Who the FOAF knows Alice? RDF Revocation in DBin 2.0*

Christian Morbidoni[2], Axel Polleres[1], and Giovanni Tummarello[1]

[1] DERI Galway, National University of Ireland, Galway
{firstname.lastname}@deri.org
[2] SeMedia Group, Universita' Politecnica delle Marche, Ancona, Italy
christian@deit.univpm.it

Abstract. In this paper we take a view from the bottom to RDF(S) reasoning. We discuss some issues and requirements on reasoning towards effectively building Semantic Web Pipes, aggregating and patching RDF data from various distributed sources. Even if we leave out complex description logics reasoning and restrict ourselves to the RDF world, it turns out that some problems, in particular how to deal with contradicting RDF statements and patching RDF graphs, do not yet find their proper solutions within the current Semantic Web Stack. Besides theoretical solutions which involve full DL reasoning, we believe that more practical and probably more scalable solutions are conceivable one of which we discuss in this paper. Namely, we provide means to express revocations in RDF and resolve such revocations by means of a specialized RDF merge procedure. We have implemented this conflict-resolving merge procedure in the DBin 2.0 system.

1 Introduction

Publishing RDF files on the Web is bound to become more and more a way to state facts that are asserted or believed to be true by the producer of the source itself. DBpedia [1], for example, publishes a large collection of such facts by extracting them from the collective works of the Wikipedia communities. FOAF [5] files are personal RDF models which are created by individuals to state facts about, typically, themselves. Nothing, however, prevents them in general to state facts about other entities and this is in fact a fundamental feature of the "Semantic Web", everyone is allowed to "state" about, virtually, anything. In some cases one might even be inclined to trust third-party information more than self-descriptions, for instance comments about an enterprise or a product one considers to buy. The sum of RDF statements, currently known to be HTTP retrievable, is now in the order of billions with millions of individual HTTP locations (sources) hosted on tens of thousands of web sites, rapidly increasing. Along with this increased take-up of RDF on the Web, upcoming query language standards

* A preliminary version of this paper has been presented at the ISWC 2007 Workshop on New forms of Reasoning for the Semantic Web. This work has been partially supported by the European FP6 project inContext (IST-034718), by Science Foundation Ireland under the Lion project (SFI/02/CE1/I131), and by the European project DISCOVERY(ECP-2005-CULT-038206).

like SPARQL [14], or RDF search engines like SWSE [7] or Sindice [15] shall finally enable structured querying over Web data. Unfortunately however, there is no clear and established model on how to use such amounts of information coming from many diverse sources. Using any available source directly, e.g. crawling/downloading and using it might not be advisable or sufficient. More information might be needed such as, for example, patches to the original data. Other cases include when a source is in general considered useful but is known to contain statements which need to be removed, e.g. outdated facts (a "negative" information patch is needed), or subjective assertions which can be accepted or not depending on who is reading the data. In general, getting information from the Web into one's own semantic client or system is very likely to require, or at least benefit, from a series of custom steps to be performed involving a number of external or internal sources before having a version which can be used directly. Also, facing the sheer amount of data to be expected, more complex tasks such as ontological inferences or complex query answering will profit from such preprocessing which only preserves relevant and useful information. In this paper, we focus on one facet of such preprocessing, namely allowing to retract unwanted RDF data, and present a practical solution for this problem.

Along these lines, the remainder of this paper discusses the following issues: In Section 2 we introduce the idea of "Semantic Web Pipes", i.e. how a new breed of applications composed of small building blocks to aggregate, filter and preprocess junks of RDF data could contribute to make the Semantic Web real. In such aggregations from arbitrary sources on the Web we will naturally have to deal with contradicting statements. We will have a look on how current Semantic Web languages could support the expression of such contradicting/negative statements and how the resolution of conflicts is being addressed in Section 3. Actually, we will come to the conclusion that current languages do *not* properly address this problem so far. Based on this observation and in an attempt to address the problem with a technique we already successfully applied in a related domain (for synchronising RDF resources), we propose to express revocations of RDF statements by means of so called RDF MSG hashes. We discuss this approach and its implications in Section 4. A prototype implemented on top of the DBin 2.0 system is briefly described in Section 5, before we conclude with an outlook to future work.

2 Towards Semantic Web Pipes

Yahoo Web Pipes[1] are a recent development which has certainly had already a big impact to the latest wave of web development by showing how customized services and information streams can be implemented by sequentially processing and interleaving existing feeds and services. With Pipes, resources, e.g. RSS feeds, can be merged with one another, filtered according to specific pipe rules, used as an input for an on-line restful API to get yet more results, etc. Most interestingly, this all happens without the original providers of informations and services had to change anything on their side or reach any form of agreement if not to use HTTP and possibly RSS. Current mashup

[1] `http://pipes.yahoo.com/`

models like Yahoo Pipes are however limited to "streams" of information (e.g. news feeds) or single, simple API invocations on a remote site (e.g. a search for a specific word, or, more general, one-shot Web service invocations).

In the same way as a Web Pipe enables an existing Web information stream to be customized, extended and reused for a specific purpose as decided by the pipe creator, we see a very clear interest in trying to use this model to address the issue we highlighted before: how to make use of web published RDF sources? We might for example want to use DBpedia knowledge about a topic, but yet sum it with the knowledge coming from certain specific sites and correcting it by eliminating some statements we believe to be false. The Web Pipe model teaches us that we do not really want to download the DBpedia RDF dump, and operate directly on a local version of it, e.g. by adding and subtracting triples in a complex SPARQL query (see also the following Section). By doing so once and in a static manner, we would create a customized knowledge base at the beginning but would miss any new information that any of the composing sources might later add. A much more dynamic and useful model would therefore be a "Semantic Web Pipes" model where an RDF piping engine can on the fly and on demand work out the customized composition and processing of a set of Web sources according to our specific needs. In case where information needs to be simply added, the RDF semantics [8] specifies how to merge two models: the piping engine has therefore to do not much more than downloading the files and putting them together in the same store, standardizing apart blank nodes. But what to do when information needs to be patched in a traditional sense, i.e. in part both removed and added?

As a use case, let us take the case where Bob is stating that Charles knows Alice in his FOAF [5] file. Alice has a questionable reputation, and Charles, clearly, has no control on Bob's FOAF file. Clearly, a minimal requirement on distributed metadata is the ability to counter such false statements, thus giving Charles a way to state in his FOAF file a simple and unambiguous statement: "I don't know Alice". We aim to provide a simple and minimalistic solution to this problem, thus avoiding unnecessarily complex reasoning.

3 Related Works: Expressing Negative RDF Statements

First, we note that neither RDF nor RDF Schema provide means to make negative statements such as "Charles doesn't `foaf:know` Alice", see last statement in Figure 1(b).

```
@prefix : <http://examp.org/ bob#>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
:me foaf:name ''Bob''.
:me foaf:knows <http://alice.exa.org/i> .
:me foaf:knows <http://ex.org/charles#me>.
<http://ex.org/ charles#me> foaf:knows
        <http://alice.exa.org/i>.
...
```

(a) Bob's FOAF file

```
@prefix : <http://ex.org/ charles#>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix rdf: <http://www...rdf-syntax-ns#>
:me rdf:type foaf:Person;
    foaf:name "Charles".
:me foaf:knows <http://examp.org/bob#me>.
:me foaf:knows <http://alice.exa.org/i>.
...
```

(b) Charles' FOAF file

Fig. 1. Personal information in FOAF

The semantics of RDF(S) is purely monotonic and described in terms of positive inference rules, so even if Charles added instead a new statement

```
:me  myfoaf:doesntknow  <http://alice.exa.org/i> .
```

he would not be able to state that statements with the property `myfoaf:doesntknow` should single out[2] `foaf:knows` statements.

N3

Tim Berners-Lee's Notation 3 (N3) [2] provides to some extent means to express what we are looking for by the ability to declare falsehood over reified statements which would be written as:

```
{ :me foaf:knows <http://alice.exa.org/i> } a n3:falsehood .
```

Nonetheless, this solution is somewhat unsatisfactory, due to the lack of formal semantics for N3; N3's operational semantics is mainly defined in terms of its implementation cwm[3] only.

OWL

The falsehood of Charles knowing Alice can be expressed in OWL, however in a pretty contrived way, as follows (for the sake of brevity we use DL notation here, the reader might translate this to OWL syntax straightforwardly):

$$\{charles\} \in \forall \texttt{foaf:knows}.\neg\{alice\}$$

Reasoning with such statements firstly involves OWL reasoning with nominals, which most DL reasoners are not particularly good at, and secondly does not buy us too much, as the simple merge of this DL statement with the information in Bob's FOAF file would just generate a contradiction, invalidating all, even the useful answers. Para-consistent reasoning on top of OWL, such as for instance proposed in [9] and related approaches, solve this problem of classical inference, but still requiring full OWL DL reasoning.

SPARQL

Finally, more along the Pipes idea, one could as a naive solution, deploy an off-the-shelf SPARQL engine and filter Bob's FOAF file by a query, leaving just the clean statements. Imagine that Charles stores his unwanted statements in the RDF Web source `<http://ex.org/~charles/badstatements.rdf>`, then such a query filtering the information from merging Bob's and Charles' FOAF files could look as follows:

```
CONSTRUCT { ?S ?P ?O }
FROM <http://ex.org/~charles/foaf.rdf>
FROM <http://ex.org/~bob/foaf.rdf>
FROM NAMED <http://ex.org/~charles/badstatements.rdf>
```

[2] In fact, we mean here overriding instead of simply contradicting in the pure logical sense.

[3] `http://www.w3.org/2000/10/swap/doc/cwm`

```
WHERE { ?S ?P ?O .
        OPTIONAL { GRAPH <http://ex.org/~charles/badstatements.rdf>
                         { ?S1 ?P1 ?O1 . }
                   FILTER (?S1 = ?S && ?P1 = ?P && ?O1 = ?O &&) }
        FILTER ( !Bound(?S1) )  }
```

However, simply putting the bad information in a separate file is not a proper solution for the scenario we outlined, as it is not clear how a Crawler stumbling over <http://ex.org/~charles/badstatements.rdf> should disambiguate this data from valid RDF information. Rather, we would need to reify the negative statements using for instance the N3 version outlined before, or the "native" RDF reification vocabulary[4] which would – besides blowing up metadata by unhandy reified statements – further complicate SPARQL querying of that Data[5] to filter out the "good" data.

In the following, we will sketch a more practical solution to the problem, exploiting previous work on Minimum Self Contained Graphs.

4 Implementing RDF revocations based on MSG hashes

Any RDF graph may be viewed as set of triples. Triple level processing of distributed RDF files, particularly identifying the same RDF graphs, is made very complex by the existence of blank nodes. For this reason, the RDFSync algorithm, which we presented in previous work, introduced the notion of Minimum Self Contained Graph (MSGs) [16].

Simply said, an MSG is constructed starting from a triple and collecting, for each blank node in it, all the other triples attached to these until no more blank nodes are involved. Such "closure" makes sure that a graph can be recomposed at a different location simply by merging all the MSGs by which it is composed, even if these are transferred one at a time.

As MSGs are stand-alone RDF graphs, they can be processed with algorithms such as canonical serialization.We use an implementation of the algorithm described in [4], which is part of the RDFContextTools Java library[6] to obtain a canonical string representing the MSG and then we hash it to an appropriate number of bits to reasonably avoid collisions. This hash acts as a unique identifier for the MSG with the fundamental property of being content based, which implies that two remote peers would derive the same hash-ID for the same MSGs in their Databases.

Each graph can be therefore treated as a set of digital hashes each one representing an MSGs. In the context of the problem addressed in the present work, we use such digital hashes to refer to the MSG itself, ie. the finest granularity at which we allow to revoke RDF statements is at the level of MSGs. The hash function we use for MSGs takes the form of a literal encoding the 16 bytes of the MD5 hash of the canonical graph serialization mentioned above.

[4] Using `rdf:Statement`,`rdf:subject`, `rdf:predicate`,`rdf:object`

[5] Note that, in the FILTER query, we exploit the admittedly awkward way to model set difference in SPARQL which as such might already not be considered intuitive unanimously.

[6] http://www.dbin.org/RDFContextTools.php

Stating that an MSG is false/revoked is therefore as easy as stating one triple where the subject is a blank node, the predicate is a designated one[7] and the object is a 16 bytes literal containing the MSG hash. So the negative statement could be made directly within Charles' FOAF file or in a separate file as follows:

```
@prefix pipes: <http://pipes.deri.org/2007/10/ns#> .
_:a  pipes:revokesMSGHash
           "HASH_OF_:ME_FOAF:KNOWS_ALICE"^^xsd:string .
```

Storing MSG hashes instead of reifying statements has (except saving storage space) some other interesting implications: This solution allows revoking sets of statements which involve blank nodes. This would not be possible using reification due to the arising ambiguity. Digital hashes over MSGs, which are agnostic about blank node IDs, avoid this problem. Some particular cases, however, require further discussion (see next Session);

A drawback of the solution to quasi "encode" the negative statements in MSG hashes which in fact possibly turns out to be a feature in certain use cases, is that the negated statements are not clearly "readable", e.g. by direct inspection of the RDF file. This can be considered a feature rather than a bug for instance when one cares that denied statements are not to be known by third-parties upfront.[8]

If, on the contrary, the denied statements should be made legible, one could think of adding auxiliary statements for this purposes (such as the above-mentioned reified N3 statements, or using agreed complementary predicate URIs modified, e.g. to adding "`not:`" in front as part of the URI or as a designated URI Scheme).

4.1 MSGs involving blank nodes: issues and considerations

Our approach do not allows to revoke single statements composing an MSGs, but only the whole MSG itself. In the case the MSG in question contains blank nodes this means that if we imagine Charles would be revoking the MSG hash for

```
MSG 1:
<http://ex.org/~charles/foaf.rdf#me>  foaf:knows _:a .
 _:a foaf:name ``Alice''.             .
```

that would not have any effect if Bob had stated for instance:

```
MSG 2:
<http://ex.org/~charles/foaf.rdf#me>  foaf:knows _:a .
 _:a foaf:name ``Alice''; foaf:homepage <http://alice.exa.org/> .
```

in his graph, as the two sets of statements are actually two distinct MSGs.

Let us consider again the *MSG 1* of the previous example. As, with respect to RDF Semantics, blank nodes should be given the meaning of existential quantified variables, denying *MSG 1* would mean to deny any instance of such MSG (that is isomorphic

[7] The prefix `http://pipes.deri.org/2007/10/ns\#` defines various other properties and classes to annotate and describe revocations, see [10] for details.

[8] Although, by some additional machinery particular negated statements could be revealed quite easily in our current approach.

MSGs with a URI in place of the blank node). If, for instance, a graph contains the following statements:

```
<http://ex.org/~charles/foaf.rdf#me>  foaf:knows
<http://alice.exa.org/i> ; foaf:name ``Alice'' .
```

one might expect the revocation to affect them. The MSG based implementation, however, would left them untouched. In real cases, where blank nodes are seldom used as existential quantified variables (but rather as individual without name, as it usually happens for FOAF persons), we claim that our approach still gives correct (with respect to user expectations) results.

4.2 Computational load

Decomposing a graph into MSGs and calculating MSG hashes might be computationally expensive if the graph is big, contains a large number of bnodes, and/or highly connected bnodes. As there is no way to retrieve an MSG starting from its hash, if not decomposing the graph into MSGs and computing the hashes to find a match, the operation of applying revocations might be time consuming. To deal whit this issue, we could add additional information to revocations, namely one extra statement pointing to one, randomly chosen URI involved in the original MSG. Such an extended revocation could look as follows:

```
_:a  pipes:revokesMSGHash
          "HASH_OF_:ME_FOAF:KNOWS_ALICE"^^xsd:string .
_:a  pipes:involvedResource
          <http://alice.exa.org/i> .
```

When applying such a revocation, we only need to calculate the hashes of those MSGs which – as a sufficient condition – contain at least one statement involving the chosen URIs for revoked MSGs (in this case `<http://alice.exa.org/i>`), thus avoiding a complete graph decomposition.

Another way to go might be to do a complete MSG decomposition once, when the graph is originally loaded, and to keep an index of MSG hashes to original triples. Such initial computational effort would however result in faster operations for repeated pipe calculation. Furthermore we notice that MSG decomposition might be needed anyway for other purposes, for example to perform remote RDF synchronization [16].

5 A Simple Semantic Web Pipe Execution Engine: Description and Implementation

Having explained the idea to encapsulate negative statements in MSG hashes and its possible benefits, we have implemented a first prototypical Semantic Web Pipe engine at the heart of the DBin 2.0 Semantic Web client and authoring tool, which we conceive to be the basis of an effective Semantic Web application middle-ware. While DBin 0.x [11] based on a P2P infrastructure where information “flows” across peers, DBin 2.0 simply provides the user with a more controlled way to define the order and the

location of the sources to import and then "executes" the pipe to generate a final RDF base which is then browsed and queried.

For our simple prototype, we exploit this order in evaluating RDF statements to be overridden: In the DBin piping engine, RDF sources can be either local or remote. These are ordered in a stack according to the priority selected by the user. At execution stage, a new empty triplestore is created which will contain the graph resulting from the pipe, let us call it 'T'. The sources are then processed one by one, from the one with the lowest priority to the one with the highest priority.[9] Naming the currently processed graph 'G', the "ordered merge" procedure is the following:

1. G is cleaned by any negative MSG that overwrites a positive MSG in G (this means that if G expresses "X" and "not X" we delete both the assertions);
2. The content of G is added to T;
3. Negative statements are "applied", i.e., if positive statements exist in T corresponding to statements revoked in G, the lower priority positive statements are removed (this step is the same of the first one except that it is applied to the resulting graph T);
4. Any remaining revocations are dropped, as they must not have effects on the higher priority graphs considered in next cycles.

Once this ordered, conflict-resolving "merge" procedure has been performed for all the RDF sources, T contains the final RDF model and DBin applies RDFS reasoning on it. We remark that the result in absence of negated statements tantamounts to exactly the common RDF merge.

Clearly, by handling conflict resolution at the RDF merge level, and applying RDFS reasoning only at the last step many issues are solved in a simple, intuitive and, at the same time, efficient manner. By removing at each step any remaining negative statement we opt for a "non symmetric" approach where positive statements are somehow considered more important and persistent than "negative" ones. Moreover, the remaining RDF set is clearly consistent (being simple RDF).

We note however, that there could also be possibly problematic corner cases. For instance, imagine that Bob sneaks in the unwanted statement about Alice as follows:

```
<http://ex.org/~charles#me>
       myfoaf:likes <http://alice.exa.org/i>.
myfoaf:likes rdfs:subPropertyOf foaf:knows.
```

In this disguise, even if Bob's FOAF data is given lower priority than Charles' FOAF file, the unwanted statement would survive the conflict resolution during our ordered merge, since we do not do RDFS inference in this process.

We are currently, investigating repairs to our approach which remedy this situation, e.g. by labeling inferred triples with the priority of the lowest statement contributing to their inference and, in a recursive process removing conflicting inferred triples in a post processing step. Unfortunately, we conjecture that finding this lowest statement is,

[9] In the current implementation, the priorities are implicitly given through a simple sequence of sources which is processed one by one and priority is thus totally ordered.

in the general intractable[10], but we hope that an approximative solution, which at least guarantees that only overall sound triples are inferred might be achievable.

Another drawback of the current approach is that the priority order among considered RDF sources has to be given upfront as user input to DBin, which might not be a problem for smaller scale pipe examples, but be undesirable as the number of known sources grow to large scale. Trust negotiation policies, see e.g. [3], encoded directly as RDFstatements within the sources could help to assess priorities among RDF sources as we require them directly from RDF data in those resources.

Finally we notice that, in some cases, the end user might want to have more options than simply putting the considered sources in a total order, i.e. the pipe being a strict sequence of sources. To allow more flexible handling of overriding statements, allowing to consider multiple sources at the same priority, we are working to add support for a partial rather than a total order of sources. There might be different ways to handle revocations within a set of sources that have equal priority. A "cautious" solution might be to allow each source to revoke both MSGs from any of the equally prioritized sources and MSGs which are stated by sources with lower priority. An other approach, that we call "brave ", might be to ignore revocations coming from equally prioritized sources and to apply only those that come from a higher prioritized source.

6 Conclusions and future works

We outlined in the present work a practical solution to add negative statements to RDF without generating overall logical inconsistency. Even leaving aside full OWL inference, we believe that being able to override RDF statements based on user priorities on which Web resources are more or less trustworthy, is a crucial feature in Semantic Web applications. In this paper we first analyzed how negative statements can at all be expressed in current Semantic Web languages and came to the conclusion these languages do not properly address this problem, not providing means to override statements in a user defined priority order among RDF sources on the Web. Based on this observation, we presented a practical solution to the problem which is implemented on top of the DBin 2.0 system.

Our general ideas are based on the assumption that we believe only partially in Web scale DL reasoning, i.e. handling complete OWL inferencing, to be feasible in the near future. Our approach is a more practical one dealing with the increasing number of RDF data out there in an effective and arguably feasible manner. Negative statements treated in this work, which is still in a preliminary stage, are a first example of practical necessities we plan to address when effectively and efficiently processing Semantic Web data for useful Semantic Web applications in the spirit of "Semantic Web Pipes". In this sense, this work is conceived to spark discussions for more practical solutions towards making the Semantic Web real, which might also raise controversy among "purists" in terms of what the term "Semantic Web Reasoning" comprises and what not. More examples of issues we want handle in practical implementations include linking RDF data by adding views (see also [6, Section 2.10]), possibly involving scoped negation [13, 12] and evaluate scalability of such extensions in practical scenarios.

[10] A concrete algorithm and complexity studies for such an algorithm are still on our agenda

References

1. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *6th Int'l Semantic Web Conference*, Busan, Korea, Nov. 2007.
2. T. Berners-Lee. Notation 3, since 1998. Available at `http://www.w3.org/DesignIssues/Notation3.html`.
3. P. A. Bonatti and D. Olmedilla. Rule-based policy representation and reasoning for the semantic web. In *Reasoning Web - Third International Summer School*, pages 240–268, Dresden, Germany, Sept. 2007.
4. J. J. Carroll. Signing rdf graphs. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference*, pages 369–384, Sanibel Island, FL, USA, Oct. 2003.
5. D. Brickley and L. Miller. Friend of a Friend (FOAF) Vocabulary Specification 0.9. Namespace Document, May 2007, available at `http://xmlns.com/foaf/spec/20070524.html`,.
6. A. Ginsberg, D. Hirtle, F. McCabe, and P. Patranjan (eds.). RIF Core Design. W3C Working Draft 10 July 2006, available at `http://www.w3.org/TR/2006/WD-rif-ucr-20060710/`.
7. A. Harth, J. Umbrich, and S. Decker. Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In *5th International Semantic Web Conference*, Athens, GA, USA, Nov. 2006.
8. P. Hayes. RDF semantics. W3C Recommendation, February 2004, available at `http://www.w3.org/TR/rdf-mt/`.
9. Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, Edinburgh, Scotland, Aug. 2005.
10. C. Morbidoni, A. Polleres, G. Tummarello, and D. Le Phuoc Semantic Web Pipes. Technical Report DERI-TR-2007-11-07, available at `http://www.deri.ie/fileadmin/documents/DERI-TR-2007-11-07.pdf`, Nov. 2007.
11. M. Nucci, C. Morbidoni, and G. Tummarello. Enabling semantic web communities with dbin: an overview. In *ISWC2006 Semantic Web challenge*, Athens, GA, USA, 2006. Finalist.
12. A. Polleres, C. Feier, and A. Harth. Rules with contextually scoped negation. In *3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.
13. A. Polleres, F. Scharffe, and R. Schindlauer. SPARQL++ for mapping between RDF vocabularies. In *6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007)*, Vilamoura, Algarve, Portugal, Nov. 2007. To appear.
14. E. Prud'hommeaux and A. Seaborne (eds.). SPARQL Query Language for RDF. W3C Candidate Recommendation, June 2007, available at `http://www.w3.org/TR/2007/CR-rdf-sparql-query-20070614/`.
15. G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the open linked data. In *Proceedings of the International Semantic Web Conference (ISWC)*, Nov. 2007. To appear.
16. G. Tummarello, C. Morbidoni, P. Puliti, and F. Piazza. Signing individual fragments of an RDF graph. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, Chiba, Japan, 2005.