POWDER and the Multi Million-Triple Store

Stasinos Konstantopoulos
Institute of Informatics and Telecommunications
NCSR 'Demokritos', Athens, Greece
konstant@iit.demokritos.gr

Phil Archer
i-sieve Technologies
Athens, Greece
phil@i-sieve.com

ABSTRACT

In this paper we present and discuss the implementation and deployment of the Protocol for Web Description Resources (POWDER) W3C Recommendation for a large RDF repository containing millions of triples. POWDER enables taking advantage of natural groupings of URIs and their reflection on the denoted things' properties; our application implements a POWDER service that intercepts the API between the RDF store and the inference layer above it and provides annotations that appear as explicit statements to the inference service. The approach is tested on a multi million-triple store of news documents and events, where it achieves dramatic savings on storage space without impacting querying time.

Categories and Subject Descriptors

I.2.4 [**Knowledge Representation Formalisms and Methods**]: Semantic networks; H.3.1 [**Information Storage and Retrieval**]: Indexing methods; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representation Languages

General Terms

Experimentation

Keywords

RDF, POWDER, Efficient Storage

1. INTRODUCTION

In this paper we present and discuss the implementation and deployment of the Protocol for Web Description Resources (POWDER) W3C Recommendation for a large RDF repository containing millions of triples. POWDER enables taking advantage of natural groupings of URIs and their reflection on the denoted things' properties and is primarily meant as a format for publishing metadata.

SWIM 2011, June 12, 2011, Athens, Greece.

The SYNC3 system extracts and serves metadata about news articles, blog posts, and the events reported in them, discovering links between blogs and conventional news portals. In this context, we implemented and deployed a POWDER inference engine that compresses the repository size by avoiding storing properties that cannot be predicted by semantic inference, but can be predicted from a resource's URI.

The rest of this paper is organized as follows: after first introducing POWDER and previous applications in Section 2, we proceed to present the SYNC3 system and repository (Section 3) and how POWDER was implemented in it as well as the effect POWDER had (Section 4). We then close with related work (Section 5) and conclusions (Section 6).

2. THE POWDER RECOMMENDATION

POWDER can be summed up most succinctly as a solution to the `rdf:aboutEach` problem, as exposed by, among others, Gibbins and Shadbolt [1]. POWDER allows one or more properties and their values to be associated with an arbitrary number of subjects within a fully-defined semantic framework. For example, one might have some repetitive data such as the data shown in Figure 1.

Half of the triples in this example make an assertion that is already obvious to a human: that URIs on the `nasa.dataincubator.org` domain that have a path beginning with `/spacecraft` are instances of the class `space:Spacecraft`. POWDER allows good URI design like this to lead directly to assertions being made in a much more terse manner.

2.1 Description Resources

The fundamental 'unit of information' in POWDER is the *Description Resource* (DR). Using some syntactic sugar (and ignoring namespaces for clarity) we can re-cast the repetitive URIs in the example of Figure 1 as shown in Figure 2.

If the context makes it appropriate, then POWDER can be processed simply as XML. However, the semantics are strictly defined such that it can be transformed into an OWL ontology via a two-stage process. This is implemented as a GRDDL transform associated with the POWDER namespace. First, POWDER reduces the syntactic sugar of the URI set into just two primitive properties: `includeregex` and `excluderegex`. As defined in the POWDER *Formal Semantics* Recommendation [2], handy XML elements like `includepathstartswith` are converted to (well tested) regular expressions. If a specific URI matches the regular expressions that are values of all includeregex elements that are children of the iriset element and do not match all the

```
@prefix space:
 <http://purl.org/net/schemas/space/> .
@prefix foaf:
 <http://xmlns.com/foaf/0.1/> .

<http://nasa.dataincubator.org/spacecraft/1969-
059A>
  a space:Spacecraft;
  foaf:name "Apollo 11 Command and Service Mod-
ule (CSM)".
<http://nasa.dataincubator.org/spacecraft/1969-
059B>
  a space:Spacecraft;
  foaf:name "Apollo 11 SIVB" .
<http://nasa.dataincubator.org/spacecraft/1969-
099A>
  a space:Spacecraft;
  foaf:name "Apollo 12 Command and Service Mod-
ule (CSM)".
<http://nasa.dataincubator.org/spacecraft/1969-
099B>
  a space:Spacecraft;
  foaf:name "Apollo 12 SIVB" .
```

Figure 1: RDF data from the `nasa.dataincubator.org` domain.

excluderegex elements, then the resource identified by that URI has the properties and values given in the descriptor set. This form of POWDER, that only allows includeregex and excluderegex as child elements of iriset, is known as POWDER-BASE. POWDER-BASE documents are a subclass of POWDER documents with all other elements and features of POWDER being available.

The second stage of the transformation takes us away from an XML-centric world and fully into semantic technologies such that the iriset above can be encoded in Turtle as shown in Figure 3.

Notice the two instances of the `wdrs:matchesregex` property. POWDER defines a semantic extension that allows us to access and manipulate the string representation of a URI (the formal definition is given in Section 4.3 of the POWDER Formal Semantics document). Clearly this requires a processing step within an environment where the semantic extension can be implemented but, as we show below, this has been achieved with minimal effort within the Sesame framework and was the key step in greatly increasing query performance.

For the sake of brevity and clarity, we are only showing the key elements of the technology relevant to the current discourse. However, it is worth noting that valid POWDER documents must be attributed, that is, carry their own provenance information. Attempts to validate POWDER against its schema without the attribution element will fail. The reasons for this become clear when considering, for example, the use of POWDER by the Wholesale Applications Community in their widget security and privacy specification[1] to describe the application—which comprises a collection of files—and its relevant features. In such a scenario, POWDER is being used to publish metadata for others to

[1]See `http://kwz.me/o1` for more details.

```
<dr>
  <iriset>
    <includehosts>
      nasa.dataincubator.org
    </includehosts>
    <includepathstartswith>
      spacecraft
    </includepathstartswith>
  </iriset>
  <descriptorset>
    <typeof
src="http://purl.org/net/schemas/space/Spacecraft"/>
  </descriptorset>
</dr>
```

Figure 2: POWDER formulation of data from the `nasa.dataincubator.org` domain.

consume. Another emerging use case where provenance is important is resource discovery, such as a POWDER document directing RDF consumers to the appropriate SPARQL endpoint.

When transformed via the two-step process outlined above to create 'semantic powder', known as POWDER-S, the attribution data become OWL ontology headers. Where POWDER-S is being created and manipulated entirely within a semantic environment as a means to reduce the number of triples that need to be queried, the provision of provenance data is optional and likely to be unnecessary. This is the case in SYNC3 where the POWDER data is not exposed outside the implementation, but rather used as an internal optimization.

2.2 A Note on POWDER Semantics

The logical foundations of POWDER, as laid out in the Formal Semantics document, are based on an extension of RDF semantics that assigns the `wdrs:matchesregex` datatype property to resources based on their URI's matching the regular expression that fills the property.

Under this extension, the semantics of POWDER documents can be formulated in OWL, allowing POWDER statements to be involved in further inference about the properties of *given* resources known by URI. The POWDER Recommendation safeguards its semantics against involving OWL inference results in the process of calculating the extension of the `wdrs:matchesregex` property, so that the POWDER extension can be implemented independently of any inference based on POWDER assertions.

In other words, there is no POWDER construct such that its semantics make a URI class either equivalent or subsuming an OWL class; the one and only way to populate `wdrs:matchesregex` and, subsequently, the `owl:Restriction` expressions that express URI sets is to have a matching URI. This implies that POWDER semantics is only applicable to previously known resources and it does not guarantee that POWDER assertions are safe to use as *guards* in query expressions under open world semantics: query paths retrieving POWDER descriptions may only involve URI subjects or previously (in the query) bound subject variables.

Given the above, a straightforward implementation of a POWDER-enabled repository would require transforming POWDER descriptions into their OWL equivalents, assert-

```
@prefix wdrs: <http://www.w3.org/2007/05/powder-s#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[] rdf:type owl:Class ;
   owl:equivalentClass
     [ rdf:type owl:Class ;
       owl:intersectionOf ([ rdf:type owl:Restriction ;
         owl:hasValue ":///([^/?#]*)@)?([^:/?#@]+\.)?(nasa\.dataincubator\.org)(:([0-
9]+))?/"^^xsd:string ;
         owl:onProperty wdrs:matchesregex
      ] [ rdf:type owl:Restriction ;
         owl:hasValue ":///([^/?#]*)@)?([^:/?#@]*)(:([0-9]+))?(/spacecraft)"^^xsd:string ;
         owl:onProperty wdrs:matchesregex
      ])
    ] .
```

Figure 3: POWDER-S formulation of RDF data from the `nasa.dataincubator.org` domain.

ing the extension of the `wdrs:matchesregex` property for all instances of `rdfs:Resource` in the repository, and querying the closure of this knowledge base and our RDFS domain model. This is the approach taken by SemPP, one of reference implementations produced by the POWDER Working Group.[2] There are two reasons why this approach is inefficient to the point of being inapplicable: although some optimization is achieved in cases where multiple properties are implied for each `wdrs:matchesregex`, further gains can be made as the same result can be reached without making any explicit assertions whatsoever. Furthermore, the requirement to use OWL inference might be an unreasonable price for applications where only RDFS or, even more more so, only retrieval of explicit triples, is required.

3. THE SYNC3 DOMAIN

The SYNC3 domain is that of news and events described in news articles and blog posts, so that the concepts of a text *document* and of a news-worthy *event* reported in it are prominently situated in the SYNC3 model.

We shall not delve into the details of the linguistic processing pipeline of SYNC3, but it suffices to say that it involves the following steps:

- Web-site boilerplate detection and removal [3]
- Named-entity recognition and categorization [4],
- Thematic annotation with terms from the IPTC hierarchy [5]
- Thematic clustering of semantically homogeneous document fragments (hereafter, *segments*) into classes corresponding to the incident they report on.
- Short textual labelling, and geographical and temporal grounding for each incident [6].

At the end of this processing, the following information about documents and events has been extracted:

- Document metadata, including title, date of publication, and source.
- A breakdown of documents into *segments*, each being a list of consecutive syntactic elements of the document which document the same event.
- The resolution of the *abstract domain entity* that each concrete term, pronoun or other anaphora in the document refers to.
- The *semantics* of each segment, which comprises the event that is being documented as well as the subset of all terms appearing in the segment that refer to entities participating in the event.
- The geographical and temporal grounding of an event, as well as a numerical valuation of the participation of the various domain entities in an event.

The SYNC3 ontology[3] extends the DOLCE+DnS Ultra-Lite top-level ontology[4] to capture and index this information in a semantic repository supporting multi-faceted and semantic querying. Documents, segments, named entities, and events in SYNC3 receive a unique numerical ID as soon as they are crawled or extracted by the linguistic processing tools. This ID can be used to retrieve from an XML database information about them via Web services located at URLs constructed from their IDs as shown in the following examples:

- News article with ID 10123:
  `http://sync3.atc.gr/Sync3MultiMediaInfo/resources/newsArticles/10123`
- The second segment of the document above:
  `http://sync3.atc.gr/Sync3MultiMediaInfo/resources/newsArticles/10123/segments/10123_2`
- Named entity of type *person* with ID 42:
  `http://sync3.atc.gr/Sync3MultiMediaInfo/resources/persons/42`

[2]See section *Semantic POWDER Processor* at `http://www.w3.org/2007/powder` and also `http://transonto.sourceforge.net`

[3]`http://www.sync3.eu/rdf/sync3` abbreviated hereafter to `sync3:`

[4]`http://www.loa-cnr.it/ontologies/DUL.owl` abbreviated hereafter to `dul:`

```
<dr>
  <iriset>
    <includehosts>sync3.atc.gr</includehosts>
    <includepathstartswith>
      Sync3MultiMediaInfo/resources/news
    </includepathstartswith>
  </iriset>
  <descriptorset>
    <typeof
src="http://sync3.eu/rdf/sync3#Content" />
    <dul:isDescribedBy
rdf:resource="http://sync3.eu/rdf/sync3#NewsItem"/>
  </descriptorset>
</dr>

<dr>
  <iriset>
    <includehosts>sync3.atc.gr</includehosts>
    <includepathstartswith>
      Sync3MultiMediaInfo/resources/persons
    </includepathstartswith>
  </iriset>
  <descriptorset>
    <typeof
src="http://www.loa-cnr.it/ontologies/DUL.owl#Person"/>
  </descriptorset>
</dr>
```

Figure 4: Sample Description Resources from the POWDER document used to describe content and domain entity instances in SYNC3.

- Event with ID 1990: `http://sync3.atc.gr/Sync3MultiMediaInfo/resources/events/1990`

Although membership in some of these concepts (for example, dul:Event) can be directly inferred from property domain and range restrictions, some distinctions cannot be made by ontological axioms, but are useful information for answering queries. So, for example, both news items and blog posts have the same structure and are, as a consequence, identically modelled as instances of the sync3:Content concept. The distinction, however, between news items and blog posts must be retrievable by repository clients, and so sync3:Content instances are dul:describedBy either the resource sync3:newsItem or the resource sync3:blogPost, both instances of dul:Description.

Similarly for the sub-types of named entities (person, location, organization, etc.) which also need to be available to queries, membership in particular DUL types cannot be inferred but needs to be explicitly asserted.

Such resource groupings can be very naturally expressed in POWDER, as shown in Figure 4. The full POWDER document for the XML database is available at `http://sync3.atc.gr/powder.xml`

4. IMPLEMENTATION

In this section we present a *layered inference* approach for optimizing the size of a repository by using POWDER to implicitly annotate millions of resources with properties that cannot be inferred by either RDFS or OWL.

The implementation is based on the *Storage And Inference Layer* (SAIL) architecture of the Sesame framework, described immediately below, with the rest of the section focusing on our implementation of the idea presented in Section 3 above.

4.1 Sesame SAILs

Sesame is an open source Java framework for storing and querying RDF data, developed as a community project led by Aduna.[5] The framework is fully extensible and configurable with respect to storage mechanisms, inference engines, RDF file formats, query result formats, and query languages.

Sesame provides an interface for implementing *Storage And Inference Layers* (SAILs), stackable components that infer implicit RDF triples from the (explicit or also implicit) data they receive from the layer immediately below. The Sesame distribution bundles a number of components for this framework, including SPARQL and SeRQL[6] query engines, a memory-based and a disk-based bottom SAIL (triple store), and an RDFS inference SAIL.

Sesame provides access to the store, or rather access to the perspective of the store seen from the top SAIL in the stack, via *repository connection* objects through which one can:

- Assert a subject-predicate-object-context quad.
- Apply a triple pattern to obtain a *repository result*. Triple patterns are subject-predicate-object triples where any of these elements might be left blank to match any resource. Repository results are obtained either from particular context or contexts, or from the whole repository. Repository results are iterators that progressively retrieve matching statements.
- Apply a SPARQL or SeRQL query to obtain a *query result*. Query results are iterators that progressively retrieve query variable bindings that satisfy the query.

What is important to note about the repository and query result mechanisms is that both retrieve each triple or binding from the store upon request, without retrieving and keeping in memory the complete result. This is particularly pertinent for large scale repositories, as it allows for the successful completion of retrieval operations that would not have been possible otherwise.

SAIL components define *wrappers* over the standard repository connections, allowing them to intercept population, retrieval, or querying calls by the application and act upon the results or actions of the SAIL above them (for population calls) or below them (for retrieval and querying calls).

4.2 The POWDER SAIL

Within this framework, we have implemented the POWDER SAIL as an inference layer between the metadata store and other semantic inference SAILs (RDFS in our application). As shown in Figure 5, the POWDER layer allows population actions to fall through to the store and intercepts requests for repository results and query results.

[5] `http://www.openrdf.org`

[6] The *Sesame RDF Query Language*, specified at `http://www.openrdf.org/doc/sesame/users/ch06.html`

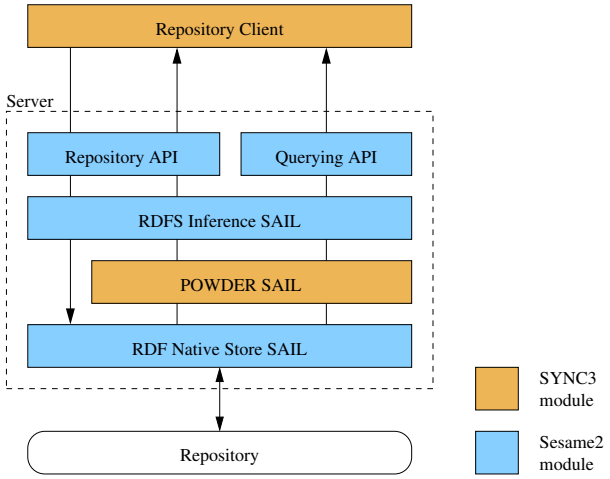

Figure 5: The SYNC3 repository SAIL stack.

Repository results are handled by wrapping the Sesame repository results implementation with one that augments the results returned by the base SAIL as follows:

- If the subject of the triple pattern is blank, then no triples are added.
- Otherwise:
  - If the predicate is blank, the subject's URI is tested against the regular expressions of all DRs.
  - Otherwise, the subject's URI is tested against the regular expressions of all DRs that assign the predicate specified in the pattern.

  Matching DRs provide potential triples, stored in an in-memory data structure in the repository results object.
  - If the object is blank, all potential triples are kept.
  - Otherwise, only potential triples with the same object as the one in the pattern are kept.

As the calling application iterates through the repository results, it first responds with the POWDER-inferred triples. As soon as these are exhausted, it proceeds to let subsequent calls fall through to the repository results provided by the SAIL under it.

It should be noted at this point that, as already discussed in Section 2, the *only* test that the POWDER SAIL needs to be able to apply is matching URIs against regular expressions; the various URI tests foreseen by the POWDER specification itself, as well as any tests defined in subsequent extensions, are required to be defined in terms of regular expressions. The POWDER SAIL loads in the POWDER document specified for the application and performs the transformation prescribed by the POWDER recommendation to obtain its POWDER-BASE representation. This is used to initialize the internal structures used to actually perform the run-time tests. As POWDER descriptions are application-wide, these structures are static and only initialized once at application initialization time, so that POWDER repository results objects do not need to repeat this process at construction time.

Table 1: Compression, in terms of number of triples and as percentage of the original size, achieved by applying POWDER to the SYNC3 repository.

Size in Triples		Difference	
Raw	POWDER	Absolute	Perc
1,878,476	1,264,140	614,336	-39.2%
2,817,714	2,156,218	661,496	-30.0%
3,663,028	2,957,805	705,223	-25.7%
4,415,114	3,383,613	1,031,501	-23.4%

4.3 Query Manipulation

Obtaining query results over the POWDER SAIL is handled in a similar fashion, that is, by extending the results obtained by the underlying SAIL.

In Sesame, queries in SPARQL or SeRQL are parsed into an internal representation of a tree of objects that represent (joins of) query paths, unions and intersections of subqueries, filters, and so on, covering the full expressivity of both query languages. SAILs do not need to be aware of the language-specific query string used to construct such a query tree, but directly operate upon this tree representation.

The POWDER SAIL traverses this representation looking for triple patterns that could potentially be satisfied by POWDER-inferred triples, that is, patterns where the predicate is bound to one of the properties assigned by the DRs in the POWDER document. These patterns are removed from the tree and replaced by FILTER clauses that enforce the appropriate regular-expression restriction upon the acceptable bindings of the subject variable.

In other words, the query results are first *extended* to accept *any* resource as having the POWDER-assigned properties, and then *restricted* back to only keeping those bindings that satisfy the URI restrictions specified in the POWDER document.

Special care is taken to never leave empty query paths. However, authoring queries with variables that are not guarded by patterns involving non POWDER-inferred properties is (possible, if necessary, but) bad practice. To explain this, consider how POWDER only makes sense for mass-annotating large volumes of URIs, such as all sync3:Content instances in SYNC3; so that trying to retrieve all instances having a POWDER-inferred property is likely to amount to retrieving millions of triples.

4.4 Results

The amount of triples saved by using POWDER to mass-annotate resources depends on the application and the density of the URI-predictable properties. In our application, the savings on content type assertions (news or blog) is linearly related to the amount of content stored in the repository, but the savings on domain entity type assertions (as extracted by the Named Entity Recognizer, cf. Section 3) depends heavily on the density of newly encountered domain entities, as these require a new domain entity instance to be typed.

In order to estimate the amount of triples saved, we used SYNC3 data to instantiate two Sesame2 Native Store instances, one full and one where POWDER-inferable triples were excluded. This data represents a week's worth of ex-

tracted and analysed news articles. As can be seen in Table 1, the compression is substantial but also shows a diminishing gains effect; the rate, however, at which the compression rate drops is also falling, converging to a steady compression rate of around 20%.

This is explained by the re-use of domain entities as they repeatedly appear in articles, so that fewer domain entity-type assertions are required. The compression rate is, naturally, specific to the particular application, but also the density of major events: new headline-making events will, naturally, introduce new named entities and increase compression rates.

What should be particularly noted is that querying time is practically the same (around 200 msec to answer queries that retrieve about a dozen tuples) between the two repositories, demonstrating how having to search through smaller indexes compensates for the overhead of the POWDER SAIL.

5. RELEVANT WORK

As already discussed in Section 2.2, the straightforward approach taken in the reference Semantic POWDER Processor implementation of constructing RDF graphs that include the POWDER-derived triples to run queries upon, is not optimal, mainly for two reasons: it requires OWL inference, and it also requires that the `wdrs:matchesregex` properties be stored. In SYNC3, both are important problems, as (a) only RDFS inference is required by the application and (b) only one or two properties per matching instance are assigned by the POWDER descriptions, so that 'spending' one triple per instance per URI set would actually *increase* the size of the repository.

Instead, we take a *layered inference* approach akin to meta-modelling [7]: in meta-modelling the inference layers (model and meta-model) are reasoned about independently and *spanning instances* that link the different layers are used to propagate results between layers. Our POWDER implementation is similar in gist, except that we transfer results from classes of URIs (rather than classes of classes) to classes of things. The meta-modelling constraint of not pushing inference results back to an already-reasoned-about layer naturally applies in reasoning about POWDER URI sets: membership in a URI set can *only* be decided by URI form and cannot be inferred from abstract properties (cf. Section 2.2 and [2, Section 4.3]).

In general, our approach is similar to various technologies related to combining semantic reasoning with triples inferred outside of OWL or RDFS semantics: Haarslev and Moller [8], for example, perform complex concrete domain inference by embedding concrete constraint-satisfaction sub-tasks in DL reasoning and Kifer *et al* [9] propose a general mechanism for interfacing Semantic Web inference technologies with different semantics, such as OWL and Horn rules.

6. CONCLUSIONS

We presented the deployment of a POWDER metadata service as part of a large RDF repository containing millions of triples. The repository implements a POWDER service that intercepts the API between the RDF store and the inference layer above it and provides annotations that appear as explicit statements to the inference service.

This is a novel application of POWDER, which is typically used in order to *publish* metadata (cf. Section 2). In SYNC3, we internally deploy a POWDER processor as part of an inference stack, exploiting the ability of POWDER to mass-annotate resources. Our approach achieves impressive compression rates for a multi million-triple store without impacting its responsiveness (cf. Section 4).

In the future, we plan to investigate how this approach compares against the Semantic POWDER Processor in applications where OWL inference is required, as well as how both approaches behave in various domains and datasets.

We are also planning to extend the POWDER SAIL so that it intercepts population actions that would assert POWDER-inferred triples. This functionality is currently embedded in the SYNC3 application logic, and not part of the POWDER SAIL, which simply lets population actions fall thought. At that point, we will publicly release our implementation as a general-purpose tool for RDF repository optimization.

Acknowledgements

The first author wishes to acknowledge the support of the European FP7-ICT project SYNC3. Please see `http://www.sync3.eu` for more information.

7. REFERENCES

[1] Gibbins, N., Shadbolt, N.: Resource Description Framework (RDF). In: Encyclopedia of Library and Information Sciences. Third edn. CRC Press (2010)

[2] Konstantopoulos, S., Archer, P. (eds): Protocol for web description resources (POWDER): Formal semantics. W3C Recommendation, 1 September 2009. (2009)

[3] Kohlschütter, C., Fankhauser, P., Nejdl, W.: Boilerplate detection using shallow text features. In: Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM 2010), February 3-6, New York City, USA. (2010)

[4] Aït-Mokhtar, S., Chanod, J.P., Roux, C.: Robustness beyond shallowness: incremental dependency parsing. Natural Language Engineering **8**(3) (June 2002)

[5] Ha-Thuc, V., Renders, J.M.: Large-scale hierarchical text classification without labelled data. In: Proceedings 4th International Conference on Web Search and Data Mining (WSDM 2011), Feb 9–12, 2011, Hong Kong. (2011)

[6] Alex, B., Grover, C.: Labelling and spatio-temporal grounding of news events. In: Proceedings of the workshop on Computational Linguistics in a World of Social Media at NAACL 2010, 6 June, 2010, Los Angeles, USA. (2010)

[7] Pan, J.Z., Horrocks, I., Schreiber, G.: OWL FA: a metamodeling extension of OWL DL. In: Proc. of the Intl. Workshop on OWL: Experience and Directions (OWL-ED 2005). (2006)

[8] Haarslev, V., Möller, R.: Practical reasoning in RACER with a concrete domain for linear inequations. In: Proc. of the Intl. Workshop on Description Logics (DL-2002), Toulouse, France, April 19-21. (2002)

[9] Kifer, M., de Bruijn, J., Boley, H., Fensel, D.: A realistic architecture for the Semantic Web. In: Proceedings of the International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005), Galway, Ireland. (2005)