., Polleres, A.: Scalable authoritative OWL reasoning for the web. Information Systems 5(2), 49–90 (2009)
8. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in owl ontologies. Journal of Web Semantics 3(4), 268–293 (2005)
9. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 1099–1110. ACM (2008)
10. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging owl ontologies. In: International World Wide Web Conference. pp. 633–640 (2005)
11. Quan, T., Hui, S., Cao, T.: A fuzzy FCA-based approach to conceptual clustering for automatic generation of concept hierarchy on uncertainty data. In: Proc. of the 2004 Concept Lattices and Their Applications Workshop. pp. 1–12. Citeseer (2004)
12. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^ 2Bench: A SPARQL Performance Benchmark. In: Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. pp. 222–233. IEEE (2009)
13. Tao, J., Ding, L., McGuinness, D.L.: Instance data evaluation for semantic web-based knowledge management systems. In: Proceedings of the 42th Hawaii International Conference on System Sciences (HICSS-42). pp. 5–8. Big Island, Hawaii (2009)
14. Tolle, K.: Validating RDF Parser: A Tool for Parsing and Validating RDF Metadata and Schemas. Master's thesis, University of Hannover (2000)
15. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable distributed reasoning using mapreduce. The Semantic Web-ISWC 2009 pp. 634–649 (2009)