Will Linked Data Benefit from Inverse Link Traversal?

Position Paper

Stefan Scheglmann
University of Koblenz-Landau, Germany
Institute for Web Science and Techologies
schegi@uni-koblenz.de

Ansgar Scherp
Kiel University, Germany
Leibniz Information Center for Economics
asc@informatik.uni-kiel.de

ABSTRACT

Query execution using link-traversal is a promising approach for retrieving and accessing data on the web. However, this approach finds its limitation when it comes to query patterns such as ?s rdf:type ex:Employee, where one does not know the subject URI. Such queries are quite useful for different application needs. In this paper, we conduct an empirical analysis on the use of such patterns in SPARQL query logs. We present different solution approaches to extend the current Linked Open Data principles with the ability for inverse link traversal. We discuss the advantages and disadvantages of the different approaches.

Categories and Subject Descriptors

H.1 [**Information Systems Applications**]: Models and Principles; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Measurement

Keywords

Querying of Linked Open Data; Link Traversal

1. INTRODUCTION

The four principles of Linked Open Data (LOD) as published in 2006[1] have gained widespread adoption in research, industries, as well as governmental and non-profit organizations. Today, we find various approaches for publishing linked data such as dedicated data stores like DBpedia, SPARQL engines with a SPARQL endpoint, plain RDF files, as well as data embedded into websites such as RDFa. After the "early years" of individually implemented solutions and proofs of concepts, we now more and more see the development of common frameworks and standardization efforts around the four LOD principles. One example is the current W3C working draft on a Linked Data Platform (LDP)[2,3], which aims at providing further clarifications and extensions to the four Linked Data Principles defined by Tim Berners-Lee. It describes a generic infrastructure for providing LOD resources as well as modifying them. The principle idea is to offer so-called LDP resources (LDPR) that can be dereferenced in order to obtain information about it (read), modify or delete the resource, or create a new resource. In addition to the notion of LDPR, the draft also suggests the concept of LDP collections (LDPC). A LDPC represents a set of "same-subject, same-predicate triples", which can be accessed and modified through a common URI. Another example of a standardization effort is the SPARQL Graph Protocol[4], a W3C recommendation that operates on the level of graphs as entities for creation, modification, and deletion. Both specifications overlap in certain points and seek to align their efforts as much as possible.[5]

We very much appreciate these efforts as they allow for developing more robust and mature LOD services. However, we believe that still improvements are needed regarding an efficient access and retrieval of linked data. Searching on linked data can in principle be categorized into two different approaches: a) Crawling all the data in a first step and subsequently indexing it to make it available to the users [6, 12, 19, 15, 5, 18, 9] and b) On-the-fly retrieval of the data according to the follow-your-nose-principle [11, 10]. While the first approach adopts the typical search scenario as it is known from the web, the second approach comes more natural with the LOD approach. This link traversal based query execution seems one of the most promising approaches to execute queries over the Web of Linked Data. No fixed, predefined set of relevant data sources has to be defined and relevant sources can be discovered during query execution. Hartig et al. [11] defined the semantics of link-traversal query execution on the web of data. However, it has its limitations. In

[1] http://www.w3.org/DesignIssues/LinkedData.html last visit 16th February, 2014

[2] http://www.w3.org/TR/2013/WD-ldp-20130730/ last visit 16th February, 2014

[3] https://dvcs.w3.org/hg/ldpwg/raw-file/default/ldp-bp/ldp-bp.html last visit 16th February, 2014

[4] http://www.w3.org/TR/sparql11-http-rdf-update/ last visit 16th February, 2014

[5] A detailed discussion on the overlaps and resulting conflicts can be found here: http://www.w3.org/2012/ldp/wiki/Main_Page#Linked_Data_Platform_.28LDP.29_vs_SPARQL_Graph_Store_HTTP_Protocol_.28GSP.29 last visit 16th February, 2014

LDOW2014 April 8, 2014, Seoul, Korea.

order to illustrate this, let us consider a simple example taken from Hartig and Freytag [11] as shown in Listing 1 below.

Listing 1: Example query on LOD (from [11]).

```
SELECT ?p ?l WHERE {
  <http://bob.name> <http://.../knows> ?p.
  ?p <http://.../currentProject> ?pr.
  ?pr <http://.../label> ?l.
}
```

The query specifies the information need to obtain a list of project names that friends of <http://bob.name> work on. This query can be executed over LOD as shown by the authors [11]. Still, executing queries on LOD has significant limitations when it comes even to slighty different information needs. For example, we might not know a-priori the existence of Bob or even if we know about him as person, we might not be aware of <http://bob.name> being the URI representing him. Thus, we like to execute a query on LOD as shown in Listing 2 where we do not know the subject URI of the first query pattern. Such a query is useful, e. g., on data providers such as the fictitious media company Big Lynx described in the Linked Data book by Heath and Bizer[6] to first determine *which employees* are working with the company, before finding out further information about each person.

Listing 2: Query requiring inverse traversal.

```
SELECT ?s WHERE
  ?s <http://.../type>
    <http://.../BigLynx/Employee> .
}
```

However, this and similar queries cannot be executed on the LOD today as it requires an inverse link traversal from the Big Lynx specific vocabulary defining the concept Employee to the resource URIs that are typed with this concept. Please note that our discussion of inverse traversal mainly addresses the rdf:type triples. However, the ideas can be applied in principle to triples with any kind of properties. We refer to this at some parts of the paper.

Below, we first further illustrate and motivate the need for an local inverse link traversal of rdf:type predicates on LOD. Subsequently, we conduct an empirical analysis on the use of SPARQL queries involving query patterns such as the one shown in Listing 2. We present possible approaches to extend the current LOD principles with the ability for inverse link traversal and discuss its advantages and disadvantages. Finally, we present related work before we conclude.

2. QUERIES ON THE SEMANTIC WEB

In order to motivate the need for an inverse link-traversal for querying LOD, we first look into further detail how queries on the Semantic Web are executed. To this end, we consider and discuss the following two questions:

(1) Why is it not sufficient to fall back to other query paradigms like SPARQL in cases where we need to resolve query patterns such as the example in Listing 2.

(2) Is this pattern important enough and would the local execution generate enough benefit to justify such an intervention into existing standards?

First, we address Question 1: As already mentioned, one might argue that one should fall back to SPARQL in cases where link traversal based query execution is not sufficient. But this might not always be possible. According to the state of the LOD cloud[7] in 2011, only 68% of the data sources in the LOD cloud provide SPARQL endpoint to allow expressive queries to be asked against the datasets. The remaining 31% are only accessible by link traversal queries.

But also the stability of provided SPARQL endpoints is of interest. Buil-Aranda et al. [4] studied the current state of available public SPARQL endpoints. They conducted various experiments to test discoverability, interoperability, performance, and availability. According to them availability of SPARQL endpoints is still an issue. In their experiments, they monitored 427 public SPARQL endpoints which are registered at DataHub[8] over a 27-month long experiment. Regarding the availability, they conclude that only around 32% of the endpoints reach an uptime of 99 – 100%, 59% an uptime over 75%, and 29.3% are available less than 5% of the time.

In order to answer Question 2, we rely on SPARQL query logs. Different analyses about the current usage of SPARQL already provide a well substantiated line of argumentation. For instance, Gallego et al. [2] analyzed the USEWOD2011 dataset. Their results show that more than 90% of the queries consist of less than 3 query patterns. SPARQL query patterns with the subject unbound while given predicate and object corresponds to our rdf:type pattern. They are the third most used type of patterns, with 7% in DBpedia and 46% in Semantic Web Dog Food (SWDF)[9] conference metadata. Möller et al. [14] applied similar analysis on a different dataset. They conducted an analysis on query logs of SWDF, DBpedia, DBtune, and RKBExplorer (RKB). In their analysis they came to comparable results, 90% consist of less than three triple patterns. The query pattern with unbound subject is used in 43% of the SWDF queries, 68% of the DBTune queries, 68% of the RKB queries, and 50% of the DBpedia queries.

Nethertheless, we conduced our own analysis, in order to show the frequent use of rdf:type in SPARQL query patterns. We extracted SELECT queries taken from the USEWOD2013[10] data challenge files. The USEWOD2013 queries where taken from Apache CLF[11] logs of four different linked open data sources, Linked Geo Data (LGD)[12], DBpedia[13], bio2RDF[14], and Semantic Web Dog Food (SWDF) conference metadata. Table 1 shows the results of our analysis.

We could show that around 11% of all SELECT queries from the aforementioned logs contain at least one pattern with rdf:type as predicate. This ratio ranges from less than 1% in the bio2rdf queries up to 14.1% in the DBpedia queries. Please note that there is also the possibility that a triple pattern like ?pr rdf:type ex:ResearchProject is contained in queries like that one in Listing 1. In such cases, the query does not

[6] http://linkeddatabook.com/editions/1.0/ last visit 16th February, 2014

[7] http://lod-cloud.net/state/ last visit 16th February, 2014

[8] http://datahub.io/en/dataset?res_format=api\%2Fsparql last visit 16th February, 2014

[9] http://data.semanticweb.org last visit 16th February, 2014

[10] http://data.semanticweb.org/usewod/2013/ last visit 16th February, 2014

[11] Common Log Format, an informal standard http://en.wikipedia.org/wiki/Common_Log_Format last visit 16th February, 2014

[12] http://linkedgeodata.org/About last visit 16th February, 2014

[13] http://DBpedia.org/About last visit 16th February, 2014

[14] http://bio2rdf.org/ last visit 16th February, 2014

DataSet DataSet	#SELECT Queries	#Queries w. rdf:type	Ratio %
LGD	1,639,889	139,788	8.52
DBpedia	27,089,369	3,819,413	14.10
SWDF	11,157,747	556,783	4.99
bio2rdf	179,370	1	0.00
total	40,066,375	4,515,985	11.27

Table 1: SPARQL query statistics on the USEWOD2013 dataset

necessarily need the possibility to inverse traversal of rdf:type links. However, investigating these cases was beyond scope of our analysis.

3. POSSIBLE SOLUTION APPROACHES

In Linked Data, each resource has an explicit URI which can be resolved in order to retrieve additional information about this entity. However, how can one do this for RDF types, i. e., how to dereference RDF type URIs? Here, two principal challenges have to be addressed:

- RDF types used in linked data sources are often not defined by the source itself, e. g., foaf:Person. Thus, the provider hosting the vocabulary does not know anything about the data sources that are using the vocabulary to describe the resources.
- In addition, resolving RDF types such as foaf:Person in order to retrieve its instances can potentially lead to a very large amount of results. In an extreme case, i. .e, when we aim at resolving the RDF type rdf:Resource, we end up retrieving all instances.

In order to address the challenges, one cannot operate on a global level, i. e., on the entire Web of Linked Data. Rather, we aim to provide resources of RDF types only on a local level, i. e., resources hosted by data sources that are run by a single organization or hosted on a single pay-level domain. Thus, when resolving a RDF type such as foaf:Person, a data provider operating a specific pay-level domain may return a reference containing information about all instances it defines using this type, e. g., by the semantic pingback mechanism [17]. In summary, we can come up with the following solution approaches based on common REST practices to allow inverse link traversal on Linked Data:

Approach 1: Simple URI Queries. It is possible that linked data providers extend their servers with mechanisms that allow for HTTP GET requests with embedded queries. This is inspired by the W3C standardization of Media Fragments[15], i. e., the definition of a unique URI scheme to access fragments of media assets such as images and videos on the web. A media fragment URI specification supports a lightweight mechanism to embed queries in such URIs. According to the standard every URI consists of four parts:

Listing 3: Media Fragment URI Schema

```
1 <scheme name> : <hierarchical part>
2                 [ ? <query> ] [ # <fragment> ]
```

[15] http://www.w3.org/TR/media-frags/#standardisation-URI-queries last visit 16th February, 2014

For inverse rdf:type traversal like in Listing 2 such an URI with an embedded query might look like the examples displayed in Listing 4. Query (1) introduces "instanceOf" as keyword to indicate inverse of rdf:type. This would allow to retrieve all instances of a specific RDF type. Query (2) extends this to "subjectOf" and allows to specify with another parameter the property URI like here the rdf:type. This allows to embed queries for arbitrary inverse properties.

Listing 4: Embedded Queries

```
(1) http://ex.com/instanceOf?
                <http://.../BigLynx/Employee>
(2) http://ex.com/subjectOf?
                p=<http://.../#type>&
                o=<http://.../BigLynx/Employee>
```

In order to address the problem that a data provider hosts a very large number of resources of specific RDF type, the data can be retrieved in a paginated manner. For example, the simple HTTP GET query can be extended by determining a limit and an offset for the data, e. g., http://ex.com/instancesOf?http://.../BigLynx/Employee&offset=100&limit=40.
This retrieves forty instances starting at an offset of 100.

The queries above could be provided as new features similar to the URI lookup endpoint feature of VoiD.[16] This feature allows to determine a dedicated URI that can be used for search by appending the *keywords* to this URI.

Approach 2: Dedicated Schema URI. Instead of providing a query mechanism, information about rdf:type entities could also be made accessible by a specific URI per type. Depending on the number of types in a dataset, this could lead to a potentially large number of entity-URIs. An alternative solution is that linked data providers could provide a special kind of RDF schema document. This schema document has to be made available under an generally accepted schema URI, e. g., http://example.org/.well-known/schema/instance-types.[17] A schema document consists of triples in the form <entities-URI> rdf:type <type-URI>, thereby the entities-URI provides a direct lookup mechanism to find instances of the specific type. For example, assuming two entities of foaf:Person on a server, namely http://bob.name and http://tim.name, its entities document would consist of two triples, cf. Listing 5. If this list is very long, i. e., there exist a large number of instances, a pagination approach can be implemented by connecting multiple RDF schema documents via rdf:List.

Listing 5: Entities document for foaf:Person

```
<http://bob.name> rdf:type <foaf:Person>.
<http://tim.name> rdf:type <foaf:Person>.
```

A globally agreed URI schema like http://example.org/.well-known/schema allows to instantly access any kind of relevant schema information for the entities in a data source such as instances of specific types. However, it is debatable whether standardization efforts should agree on generally accepted URIs for storing schema information. Thus, another way

[16] http://www.w3.org/TR/2011/NOTE-void-20110303/#lookup last visit 16th February, 2014

[17] Please note the use of the path .well-known/ for modeling a "well-known location" for common data like our schema data as proposed in RFC 5785 available from: https://tools.ietf.org/html/rfc5785

would be to add a link to an instances list to a VoID metadata description of the datasets.

Implementation Issues.

Finally, we have to think about how the response to a query (Approach 1) or the type-entity documents (Approach 2) can be generated. In general, Linked Data can be provided in different means. One can distinguish three different ways of publishing and accessing LOD on the web:

(a) Linked data sources using a dedicated RDF infrastructure, like a triple store in the back-end. These sources may provide a SPARQL endpoint to enable for complex queries. An example of such a source is DBpedia.

(b) Linked data may also be provided by sources as a set of different RDF files or data dumps. An example for this could be a web server just providing a couple of FOAF[18] documents, in addition to personal information published on hosted websites.

(c) Linked data could even be embedded into existing (X)HTML documents of a website using RDF in attributes (RDFa)[19].

For sources of type (a) the response to an inverse link traversal could just be generated by executing a query corresponding to the request directly on the triple store. Sources of type (b) and (c) need a more sophisticated mechanism. Servers providing linked data in that way have to somehow collect the information first. If the linked data is provided in multiple RDF files, e.g. like on a FOAF server, we might extend the server by an indexing mechanism, which provides all RDF files and generates the response set. In order to also support embedded RDF, this indexing mechanism has to be extended so that it can provide arbitrary files that allow to embed RDF and identify and extract the desired information from those files.

Should an indexing service generate the pages dynamically or should we go for static index pages? The former would be space saving and more robust regarding changes in the data but has several disadvantages in terms of computational power and response times. The latter is clearly more space consuming. Depending on the data, the worse case leads to redundancy in the URIs that is linear to the number of used types. It is also less robust regarding changes in the data. There might be indices affected by the change in the data and those have to be updated. But this method has clear advantages regarding performance and response times, because all required answer sets already exist. The concrete decision on how such an indexing service is finally implemented depends on the concrete use case and the data. A static indexing brings some advantages if space is not critical but computational power, or the data is static and changes infrequent, or the amount of entities per type is reasonable, or there are strong requirements regarding the response times. In other cases, e.g., for frequently changing data or datasets with very large amounts of entities per type, dynamic indexing might be more convenient. We could also think of adaptive indexing strategies, e.g. computing only those indexes dynamically which lead to a high amount of redundancy and precompute indexes for frequently requested types.

[18]The **f**riend **o**f **a** **f**riend ontology `http://www.foaf-project.org/` last visit 16th February, 2014

[19]The RDFa Primer `http://www.w3.org/TR/xhtml-rdfa-primer/` last visit 16th February, 2014

4. RELATED WORK

We find a plethora of different approaches for searching Linked Open Data (and the Semantic Web in general) such as Swoogle [6], SemSearch [12], Sig.ma [19], Sindice [15], Falcons [5], Hermes [18], and LODatio [9].[20] Despite particular features and differences these systems have, they all have in common that they index a crawled set of semantic data. As such, they require the semantic data readily at hand for search in order to provide answers to the users' queries. LODatio does not store the instance data itself but solely relies on a schema-level index. This schema-level index allows to find sources of information on the web of data without keeping the original data. However, it still requires an initial crawl of LOD to compute the schema-level index.

Among the different approaches for semantic search, there are also some that exploit information from existing query logs. Examples are Adeyanju et al. [1] who use query logs for query term recommendations such as generalizations and specializations. Meij et al. [13] align query logs with concepts from DBpedia and conduct recommendations based on the number of concepts linking to or from these concepts. Overall, query logs provide useful information about the actual information needs a LOD client has. However, they are not used by LOD providers to compute some a-priori available indices that ease the consumption of their data. Finally, we find tools that solely rely on exploring the data in direct communication with the LOD providers. A well known example is the Tabulator system by Tim Berners-Lee [3] that allows besides viewing and browsing also editing and thus updating the data. Also Marbles[21] allows for browsing the LOD graph by following the outgoing edges. Given a user provided URI, it retrieves all information about it by dereferencing it. In addition, Marbles queries search engines such as Sindice and Falcons and follows RDF predicates like owl:sameAs and rdfs:seeAlso to gain further information about a resource. While the data is retrieved live, Marbles can be considered as hybrid solution as it not only makes use of other semantic search engines but also makes the graph data persistent across user sessions to allow for a more efficient access to the data. However, it does not allow to browse the data providers, e. g., by inverse type links or some simple lookup features to find the instances of particular RDF types on a LOD data source. A comprehensive overview of further LOD components and clients can be found online and includes Linked Data browsers, crawlers, data extractors, and mashup frameworks.[22].

Hartig et al. [11] formalized traversal-based query execution on LOD, provide semantics for queries and analyse characteristics of such queries. Examples of graph traversal languages for RDF data are nSPARQL [16], a language with focus on navigation through RDF data. With NautiLOD [7], Fionda et al. introduced a declarative language which allows to specify portions of the web of data, defines routes and instructs agents to perform actions. They also introduce

[20]For a comprehensive overview, please refer to: `http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/SemanticWebSearchEngines` last visit 16th February, 2014

[21]`http://mes.github.io/marbles/` last visit 16th February, 2014

[22]`http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/SemWebClients` last visit 16th February, 2014

swget, a tool that allows for the use of NautiLOD on the web of data. GuLP [8] is a query execution language and framework based on the link-traversal paradigm, which can declaratively include preferential attachment into its queries to order the answer in terms of node/edge attributes. With SQUIN [10], Hartig presented a query execution framework that integrates the traversal of data links into the result construction process. All these approaches would clearly benefit from our proposed extension.

The VoID Vocabulary[23] allows for describing Linked Datasets in terms of providing metadata information such as a SPARQL endpoint location, example resources, and the number of triples, entities, classes, and properties in the data set. However, this does not help in identifying resources of a specific type. Most relevant to our work is the VoiD feature to denote root resources (named using the void:rootResource property). Here, it is assumed that the "entire dataset can be crawled by resolving the root resource(s) and recursively following links to other URIs in the retrieved RDF responses". Thus, in principle it is possible to list all relevant entities here. However, it cannot be stated that specific resources are of a particular RDF type. In addition, finding the relevant resources requires that the data has to be crawled first.

5. CONCLUSIONS

In our opinion traversal-based query execution is one of the most promising approaches to execute queries over the Web of Linked Data. The enormous freedom of this approach, its flexibility and reliability, the independence from centralized indexes, the on-the-fly discovery of new sources and its integration with common Web protocols, has the potential to make this paradigm the preferred mechanism to execute queries on Linked Data. From our point of view this paradigm has still some weaknesses. The analysis of SPARQL query logs has shown that inverse traversal of links in RDF data is crucial in order to make full use of the potential of Linked Data. From our point of view, the possibility to inverse traversal of rdf:type links would imply a tremendous extension to the possibilities of traversal-based query execution. We propose a set of ideas to ongoing standardization efforts of linked data and hope that these ideas will lead to a discussion in the community. Part of our future work is to conduct some experimental research that actually investigates the merits and drawbacks of the approaches outlined in the paper.

6. REFERENCES

[1] I. A. Adeyanju, D. Song, M.-D. Albakour, U. Kruschwitz, A. De Roeck, and M. Fasli. Adaptation of the concept hierarchy model with search logs for query recommendation on intranets. In *SIGIR*, pages 5–14. ACM, 2012.

[2] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An Empirical Study of Real-World SPARQL Queries. *CoRR*, abs/1103.5043, 2011.

[3] T. Berners-Lee, J. Hollenbach, K. Lu, J. Presbrey, E. Prud'hommeaux, and M. M. C. Schraefel. Tabulator redux: Browsing and writing linked data. In *LDOW*, volume 369, 2008.

[4] C. Buil-Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbusshe. Sparql web-querying infrastructure: Ready for action? In *Proceedings of the 12th International Semantic Web Conference, Sydney, Australia*, 2013.

[5] G. Cheng, W. Ge, and Y. Qu. Falcons: searching and browsing entities on the semantic web. In *WWW*, pages 1101–1102. ACM, 2008.

[6] L. Ding, T. W. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM*. ACM, 2004.

[7] V. Fionda, C. Gutierrez, and G. PirrŸ. Semantic navigation on the web of data: specification of routes, web fragments and actions. In A. Mille, F. L. Gandon, J. Misselis, M. Rabinovich, and S. Staab, editors, *WWW*, pages 281–290. ACM, 2012.

[8] V. Fionda and G. Pirrò. Querying graphs with preferences. In *CIKM*, pages 929–938, 2013.

[9] T. Gottron, A. Scherp, B. Krayer, and A. Peters. Lodatio: using a schema-level index to support users infinding relevant sources of linked data. In *K-CAP*, pages 105–108. ACM, 2013.

[10] O. Hartig. Squin: a traversal based query execution system for the web of linked data. In K. A. Ross, D. Srivastava, and D. Papadias, editors, *SIGMOD Conference*, pages 1081–1084. ACM, 2013.

[11] O. Hartig and J.-C. Freytag. Foundations of traversal based query execution over linked data. In *HT*, pages 43–52. ACM, 2012.

[12] Y. Lei, V. S. Uren, and E. Motta. Semsearch: A search engine for the semantic web. In *EKAW*, volume 4248 of *Lecture Notes in Computer Science*, pages 238–245. Springer, 2006.

[13] E. Meij, M. Bron, L. Hollink, B. Huurnink, and M. de Rijke. Learning semantic query suggestions. In *ISWC*, pages 424–440. Springer, 2009.

[14] K. Möller, M. Hausenblas, R. Cyganiak, and G. A. Grimnes. Learning from linked open data usage: Patterns & metrics. In *Proceedings of the WebSci10: Extending the Frontiers of Society On-Line*, 2010.

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

[16] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[17] S. Tramp, P. Frischmuth, T. Ermilov, and S. Auer. Weaving a Social Data Web with Semantic Pingback. In *Proceedings of the EKAW 2010*, pages 135–149, Berlin / Heidelberg, October 2010. Springer.

[18] T. Tran, H. Wang, and P. Haase. Hermes: Data web search on a pay-as-you-go integration infrastructure. *J. Web Sem.*, 7(3):189–203, 2009.

[19] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker. Sig.ma: Live views on the web of data. *J. Web Sem.*, 8(4):355–364, 2010.

[23] `http://www.w3.org/TR/2011/NOTE-void-20110303/` last visit 16th February, 2014