Semantic Web in Enterprise
An Agile Startup Perspective

Jeen Broekstra

InsightNG Solutions Limited
http://www.insightng.com
jeen.broekstra@insightng.com

Abstract. Since the Agile Manifesto was first published, uptake in enterprise of agile methods such as Scrum has been significant. In this keynote speech, the speaker explored how, for data-intensive projects that aim to be agile, a Semantic Web technology stack can have several important benefits over other approaches.

1 Agile Development

Agile development methods such as Scrum and eXtreme Programming (XP) have over the course of the last several years seen extensive adoption, particularly in startups and organisations adhering to Lean principles. While these methods differ somewhat in details, they share a set of base priorities:

- **Individuals and interactions**
  over processes and tools
- **Working software**
  over comprehensive documentation
- **Customer collaboration**
  over contract negotiation
- **Responding to change**
  over following a plan

In essence, a successfully managed agile startup is low on bureaucracy and overhead and high in flexibility and adaptability, which gives it key competitive advantages when compared to a traditionally run corporation.

2 Semantic Web in Startups

The speaker's own tech startup, InsightNG, develops a platform for information gathering, mapping and contextualisation, providing insight and answers supporting knowledge workers in complex decision-making and problem-solving situations. In order to support the user, data and vocabularies from several Linked Data sources are reused. The use of such Semantic Web resources provides InsightNG with several benefits, which the speaker believes can be applied to other startups also:

– quick integration of heterogeneous data sources
– flexible modeling of relevant information
– powerful querying capabilities

All of these benefits are of great use in a rapidly developing situation with changing requirements, such as is common in the development of a new software product. The data sources used today may differ significantly from the data sources needed yesterday or tomorrow. By employing Semantic Web ontologies to structure data, schema changes can be made cheaply and easily, avoiding the time consuming Extract-Transform-Load type processes commonly associated with changing schemas in traditional RDBMS systems. Additionally, the low cost associated with schema changes mean that data integration challenges can be met on a pay-as-you-go basis, being driven by direct business needs, rather than be planned far ahead.

The use of Semantic Web ontologies as knowledge models reduces the need for documentation, since the model becomes the documentation. This benefit is particularly useful to small organisations lacking resources for ambitious documentation projects. It is also helpful in teams with differing language skill sets, as is common in geographically distributed teams.

In terms of pitching Semantic Web technologies to investors, the reuse of real-world Linked Data resources enables an agile company to easily provide venture capitalists with more realistic demos of their product running on top of real data. This enables the startup to already from the outset focus on showing how their product or service provides customer value, which is essential in obtaining funding.

All of these benefits illustrate why startups wishing to be agile could themselves gain from using Semantic Web technologies. It is however also worth keeping in mind that those startups would not be the only beneficiaries of Semantic Web technology use, but that the increased use and adoption of these technologies could benefit the quality of the Web as a whole. Without business backing and solid customer value being generated, we are unlikely to see a true Web-scale adoption of these helpful technologies. By getting startups on board, the first few steps towards a more Open and Semantic Web are being taken.

3 Take Home Message

In summary, academics and practitioners attending this keynote should remember that:

– Agile methods and Semantic Web technology are natural partners in crime
– Pitching Semantic Web technology should focus on its flexibility and its fit with agile principles
– Openness will follow

Semantic Web and Best Practice in Watson

Chris Welty

IBM Research

1 About the Speaker

Dr Chris Welty is a Research Scientist at the IBM T.J. Watson Research Center in New York. His principal area of research is Knowledge Representation, specifically ontologies and the semantic web, and he spends most of his time applying this technology to Natural Language Question Answering as a member of the DeepQA/Watson team. He is best known as a co-developer of the OntoClean methodology with Nicola Guarino, and as the co-chair of the W3C RIF working group.

2 Speech Abstract

IBM's revolutionary Watson system has successfully beaten human Jeopardy champions, and is now being extended and used in other domains, such as healthcare question answering, and financial data analysis. It is a common misconception that Watson is a through-and-through formally semantic system, which translates questions into formal language queries, and returns answers by executing those queries over a large knowledge base. In actual fact, Watson uses a variety of technologies to produce candidate answers to each question, and semantic technologies are primarily used in the subsequent candidate answer ranking components. In particular, linked data sources are used to provide typing evidence for candidate answers, but also several other answer ranking components rely more or less on semantics and linked data.

In this speech Dr Welty discusses these semantic components and the data upon which they operate, giving examples of expected and unexpected behaviours, and how these affect the resulting answers returned by the system. He also touches upon the methods and practices employed in developing and testing Watson, giving useful suggestions for practitioners building real world large-scale cognitive systems.

Boosting RDF Adoption in Ruby with Goo

Manuel Salvadores, Paul R. Alexander, Ray W. Fergerson,
Natalya F. Noy, and Mark A. Musen

Stanford Center for Biomedical Informatics Research
Stanford University, US
{manuelso,palexander,ray.fergerson,noy,musen}@stanford.edu

Abstract. For the last year, the BioPortal team has been working on a new iteration that will incorporate major modifications to the existing services and architecture. As part of this work, we transitioned BioPortal to an architecture where RDF is the main data model and where triple stores are the main database systems. We have a component (called "Goo") that interacts with RDF data using SPARQL, and provides a clean API to perform CRUD operations on RDF stores. Using RDF and SPARQL for a real-world large-scale application creates challenges in terms of both scalability and technology adoption. In BioPortal, Goo helped us overcome that barrier using the technology that developers were familiar with, an ORM-alike API.

Keywords: SPARQL, RDF, ORM, Ontologies

1 Why Goo? Why a Framework?

BioPortal, developed in our laboratory, provides access to semantic artifacts such as ontologies [9]. Our team has developed a new iteration of the BioPortal REST API and related infrastructure. The most significant architectural change is the replacement of the backend systems with an RDF triplestore. This single RDF triplestore replaces a variety of custom database schemas that were used to represent ontologies originated in different languages.

The BioPortal REST API provides search across all ontologies in its collection, a repository of automatically and manually generated mappings between classes in different ontologies, ontology reviews, new term requests, and discussions generated by the ontology users in the community [9]. Most importantly, our API provides uniform access to the terminologies regardless of the language used to develop them. Naturally, we did not expect the majority of developers on our team and others who access the REST API to understand which specific SPARQL query to use to access this complex information. Rather, it became much more efficient to abstract the SPARQL queries into an API that operates at the resource level. Goo (which stands for "Graph Oriented Objects") is the library that we developed for this purpose. In many ways, Goo contains characteristics of traditional Object-Relational Mapping libraries (ORMs). ORMs are widely used to handle persistency in relational databases and to provide

an abstraction over the physical structure of the data and the raw, underlying SQL queries. They also help to map data between relational models and object-oriented programming languages. Popular ORMs include Hibernate, ActiveRecord and SQLAlchemy for Java, Ruby and Python respectively. Goo is therefore an ORM specifically designed to work with SPARQL backends. The Goo library frees developers from thinking about the intricacies of SPARQL, while still exposing the power of RDF's ability to interconnect data.

The driving requirements for our design are the following:

Abstraction: Though BioPortal uses Semantic Web technologies, not all of the BioPortal development team has been exposed to RDF and SPARQL. This situation is probably common in many other development teams. At the same time, most professional developers have dealt extensively with ORMs–like Hibernate and ActiveRecord–and most developers feel very comfortable working with them.

Scalability: Our data-access layer must be aware of the query patterns for which the triplestore performance excels and must try to rely on those patterns as much as possible.

Flexibility: The schemaless nature of triplestores supports heterogeneity very well. Our store contains 2,541 different predicates, but the application needs to provide special handling for only a small portion of those [7]. Our framework needs to be flexible enough to let the developer choose what data attributes get included in the retrieval.

A number of libraries for different platforms offer ORM-like capabilities for RDF and SPARQL. Jenabean uses Jena's flexible RDF/OWL API to persist Java Beans [8]. But Jenabeans approach is driven by the Java object model rather than an OWL or RDF schema. A number of tools use OWL schemas to generate Java classes [1, 2]. These tools enable Model Driven Architecture development, but do not provide support for triple stores. For Python, RDFAlchemy provides an object-type API to access RDF data from triplestores. It supports both SPARQL backends and Python RDFLib memory models [5]. ActiveRDF is a library for accessing RDF data from Ruby programs. It can be used as data layer in Ruby-on-Rails, similar to ActiveRecord, and it provides an API to build SPARQL queries programmatically [6]. The SPIRA project, also for Ruby, provides a useful API for using information in RDF repositories that can be exposed via the RDF.rb Ruby library [4, 3]. Because we use Ruby in the new BioPortal platform, we considered SPIRA as our ORM. However, SPIRA's query strategy was not built to handle very large collections of artifacts and the query API did not allow for the complex query construction that BioPortal needs.

2 Goo's API in a Nutshell

This section briefly introduces the Goo API. In this section, and the rest of the paper, we describe the API using a subset of BioPortal models that are complex enough to help us outline Goo's capabilities.

Fig. 1. The top of this figure is an example of a Goo model definition. The settings in this model establish object validations and how this model is represented in RDF. For more details on each of these settings see the project documentation page at http://ncbo.github.io/goo/. The second part of this figure is a script that shows how we achieve persistence like similar ORM-alike libraries.

The following set of models is mentioned in the remainder of the paper: User (describes a user profile), Role (describes different roles of users in the application, such as an administrator), Note (describes comments on ontologies provided by users), Ontology (describes the object that represents an ontology entry in our repository) and OWLClass. We do not provide a fully detailed schema for these objects; each example is self-contained and the relations between objects should be clear to the reader. The full documentation for the Goo API is available at `http://ncbo.github.io/goo/`.

Goo models are regular Ruby classes. To enable RDF support each model needs to extend the `Goo::Base::Resource` class. Figure 1 gives a brief description of how models get defined. In the same figure it is shown how Ruby developers can save and validate the object without having to deal with RDF and/or SPARQL.

Once objects are defined, Goo provides a framework for creating, saving, updating and deleting object instances. Goo assures uniqueness of RDF IDs in collections and tracks modified attributes in persisted objects. The DSL allows us to provide both validation rules and define how objects are interlinked.

Ruby is a typeless language and thus developers can assign arbitrary value types to variables and attributes (i.e: the language does not enforce the assignment of `Date` values to a property that should only accept `Date` objects). We rely on Goo to perform these operations automatically and transparently, notifying when a validation fails. Goo incorporates multiple built-in validations for

data types like email, URI, integer, float, date, etc. Moreover, the framework can be extended by using Ruby lambdas which can be needed to perform custom validations. For example, a custom ISBN format validation can be included in the set of validations for a model with the following `attribute` definition:

```
attribute :isbn, enforce: [ lambda { |self| isbn_valid?(self.isbn) } ]
```

2.1 Querying: From Graphs of Triples to Graphs of Objects

Goo's most important feature is its flexible query API. The API allows retrieving individual objects, their attributes, collections, and so on.

Retrieving individual objects One can retrieve single resource instances using `Resource.find`. This call is useful when querying by unique attributes or when the URI that identifies a resource is known.

Retrieving object attributes By default, none of the query API calls attach any attribute values to the instance object that they return. If we try to access an attribute that has not been included (i.e., retrieved from the triplestore), Goo throws an `AttributeNotLoaded` exception. Our design always defaults to strategies that imply minimum data movements. This strategy improves efficiency by retrieving only the attributes that the application cares about. Data attributes are loaded into objects by using the `include` command (Figure 2).

Incremental Object Retrieval We have encountered situations in which one might not know exactly what attributes need to be loaded in an object. Goo allows incremental, in-place retrieval of attributes. An array of already loaded objects can be populated with more attributes. This operation is in-place because Goo will not create a new array of objects but will populate the objects that are passed into the query via the `models` call.

```
Users.where.models(users).include(notes: [:content])
```

Pagination Our REST API outputs large collections of data and in some cases we have to implement pagination over the responses. Pagination in SPARQL, with `LIMIT` and `OFFSET`, works at the triple level but it is not trivial for non SPARQL experts to develop the queries that retrieve a paginated collection of items. Goo provides capabilities that abstract the intricacies of triple level pagination and leverages this capability to the Ruby objects. Every query in Goo can be paginated, the underlying SPARQL query uses SPARQL `LIMIT` and `OFFSET` to assure low data transfers and minimun object instantiation. A Goo API paginated call is shown in Figure 2.4.

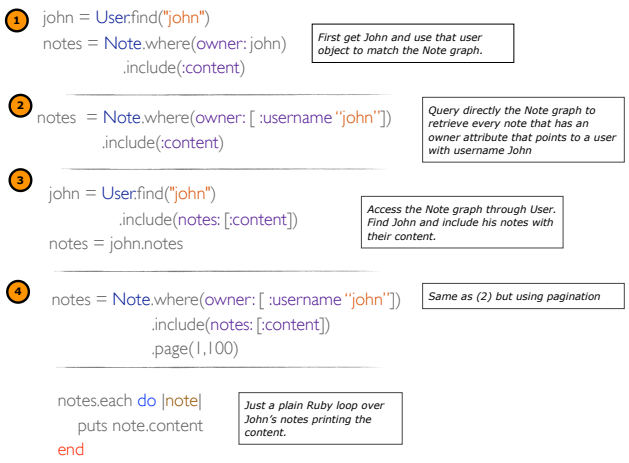

Fig. 2. Four different ways to retrieve John's notes. 1: First retrieve the user and then the notes. 2: Match the graph with a slightly more complex pattern. 3: Extract the notes by including them in a User instance. 4: same as (2) but uses pagination.

Creating complex queries The API also allows for more complex query definitions. One can combine calls with `or` and `join`, and with these we internally construct SPARQL `UNION` blocks that can be combined with SPARQL joins. Range queries can be also defined using the `Filter` object and the `filter` method. All of these operations can be combined to build complex queries.

```
filter_on_created =
      (Goo::Filter.new(:created) > DateTime.parse('2011-01-01'))
      .and(Goo::Filter.new(:created) < DateTime.parse('2011-12-31'))

Users.where(notes: [ ontology: [ acronym: "SNOMEDCT"] ])
            .or(notes: [ ontology: [ acronym: "NCIT"] ])
            .join(notes: [ :visibility [ code: "public"]])
            .filter(filter_on_created)
            .include(:username, :affiliation)
```

The Ruby code above represents a Goo query that retrieves the list of users, with their `:username` and `:affiliation`, that submitted notes to the ontologies `"SNOMEDCT"` or `"NCIT"` and these notes have visibility code `"public"`. In addition, a filter is created to filter users to just the ones that were created in the system between a range of dates. This query has an extra complexity, the `notes` attribute in `User` is defined as an inverse attribute. Goo is able to reverse the SPARQL query patterns to match the graph with the correct pattern directionality. The filtering implementation also allows for retrieval of nonexistent graph patterns. To retrieve the list of users that never submitted a note we simply use the `unbound` call in `Filter`.[1]

[1] See usage of `Filter.unbound` at `http://ncbo.github.io/goo/`

3 Goo's Query Strategy

Different query strategies can lead to starkly different performance in SPARQL. A triplestore might have an efficient query implementation, but if our application, for example, moves data around a lot, our queries will not perform well. Indeed, one often hears complaints about the performance of SPARQL engines whereas the real issue is the client who is not using SPARQL efficiently. Our key rationale with implementing query strategies in Goo is to provide a layer that ensures efficient access and query decomposition for SPARQL without the developer having to worry about it.

Goo's strategy navigates the graph of included patterns recursively. The first query focusses on constraining the graph and retrieving attributes that are adjacent to the resource type. To retrieve data attributes located more than one hop away, Goo runs additional queries—as many of these queries are types of resources that are involved in the retrieval request. As a result, when a developer uses Goo to request attributes of dependent resources, Goo will decompose the request into multiple queries.

Goo traverses the graph patterns recursively using a Depth First Search (DFS); it focuses on individual resource types in each step. Figure 3 (right side) shows how we chain these queries together with SPARQL `FILTER`s that join sequences of `OR` operations. These filters help each subsequent query to retrieve only attributes for the dependent models and not the entire collection.[2]

Consider the following example. In BioPortal, a user can attach notes to ontologies. A note description links to a user (author of the note) and the ontology that the note refers to. Our testing dataset contains 400 ontologies, 10K notes, 900 users and 10 roles. We have intentionally skewed our testing dataset so that the top 10% ontologies account for 55% of the notes—the distribution that reflects the actual state of affairs in BioPortal. Figure 3 highlights the performance gain when retrieving notes for each ontology in BioPortal using Goo. In the Naive approach performance degrades when we start projecting, into tabular form, n-ary relations that are hidden in the graph. We have seen that it is often the case that hand-written SPARQL queries retrieve too much data at once, and this can cause combinatorial data explosions due to the hidden n-ary relations.

As it can be seen in Figure 3, we can demonstrate the difference in the performance of different query strategies even on this small dataset. Goo's strategy will recursively navigate different types of objects in the retrieval, avoiding combinatorial explosions.

4 Discussion

Traditional software development and database development has a long history of reliable frameworks to access backend systems. The majority of Semantic Web

[2] For this experiment we used 4store 1.1.5 in a cluster setup with 8 backend nodes.

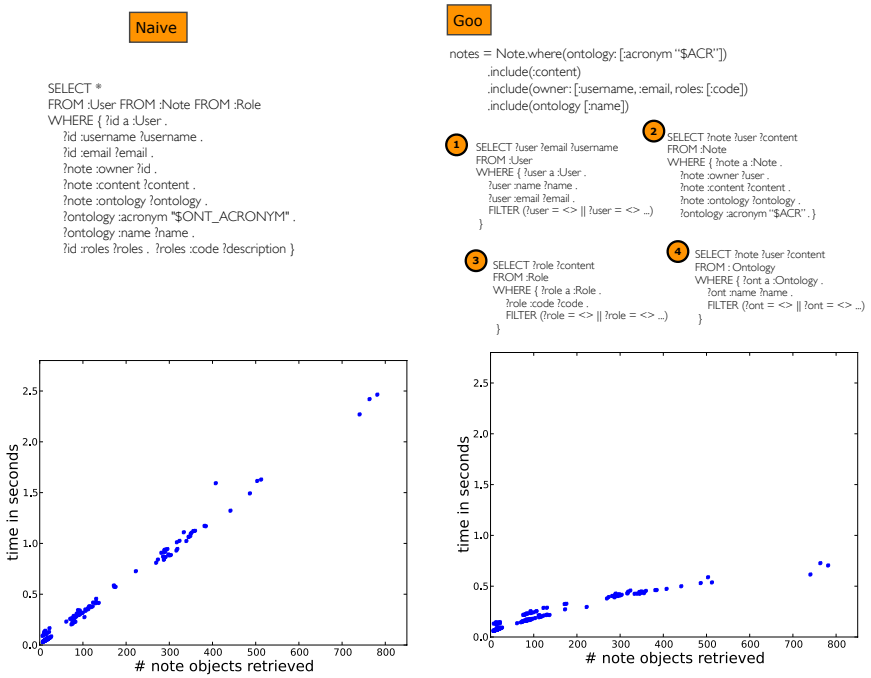

Fig. 3. Left-hand side shows a SPARQL query template that returns all notes and user information for a ontology in BioPortal. Right hand side shows the same data retrieval in Goo.

development today often includes writing SPARQL queries. In many cases, software developers are unaware of what queries the SPARQL server is optimized for and what queries they should use. Furthermore, many software-development teams do not yet have SPARQL experts, which makes relying on triplestores as components in large software systems problematic. Indeed, even in our team we have experienced this problem as we were redesigning the BioPortal backend to use a triplestore and not all of our developers were proficient in writing efficient SPARQL queries. The development of Goo enabled our team members to overcome that barrier using the technology that they were familiar with (Ruby). Goo is a completely general ORM for SPARQL and therefore other developers can use it in their projects defining their models that Goo will validate and relying on the SPARQL query optimization in Goo to access their triplestores.

In this paper we show one example of how using naive SPARQL can lead to unexpected bad performance (see Figure 3). Our preliminary study shows that the combination of n-ary relations in hand-written queries result in query time distributions that may not perform well. Figure 3 shows that a simple partioning strategy can help to alleviate this issue.

We are proponents of semantic technologies, and thus we were drawn to the idea of using ontologies to define our schemas, including the domains and the

allowed values for attributes. However, we see two major advantages to a DSL like Goo. First, one of our goals was to make schema definitions easy to use for developers who are not familiar with ontologies or OWL, and using ontologies counteracts that goal. Second, ontologies traditionally entail new information and are not designed to validate constraints.

Goo enables software developers without significant experience with semantic technologies, to use SPARQL and RDF naturally and efficiently. The BioPortal developers have found it easy to start working with the Goo API; as a result, the transition to RDF and triple store technology within BioPortal was much faster and smoother than it would have been otherwise. Basic query definitions in Goo are intuitive because the combination of hashes and arrays to construct Goo query patterns resembles JSON structures and developers are very familiar with them. Knowing that following a few simple restrictions, such as only querying for attributes that are needed, freed them from having to worry about the performance of the data storage layer and allowed them just to focus on the business logic, which sped development time significantly.

Acknowledgments This work was supported by the National Center for Biomedical Ontology, under grant U54 HG004028 from the National Institutes of Health.

References

1. OWL2Java: A Java Code Generator for OWL. `http://www.incunabulum.de/projects/it/owl2java`
2. Kalyanpur, A., Jimenez, D.: Automatic Mapping of OWL Ontologies into Java. In: Proceedings of Software Engineering and Knowledge Engineering (2004)
3. Arto Bendiken, Gregg Kellogg, B.L., Borkum, M.: RDF.rb: Linked Data for Ruby. `http://rdf.rubyforge.org/`
4. Ben Lavender, A.B., Humfrey, N.J.: Spira: A Linked Data ORM for Ruby. `https://github.com/datagraph/spira`
5. Graham Higgins, P.C.: RDF Alchemy, `http://www.openvest.com/trac/wiki/RDFAlchemy`
6. Oren, E., Delbru, R., Gerke, S., Haller, A., Decker, S.: Activerdf: Object-oriented Semantic Web Programming. In: Proceedings of the 16th International Conference on World Wide Web. pp. 817–824. WWW '07, ACM, New York, NY, USA (2007), `http://doi.acm.org/10.1145/1242572.1242682`
7. Salvadores, M., Horridge, M., Alexander, P.R., Fergerson, R.W., Musen, M.A., Noy, N.F.: Using SPARQL to Query BioPortal Ontologies and Metadata. In: International Semantic Web Conference (2). pp. 180–195 (2012)
8. Vollel, M.: Jenabean: A library for persisting java beans to RDF. `http://code.google.com/p/jenabean/`
9. Whetzel, P.L., Noy, N.F., Shah, N.H., Alexander, P.R., Nyulas, C.I., Tudorache, T., Musen, M.A.: BioPortal: Enhanced functionality via new web services from the national center for biomedical ontology to access and use ontologies in software applications. Nucleic Acids Research (NAR) 39(Web Server issue), W541–5 (2011)

Harmonizing services for LOD vocabularies: a case study

Ghislain Auguste Atemezing[1], Bernard Vatant[2], Raphaël Troncy[1] and Pierre-Yves Vanderbussche[3]

[1] EURECOM, Sophia Antipolis, France, <firstName.lastName@eurecom.fr>
[2] Mondeca, Paris, France, <bernard.vatant@mondeca.com>
[3] Fujitsu, Galway, Ireland, <py.vanderbussche@fujitsu.com>

Abstract. Vocabularies are more and more (re)-used in the Linked Data ecosystem. However, managing the prefixes associated to their Uniform Resource Identifiers (URIs) is still cumbersome and namespaces are sometimes referring to different pair <prefix, URI>. In this paper, we propose to align two well-known services with the aim of managing and harmonizing vocabularies' namespaces. We use prefix.cc that provides a look up service for namespaces in general and Linked Open Vocabularies (LOV) that extracts vocabularies metadata. Our method enables to identify three different scenarios: (i) conflicts between prefix.cc and LOV; (ii) prefixes in LOV not present in prefix.cc and (iii) URIs in prefix.cc that are actually LOV-able vocabularies. We describe how we solve each of these issues, with actions ranging from updating the different services to contacting the editors of the vocabularies to fix clashes among prefixes. Finally, we present the new LOV API that enables to check whether those namespaces in prefix.cc can actually be vocabularies to be inserted in the LOV ecosystem or not.

Keywords: Vocabulary discovery, Linked Open Vocabularies, prefix.cc, namespaces reconciliation, vocabulary management

1 Introduction

RDF vocabularies bring their meaning to linked data by defining classes and properties, and their formal semantics. Relying on W3C standards RDFS or OWL, those vocabularies are a fundamental layer in the architecture of the Semantic Web. Without the explicit semantics declared in vocabularies, linked data, even using RDF, would be just linked pieces of information where links have no meaning. Interoperability between data and datasets rely heavily on shared vocabularies, but given the distributed nature of the Web, vocabularies are published by independent parties and there is no centralized coordination of this publication, nor should it be. Various independent services have been developed in order to discover vocabularies and provide information about them, and the community of data publishers and vocabulary managers have all interest

in complementarity and coordination between such services. In this paper, we focus on a specific aspect of vocabularies: their identification by namespaces and associated prefixes.

In the original XML syntax of RDF, prefixes are simply local shortcuts associated with XML namespaces using `xmlns` declarations. The usage of prefixes has been further extended to other syntaxes of RDF such as N3 and Turtle. Although a prefix to namespace association is syntactically limited to the local context of the file in which it is declared, common prefixes such as `rdf:`, `rdfs:`, `owl:`, `skos:`, `foaf:` and many more have become de facto standards. For example, RDFa has 1.1 has a default profile made of 11 well-used vocabularies based on their general usage on the Semantic Web according to the crawl of Yahoo! and Sindice as of March 2013[1]. Similarly, the YASGUI SPARQL editor has a list of built-in prefix-namespace associations to ease the construction of SPARQL queries. However, this list of "standard" prefixes is open-ended. Interfaces such as SPARQL endpoints (e.g. Virtuoso) use a list of built-in prefixes declaration for more and more namespaces but the choice of entries in this list is all but transparent. Hence, the reason of a given namespace being or not in this list could be interpreted in many ways, a potential source of technical and social conflicts. Therefore, the notion has been slowly spreading, at least implicitly, that common prefixes could and indeed should have a global use, implying some kind of governance and good practices. More and more vocabularies explicitly recommend the prefix that should be used for their namespace, generally using a common if not written good practice to avoid frontal clashes by recommending a prefix not already used. But there is no global policy except implicit rules of fair use to avoid potential conflicts resulting from polysemy (different namespaces using or recommending the same prefix) or synonymy (different prefixes used for the same namespace).

A vocabulary publisher needs to have access to some services capable of monitoring the existing prefixes usage in order to stick to those rules. In this paper, we focus on two services providing such information on prefixes usage namely prefix.cc[2] and LOV (Linked Open Vocabularies) [5]. Both services provide associations between prefixes and namespaces but following a different logic. The prefix.cc service allows anybody to suggest a prefix to namespace association. It supports polysemy and synonymy, and has a very loose control on its crowdsourced information. What it provides is more a measure of popularity of prefixes and namespaces than a way to put order in them. LOV has a much more strict policy forbidding polysemy and synonymy, enforced by a dedicated back-office database infrastructure, ensuring that each vocabulary in the LOV database is uniquely identified by a prefix, this unique identification allowing the usage of prefixes in various LOV publication URIs. This requirement leads sometimes to a situation where LOV uses prefixes different from the ones recommended by the vocabulary publishers.

[1] `http://www.w3.org/2010/02/rdfa/profile/data/`

[2] Service: `http://prefix.cc/`; Code: `https://github.com/cygri/prefix.cc`

The initial motivation of the work presented in this paper was to provide some kind of harmonization between those two services, from simple obvious tasks such as checking that prefix.cc provides all prefixes present in LOV and add them as necessary, to more complex ones such as detection and possible resolution of conflicts. We describe an approach for discovering new vocabularies in the wild by reconciling vocabularies in prefix.cc using SPARQL federated queries. This work was made semi-manually and involved collaboration between the two services managers to exchange data and take actions on each side. The remainder of this paper is structured as follows. In Section 2, we provide an overview of related work and services that support vocabulary management including the current approaches implemented by the LOV and prefix.cc services. In Section 3, we present how we have aligned those two services, detected conflicts and resolved them. In Section 4, we describe a method enabling to find new LOV-able vocabularies from the prefix.cc service. Finally, we discuss some lessons learned and outline future work in Section 5.

2 Related Work

Many different type of repositories exist to support users and developers to find controlled terms and entire vocabularies or ontologies on the web of data. We first describe the LOV initiative (Section 2.1) and we propose then our own classification based on the content, the domain, the purpose and the way such catalogs are populated or index created (Section 2.2).

2.1 Linked Open Vocabulary (LOV)

The Linked Open Vocabularies (LOV) initiative aims to bring more insights about published vocabularies in order to foster their reuse. Compared to other projects, LOV benefits from a community:

- to assess the quality (including documentation, metadata) and the reuse potential of a vocabulary before it is indexed. LOV contains currently 350+ reusable and well-documented vocabularies;
- to augment vocabularies with explicit information not originally defined in the RDF vocabulary. For example, only 55% of vocabularies have explicit metadata of at least one creator, contributor or editor. In LOV, we augmented this information leading to more than 85% of vocabularies with this information;
- to automatically extract the implicit relations between vocabularies using the Vocabulary Of Friend[3] (VOAF) ontology. These relations can be used as a new metric for ranking terms based on their popularity at the schema level;
- to consider vocabulary semantic in the result ranking: a literal value matched for the `rdfs:label` property has a higher score than for the `dcterms:comment` property.

[3] `http://lov.okfn.org/vocab/voaf/`

The way vocabularies are considered in LOV is similar to the way datasets are considered in the LOD cloud [2]. Hence, while the Vocabulary of Interlinked Datasets (VoiD) is used to describe relationships between datasets and their vocabularies [1], VOAF is used to describe the mutual relationships between vocabularies. VOAF itself reuses over popular vocabularies such as Dublin Core Terms (dcterms), Vocabulary Of Interlinked Datasets (VoiD), Vocabulary for ANNotating vocabulary (vann) and the BIBliographic Ontology (bibo). The vocabulary also introduces new classes such as `voaf:Vocabulary` and `voaf:VocabularySpace`.

The LOV-Bot is the tool that automatically keeps up-to-date the relationships and the metadata about the vocabularies indexed in LOV, using the following steps:

- LOV-Bot daily checks for vocabularies update (any difference in the vocabulary formal description fetched using content negotiation);
- LOV-Bot uses SPARQL constructs to detect relationships and metadata and creates explicit metadata descriptions in the LOV dataset;
- LOV-Bot annotations are then listed in a back-office administration dashboard in order to be reviewed. This manual part enables LOV curators to interact with vocabularies authors and the wider community to raise issues and make remarks or suggestions.

The LOV dataset is synchronized with the information presented in the web site. The latter allows a human user to browse LOV information. The Linked Open Vocabularies initiative does not only monitor the current state of the ecosystem. It also aims at storing and giving access to vocabularies history. To achieve this goal, the LOV database contains every different version of a vocabulary over the time since its first issue. For each version, a user can access the file and a log of modifications since the previous version.

2.2 Ontology Repositories

While we refer the reader to [3] for a systematic survey of ontology libraries, we give our own classification of ontology repositories (Table 1). In particular, we distinguish six categories of catalogs:

- *Catalogs of generic vocabularies/schemas* similar to the LOV catalog, but without any relations among the vocabularies. Example of catalogs falling in this category are vocab.org[4], ontologi.es[5], JoinUp Semantic Assets or the Open Metadata Registry.
- *Catalogs of ontologies for a specific domain* such as biomedicine with the BioPortal[6], geospatial ontologies with SOCoP+OOR[7], Marine Metadata Interoperability and the SWEET ontologies[8].

[4] `http://vocab.org/`
[5] `http://ontologi.es/`
[6] `http://bioportal.bioontology.org/`
[7] `http://socop.oor.net/`
[8] `http://sweet.jpl.nasa.gov/2.1/`

– *Catalogs of ontologies from a project* such as the famous DAML repository of ontologies[9].
– *Catalogs of ontology Design Patterns (ODP)* focused on reusable patterns in ontology engineering.
– *Catalogs of editors' ontologies* used to test some features of a tool and to keep track of the ontologies built by a tool, such as Web Protégé or TONES.
– *Catalogs of ontologies maintained by a single organization* which often uses a platform such as Neologism[10] for publishing vocabularies.
– *Vocabularies crawled by Semantic Web search engines* containing snapshots at the time of the crawsuch as Watson[11], Sindice[12], Falcon-s[13] or Swoogle.

Catalog name	Number of vocabularies	Search Feature	Category	Vocabulary maintenance
vocab.org	19	No	Catalog of generic vocabularies	N/A
ontologi.es	39	No	-//-	N/A
Joinup Semantic Assets	112	Yes	-//-	Yes
Open Metadata Registry	308	Yes	–//–	Yes
BioPortal	355	Yes	Catalog of Domain vocabularies	Yes
SOCoP + OOR	40	Yes	–//–	Yes
Marine Metadata Interoperability	55	Yes	–//–	Yes
SWEEET 2.2	200	No	–//–	N/A
DAML	282	No	–//–	No
ODPs	101	No	Catalog of ODPs	Yes
vocab.derie.ie	68	No	Catalog of Organizations	Yes
data.lirmm.fr ontologies	15	No	–//-	Yes
TONES	219	No	Catalog of editors' vocabularies	N/A
Web Protégé	69	No	–//–	Yes

Table 1. Catalogs of vocabularies with respectively the number of the ontologies, the presence of a search feature, the catalog category and whether it is maintained or not

[9] http://daml.org/ontologies/
[10] http://neologism.deri.ie
[11] http://watson.kmi.open.ac.uk/
[12] http://www.sindice.com
[13] http://ws.nju.edu.cn/falcons/

We observe that the existing catalogs of vocabularies in the literature have some limitations compared with LOV. In terms of coverage, the number of vocabularies indexed by LOV is constantly growing and it is the only catalog, to the best of our knowledge, that provide all types of search criteria (metadata search, within/across ontologies search), both an API and a SPARQL endpoint access and that can be as well classified as an "Application platform" apart from being at the same time an ontology directory and an ontology registry. According to the categories of ontology libraries defined in [3], LOV falls under the category of "curated ontology directory" and an "application platform" because the ontologies are curated manually with statistics automatically generated, and because it exposes its data via an API. Furthermore, LOV provides an answer to some of the issues mentioned in the survey reported in [3], such as "where has an ontology been used before?" or "is this ontology compatible with mine?". In particular, LOV provides vocabulary usage statistics of the LOD Cloud datasets and it exposes vocabularies dependency using the Vocabulary-of-A-Friend (VOAF) ontology.

vocab.cc[14] is a service which is similar to prefix.cc since it enables to look up and search for Linked Data vocabularies while providing more specific information about the usage of a particular class or property in the Billion Triple Challenge Dataset (BTCD). It also provides the ranking of those properties or classes. The authors mentioned that "common prefixes are resolved with data from prefix.cc". Although they don't give further details, this service is somehow related to prefix.cc. Triple-Checker[15] is a web service based on prefix.cc which aims at finding typos and common errors in RDF data. It parses a given URI/URL and the output is divided in two sections: the namespaces and the term section. The former matches against prefix.cc to determine whether they are "common prefixes" and the latter provides the term definition.

3 Aligning LOV with Prefix.cc

In this section, we present how we perform the alignment between the two services LOV and prefix.cc. Figure 1 shows the evolution of the number of prefixes registered in these two services between April 2009 and July 2013. Our main goals are to align Qnames (prefix) to a unique URI in LOV and to make sure that all the vocabularies in LOV are actually inserted in prefix.cc.

We propose to perform SPARQL queries over all the files of prefix.cc at `http://prefix.cc/popular/all.file.vann` in the FROM clause and compare them to the content of the LOV SPARQL endpoint[16] via a SERVICE[17] call. The SERVICE keyword defined in the SPARQL 1.1 Query Language instructs a federated query processor to invoke a portion of a SPARQL query against a remote SPARQL endpoint [4]. Results are returned to the federated query

[14] `http://vocab.cc`

[15] `https://github.com/cgutteridge/TripleChecker`

[16] `http://lov.okfn.org/endpoint/lov`

[17] `http://www.w3.org/2009/sparql/docs/fed/service`

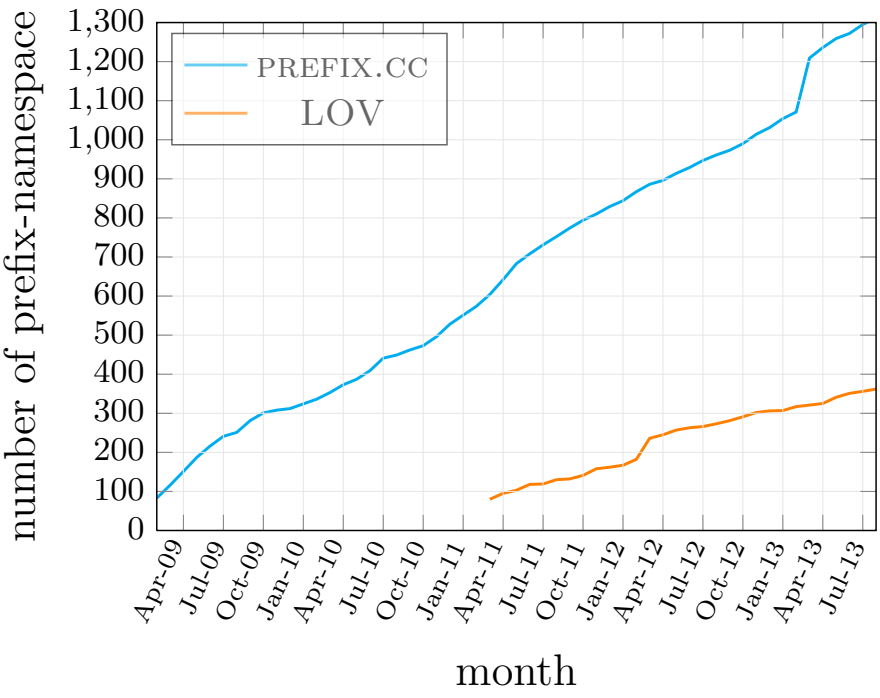

Fig. 1. Evolution of the number of prefix-namespace pairs registered in prefix.cc and LOV

processor and are combined with results from the rest of the query. To be more generic and standards-compliant, the queries could be run with the Jena ARQ command-line tool to produce a CSV or a JSON serialization that could be easily consumed either by the prefix.cc backend via phpMyAdmin or by the LOV backend.

3.1 First Task: prefixes in LOV not present in Prefix.cc

First, we compute $< LOV > INTERSECTS < PREFIX.CC >$ and $< LOV > MINUS \{< LOV > INTERSECTS < PREFIX.CC >\}$. The following SPARQL query finds namespace URIs in LOV that do not exist in prefix.cc along with their LOV prefix.

```
PREFIX vann: <http://purl.org/vocab/vann/>
SELECT ?prefix ?lovURI
FROM <http://prefix.cc/popular/all.file.vann> {
  SERVICE <http://lov.okfn.org/endpoint/lov> {
    SELECT ?prefix ?lovURI {
```

```
      [] vann:preferredNamespacePrefix ?prefix;
        vann:preferredNamespaceUri ?lovURI;
    }
  }
  FILTER (NOT EXISTS { [] vann:preferredNamespaceUri ?lovURI })
  OPTIONAL {
    [] vann:preferredNamespacePrefix ?prefix;
      vann:preferredNamespaceUri ?pccURI;
  }
}
ORDER BY ?prefix
```

The first results[18] shown the following: $card(LOV) \bigcap card(PREFIX.cc) = 188$[19] and $card(LOV) - card(PREFIX.cc) = 133$[20] prefixes in LOV not yet registered in prefix.cc. At this point, a first batch of 80 prefixes/namespaces from LOV were safely imported in prefix.cc since there were no conflicts. For the remaining conflicting ones, they needed more in-depth analysis.

3.2 Second Task: Dealing with Conflicts between Prefix.cc and LOV

In the process of alignment, there were two types of conflicts and we provide appropriate actions and/or solutions accordingly:

- Clashes: cases where we have in both services the same prefix but different URIs;
- Disagreements on preferred namespace: cases where for the same URI, we found different prefixes.

Clashes. We performed a SPARQL query as above to identify clashes in vocabularies (30). In Table 2, we identify seven different types of issues to deal with, such as (i) real conflicts, (ii) URIs are 404, (iii) URIs are obsolete versions and (iv) two URIs redirecting to the same resource.

Disagreements on namespace URIs. The general idea is that if vocabulary editors have not included explicitly a `vann:preferredNamespacePrefix` in their description, the curators of LOV are free to change it and put whatever seems appropriate. At the same time, in prefix.cc, having multiple prefixes for the same namespace IRI in not a problem. However, we computed those prefixes in LOV that have different prefixes in prefix.cc. The following query retrieves the URIs falling in those disagreements:

[18] This query was performed in two weeks between March, 2nd and March, 20th 2013 and at this time, card(LOV) = 321 vocabularies while card(Prefix.cc) = 925

[19] http://www.eurecom.fr/~atemezin/iswc2013/experiments/firstAlignments/intersection-prefixLOV-02-03.csv

[20] http://www.eurecom.fr/~atemezin/iswc2013/experiments/firstAlignments/inLovNotINPrefixcc-02-03.csv

Type of issue	# Vocabularies	Percentage
pccURI and lovURI redirect to same resource	8	26.67%
lovURI already in prefix.cc as secondary	7	23.3%
Real conflicts	6	20%
pccURI is 404	4	13.3%
pccURI is an obsolete version	3	10%
lovURI is 404	1	3.3%
lovURI is an obsolete version	1	3.3%

Table 2. Type of issues encountered for vocabulary clashes

```
PREFIX vann: <http://purl.org/vocab/vann/>
SELECT ?prefix ?lovURI ?prefixcc
FROM <http://prefix.cc/popular/all.file.vann> {
  SERVICE <http://lov.okfn.org/endpoint/lov> {
    SELECT ?prefix ?lovURI {
      [] vann:preferredNamespacePrefix ?prefix;
        vann:preferredNamespaceUri ?lovURI;
    }
  }
  FILTER (?pccURI = ?lovURI && ?prefix != ?prefixcc)
  OPTIONAL {
    [] vann:preferredNamespacePrefix ?prefixcc;
      vann:preferredNamespaceUri ?pccURI;
  }
}
ORDER BY ?prefix
```

From the results of this query (61 cases), we have three actions to perform:

- add the lovPrefix (prefix in LOV) in prefix.cc (e.g: adding `geod:http://vocab.lenka.no/geo-deling#`) to the existing `ngeoi` in *pccPrefix*.)
- add more alternative URIs to the existing prefix in prefix.cc (e.g: adding `prov:http://purl.org/net/provenance/ns#`) to the existing `hartigprov, prv` in *pccPrefix*)
- change a prefix in LOV[21] (e.g: lovPrefix `dc` for `http://purl.org/dc/terms` not in the list $\{dcterm, dcq, dct, dcterms\}$ has been replaced by `dce` in LOV).
- No changes when the lovPrefix is contained in the set of prefixes of prefix.cc.

3.3 Social Aspects

Several vocabularies are maintained by a community of users. As part of the alignment process, we contacted the authors, creators or maintainers (if they

[21] `http://www.eurecom.fr/~atemezin/iswc2013/material/action-sameUriDifferentPrefixes.pdf`

exist) of vocabularies to involve them as well in the process of changing prefixes, and agree with them to fix some issues regarding their vocabularies. From the homepages of the vocabulary authors and editors collected in LOV, we connect to their social platform accounts such as LinkedIn, Google+ or Twitter. Table 3 summarizes some cases of real conflicts where the LOV curators have to find and contact the editors of the vocabularies for negotiation.

prefix	lovURI	Remark
sp	http://data.lirmm.fr/ontologies/sp#	contact editor at LIRMM ($sp \Rightarrow osp$)
scot	http://scot-project.net/scot/ns#	contact editors at lovURI
media	http://purl.org/media#	contact editors for negotiation
pro	http://purl.org/spar/pro/	contact editors for negotiation
swp	http://www.w3.org/2004/03/trix/swp-1/	contact editors, fix on LOV side
wo	http://purl.org/ontology/wo/core#	contact editors
idemo	http://rdf.insee.fr/def/demo#	to resolve with INSEE

Table 3. LOV and prefix.cc conflicts resolution leading to contact vocabularies editors for negotiation. We provide the prefix, the URI in LOV and the action undertaken.

4 Finding Vocabularies in Prefix.cc

We want to find out in prefix.cc, which of the couples (prefix, URI) could be potentially a vocabulary to be further assess to be included in the LOV catalog. To address this question, we first compute all the differences on prefix.cc NOT in LOV, i.e. $PREFIX.CC$ MINUS ($LOV < INTERSECT > PREFIX.CC$), performing using a SPARQL query. This results in 742 URIs to be checked[22].

4.1 LOV Check API

We have implemented an API[23] that allows a user to run the LOV-Bot over a distant vocabulary. It takes as parameter the vocabulary URI to process and the time out (integer) specified to stop the process. The result of this action is a set of 26 property-values from which we are interested in using only 8 of them, namely:

- **uri** (string) – uri of the vocabulary.
- **namespace** (string) – namespace of the vocabulary.
- **prefix** (string) – prefix of the vocabulary
- **inLOV** (boolean) – indicates if the vocabulary is already in the Linked Open Vocabularies ecosystem.

[22] http://www.eurecom.fr/~atemezin/iswc2013/experiments/input/notInLOV.json

[23] http://lov.okfn.org/dataset/lov/apidoc/

– **nbClasses** (int) – Number of classes defined in the vocabulary namespace.
– **nbProperties** (int) – Number of properties defined in the vocabulary namespace.
– **dateIssued** (string) – Vocabulary date of issue.
– **title** (Taxonomy) – List of titles with language information if available.

The code below gives the response of our algorithm for the vocabulary identified at `http://ns.aksw.org/Evolution/`.

```
[caption={Sample output of a response of the Check API}]
{
 "dateIssued": "None",
 "inLOV": false,
 "namespace": "http://www.agfa.com/w3c/2009/clinicalProcedure#",
 "nbClasses": 47,
 "nbProperties": 29,
 "pccURI": "http://www.agfa.com/w3c/2009/clinicalProcedure",
 "prefix": "clinproc",
 "title": [
   {
     "dataType": null,
     "language": "en",
     "value": "Clinical Procedure"
   }
 ],
 "uri": "http://www.agfa.com/w3c/2009/clinicalProcedure"
},
```

4.2 Experiments

We wrote a script calling the LOV Check API on the URIs in prefix.cc for determining the candidates vocabularies to be inserted in LOV using the algorithm in Listing 1. We ran four times the experiments (possibly due to some network instabilities) in order to determine from which results what should be assessed. Table 4 gives an overview of the number of URIs with respectively the attribute "inLOV=false"(TP), "inLOV=true"(FP) and the errors occurred (Null returned, http/proxy or time out reached by the API).

Regarding the experiments, `Experiment4` gives stable results with less network errors. Therefore, we stick on this experiment to report our findings and analysis. We found that 227 (43.48%) are vocabularies in the sense of LOV since they have at least one property or one class. 297 vocabularies (56.51%) might have some problems (or are even not vocabularies at all) as they have neither classes nor properties. Regarding the presence of prefixes, we found 140 (61.67%) of them. The 227 vocabularies could all be inserted in the LOV catalog since they fulfill the current requirements of what is a " LOV-able vocabulary". In this list, we found vocabularies such as `rdf, rdfs, owl` that are used to build other vocabularies but are not yet integrated in the LOV catalog.

	TP(inLOV=false)	FP(inLOV=true)	Errors
Experiment1	525	44	173
Experiment2	403	26	313
Experiment3	351	28	363
Experiment4	522	44	176

Table 4. Experiments looking for stable results of finding vocabularies in prefix.cc.

Algorithm 1 finding vocabularies NOT in LOV from prefix.cc algorithm

Open $notInLOV.jsonfile$ containing the prefix.cc URIs **not** in LOV
initialize *item* as List
Initialize *result* as collection of *item*
for each $pccURI \in notInLOVfile$ **do**
  $uri \leftarrow$ value of $pccURI$
  $uriv \leftarrow$ construct-valid uri
  call LOV-Check API with parameter $uriv$
  try/catch HTTPError, URLError, IOError, ValueError
  while no error raised **do**
    initialize *item* to an empty List
    append $pccURI, prefix, inLOV, namespace, title, dateIssued, nbClasses, nbProperties$ in *item* List
    append *item* to *result*
  end while
end for
print $output - result$

From the list of URIs that were not LOV-able vocabularies, we wanted to do more analysis by checking the RDF files using the Triple-Checker tool. Our aim is to be sure if we did not leave out some candidate vocabularies or if there are other type of errors such as parsing errors. Table 5 provides results classified into 4 categories:

- General errors such as loading files or proxy errors: 78.30%
- Candidate LOV-able vocabularies: 12.20%
- Clearly not vocabularies (`nbClasses` = `nbProperties` = 0), typically instances, datasets, html pages: 6.45%
- Others (mainly parsing errors): 3.05%

5 Conclusion

In this paper, we have analyzed numerous vocabularies referenced in LOV and in prefix.cc and we have presented a way to manage the prefixes of those vocabularies. We have shown that in the process of mapping namespaces with prefixes, some conflicts have to be resolved, often by contacting the editors themselves.

Total URIs	295	100%
Loading/404 errors	182	61.69%
Vocabularies	36	12.20%
Proxy errors	27	9.15%
50x, 40x errors	22	7.45%
Parsing errors	9	3.05%
Web Pages containers	9	3.05%
No triples found	8	2.71%
RDF data	2	0.67%

Table 5. Analysis of the URIs with no classes and no properties while using the LOV-Bot API

One future work is to develop a new strategy for the LOV-Bot API to take into account vocabularies published in other formats such as n3 and turtle. This would require to first test the validity of those formats and to adapt the way namespaces are obtained in order to not check only the presence of the `vann:preferredNamespace` property but to rely on similarity algorithm in order to guess the closest namespace given a URI vocabulary and some statistics of the number of classes and properties.

The work presented in this paper can be extended in several directions. Sticking to the two services we have studied and already contributed to harmonize, the possible next steps would be to automate as far as possible the tasks that have been made semi-automatically so far: *i)* developing a unique interface for submitting namespaces and prefixes to both services; *ii)* bridging the LOV back-office and the prefix-cc database using both services API in order to publish a list of common recommended prefixes. The latter goes beyond the limited framework of the two original services since such a list could be consolidated and endorsed by the main actors in vocabulary publication and management, and recommended for use in linked data applications. This could be picked up by the upcoming W3C Vocabulary Management Working Group as part of the new Data Activity[24].

This (apparently) simple issue of prefixes and namespaces is providing a good illustration of why some kind of governance is needed in the distributed ecosystem of vocabularies and linked data, pointing to both technical and social aspects, and proposing concrete examples of conflict resolution. There is no, and certainly there should never be any, central attribution authority for prefixes, and the needed regulation has to be made a posteriori, including good practices of cooperation and negotiation between vocabulary publishers. Development and harmonization of services such as LOV and prefix.cc is then to be considered as

[24] `http://www.w3.org/2013/05/odbp-charter.html`

part of the current and more general effort already started by the DCMI[25] and W3C[26] for a sustainable governance of vocabularies.

Acknowledgments

This work is partially supported by the project Datalift funded by the French Research Agency (ANR) under grant number ANR-10-CORD-009. The Linked Open Vocabularies initiative is hosted by the Open Knowledge Foundation. The authors are very grateful for the support and help of Richard Cyganiak, author and maintainer of the prefix.cc service.

References

1. K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. Describing linked datasets. In *2nd Workshop on Linked Data on the Web (LDOW)*, Madrid, Spain, 2009.
2. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
3. M. DÁquin and N. Noy. Where to publish and find ontologies? a survey of ontology libraries. *Web Semantics: Science, Services and Agents on the World Wide Web*, 11(0):96–111, 2012.
4. E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 Federated Query. W3C Recommendation, 2013. http://www.w3.org/TR/sparql11-federated-query/.
5. B. Vatant and P.-Y. Vandenbussche. Catalogue de Vocabulaires. Datalift, D2.2, 2013. http://datalift.org/en/node/18.

[25] Long-term Preservation and Governance of RDF Vocabularies: http://dcevents.dublincore.org/IntConf/index/pages/view/vocPres

[26] W3C Vocabulary Services: http://www.w3.org/2013/04/vocabs/

Towards Linked Data based Enterprise Information Integration

Philipp Frischmuth[1], Sören Auer[1], Sebastian Tramp[1], Jörg Unbehauen[1], Kai Holzweißig[2] and Carl-Martin Marquardt[2]

[1] Universität Leipzig, Institut für Informatik, AKSW,
{lastname}@informatik.uni-leipzig.de
[2] Enterprise Services Portal, CMS & Search, Daimler AG, Plant 096-0191, 70546 Stuttgart, Germany
{firstname.lastname}@daimler.com

Abstract. Data integration in large enterprises is a crucial but at the same time costly, long lasting and challenging problem. In the last decade, the prevalent data integration approaches were primarily based on XML, Web Services and Service Oriented Architectures (SOA). We argue that classic SOA architectures may be well-suited for transaction processing, however more efficient technologies can be employed for enterprise data integration. In particular, the use of the Linked Data paradigm appears to be a very promising approach. In this article we explore challenges large enterprises are still facing with regard to data integration. We discuss Linked Data approaches in these areas and present some examples of successful applications of the Linked Data principles in that context.

1 Introduction

Data integration in large enterprises is a crucial but at the same time costly, long lasting and challenging problem. While business-critical information is often already gathered in integrated information systems such as enterprise resource planning (ERP), customer relationship management (CRM) and supply chain management (SCM) systems, the integration of these systems itself as well as the integration with the abundance of other information sources is still a major challenge. In the last decade, the prevalent data integration approaches were primarily based on the Extensible Markup Language (XML), Web Services and Service Oriented Architectures (SOA) [6]. However, we become increasingly aware that these technologies are not sufficient to ultimately solve the data integration challenge in large enterprises. We argue that classic SOA architectures are well-suited for transaction processing, but more efficient technologies are available that can be deployed for solving the data integration challenge. With the use of the Linked Data paradigm for integrating enterprise information, data intranets can complement the intranets and SOA landscapes currently found in large enterprises.

In this paper, we explore the challenges large enterprises are still facing with regard to data integration. These include, but are not limited to, the development, management and interlinking of enterprise taxonomies, domain databases, wikis and other enterprise information sources. We discuss Linked Data approaches in these areas and present some examples of successful applications of the Linked Data principles in that context.

2 Data Integration Challenges in the Enterprise

We identified crucial areas where data integration challenges arise in large enterprises. In the following section we investigate those challenges, each by considering the current situation first. We then examine the benefits of employing Linked Data technologies in order to tackle the respective challenge. Finally, we describe the challenges that need to be addressed to make the transition from the current state of the art to the Linked Data approach feasible.

2.1 Enterprise Taxonomies

Nowadays, almost every large enterprise uses taxonomies to provide a shared linguistic model. It is widely agreed that taxonomies are usable, however, there are multiple challenges that must be addressed in order for taxonomies to work correctly [5]. A problem that arises is that different metadata creators use different terminologies and therefore the same object may receive different metadata descriptions by different people [4]. Another challenge is that large taxonomies require certain time for the users to get their bearings so that they can start to use the taxonomies correctly and avoid creating duplicities and other errors. In [15], the author discusses whether taxonomies are really necessary and stresses the importance of establishing relations to documents via URLs, which indicates already a clear shift towards the Linked Data vision.

If we take a look at commercial implementations, there is Microsoft SharePoint's[3] Term Store (also referred to as Managed Metadata), which enables enterprises using SharePoint to tag objects stored in SharePoint with terms from a taxonomy. However, there are some strong limitations to this approach. There is very restricted multilingual support – separate SharePoint language packs need to be installed for each language to be used in the taxonomy. Also, the implementation is proprietary, thus hindering the integration with taxonomies or data outside of SharePoint.

Linked Data Approach We propose to represent enterprise taxonomies in RDF employing the standardized and widely used SKOS [12] vocabulary as well as publishing term definitions via the Linked Data principles. This approach entails the following main benefits: (1) Since terms are attached to URIs, which can be dereferenced using HTTP, term definitions can be obtained without the need for additional software (a browser is sufficient). (2) For the same reason, term creation and management can be realized in a distributed scenario, where, for example, certain departments are responsible for different parts of the taxonomy. Terms can then be interlinked and re-used regardless of department boundaries. (3) By employing the SKOS vocabulary, terms can have a hierarchical order and thus the problem of different metadata granularity can be easily solved. (4) Also, since data is represented using RDF, which works with arbitrary vocabularies and properties, overlapping, coinciding or conflicting term definitions (e.g. by different departments) can be interlinked by explicitly stating the relationship between terms via links. (5) Terms can be assigned multiple labels, which are represented as RDF literals. As such they can be furnished with a language tag resulting in

[3] `http://sharepoint.microsoft.com/`

multilingual taxonomies with very little additional effort. (6) Ultimately the result of employing the Linked Data approach for enterprise taxonomies is, that terms can be easily re-used in other scenarios as the originally intended ones. For example, a detailed term definition (with alternative labels in multiple languages) can be very useful for search applications, where terms are associated with documents which then become reachable via a variety of keywords.

However, the initial costs for building a basic taxonomy may still be quite high. The problem of finding initial terms (and their definitions) can be solved by integrating a service like DBpedia Spotlight [11]. With this service texts can be annotated with resources from DBpedia [1], which already contain a detailed description in multiple languages in many cases. For enterprise-specific terms, where a description is not available via DBpedia, a keyword extraction service like FOX[4] can be used instead. Thus an initial set of term URIs can be gathered, which can then be annotated manually with for example a data wiki like OntoWiki (see subsection 2.2).

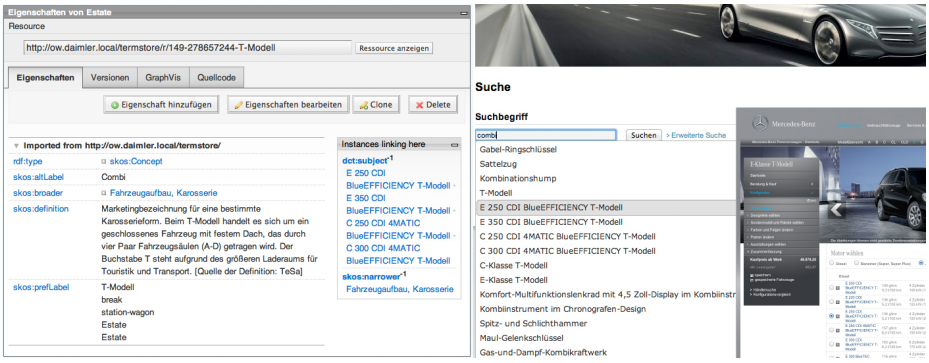

Fig. 1. The left side shows OntoWiki, which displays a term definition and resources linking to it. The right side shows a search application, which employs the term metadata for finding and suggesting relevant content.

Figure 1 shows two screenshots, which demonstrate some of the advantages described in this section. The left side shows OntoWiki, which displays the definition of the term *T-Modell* along with some additional information. The location bar on the top of the screen displays the URI used for this very concept, which other resources can link to. It is also possible to directly de-reference this identifier and obtain the description for this resource in a machine-readable format. A dedicated list shows other resources that link to this concept, in this case certain car models. This circumstance is used in a search application, which is shown on the right side of Figure 1. When a user types the keyword *combi*, the knowledge base is used to obtain the fact, that this search term is a synonym for the concept *T-Modell*. Once this is done, all linked car models are retrieved

[4] `http://aksw.org/Projects/FOX`

and shown to the user. The depicted scenario is a good example of an application of a taxonomy outside the scope of the originally intended use. One main reason for the efficient realization of this application is that data from multiple sources (term store and car configuration metadata) was made available via the Linked Data principles.

Challenges Currently terminology in large enterprises is managed in a centralized manner mostly by a dedicated and independently acting department, which is in charge to standardize all corporate terms. As a result they create a variety of dictionary files for different scopes that are not interconnected. An employee that wants to look up a certain term, needs to know which dictionary to use in that very context, as well as where to retrieve the currently approved version of it. The main challenge in the area of enterprise taxonomies is defragmentation of term definitions without centralization of taxonomy management.

Defragmentation. The proposed Linked Data approach can be implemented by keeping the centralized structure and assign the corporate language management (CLM) department the task to create a knowledge base that contains the complete terminology of the company. This solves the fragmentation problem occurring with the dictionary approach, but it also keeps the barrier for participation high, since a single department still is in charge for maintaining the knowledge base.

Decentralization. On the other hand an entire decentralized solution can be implemented, by assigning each department in the enterprise it's own taxonomy namespace. Adding new terms or refactoring existing terms becomes very easy with this approach, due to the reduced communication overhead with other departments. Nevertheless the problem of fragmentation arises again.

2.2 Wikis

Wikis have become increasingly common through the last years reaching from small personal wikis to the largest Internet encyclopedia Wikipedia. Since large companies have special requirements, such as fine-grained access-control, enterprise scalability, security integration and the like, a special type of wikis – enterprise wikis – emerged. A survey [10] about the use of corporate wikis pointed out, that corporate users expect three benefits from using wikis, namely improved reputation, relaxation of work and helping in the advancement of processes. Widely utilized wikis in the enterprise context are Confluence[5] and Jive[6]. Popular open-source wikis include FOS wiki[7] and TWiki[8]. These tools differ in their provided functionality, but they are all mainly centered around textual content, although some wikis provide limited support for managing structured information (e.g. FOS wiki via data forms). Consequently, the knowledge contained in those wikis can in most cases only be extracted by human reading of the documents and not by other applications used within the company.

[5] `http://www.atlassian.com/software/confluence/overview`
[6] `http://www.jivesoftware.com/social-business/platform`
[7] `http://foswiki.org/`
[8] `http://twiki.org/`

Linked Data Approach In addition to traditional wikis, there is also another category of wikis which are called semantic wikis. Those can again be divided into two categories: semantic text wikis and semantic data wikis. Wikis of this kind are not yet commonly used in enterprises, but crucial for enterprise data integration since they make (at least some of) the information contained in a wiki machine-accessible. Text-based semantic wikis are conventional wikis (where text is still the main content type), which allow users to add some semantic annotations to the texts (e.g. typed links). The semantically enriched content can then be used within the wiki itself (e.g. for dynamically created wiki pages) or can be queried, when the structured data is stored in a separate data store. An example is Semantic MediaWiki [9] and its enterprise counterpart SMW+. It extends the well-known MediaWiki engine (which powers Wikipedia) with syntax for typecasting links and data, classifying articles and creating dynamic pages. The knowledge in a wiki (KiWi) [14] project also developed a semantic wiki, which provides an adaptable platform for building semantic and social tools.

We propose the usage of semantic data wikis such as OntoWiki [2,8] in enterprises for the following main reasons: (1) Data wikis focus on structured information, which is kept as such and thus can be easily re-used by other applications consuming the data. (2) Since OntoWiki is solely based on RDF, all information is automatically published via the Linked Data principles, making it trivial for other parties to consume the data. (3) Information fragments can be interlinked with other resources within an enterprise (e.g. taxonomies, XML schemas, databases, Web services), which leads to a better reuse of information and thus to better maintained data. (4) Since textual information can also be represented in RDF (via literals), text wikis can be emulated and thus (additional) human-friendly information can be added. Such annotations and the structured information can then be used to create customized views on the data.

Challenges A challenge is to train users of wikis to actually create semantically enriched information. For example, the value of a fact can be either represented as a plain literal, or as a relation to another information resource (eventually already attached with some metadata). The more users are urged to reuse information wherever appropriate, the more all participants can benefit from the data. It should be part of the design of the wiki application (especially the user interface), to make it easy for users to build quality knowledge bases (e.g. through auto-suggestion of URIs within authoring widgets).

Since data in RDF is represented in the form of simple statements, information that naturally is intended to be stored in conjunction (e.g. geographic coordinates) is not visible as such per se. The same applies for information which users are accustomed to edit in a certain order (e.g. address data). A non-rational editing workflow, where the end-users are confronted with a random list of property values may result in invalid or incomplete information. The challenge here is to develop a *choreography of authoring widgets* in order to provide users with a more logical editing workflow.

Another defiance to tackle is to make the deployed wiki systems available to as many stakeholders as possible (i.e. cross department boundaries) to allow for an improved information re-use. Once Linked Data resources and potentially attached information are re-used (e.g. by importing such data), it becomes crucial to keep them in sync with the original source. Therefore mechanisms for *syndication* (i.e. propagation

of changes) and *synchronization* need to be developed, both for intra- and extranet semantic wiki resources.

2.3 Web Portal and Intranet Search

Current state-of-the-art intranet data management system with proper full text search and a comfortable user interface include Microsoft's FAST Search[9], SAP's Netweaver Enterprise Search[10] or Autonomy's IDOL Universal Search[11]. These search engines are based on full-text search and offer taxonomy support, custom ranking and context awareness. Even though the search engines are quite sophisticated, there is still a lot of room for improvement that can be tackled by publishing the data as Linked Data and allowing it to be queried as such [5]. In [7], the author identifies several challenges in enterprise search, one of them being internal multi-source search, which is when a user has a precisely formulated question but only a keyword search is available. The author uses an example where a manager in an oil company wants to identify all wells previously drilled by the company in the Black Gold field where the problem known as "stuck pipe" was experienced. The manager searches for "Black Gold stuck pipe", but he must go through all the found documents, identifying the wells and so on. If the company data was stored or internally published as Linked Data using an ontology describing the oil drilling domain, the result could be gained using a single SPARQL query.

Linked Data Approach In an enterprise exist at least two distinct areas where search technology needs to be applied. On the one hand, there is corporate internal search, which enables employees to find relevant information required for their work. On the other hand, all large enterprises need at least simple search capabilities on their public web portal(s), since otherwise the huge amounts of information provided may not be reachable for potential customers. Some dedicated companies (e.g. automotive companies) would actually have a need for more sophisticated query capabilities, since the complexity of offered products is very high. Nevertheless, in reality, search, both internal and external, is often solely based on keyword matching. We argue that by employing the Linked Data paradigm in enterprises the classical keyword based search can be enhanced. Additionally, more sophisticated search mechanisms can be easily realized since more information is available in a uniform and machine-processable format.

In cooperation with Daimler a prototype that employs multiple Linked Data sources to provide a uniform search application was developed. By entering simple keywords users can (a) find documents that are attached to terms from the taxonomy that match the given query or (b) find specific car models that match the criteria given by the user (e.g. more than 6 seats). In the first case the advantage is, that documents can be found

[9] `http://sharepoint.microsoft.com/en-us/product/capabilities/search/Pages/Fast-Search.aspx`

[10] `http://www.sap.com/platform/netweaver/components/enterprisesearch/index.epx`

[11] `http://www.autonomy.com/content/Products/idol-universal-search/index.en.html`

even if the content does not mention the keyword. The second case would not be even possible without taking another datasource into account, namely structured information about possible car configurations. Thus, when a user queries for a keyword that matches a term that is linked to a car related property and also provides a value restriction (e.g. less than 10), the system can obtain a list of matching cars (via SPARQL queries) and return them to the user together with some metadata about the models.

Challenges In order to implement search systems that are based on a Linked Data approach and that provide a substantial benefit in comparison with traditional search applications, the challenge of *bootstrapping an initial set of high-quality RDF datasources* needs to be tackled first. Mechanisms then need to be established to automatically create high-quality links between datasets.

Finally, although a search engine that queries RDF data directly works (in fact the prototype described above was implemented using this approach), it results in suboptimal performance. The challenge here is to develop methods for improving performance to match traditional search engines, while keeping the advantages of using SPARQL directly.

2.4 Database Integration

Relational Database Management Systems (RDBMS) are the predominant mode of data storage in the enterprise context. We therefore deem the integration of relation data into Linked Data a crucial Enterprise Data Integration technique. For providing a unified view over different databases multiple methods like data warehousing, schema mediation and query federation have been devised and successfully used. However, problems arise with more heterogeneous data landscapes, where strict schema adherence can not be guaranteed and external data is utilized. The integration of heterogeneous sources requires a costly transformation of the data into the relational model. This has the effect, that only key data sources and thus only a small fraction of the RDBMSes in a typical enterprise are integrated.

Linked Data Approach The mapping of relational data to the RDF data model adopts relational database integration techniques and augments them. By employing a mapping from relational data to RDF, data can be integrated into an internal or external data cloud. By using URIs for identifying resources, integration with non-relational and external data is facilitated. The RDB to RDF Mapping Language (R2RML) standard describes how a relational database can be transformed into RDF by means of term maps and triple maps. In order to avoid a costly materialization step, R2RML implementations can dynamically map an input SPARQL query into a corresponding SQL query, which renders exactly the same results as the SPARQL query being executed against a materialized RDF dump. By avoiding a costly materialization of the relational data into a dedicated triple store, a light-weight integration into existing architectures is possible. Consequently, semantic wikis, query federation tools and interlinking tools can work with the data of relation databases. The usage of SPARQL 1.1 query federation [13] allows relational databases to be integrated into query federation systems with queries spanning over multiple databases.

Challenges A primary concern when integrating relational data is *scalability and query performance*. With our R2RML based tool SparqlMap[12] we show that an efficient query translation is possible, thus avoiding the higher deployment costs associated with the data duplication inherent in ETL approaches. The challenge of closing the gap between triple stores and relational databases is also present in SPARQL-to-SQL mappers and drives research. The standardization of the RDB to RDF Mapping Language (R2RML) by the W3C RDB2RDF Working Group establishes a common ground for an interoperable ecosystem of tools. However, there is a lack of mature tools for the creation and application of R2RML mappings. A challenge lies in the creation of user friendly interfaces and establish best practices for creating, integrating and maintaining those mappings. Finally, for a *read-write integration* updates on the mapped data need to be propagated back into the underlying RDBMS. An initial solution is presented in [3]. In the context of Enterprise Data an integration with *granular access control* mechanisms is of vital importance.

3 Conclusions

In this work we identified several data integration challenges that arise in corporate environments. We discussed the use of Linked Data technologies in those contexts and presented some insights gained during the development of corresponding prototypes for Daimler. We conclude from our experiments, that the deployment of Linked Data approaches in enterprise scenarios has huge potential and can result in extensive benefits. However, we are aware that more challenges than the aforementioned need to be tackled when trying to create sophisticated enterprise knowledge intranets. We consider as future work to investigate XML schema governance (due to the numerous applications that employ XML for information exchange) and enterprise single sign-on from a Linked Data perspective. In order to ultimately establish Linked Data as a strategy for enterprise data integration also many organizational challenges have to be tackled. For example, it is relatively easy to determine the return-on-investment for an integration of two information systems, while it is very difficult to precisely assess the cost savings of the Linked Data approach. Also, the added value of the Linked Data approach might only become visible after a critical mass of Linked Data interfaces and resources are already established in the enterprise.

References

1. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. *The Semantic Web*, pages 722–735, 2007.
2. S. Auer, S. Dietzold, and T. Riechert. OntoWiki - A Tool for Social, Semantic Collaboration. volume 4273, pages 736–749. 2006.
3. V. Eisenberg and Y. Kanza. D2RQ/update: updating relational data via virtual RDF. In *WWW (Companion Volume)*, pages 497–498, 2012.
4. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *ACM*, pages 964–971, Nov. 1987.

[12] `http://askw.org/Projects/SparqlMap`

5. J. Grudin. Enterprise Knowledge Management and Emerging Technologies. In *System Sciences, 2006. HICSS '06*, volume 3, page 57a, 2006.
6. A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In *VLDB '06*, pages 9–16. VLDB Endowment, 2006.
7. D. Hawking. Challenges in enterprise search. In *ADC '04*, ADC '04, pages 15–24, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
8. N. Heino, S. Dietzold, M. Martin, and S. Auer. *Developing Semantic Web Applications with the OntoWiki Framework*. Networked Knowledge. Springer, 2009.
9. M. Krötzsch, D. Vrandečić, and M. Völkel. Semantic MediaWiki. *The Semantic Web-ISWC 2006*, pages 935–942, 2006.
10. A. Majchrzak, C. Wagner, and D. Yates. Corporate wiki users: results of a survey. In *WikiSym '06*. ACM, Aug. 2006.
11. P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. DBpedia spotlight: shedding light on the web of documents. In *I-Semantics '11*, pages 1–8. ACM, 2011.
12. A. Miles and S. Bechhofer. SKOS Simple Knowledge Organization System Reference. *W3C Recommendation*, 2008.
13. E. Prud'hommeaux. SPARQL 1.1 Federation Extensions, November 2011. `http://www.w3.org/TR/sparql11-federated-query/`.
14. S. Schaffert, J. Eder, S. Grünwald, T. Kurz, and M. Radulescu. Kiwi–a platform for semantic social software (demo). *The Semantic Web*, pages 888–892, 2009.
15. C. Shirky. Ontology is overrated: Categories, links, and tags, 2005.

Adaptive Semantic Publishing

Georgi Georgiev, Borislav Popov, Petya Osenova, Marin Dimitrov

Ontotext AD, Bulgaria
{borislav.popov, georgiev, petya.osenova,
marin.dimitrov}@ontotext.com

Abstract. The paper describes the approach, methodology and main software components of an Adaptive Semantic Publishing Platform for digital medias; applied previously to numerous use cases and publishers like the BBC, Euro-Money and Press Association. The semantic publishing relies on the interaction among the common sense model in ontologies, the world knowledge in Linked Open Data (LOD), the named entity categorization and the set of domain-specific keywords. Hence, the contribution of the related LOD datasets is briefly considered. The adaptive publishing relies on the user's requirements (interests, searches, activities) provided as summaries of articles on selected topics (sports, politics, society, etc.). Also, approaches to gold standard data are presented, which enable the fast and high quality clusterization of numerous information streams per topic.

Keywords: Semantic Publishing, Personalization, Clustering, Ontologies, Linked Open Data, Summarization

Introduction

In recent years Semantic publishing applications get more and more user-oriented in several aspects, among which: customization and re-purpose of data and content reflecting the user needs; focused summaries with respect to user interests; high relevance of the retrieved information and minimal effort in receiving it.

There are various works, exploring the relation between publishing and Linked Open Data. In [4], for example, authors present their idea on a life cycle model (specification, modeling, generation, linking, publication, exploitation) and demonstrate its application within various domains. At the same time, in [3] a DBpedia service has been presented (called DBpedia Spotlight), which automatically annotates text documents with DBpedia URI's using the DBpedia in-house ontology. Similarly, Zemanta[1] provides a plug-in to content creators, which recommends links to relevant content (articles, keywords, tags). Our approach is generally in-line with these ideas and services – domain specific applications, automatic semantic annotation, adding relevant linked content. However, our focus is preferably on: the trade-off between the seman-

[1] http://en.wikipedia.org/wiki/Zemanta

tic knowledge holders (ontologies, linked data) and their language reflection (domain texts), mediated by the linguistic processing pipelines; the adaptive flexibility of the constructed applications and the efficient storage and publishing of large data.

Within Ontotext, examples of mass media, semantic publishing web sites, such as the BBC's sport web[2] and the official web of the London's Olympics 2013, have proven to attract a multi-million user bases. Behind such applications, as revealed by lead engineers at the BBC[3], there lies the complex architecture of the state-of-the-art Semantic and Text Analytics technologies, such as in-house: fast RDF database management system OWLIM[4] and knowledge management platforms KIM[5]; for robust semantic annotation and search, as well as for text analytics applications.

Both platforms are incorporated into numerous successful Semantic Publishing Solutions (including the BBC Sport[6], Press Association[7], Newz[8], EuroMoney[9], Publicis[10] etc.). This paper aims to describe the approach, main software components, information architecture, text analytics and semantic annotation and indexing, used successfully in many solutions for more than 5 years, to build semantic publishing solutions.

Our approach relies on the calibration between the RDF semantic repository OWLIM, the semantic resources in KIM and the optimized Text Analytics techniques including methodologies for fast creation of gold data in the selected domain; focused curation of the automatically analyzed data and the application of advanced machine learning algorithms in data clustering. Thus, the success of our solutions lies in the customization of the advanced semantic technologies in combination with text analytics techniques, tuned to the needs of publishers and adapted to the requested domains.

The Overall Architecture of Semantic Publishing System

Our generalized system, presented on Fig. 1 below, comprises several components, connected in a life cycle. The Content Store on the left side contains the news articles, along with the associated pictures and videos. The textual content of these assets is then passed to the Text Processing Pipeline, implemented in GATE[11]. The pipeline includes various components. The generic components refer to: tokenization, POS tagging, chunking. The Gazetteer refers to named entity recognition, such as Person, Location, Organization, etc. The Rules are written in JAPE[12] style. They

[2] www.bbc.com/sport
[3] www.bbc.co.uk/blogs/bbcinternet/2012/04/sports_dynamic_semantic.html
[4] www.ontotext.com/owlim
[5] http://www.ontotext.com/kim
[6] http://www.ontotext.com/publishing
[7] http://www.pressassociation.com/
[8] newz.nl
[9] http://www.euromoney.com/
[10] http://www.publicis.de/
[11] http://gate.ac.uk/
[12] http://gate.ac.uk/sale/tao/splitch8.html#chap:jape

usually cover the relation extraction, such as Person works for Organization; Organization is located in Location, etc. Machine learning component scales up the application when there is manually annotated training data. Semantic indexing refers to URIs, which map the detected entities with real objects in the world. For example, Obama is recognized as Person, but then, more information about him is provided through a link to Obama's DBPedia[13] profile. Geo localization of the articles relies on automatic association with GeoNames[14] map coordinates. This means that the recognized country is assigned with its longitude and latitude information from the interlinked map.

Additionally, some other knowledge is provided, such as capital, currency, etc. In this way the facts are available in interconnected knowledge maps. The ontology abstracts over the text chunks and specifies the categories and relations within linked data. We initially rely on common sense ontology (such as PROTON), which might be further extended with the necessary domain knowledge depending on the required conceptualization granularity. Given the common ontology and the data, the extension is trivial and easy.

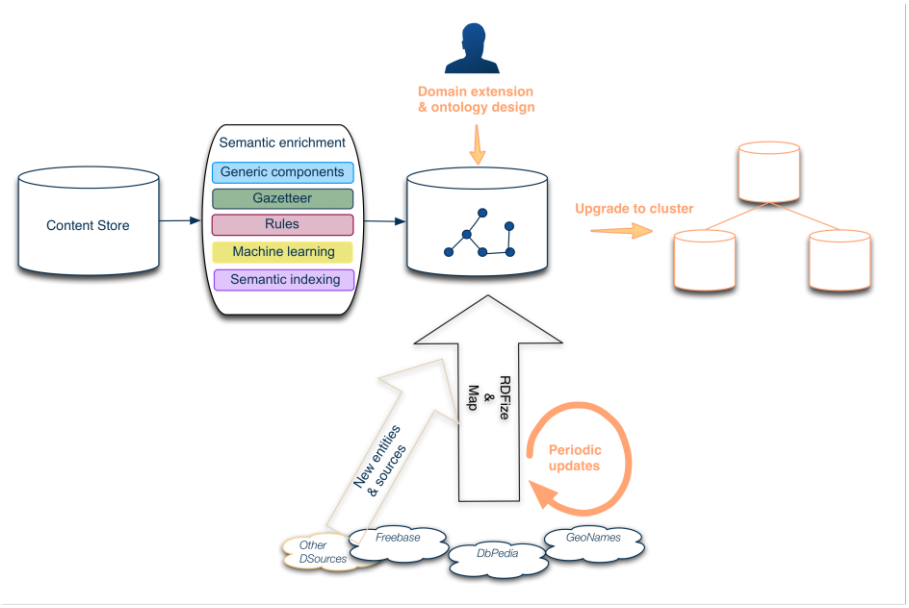

Fig. 1. Generalized Semantic Publishing Architecture.

[13] http://dbpedia.org/About
[14] http://www.geonames.org/

In Fig. 2 below the Basic Information Architecture is shown. The Domain Extension and Ontology design process include modeling of the specific domain (finance, sports, etc.) in domain ontology, connected to the upper-level PROTON ontology[15]. At the same time, the processed data is mapped to Linked Open Data. Linked Open Data also provides updates on named entities.

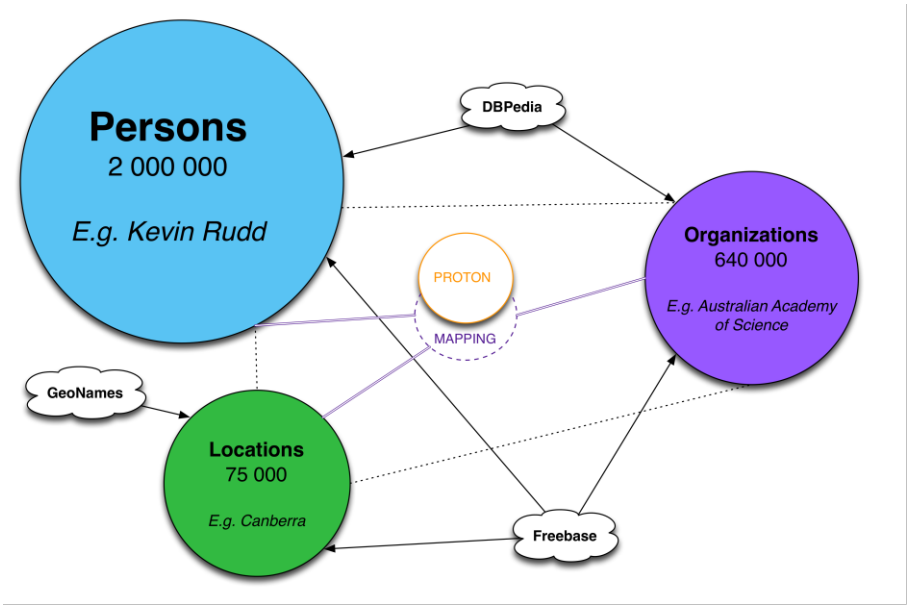

Fig. 2. Basic Information Architecture. The arrowed lines show what types of concepts are present in the specific linked resource (for example, GeoNames have only Locations). The dotted lines show that there might be various relations among Persons, Locations, Organizations (such as, Person lives-in Location). All the instances from linked data are mapped to a common ontology (PROTON).

The interconnected data then goes into clustering models for focused categorization with respect to the task. We work in a hierarchical way of knowledge recognition – first, we try to detect a referent in LOD (i.e., that Obama is a Person with features like birthday, birthplace; LivesIn; is_a president of the USA in the appointed time periods, etc.); if there is no such connection, then we aim at recognizing the more abstract concept in the Ontology (*Person with a Role in Society*, and with other appropriate features); if this information is also not available for some reason, then the named-entity is detected (*Obama is Person*); if this is not possible either, then the most important key phrases are recognized (the terminology in the domain, etc.); last, but not least, topics of documents can be categorized (Sports, Society, Politics,

[15] proton.semanticweb.org

Finance, etc.). In most of our use cases, however, all these types of recognition are available and connected to each other. Thus, clustering of data is based on detailed semantic annotations.

The recognized knowledge pieces (referents, concepts, entities, etc.) can be also cross-related. For example, the categorization of entities in Sports, will provide groups of Persons, involved in Sports activities, Types of Sports, etc. On the other hand, information might be given about most popular sports per country or about Sports careers of Persons in various Organizations.

The Semantic Annotation Solution

The Underlying Datasets.

The OWLIM repository uses the following datasets: Freebase (version jan_9_2012), DBpedia (version 3.8) and subset of Geonames. From Freebase and DBpedia all the people and organizations are integrated. The subset of Geonames includes bigger cities, countries, continents, oceans, seas, US states, selected places in Europe. All these linked open data sets have their own ontologies, which are also part of the repository. All the above-mentioned ontologies are mapped to the common sense upper-level PROTON ontology. The added value of such a mapping is that a high quality reasoning and consistency of the modeled knowledge is ensured.

Since there is duplicated information in the three data sets, which, however, is presented in different ways, a mapping effort has been performed also between Freebase and DBpedia; Geonames and DBpedia. Thus, via the mappings of two bases to DBpedia, Freebase has also its mapping to Geonames.

Gold Standard Annotation and Curation.

The gold standard annotation includes the following steps: understanding the task; preparation of annotation guidelines; manual annotation of some texts with pre-selected tags; training an algorithm over gold data; automatic annotation of big datasets; curation of the processed documents; re-training (if necessary). For more details see [1] and [2].

The curated documents are used in turn as a bigger gold standard corpus for automatic text analysis evaluation, but also for training further machine learning models over the data.

The level of granularity for the creation of Annotation Types and their Features is based on more concrete classes for Person, Organisation, Location of the PROTON Ontology, with extension of classes and properties from linked open resources, such as DBpedia, Freebase and GeoNames. Modularization depends also on the specific task. In the case of media publishing, it detects Person, Organization and Location and maps them to the descriptions in Linked Open Data sources. But it also respects the domains, in which the named entities occur (Sports, Politics, Economics, etc.)

The document curation subsumes three related tasks:

- *Instance disambiguation*. It handles cases, such as ambiguities between various people with the same name or various locations with the same name, or even

person, location and/or organization with the same name. Since such properties are very context-dependent, the possible true candidates are verified by assigning the correct URL from Linked Open Data. For example, the name 'Washington' is ambiguous, since it refers to a *Politician*, an *Actor*, a *US state* and a *US city*.

- *Tagging*. This step is applied only for **Person**, **Organisation** and **Location**. The labels **Person**, **Organisation** or **Location** assigned during the automated annotation stage are verified and new labels are added to specific words or phrases in the text. This step prevents from considering a *Person* as a *Location*, or vice versa. The step is very important, since such mis-tagged entities might not be many, but might be very frequent in the domain. On Fig 2 above the complex relations are given among named entities, such as *Persons, Organizations* and *Locations*, etc. and their Linked Open Data mappings.
- *Topic or key words correction*. This step is applied when a document is miscategorized for a topic (for example *Society* instead of *Finance*) or when some phrases are detected which do not belong to the terminology of the domain.

In Fig. 3 below the activities cycle behind the Basic Semantic Annotation Process is shown.

From the perspective of the Agent, it contains the following actions. The reports or articles, provided by the client (the person on the top), are carefully examined by the company specialists (SME – small medium enterprise).

Then some probe annotations are performed for clarifying and defining the required annotation types. Then bigger chunks of data are annotated by annotators while keeping high interannotator agreement and providing checks by a superannotator.

When the data has been automatically annotated via the trained manually produced data, then also the curator is involved.

He/she chooses the correct mappings from a set of available mappings. Very often the set contains ambiguous mappings, which have to be resolved.

He/she also adds new links or deletes some, if needed. Simultaneously, the information from Linked Open Data store is mapped to the semantic annotations (both manual and automatic).

These mappings need a careful curation for achieving a high precision and recall. The person on the right indicates the client, who might examine the annotated data before its linking to the open data and its storage. This involvement is optional.

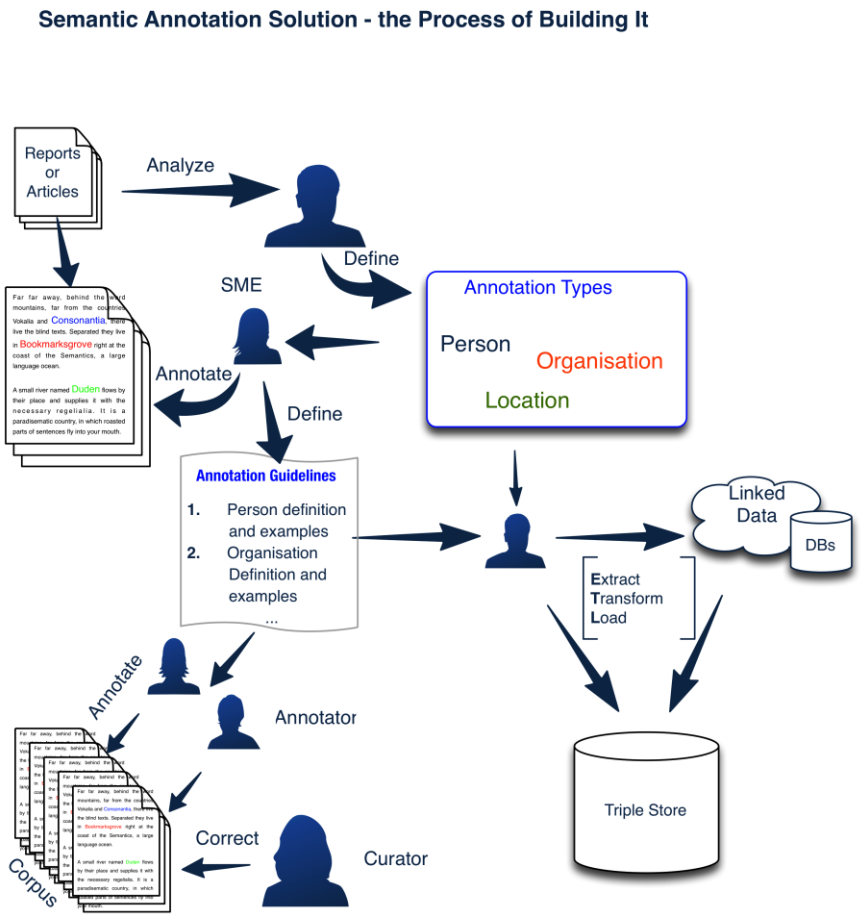

Fig. 3. Basic Semantic Annotation Architecture. The persons indicate human intervention in the automatic process.

Personal Semantic Publishing

The Semantic Publishing Platform provides the following adaptive instruments: parallel streams of excerpts from topic-related articles (trending); personalized user area (profile); search by topic-related keywords and focused summary. Let us consider each service separately.

Trending provides parallel streams of articles on various topics. The user can get oriented within the variety of topics, can select the ones of interest to him and explore them further (see Fig. 4).

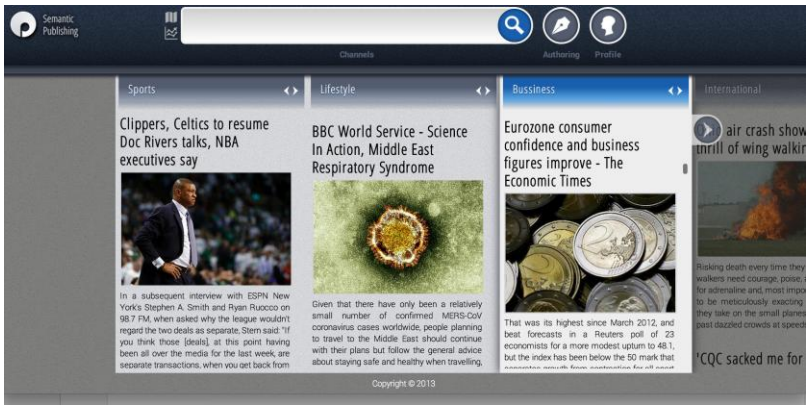

Fig. 4. Trending: parallel news streams.

The personalized user area allows the user to mark the articles and summaries that are of interest to him, to tag them with pre-selected tags and to store them for further usage. Additionally, the user can see related articles to the selected topic (see Fig. 5).

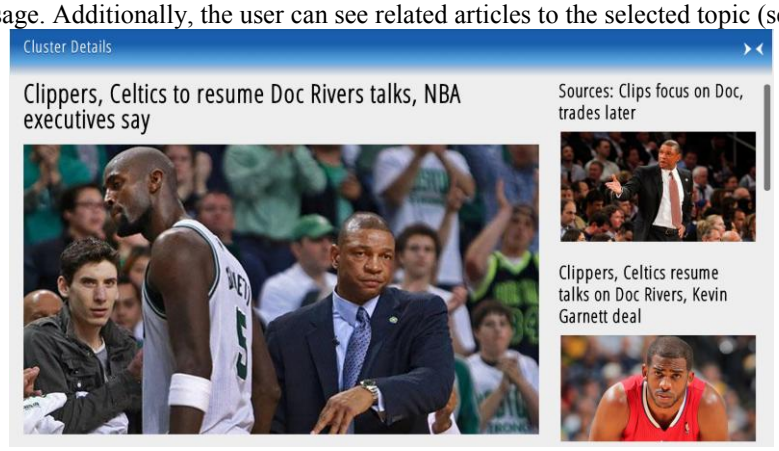

Fig. 5. Showing similar content to the chosen article.

The user can search information through sequencing of as many keywords and concepts as necessary for constraining the requested topic. Also, an autocomplete search is added as a facility (see Fig. 6).

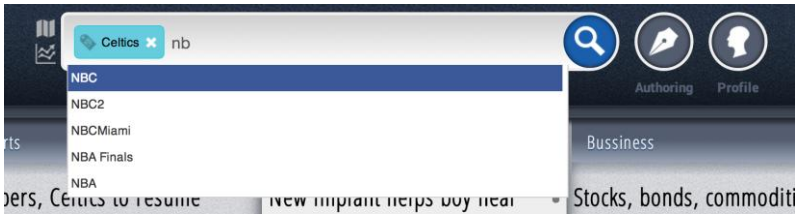

Fig. 6. Showing autocomplete options.

The user can get a summary of the topic, customized from related articles and built on the clustered data. Additionally, he/she receives the most frequent named entities and keywords in a rated list (see Fig. 7).

Fig. 7. Showing summary and related keywords.

Conclusions

The current work reveals a platform and methodology for development of Adaptive Semantic Publishing solution, applied previously in many use cases such as the BBC Sport web site. A concrete solution is described in terms of methodology and main software components and is publicly available as demonstration software.

The main user interface components: faceted search, trending, term and word search as well as channel's customization are also described with examples. The implementation of the user interfaces for personalization/profile and authoring are main areas of future work.

References

1. Kiryakov et. al 2003: Atanas Kiryakov, Borislav Popov, Damyan Ognyanoff, Dimitar Manov, Angel Kirilov, Miroslav Goranov. Semantic Annotation, Indexing, and Retrieval. 2nd International Semantic Web Conference (ISWC2003), 20-23 October 2003, Florida, USA. LNAI Vol. 2870, pp. 484-499, Springer-Verlag Berlin Heidelberg 2003.
2. Popov et al. 2003: Borislav Popov, Atanas Kiryakov, Angel Kirilov, Dimitar Manov, Damyan Ognyanoff, Miroslav Goranov. KIM – Semantic Annotation Platform,
3. 2nd International Semantic Web Conference (ISWC2003), 20-23 October 2003, Florida, USA. LNAI Vol. 2870, pp. 834-849, Springer-Verlag Berlin Heidelberg 2003.

4. Mendes et. al 2011: Pablo N. Mendes, Max Jakob, Andres Garcia-Silva and Christian Bizer. DBpedia spotlight: shedding light on the web of documents. In: Proceedings of the 7th International Conference on Semantic Systems, pp. 1-8, ACM, New York, NY, USA.
5. Villazon-Terrazas et. al 2012: Boris Villazon-Terrazas, Daniel Vila-Suero, Daniel Garijo, Luis M. Vilches-Blazquez, Maria Poveda-Villalon, Jose Mora, Oscar Corcho, and Asuncion Gomez-Perez. Publishing Linked Data - There is no One-Size-Fits-All Formula. In: Proceedings of the European Data Forum 2012, Copenague, Dinamarca.