



Scalability Solutions for the Web of Data

Michele Catasta

DERI - Digital Enterprise Research Institute

Supervisor: *Dr. Giovanni Tummarello*

Head of Department: *Prof. Dr. Stefan Decker*

July 17, 2009

A thesis submitted to the National University of Ireland, Galway
for the degree of M.Sc. in Applied Computing and Information Technology

The studies presented in this thesis were performed at the Digital Enterprise Research Institute at the National University of Ireland, Galway. The research was financially supported in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2) and in part by the FP7 EU Large-scale Integrating Project OKKAM - Enabling a Web of Entities (contract no. ICT-215032).

Contents

1	Introduction	1
1.1	Problem statement	2
1.2	Hypothesis	4
1.3	Thesis structure	4
1.4	Thesis contribution	5
2	Data Model	6
2.1	Definition and Model for the Web of Data	6
3	A Scalable Architecture for Semantic-Data Processing	9
3.1	Acquisition	9
3.2	Pre-processing	11
3.3	Services	12
3.4	Post-processing	13
4	Pre-Processing Terabytes of Web Semantics Data in Sindice	14
4.1	Semantic Sitemap: a scalable way to publish and split large-datasets . . .	14
4.2	Context-Dependent Reasoning	18
5	SIREn: a Semantic Information Retrieval Engine for the Web of Data	25
5.1	Introduction	25
5.2	SIREn Data Model	26
5.3	SIREn Query Model	27
5.4	Experimental Benchmark	30
5.5	Conclusion	33
6	DING: A Dataset-Centric Approach to Ranking	34
6.1	Introduction	34
6.2	A Two-Layer Model for Web Data	35
6.3	DING Algorithm	36

6.4	Example	38
6.5	Conclusion	38
7	A Real-World Use Case: Sig.ma	39
7.1	Introduction	39
7.2	Sig.ma: processing dataflow	40
7.3	Test driving Sig.ma: example user interactions	50
7.4	Related works and comparative evaluation	52
7.5	Sig.ma implementation and performances	57
7.6	Conclusion	58
8	Conclusions	59
8.1	Future Works	60
A	The Apache Hadoop Project	61
B	Apache Lucene and Solr	63
B.1	Apache Lucene	63
B.2	Apache Solr	64
	Bibliography	67
	List of figures	72
	List of tables	73

“To Isabella and Susanna”

Abstract

The amount of Resource Description Framework (*RDF*) documents and *Microformats* available online has grown tremendously in the past years. The Linked Open Data community has made available several billion triples equivalent of information, driven by the idea of open access to structured data. Furthermore, an increasing number of relevant Web 2.0 players (Technorati, LinkedIn, Yahoo! Locals, Last.fm, Digg, Youtube, Wordpress to name just a few) have added some form of structured data markups.

Precise measurements of this growth are not available, but partial reports combined with estimates allow us to give an educated guess of the current size of the *Web of Data*: 4-7 billions of RDF triples as well as half a billion of marked-up webpages, totaling several billion triples equivalent. The Web of Data has surpassed the critical mass which requires ad-hoc scalable services for finding relevant resources, similarly to what happened more than a decade ago with the birth of the first Web search engines.

The approaches followed by big players like Yahoo! (with the Searchmonkey project) and Google (with the support for RDFa structured snippets) are not harnessing the full potential of a structured Web. Our hypothesis is that the solutions we devised are a novel step towards a pragmatic Web of Data, supporting its vision of interoperability and simple data consolidation.

This thesis presents theoretical and pragmatic solutions to build scalable services fully exploiting the Web of Data. We address both *data discovery* and *data navigation* areas, looking at all the various aspects which should be taken into consideration when designing such systems – including optimized indices, scalable pre-processing, ranking, data consolidation and engineering issues.

Acknowledgements

*“Meglio del perdersi in fondo all’immobile
Meglio del sentirsi forti nel labile...”*
— Cristiano Godano (*Lieve*)

The journey which led me to this thesis was long and definitely not straightforward. I studied for my M.Sc. in one university (“*Sapienza*” in Rome, Italy) while I submitted my final thesis in another one (National University of Ireland, Galway). I worked in 3 different countries (Italy, Ireland, Spain) for various companies. I lived in at least 8 different flats and I had not less than 30 different flatmates. I switched my own time zone from CET to PST for various weeks while I was “dreaming California” together with my startup team. I often enjoyed the Irish dusk while I was “deadline-surfing”, usually with the support of some of my teammates.

I don’t know how much of what I have done is common in a normal student life, but for me it has been great nevertheless. Thus, all the people who I met in that period deserve my most sincere and heartfelt thanks. Unfortunately, I am aware that the process of listing them is error-prone, so I beg your pardon if I missed you. Here is the best I was able to do Randomly Accessing my Memories:

Giovanni Tummarello and Eyal Oren respectively as my supervisor and my first mentor, for having made me acquainted with DERI since the first day I joined it, and for having introduced me to the Sindice project while it was still in alpha stage.

Renaud Delbru, Nickolai Toupikov, Richard Cyganiak, Michael Hausenblas for all the effort put in our shared projects and for all the fruitful discussions we had about our research challenges.

Robert Fuller, Stephen Mulcahy, Szymon Danielczyk who reminded me how pleasant is a workplace in which highly-competent people are ready to collaborate and help you with a smile on their faces.

Gabriele Renzi and Paolo Capriotti who trusted me so much to join my team in Ireland as soon as I proposed them. The enthusiasm and contributions they provided to the Sindice project were second only to the friendship they showed me.

All the former and current DI2 members who put their daily effort in a plethora of interesting projects which inspired and influenced me. Credits for our achievements must be evenly distributed on each one of them, since I am a strongly believer that the whole is greater than the sum of its parts.

Stefan Decker and Manfred Hauswirth respectively Director and Vice-Director of DERI, who have always been able to find free time slots when I needed their experience and knowledge, despite their extremely-busy calendars.

All the DERI colleagues who taught me how to feel myself appropriate in a Semantic Web research institute, despite my different area of expertise.

Karl Aberer and Christian Morbidoni for their precious feedback as external reviewers of this thesis. Being respectively my future (at the time of writing) Ph.D. supervisor and my former B.Sc. supervisor, they were, are and will be invaluable judges of the coherence of my work and my career.

Peter Mika who mentored me during my visit at the Yahoo! Research Labs in Barcelona. I could not have asked for a better person to show me the peculiar differences between real-world research and academic research.

My italian friends who got the habit to meet me less and less frequently. I suddenly disappeared from their daily lives jumping on a plane, yet each one of them managed to stay close to me in his own manner.

Barbara Bazzanella who broadened my views both on my research and on my personal life in an exquisitely unique way.

My family for always letting me feel the warmth of their love, no matter how far I could be.

“Prying open my third eye.”
— Maynard James Keenan

Chapter 1

Introduction

Despite the enormous amounts of information the Web has made accessible, we still lack means to interconnect and link this information in a meaningful way to lift it from the level of information to the level of *Networked Knowledge*. Besides the creation of knowledge through observation, networking of knowledge is the basic process to generate new knowledge [DH08]. Networking knowledge, can produce a piece of knowledge whose information value is far beyond the mere sum of the individual pieces, i.e., it creates new knowledge. With the Web we now have a foundational infrastructure in place enabling the linking of information on a global scale. Adding meaning moves the interlinked information to the knowledge level: *Web + Semantics = Networked Knowledge*. Knowledge is the fuel of our increasingly digital service economy (versus manufacturing economy); linking information is the basis of economic productivity.

Foundation of the Networked Knowledge is the “*Web of Data*”: a world-wide network of simple statements which, through its interconnections and sheer size, could serve as global knowledge management repository [Dec02]. The Web of Data is now moving from a vision to a reality. The fundamental standards (HTTP, URI, RDF, RDFS, and OWL) and a collection of light-weight embeddable formats (Microformats, RDFa, etc.) have been developed, data is growing at fast pace, and infrastructure is emerging. With these foundations given, we can start building applications that move towards the original vision: to improve the awareness, management and reuse of our knowledge.

Albeit the vision depicted above could take years (if not decades) to be fully accomplished, incentives for exposing (with little if no effort) such data are already becoming clear. Yahoo!’s SearchMonkey¹, for example, makes Web sites containing structured data stand out from others by providing the most appropriate visualization for the end

¹<http://developer.yahoo.com/searchmonkey/>

user in the search result page. Moreover, Google Custom Search² and Yahoo! Boss³ are directly using this information for ranking and relevance purposes - returning, for example, qualitatively better results for queries that involve everyday entities such as events, locations, and people.

1.1 Problem statement

Even though we are still at the beginning of the Data Web, the amount of information already available is much larger than what could be handled by most of the current-generation triple stores⁴ ([McB02], [BKvH02]). However, recently there have been notable advances regarding distributed triple stores, among of them: YARS2 [HUHD07] and Virtuoso Cluster Edition [EM08] by OpenLink Software. The latter is of remarkable importance for this thesis, as it showcases nice scalability properties by means of harnessing the Cloud Computing model.

Let's consider the following classification exposed in the Berkeley view of Cloud Computing [AFG⁺09]:

- a Cloud which is made available in a pay-as-you-go manner to the general public is called a *Public Cloud*; the service being sold is *Utility Computing*.
- internal datacenters of big companies, which are not made available to the general public, are called *Private Clouds*.

Their claim is that the Cloud Computing model is suitable to characterize software and hardware dynamics in a small startup as well as in a search engine which receives hundreds of millions of queries per day.

Among the others, three fundamental aspects are new in Cloud Computing:

1. *The illusion of almost-infinite computing resource available on demand*, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning
2. *The elimination of an up-front commitment by Cloud users*, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs
3. *The ability to pay for use of computing resources on a short-term basis as needed* (e.g., processors by the hour and storage by the day) and release them as needed,

²<http://www.google.com/coop/cse/>

³<http://developer.yahoo.com/search/boss/>

⁴purpose-built databases for the storage and retrieval of RDF metadata

thereby rewarding conservation by letting machines and storage go when they are no longer useful.

Not only those aspects make clear what are the advantages of building an architecture which is based on the *Software as a Service* (SaaS) paradigm, but they also show why a software design which is “Cloud-friendly” will have all the prerequisites needed to scale.

Although many applications will need to work with large amounts of metadata, one particular application would certainly not exist without the capability of accessing and processing arbitrary amounts of metadata: search engines that locate the data and services that other applications need. For this reason, Semantic Web search engines and large-scale services should be the first in harnessing Cloud Computing’s power when it comes to scaling far beyond the current generation of triple stores.

To quantify the main characteristics of such services, we propose a taxonomy of information systems⁵:

Personal information systems: multi-processor support, scales to GBs of data and hundreds of thousands of entries, usually represents the backend of a user oriented application (e.g. Mail client, PIM)

Enterprise-scale information systems: data-center distributable, scales to TBs of data and hundreds of millions of entries, exposes advanced APIs (e.g. RDBMS, Data Warehousing solutions)

Web-scale information systems: geographically distributable, scales to PBs of data and hundreds of billions of entries, trades off support for complex business logics with raw performances and deployment agility (e.g. Google and Yahoo! architectures)

Given the kind of architecture we will show in this thesis, it has certain importance being able to assert Web-scale characteristics in a search engine. In general, we envision to two different ways to establish it:

1. actually building and running a real-world proven Web-scale search engine, with an amount of daily traffic similar to current big players (Google, Yahoo!, etc.)
2. adhere to the Cloud Computing requisites in terms of software design

It follows that, especially with a limited amount of resources, only the second solution is viable.

⁵specific application software that is used to store data records in a computer system and to automate some of the information-processing activities

1.2 Hypothesis

As stated in [Neu94], the scale of a distributed system has three dimensions: numerical, geographical and administrative. The numerical dimension consists of the number of users of the system and the number of objects and services encompassed. The geographical dimension consists of the distance over which the system is scattered. The administrative dimension consists of the number of organizations that exert control over pieces of the system. The three dimensions of scale affect distributed systems in many ways, among of them: reliability, performances, latency, authorization, communication, accounting, total cost of ownership, etc.

To avoid the complexity of taking into account all those aspects in a systematic and thorough way, we followed the approach advocated by the following quote: *“Any software problem can be solved by adding another layer of indirection.”* (Steven M. Bellovin). In our specific case, the Cloud Computing model represents the layer of indirection which we exploit to simplify our analysis.

Our hypothesis is that the issue of scalability on the Web of Data can be addressed by the means of mastering and extending techniques and tools which have been proved to be effective in the Cloud Computing realm.

Thus, after having proved its feasibility [ODC⁺08], we show how standard inverted index can be effectively adapted also when indexing Semantic Web documents. Furthermore, basing our model on widely-accepted scalability recipes (like MapReduce [DG04] and BigTable [CDG⁺06] of Google’s fame), we propose some solutions tailored for Web of Data pre-processing which take into account all the key aspects of scale in distributed systems.

1.3 Thesis structure

The outline of the thesis is as follows:

- Chapter 2 introduces a general model for the Web of Data, which allows multiple forms of Web Semantics to be seen as a uniform space.
- Chapter 3 provides an architectural overview of our approach in building a scalable semantic-data processing infrastructure.

- Chapters 4, 5, 6 plunge into the details regarding the main components of our architecture: namely the pre-processing pipeline, the Information Retrieval engine and the dataset-centric ranking.
- Chapter 7 presents a real-world use case which exploits all the previously described components.

1.4 Thesis contribution

This thesis will present the following contribution to the current state of the art in the Semantic Web research:

- we will show how we have been able to successfully build <http://sindice.com>, adopting tools and techniques which are “Cloud-friendly”
- we will describe in detail why the Information Retrieval engines which lie behind Sindice let us scale far more than a triplestore based approach
- we will introduce a dataset-centric ranking algorithm, devised with both scalability and qualitative reasons in mind
- we will explain why all of the previous contributions are of key importance for building real-world application⁶ which can showcase the potentiality of the Web of Data

Out of the scope of this thesis, however, is how to scale and improve triple stores. It is important to notice that our solutions are a tradeoff between performances and features (e.g.: we only partially support SPARQL⁷).

⁶Sig.ma: live views on the Web of Data – <http://sig.ma>

⁷<http://www.w3.org/TR/rdf-sparql-query/>

Chapter 2

Data Model*

This chapter introduces a data model which defines three different layer on the Web of Data, as shown in Figure 2.1. We used it as the foundation for our infrastructure, considering the various benefits it provides:

General benefits hides unnecessary information, decouples changes, breaks up complex problem into smaller manageable pieces, abstracts implementation details, etc.

Domain-specific benefits in the case of Web of Data, which embraces both RDF and embedded metadata worlds, it allows us to see multiple forms of Web Semantics as a uniform space

2.1 Definition and Model for the Web of Data

For the purpose of this chapter, we define the Web of Data as the part of the HTTP accessible Web which returns semantically-described entities, using standard interchange formats and practices. These include pages which embed Microformats as well as RDF models using different standards. The LOD community, for example, advocates the use of HTTP content negotiation and redirect⁸ to provide either machine or human readable versions of the RDF data. RDFa, on the other hand, is a standard that enables embedding of RDF directly within HTML documents. In each case, it is possible to abstract the following core elements in semi-structured data Web publishing:

A dataset is a collection of entity descriptions. One dataset is usually the content of a database which powers a Web application exposing metadata, be this a dynamic

*This chapter is (partially) based on [CDTT09]

⁸URL redirection is a directive that allows to indicate that further action, e.g. fetch a second URL, needs to be taken by the user agent in order to fulfill the request.

Web site with just partial semantic markups (e.g. Web 2.0 sites) or a Semantic Web database which exposes its content (such as the LOD datasets). Datasets, however, can also come in the form of a single RDF document, e.g. an individual FOAF file posted on a user's homepage. A dataset is said to be a RDF *Named Graph* [CBHS05] which is uniquely identified through an URI. This name, or URI, is often mapped to a *context*.

An entity description is a set of assertions about the entity which is contained within a dataset. Assertions can be solely regarding the entity (e.g. properties such as its name or type) or connecting one or more entities either inside or outside the dataset, using resolvable URIs. Thanks to the flexibility of the semi-structured RDF model, it is always possible to map assertions to RDF *statements*. When RDF is used natively by a Web site to describe entities, it is possible to furthermore distinguish between *authoritative statements* and *authoritative descriptions*, those that are made within the context where the URI name is minted, and *non-authoritative descriptions* and *non-authoritative descriptions*, those that are made outside the context where the URI name is minted such as in third parties Web sites. For example, the authoritative description about <http://dbpedia.org/resource/Galway> is the set of statements that is obtained when resolving the URI itself.

Keeping in mind that we do want to keep statements from different contexts separated, we can see the collection of all the datasets as follow. Given three infinite sets U , B and L called respectively URI references, blank nodes and literals, a contextualized RDF statement or quad (s,p,o,c) is an element q of $Q = (U \cup B) \times U \times (U \cup B \cup L) \times (U)$. Here, s is called the subject, p the predicate, o the object and c the context of the statement. The context is generally, as explained above, the identifier of the dataset.

A View generally represents a single HTTP accessible document, retrieved by resolving an URI that points to a full or partial view of the knowledge contained in the dataset. A view can be seen as a function $v : U \rightarrow Q$ that, given an URI $u \in U$, returns a semi-structured representation of the entity denoted by u that is a set of quads $Q_u \subseteq Q$. In case of LOD, a typical view is the RDF model returned by a LOD dataset when one dereferences the URI of a resource. For example when dereferencing <http://dbpedia.org/resource/Galway>, a view of the information about this URI is created by the underlying database of DBPedia and it is returned as RDF model. LOD views are 1 to 1 mapping from the URI to the complete entity description. This however is more the exception than the rule for other kind of Web data publishing, where most often only partial entity descriptions are provided in the views. For example, in the case of microformats or RDFa, views are pages that

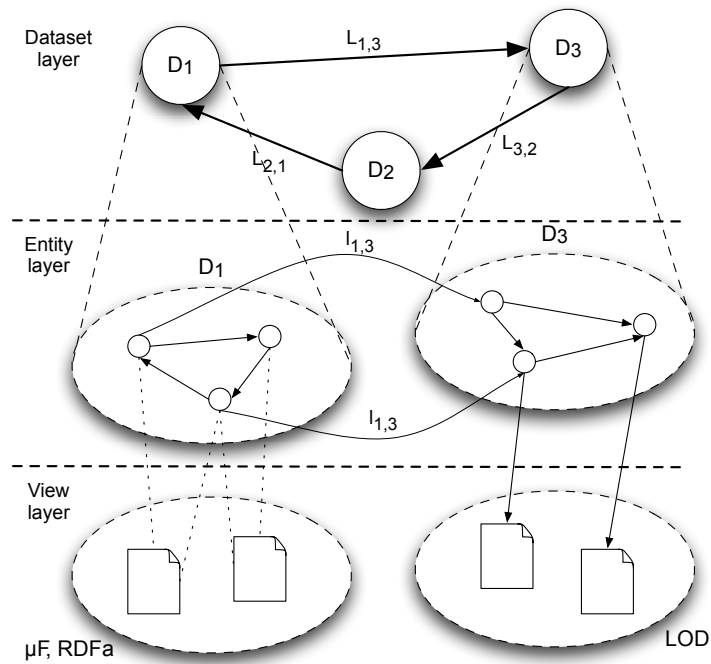


Figure 2.1: The three-layer model of the Web of Data

talk about different aspects of the entity, e.g. a page listing social contacts for a person, a separate page listing personal data as well as a page containing all the posts by a user. The union of all the views provided within a context, e.g. a Web site, might enable an agent, e.g. a crawler, to reconstruct the entire dataset, but there is no guarantee on this.

Chapter 3

A Scalable Architecture for Semantic-Data Processing*

Developed by DERI's Data Intensive Infrastructure group⁹, the Sindice project (<http://sindice.com>) aims at building scalable APIs¹⁰ to locate and make use of Semantic Data. By using the Sindice API, for example, it is possible to use keywords and semantic pattern queries to search for people, places, events and connections among these based on the semi-structured content of documents found on the Web. Sindice is designed to provide these services while fulfilling three main non-functional requirements:

- scalability
- run-time performance
- ability to cope with the many changes in standards and usage practices on the Web of Data

As in Figure 3.1, we defined four different component categories in Sindice, each one corresponding to a fundamental step in the system pipeline: *Acquisition*, *Pre-processing*, *Services*, *Post-processing*. In this chapter we will introduce each of the categories from a technical point of view, referencing following chapters for an in-depth analysis regarding some of those components.

3.1 Acquisition

Data acquisition happens by means of a crawler, a ping-manager and a sitemap-manager.

*This chapter is (partially) based on [CDTT09]

⁹<http://di2.deri.ie>

¹⁰Application Programming Interfaces

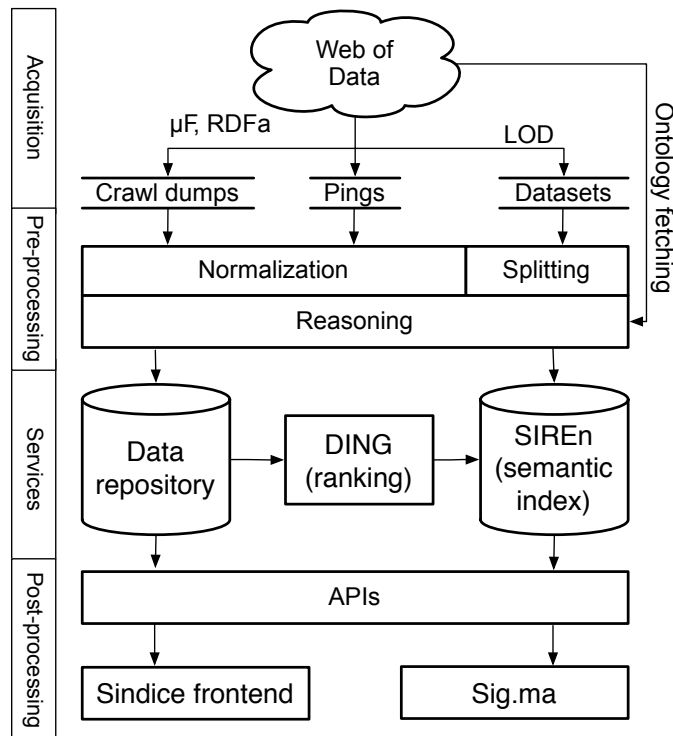


Figure 3.1: Overview graph

crawler during the development of Sindice, we based our crawler on two different technologies. At the beginning, we explored Nutch¹¹ functionalities, due to its strong relationships with Lucene and Hadoop projects (refer to Appendices A and B for more details). Eventually, due also to customization issues, we decided to migrate to Heritrix¹², the Web Archive Crawler. Since both the projects are written in Java, migrating all the code and logic we had in Nutch was not that difficult. Currently, we have an Heritrix instance running daily and we regularly optimize our crawling jobs given the results of the previous ones. While strategies for crawling the Web of Data are beyond the scope of this thesis, it is important to note that we developed algorithms which let us crawl some millions of metadata-enabled pages per day.

ping-manager Sindice has a best-effort contract with its users: being able to show fresh content within 30 minutes of having received a ping. We thus expose an API through which a developer can notify us about new URLs as well as URLs pointing to updated content. Those pings are managed by a Java sevlet that acts as a daemon¹³: the ping-manager. Basically, it gets all the requests and routes them

¹¹<http://lucene.apache.org/nutch/>

¹²<http://crawler.archive.org/>

¹³A process that runs in the background and performs a specified operation at predefined times or in response to certain events.

to different queues depending on the domain, the number of duplicate requests, etc. The queues are persisted on a replicated RDBMS, and are periodically polled by the ping-manager to actually process the URL lists. Depending on the size of the queue, we try to process the pings right away or we submit a batch job to our Hadoop cluster.

Moreover, the ping-manager takes care of periodically including the latest pings in the crawler seed list. This kind of integration not only let us refresh periodically the documents which have been explicitly pinged, but its also a way to discover interesting resources which are few hops away¹⁴ from the original source.

sitemap-manager similar in its implementation to the ping-manager, the sitemap-manager handles “Semantic Sitemaps”, a standard we proposed which facilitates the publication and harvesting of very-large datasets. More details about the standard and its advantages in Section 4.1.

3.2 Pre-processing

In this phase, the raw content harvested, e.g. the semantically annotated HTML, is analyzed by a set of parsers which we are currently publishing as open source under the name *any23*¹⁵. These parsers are used to extract RDF data from various formats. Different RDF serialization formats (RDF/XML, Notation 3, Turtle, N-Triples, RDFa) are supported, as well as several Microformats (hCard, hEvent, hListing, hResume and others). The data collected also include statistics and extra bits of information such as, for example, the title of the page, the presence of RSS or other machine readable elements, context elements such as keywords found in the text in the page, etc.

At this point, the semantics of the document (the complete or partial view over an entity), regardless of the original format, is represented in RDF. Information then reaches the “context-dependent” reasoning engine, discussed in Section 4.2, which is a part of the pre-processing step where inference is performed for each single document to be indexed.

It is of key importance to notice that the whole indexing pipeline, which includes reasoning and the feature extraction, is performed in Hadoop. This engineering effort let the Sindice indexing capabilities scale linearly with the number of nodes in our Hadoop

¹⁴by few hops away we mean a resource which is reachable following no more than 2-3 explicit links

¹⁵<http://code.google.com/p/any23/>

cluster. Refer to Appendix A for an overview about Hadoop and to Section 4.2 for a performance evaluation of our indexing pipeline.

3.3 Services

Indexing is performed by our Semantic Information Retrieval Engine (SIREn), an “entity retrieval system” designed to provide entity search capabilities over datasets as large as the entire Web of Data. SIREn supports efficient full text search with semi-structural queries and exhibits a concise index, constant time updates and inherits Information Retrieval features such as top-k queries, efficient caching and scalability via distribution over shards. SIREn is discussed in details in Chapter 5.

To be able to provide high quality top-k results, a dataset-centric ranking strategy is employed. We rank datasets using DING, a methodology which uses dataset aggregates statistics and combines the resulting values with local dataset entity ranking strategies. The resulting algorithm is shown to have desirable computational properties, e.g. allowing quasi real time ranking. DING is described in Chapter 6.

In parallel to the the IR engine, we maintain a scalable data-repository which contains both the raw data and the metadata we extract from each resource processed by our pipeline. There are several reasons for having such a component:

- it let us serve cached content to end users, useful both for historical reasons and in case of downtime of external websites
- it reduces by at least one order of magnitude the cost of fetching a document on the Web. All our components which have Web fetching requirements are redirected to the data-repository before actually hitting the Web.
- if we consider it as a snapshot of the Web of Data, it can be used to rebuild a SIREn index from scratch (disaster recovery)

For such a service, we opted for HBase¹⁶, both because it fits all our scalability requirements and because it has strong ties with Hadoop (refer to Appendix A).

¹⁶<http://hbase.org>

3.4 Post-processing

Sindice has been designed primarily for machines, and its main point of access is a REST API. All the services we offer are wrapped-up by a Ruby on Rails¹⁷ frontend, which supports HTTP Content-negotiation to serve results in RDF, JSON as well as Atom.

However, given the MVC¹⁸ pattern support in Rails, we just needed to add a thin templating-layer in front of the API to actually get a browsable interface. It proved to be useful not only for debugging reasons, but it is also heavily used by Web of Data experts to explore and showcase the potential of these new standards.

Finally, these APIs can be leveraged to create advanced applications which make use of Web of Data. This is the case of Sig.ma, an end user targeted application that enables visual “entity based search and information mashups”. Sig.ma uses an holistic approach in which large-scale semantic Web indexing, logic reasoning, data-aggregation heuristics, ad-hoc ontology consolidation, external services and responsive user interaction all play together to create rich entity descriptions. These consolidated entity descriptions then form the base for embeddable data mash-ups, machine oriented services as well as data browsing services. Sig.ma is described in details in Chapter 7.

¹⁷<http://rubyonrails.org>

¹⁸<http://en.wikipedia.org/wiki/Modelviewcontroller>

Chapter 4

Pre-Processing Terabytes of Web Semantics Data in Sindice*

In this chapter we analyze in detail the pre-processing step in Sindice (Figure 3.1). As explained in Section 3.2, we take care of the data normalization by means of a parsing suite. However, this suite does work only with small–mid sized documents, for intrinsic limitation of the parsers’ implementation. A DOM¹⁹ parser, for example, must have the entire tree in memory before any processing can begin, so the amount of memory used depends entirely on the size of the input data. Thus, for handling large RDF graphs, we proposed a standard which let us split a dataset in small entity description: the “Semantic Sitemap” (Section 4.1).

After all our data is normalized, we perform what we call “Context-Dependent Reasoning” (Section 4.2). Reasoning over semi-structured entity description enables to make explicit what would otherwise be implicit knowledge: it adds value to the information and enables an entity-centric search engine to ultimately be much more competitive in terms of precision and recall [MF03].

4.1 Semantic Sitemap: a scalable way to publish and split large-datasets

There are many ways to make information available on the Web of Data. Even restricting to the RDF world, for example, an online database might be published as one single

*This chapter is (partially) based on [CSD+08a], [DPTD08], [CDTT09]

¹⁹http://en.wikipedia.org/wiki/Document_Object_Model

RDF dump. Alternatively, the LOD paradigm is based on using resolvable URIs to offer access to individual descriptions of resources. Other datasets might offer access to its data via a SPARQL endpoint that allows clients to submit queries using the SPARQL RDF query language and protocol²⁰, or might expose HTML views that embed RDFa.

If several of these options are offered simultaneously for the same database, the choice of access method can have significant effects on the amount of networking and computing resources consumed on the client and the server side. Such choices can have serious implications. Considering that results are generated from semantic queries, which tend to be resource intensive, it might be sensible to limit the query rate to one document per second. With this limit, however, crawling Geonames' 6.4M RDF documents would take 2.5 months.

In this section we describe a methodology which Sindice supports fully that allows publishers to provide enough information about their publishing model and thus enables a smart selection of data access methods by clients and crawlers alike. The methodology, called Semantic Sitemaps, is based on extending the existing Sitemap Protocol (presented next) by introducing several new XML tags for announcing the presence of RDF data and to deal with specific RDF publishing needs. Semantic Sitemaps are used extensively to index most of the LOD datasets. They represent probably the most peculiar method of data acquisition targeted at Semantic Web data, while other methods such as direct notifications of URLs to index (called Pings), and general crawling are more comparable to those used in conventional Web search engine and will not be discussed in this chapter.

4.1.1 The Sitemap Protocol and `robots.txt`

The Sitemap Protocol²¹ defines a straightforward XML file for automatic agents (e.g. crawlers) that holds a list of URLs they should index. This is possible through tags that describe the location of each crawlable resource along with meta-information such as, for example, the expected rate of change for each individual URL or the date when this was last modified. An example of a sitemap is shown in the following listing:

Listing 4.1: Example of Sitemap

```
<?xml version="1.0" encoding="UTF-8" ?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
```

²⁰SPARQL: <http://www.w3.org/TR/rdf-sparql-query/>

²¹Sitemap Protocol: <http://www.sitemaps.org/protocol.php>

```
<url>
  <loc>http://www.example.com/</loc>
  <lastmod>2005-01-01</lastmod>
  <changefreq>monthly</changefreq>
  <priority>0.8</priority>
</url>
</urlset>
```

Once a sitemap has been created, it must be saved in a file on the server. The protocol defines a way to extend the `robot.txt` file, so that a robot can find the location of the sitemap file on a given site.

4.1.2 The Semantic Sitemaps Extension

This section introduces the Semantic Sitemaps proposal. We will start by clarifying the notion of a *dataset*, then list the key pieces of information that can be provided in a Semantic Sitemap, and finally look at two specific issues: obtaining individual resource descriptions from a large dump; and the topic of authority on the Semantic Web.

Datasets

The Semantic Sitemap extension has the concept of dataset at its core: datasets are well defined resources which can have one or more access methods. It is well defined what properties apply to a certain access method and what properties apply to a given dataset. Therefore properties that apply to that dataset will be directly related to all the data that can be obtained, independently from the access method. A publisher can host multiple datasets on the same site and can describe them independently using different sections of the Semantic Sitemap. While there is nothing that prevents information overlap or contradictions between different datasets, it is expected that this is not the case.

Adding dataset descriptions to the Sitemap protocol

The Semantic Sitemap extension allows the description of a dataset via the tag `<sc:dataset>`, to be used at the same level as `<url>` tags in a regular sitemap. Access options for the datasets are given by additional tags such as `<sc:dataDump>`, `<sc:sparqlEndpoint>`

and `<sc:linkedDataPrefix>`. If a sitemap contains several dataset definitions, they are treated independently. The following example shows a sitemap file applying the extension.

Listing 4.2: Example of Semantic Sitemap

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:sc="http://sw.deri.org/2007/07/sitemapextension">
  <sc:dataset>
    <sc:datasetLabel>
      Example Corp. Product Catalog
    </sc:datasetLabel>
    <sc:datasetURI>
      http://example.com/catalog.rdf#catalog
    </sc:datasetURI>
    <sc:linkedDataPrefix sc:slicing="subject-object">
      http://example.com/products/
    </sc:linkedDataPrefix>
    <sc:sampleURI>
      http://example.com/products/widgets/X42
    </sc:sampleURI>
    <sc:sampleURI>
      http://example.com/products/categories/all
    </sc:sampleURI>
    <sc:sparqlEndpoint sc:slicing="subject-object">
      http://example.com/sparql
    </sc:sparqlEndpoint>
    <sc:dataDump>
      http://example.com/data/catalogdump.rdf.gz
    </sc:dataDump>
    <sc:dataDump>
      http://example.org/data/catalog_archive.rdf.gz
    </sc:dataDump>
    <changefreq>weekly</changefreq>
  </sc:dataset>
</urlset>
```

The dataset is labelled as the *Example Corp. Product Catalog* and identified by <http://example.com/catalog.rdf#catalog>. Hence it is reasonable to expect further RDF annotations about the dataset <http://example.com/catalog.rdf>.

The resources described in the dataset all have identifiers starting with <http://example.com/products/>, and their descriptions are served as Linked Data. A dump of the entire dataset is available, split into two parts and the publisher states that dataset updates can be expected weekly.

RDF dataset dumps can be provided in formats such as RDF/XML, N-Triples and N-Quads²² (similar to N-Triples with a fourth element specifying the URI of the RDF document containing the triple; the same triple might be contained in several different documents). Optionally, dump files may be compressed in GZIP, ZIP, or BZIP2 format.

4.2 Context-Dependent Reasoning

After having created from the views a complete or partial entity description, the information is pushed into the indexing pipeline. The main component of the indexing pipeline is the “context-dependent” reasoning engine. Reasoning over semi-structured entity description enables to make explicit what would otherwise be implicit knowledge: it adds value to the information and enables an entity-centric search engine to ultimately be much more competitive in terms of precision and recall [MF03]. The drawback is that inference can be computationally expensive, and therefore drastically slow down the process of indexing large amounts of information.

Our work is specifically concerned on how to efficiently reason over very large collections of entity description which have been published on the Web. To reason on such descriptions, we assume that the ontologies that these descriptions refer to are either included explicitly with `owl:imports` declarations or implicitly by using property and class URIs that link directly to the data describing the ontology itself. This later case should be the standard if the W3C best practices [MBS06] for publishing ontologies and the Linked Data principles [BL06] are followed by ontology creators. As ontologies might refer to other ontologies, the import process then needs to be recursively iterated (see Sect. 4.2.2 for a detailed formalisation).

A naive approach would be to execute such recursive fetching for each entity description and create a single RDF model composed by the original description plus the

²²N-Quads: <http://sw.deri.org/2008/07/n-quads/>

ontologies. At this point the deductive closure (see Sect. 4.2.3 for a detailed formalisation) of the RDF model can be computed and the entity description – including ontologically inferred information – can be indexed.

Such a naive procedure is however obviously inefficient since a lot of processing time will be used to recalculate deductions which, as we will see, could be instead reused for possibly large classes of other entity descriptions during the indexing procedure.

To reuse previous inference results, a simple strategy has been traditionally to put several (if not all) the ontologies together compute and reuse their deductive closures across all the entity description to be indexed. While this simple approach is computationally convenient, it turns out to be sometimes inappropriate, since data publishers can reuse or extend ontology terms with divergent points of view.

For example, if an ontology other than the FOAF vocabulary itself extends `foaf:knows` as a symmetric property, an inferencing agent should not consider this axiom outside the scope of the entity description that references this particular ontology. Not doing so would severely decrease the precision of semantic querying, by diluting the results with many false positives.

For this reason, a fundamental requirement of the procedure that we developed has been to confine ontological assertions and reasoning tasks into contexts in order to track provenance of inference results. By tracking provenance of each single assertion, we are able to prevent one ontology to alter the semantics of other ontologies on a global scale.

4.2.1 Contexts on the Semantic Web

As discussed previously, the dataset URI provides a natural way to define its context. Naming an RDF graph has multiple benefits. It helps tracking the provenance of each statement. In addition, since named graphs are treated as first class objects, it enables the description and manipulation of a set of statements just as for any other resource. However, in this section, the notion of context refers to the entity URI at which the entity description is retrievable. For our purposes, it is sufficient to understand, in this section, that by *context* we identify the set of statements of an entity description.

Guha [GMF04] proposed a context mechanism for the Semantic Web which provides a formal basis to specify the aggregation of contexts. Within his framework, a context denotes the scope of validity of a statement. This scope is defined by the symbol *ist* (“is

true in context”), introduced by Guha in [Guh92]. The notation $ist(c, \varphi)$ states that a proposition φ is true in the context c .

The notion of context presented in this section is based on Guha’s context mechanism. Its aim is to enable the control of integration of ontologies and ultimately avoid the aggregation of ontologies that may result in undesirable inferred assertions.

Aggregate Context

An *Aggregate Context* is a subclass of *Context*. Its content is composed by the content retrieved by resolving its URI, and by the contents lifted from other contexts. An aggregate context must contain the full specification of what it imports. In our case, each entity description is considered an *Aggregate Context*, since it always contains the specification of what it imports through explicit or implicit import declarations, described next.

Lifting Rules

Since contexts are first class objects, it becomes possible to define expressive formulae whose domains and ranges are contexts. An example are so called *Lifting Rules* that enable to *lift* axioms from one context to another. In Guha’s context mechanism, a lifting rule is a property type whose domain is an *Aggregate Context* and range a *Context*. A lifting rule can simply import all of the statements from a context to another, but can also be defined to select precisely which statements have to be imported.

4.2.2 Import Closure of RDF Models

On the Semantic Web, ontologies are meant to be published so to be easily reused by third parties. OWL provides the `owl:imports` primitive to indicate the inclusion of a target ontology inside an RDF model. Conceptually, importing an ontology brings the content of that ontology into the RDF model.

The `owl:imports` primitive is transitive. That is, an import declaration states that, when reasoning with an ontology O , one should consider not only the axioms of O , but the entire import closure of O . The imports closure of an ontology O is the smallest set containing the axioms of O and of all the axioms of the ontologies that O imports. For

example, if ontology O_A imports O_B , and O_B imports O_C , then O_A imports both O_B and O_C .

The declaration of `owl:imports` primitives is not a common practice in Semantic Web documents, most published RDF models do not contain explicit `owl:imports` declarations. For example, among the 50 million of documents in Sindice, only 66.38 thousand are declaring at least one `owl:imports`.

Instead, RDF models generally refer to classes and properties of existing ontologies by their URIs. For example, most FOAF profile documents don't explicitly import the FOAF ontology, but instead just refer to classes and properties of the FOAF vocabulary. Following the W3C best practices [MBS06] and Linked Data principles [BL06], the URIs of the classes and properties defined in an ontology should be resolvable and provide the machine-processable content of the vocabulary.

In the presence of such dereferenceable class or property URIs, we perform what we call an *implicit import*. By dereferencing the URI, we attempt to retrieve a document containing the ontological description of the entity and to include its content inside the source RDF model.

Also implicit import is considered transitive. For example, if a RDF model refers to an entity E_A from an ontology O_A , and if O_A refers to an entity E_B in an ontology O_B , then the model imports two ontologies O_A and O_B .

4.2.3 Deductive Closure of RDF Models

In Sindice, we consider for indexing inferred statements from the deductive closure of each entity description we index. What we call here the “deductive closure” of a RDF model is the set of assertions entailed in the aggregate context composed of the entity description itself and of its ontology import closure.

When reasoning with Web data, we can not expect to deal with a level of expressiveness of OWL-DL [dBG⁺07], but would need to consider OWL Full. Since under such circumstances, we cannot strive for complete reasoning anyway, we therefore content with an incomplete but finite entailment regime based on a subset of RDFS and OWL, namely the *ter Horst* fragment [tH05] as implemented by off-the-shelf rule-based materialisation engines such as for instance OWLIM²³. Such an incomplete deductive closure is sufficient with respect to increasing the precision and recall of the search engine, and

²³<http://www.ontotext.com/owlim/>

provides useful RDF(S) and OWL inferences such as class hierarchy, equalities, property characteristics such as inverse functional properties or annotation properties.

A rule-based inference engine is currently used to compute full materialisation of the entailed statements with respect to this finite entailment regime. In fact it is in such a setting a requirement that deduction to be finite, which in terms of OWL Full can only be possible if the entailment regime is incomplete (as widely known the RDF container vocabulary alone is infinite already [Hay04]). This is deliberate and we consider a higher level of completeness hardly achievable on the Web of Data: in a setting where the target ontology to be imported may not be accessible at the time of the processing, for instance, we just ignore the imports primitives and are thus anyway incomplete from the start.

One can see that the deductive closure of an aggregate context can lead to inferred statements that are not true in any of the source contexts alone.

Definition 4.1 *Let c_1 and c_2 be two contexts with respectively two propositions φ_1 and φ_2 , $ist(c_1, \varphi_1)$ and $ist(c_2, \varphi_2)$, and $\varphi_1 \wedge \varphi_2 \models \varphi_3$, such that $\varphi_2 \not\models \varphi_3$, $\varphi_1 \not\models \varphi_3$, then we call φ_3 a newly entailed proposition in the aggregate context $c_1 \wedge c_2$. We denote the set of all newly defined propositions Δ_{c_1, c_2} :*

$$\begin{aligned} \Delta_{c_1, c_2} = & \{ist(c_1, \varphi_1) \wedge ist(c_2, \varphi_2) \models ist(c_1 \wedge c_2, \varphi_3) \\ & \text{and } \neg(ist(c_1, \varphi_3) \vee ist(c_2, \varphi_3))\} \end{aligned}$$

For example, if a context c_1 contains an instance x of the class *Person*, and a context c_2 contains a proposition stating that *Person* is a subclass of *Human*, then the entailed conclusion that x is a human is only true in the aggregate context $c_1 \wedge c_2$:

$$\begin{aligned} ist(c_1, Person(x)) \quad \wedge \quad ist(c_2, subclass(Person, Human)) \\ \rightarrow \quad ist(c_1 \wedge c_2, Human(x)) \end{aligned}$$

Note that - by considering (in our case (Horn) rule-based) RDFS/OWL inferences only - aggregate contexts enjoy the following monotonicity property:²⁴ if aggregate context $c_1 \subseteq c_2$ then $ist(c_2, \phi)$ implies $ist(c_1, \phi)$, or respectively, for overlapping contexts, if $ist(c_1 \cap c_2, \phi)$ implies both $ist(c_1, \phi)$ and $ist(c_2, \phi)$.

²⁴We remark here that under the addition of possibly non-monotonic rules to the Semantic Web architecture, this context monotonicity only holds under certain circumstances [PFH06].

Such property is exploited in a distributed ABox reasoning procedure based on an persistent TBox, called an ontology base. The ontology base is in charge of storing any ontology discovered on the Web of Data with the import relations between them. The ontology base also stores inference results that has been performed in order to reuse such computation later. More information about the implementation issues of such reasoning procedure can be found in [DPTD08].

4.2.4 Technical Overview

In Sindice, reasoning is performed for every entity description at indexing time. This is possible with the help of a distributed architecture based on Hadoop²⁵ and of Memcached²⁶. Each Hadoop node has its own ontology base which support the linked ontology and inference model described in the previous section. A single ontology base is shared across several Hadoop worker jobs by the. This architecture can be seen as a distributed ABox reasoning with a shared TBox among all the nodes.

Each Hadoop worker job acts as a reasoning agent processing an entity description at a time. It first analyses the description in order to discover the explicit `owl:imports` declarations or the implicit ones by resolving each class and property URIs. Then, the reasoning agent lift the content of the referred ontologies inside the RDF model by querying the ontology cache with the URIs of the ontologies. The ontology cache responds by providing the set of ontological assertions, unless there is a cache miss. In case of a miss, the TBox layer performs the inferencing closure on the given ontology, taking care also of updating the caching layer.

As none of the triple-stores available today support context aware reasoning, our implementation required considerable low level modifications to the Aduna Sesame framework.²⁷

4.2.5 Performance Overview

Using a three nodes Hadoop cluster and a Memcached pool on commodity servers, we are able to process²⁸ an average of 60 documents per second when building the 50 million documents index currently available in the Sindice index.

²⁵Hadoop: <http://hadoop.apache.org/core/>

²⁶<http://www.danga.com/memcached/>

²⁷OpenRDF Sesame: <http://www.openrdf.org/>

²⁸including document pre-processing and post-processing (metadata extraction, indexing, etc.)

When running the indexing with a warm cache pool, we experienced an improvement of about 20%. This is implied by the fact that the ontology networks do not need to be computed again, saving an high amount of cache misses.

The processing speed varies also depending on the type of RDF model. On models with a simple and concise representation, such as the ones found in the Geonames' dataset, the prototype is able to process up to 80 entity descriptions per second.

It is interesting to make a few observation based on the large corpus of documents processed by Sindice. We observe that on a snapshot of the index containing 6 million of documents, the original size of the corpus was 18GB whereas the total size after inference was 46GB, thus a ratio of 2.5. As a result of inference, we can then observe that in Sindice we are indexing twice as many statements as we would by just indexing explicit semantics. Also we find that the number of ontologies, defined as documents which define new classes or properties, in our knowledge base is around 95.000, most of which are fragments coming from projects such as OpenCyc, Yago and DBpedia. The ratio of ontologies to semantic documents is nevertheless low, currently 1 to 335.

4.2.6 Discussion

Reasoning of the Web of Data enables the Sindice search engine to be more effective in term of precision and recall for Semantic Web information retrieval. As an example, it would not be possible to answer a query on FOAF social networking files asking for entities which have label “giovanni” unless reasoning is performed to infer `rdfs:labels` from the property `foaf:name`.

The context mechanism allows Sindice to avoid the deduction of undesirable assertions in RDF models, a common risk when working with the Web of Data. However, this context mechanism does not restrict the freedom of expression of data publisher. Data publisher are still allowed to reuse and extend ontologies in any manner, but the consequences of their modifications will be confined in their own context, and will not alter the intended semantics of the other RDF models published on the Web.

Chapter 5

SIREn: a Semantic Information Retrieval Engine for the Web of Data*

We present the Semantic Information Retrieval Engine (SIREn), an “entity retrieval system” designed to provide entity search capabilities over datasets as large as the entire Web of Data. SIREn supports efficient full text search with semi-structural queries and exhibits a concise index, constant time updates and inherits Information Retrieval features such as top-k queries, efficient caching and scalability via distribution over shards.

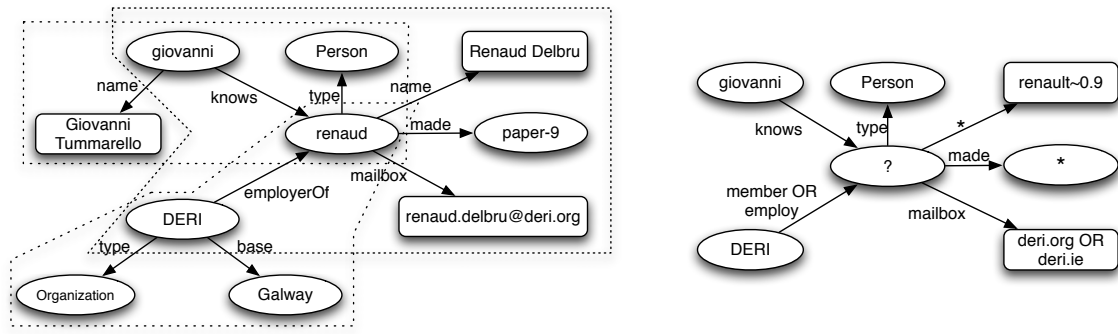
5.1 Introduction

In developing the Semantic Information Retrieval Engine (SIREn), the goal has been to be able to index the entire “Web of Data”. The requirements have therefore been:

1. Support for the multiple formats which are used on the Web of Data;
2. Support for entity centric search;
3. Support for context (provenance) of information: entity descriptions are given in the context of a Website or dataset;
4. Support for semi-structural full text search, top-k query, scalability via shard over clusters of commodity machines, efficient caching strategy and real-time dynamic index maintenance.

With respect to point 1, the problem is solved as we normalize everything to RDF as previously explained. With respect to point 2 and 3, the main use case for which SIREn

*This chapter is (partially) based on [DTCT09], [CDTT09]



(a) Visual representation of an RDF graph. The RDF graph is divided (dashed lines) into three entities identified by the node *renaud*, *giovanni* and *DERI* (b) Star-shaped query matching the entity *renaud* where *?* is the bound variable and *** a wildcard

Figure 5.1: In these graphs, oval nodes represent resources and rectangular ones represent literals. For space consideration, URIs have been replaced by their local names.

is developed is entity search: given a description of an entity, i.e. a star-shaped queries such as the one in Figure 5.1b, locate the most suitable entities and datasets. This means that, in terms of granularity, the search needs to move from “page” (as per normal Web search) to a “dataset-entity”. The Figure 5.1a shows an RDF graph and how it can be split into three entities *renaud*, *giovanni* and *DERI*. Each entity description forms a sub-graph containing the incoming and outgoing relations of the entity node.

5.2 SIREn Data Model

SIREn, similarly to XML information retrieval engine, adopts a tree data structure and orderings of tree nodes to model datasets, entities and their RDF descriptions. The data tree model is pictured in Fig. 5.2a. This model has a hierarchical structure with four different kind of nodes: context (dataset), subject (entity), predicate and object. Each node can refer to one or more terms. In case of RDF, a term is not necessarily a word (as in part of an RDF Literal), but can be an URI or a local blank node identifier.

Inverted indexes based on tree data structure enable to efficiently establish relationships between tree nodes. There are two main types of relations: *Parent-Child* and *Ancestor-Descendant*. To support these relations, the requirement is to assign unique identifiers (node labels) that encodes relationships between the tree nodes. Several node labelling schemes have been developed and the reader can refer to [SCCS09] for an

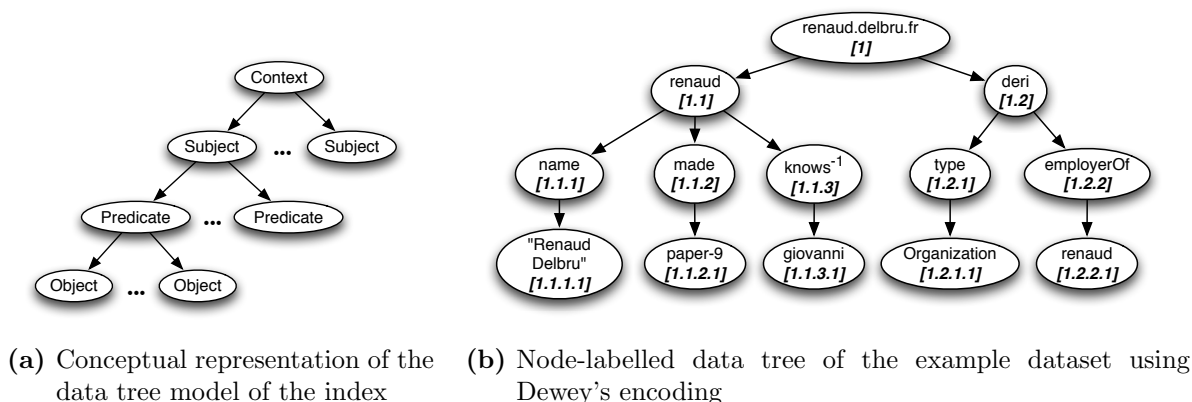


Figure 5.2: The SIREn data model

overview of them. In the rest of the chapter, we will use a simple prefix scheme, the *Dewey Order* encoding [BVT⁺02].

Using this labelling scheme, structural relationships between elements can be determined efficiently. An element u is an ancestor of an element v if $\text{label}(u)$ is a prefix of $\text{label}(v)$. Fig. 5.2b presents a data tree where nodes have been labelled using Dewey's encoding. Given the label $\langle 1.2.1.1 \rangle$ for the term **Organisation**, we can efficiently find that its parent is the predicate `rdf:type`, labelled with $\langle 1.2.1 \rangle$.

The data tree structure covers the quad relations CSPO (outgoing relations) and COPS (incoming relations). Incoming relations are symbolised by a predicate node with a -1 tag in Fig. 5.2b. The tree data structure is not limited to quad relations, and could in theory be used to encode longer paths such as 2-hop outgoing and incoming relations.

In SIREn, the node labelled tree is embedded into an inverted list. Each term occurrence is stored with its node label, i.e. the path of elements from the root node (context) to the node (predicate or object) that contains the term. A detailed description of how this tree data structure is encoded in the inverted index is given in [DTCT09].

5.3 SIREn Query Model

Since RDF is semi-structured data, we expect three types of queries: 1. full-text search (keyword based), 2. semi-structural queries (complex queries specified in a star-shaped structure), 3. or a combination of the two (where full-text search can be used on any

part of the star-shaped query). We present in this section a set of query operators over the content and structure of the data tree that cover the three types of queries.

5.3.1 SIREn Operators

Content operators The content query operators are the only ones that access the content of a node, and are orthogonal to the structure operators. They include extended boolean operations such as boolean operators (intersection, union, difference), proximity operators (phrase, near, before, after, etc.) and fuzzy or wildcard operators.

These operations allow to express complex keyword queries for each node of the tree. Interestingly, it is possible to apply these operators not only on literals, but also on URIs (subject, predicate and object), if URIs are normalized (i.e. tokenized). For example one could just use an RDF local name, e.g. `name`, to match `foaf:name` ignoring the namespace.

Structure operators In the following, we define a set of operations over the structure of the data tree. Thanks to these operations, we are able to search content to limited tree nodes, to query node relationships and to retrieve paths of nodes matching a given pattern. Joins over paths are possible using set operators, enabling the computation of entities and datasets matching a given star-shaped query.

Ancestor-Descendant: A//D A node A is the ancestor of a node D if it exists a path between A and D. For example, the SPARQL query, in Fig. 5.3a line 1, can be interpreted as an Ancestor-Descendant operator, as shown in Fig. 5.3b line 1, and will return the path $\langle 1.2.2.1 \rangle$.

Parent-Child: P/C A node P is the parent of a node C if P is an ancestor of C and C is exactly one level above P. For example, the SPARQL query, in Fig. 5.3a line 2, can be translated into a Parent-Child operator, as shown in Fig. 5.3b line 2, and will return the path $\langle 1.1.1.1 \rangle$.

Set manipulation operators These operators allow to manipulate nodes of the tree (context, subject, predicate and object) as sets, implementing union (\cup), difference (\setminus) and intersection (\cap). For example in Fig. 5.3a the SPARQL query, line 3, can be interpreted as two Parent-Child operators with the intersection operator (AND), as shown in Fig. 5.3b line 3.

1	SELECT ?g WHERE { GRAPH ?g { deri ?p renaud } }
2	SELECT ?g ?s WHERE { GRAPH ?g { ?s name "Renaud Delbru" } }
3	SELECT ?g ?o WHERE { GRAPH ?g { giovanni knows ?o . deri employerOf ?o . } }
4	SELECT ?s WHERE { GRAPH <renaud.delbru.fr> { ?s knows renaud } }
5	SELECT ?g ?s WHERE { GRAPH ?g { ?s employerOf ?o . ?o name "renaud" . } }

1	deri // renaud
2	name / "Renaud Delbru"
3	knows ⁻¹ / giovanni AND employerOf ⁻¹ / deri
4	renaud.delbru.fr // knows / renaud
5	employerOf // name / "renaud"

(a) SPARQL queries
(b) SIREn interpretation

Figure 5.3: SPARQL queries and their SIREn interpretation

SPOC	POCS	OCSP	CPSO	CSPO	OSPC
(?,*,*,?)	(?,p,*,?)	(?,*,o,?)	(?,*,*,c)	(s,*,*,c)	(s,*,o,?)
	<i>p</i>	<i>o</i>	<i>c</i>	<i>c/s</i>	<i>s//o</i>
(s,*,*,?)	(?,p,o,?)	(?,*,o,c)	(?,p,*,c)	(s,p,*,c)	
<i>s</i>	<i>p/o</i>	<i>c//o</i>	<i>c//p</i>	<i>c/s/p</i>	
(s,p,*,?)	(?,p,o,c)	(s,*,o,c)			
<i>s/p</i>	<i>c//p/o</i>	<i>c/s//o</i>			
(s,p,o,?)					
<i>s/p/o</i>					

Table 5.1: Quad patterns covered by outgoing relations and their interpretation with the SIREn operators. The ? stands for the elements that are retrieved and the * stands for a wildcard element.

In addition, operators can be nested to express longer path as shown in Fig. 5.3a, line 4 and 5. However, the later query is possible only if deeper trees have been indexed, i.e. 2-hop outgoing and incoming relations of an entity.

5.3.2 SPARQL Interpretation

In this section we discuss the extension by which, given the above discussed operators, it is possible to support a subset of the standard SPARQL query language.

By indexing outgoing relations alone, we can show to cover the quad access patterns listed in Table 5.1. A quad lookup is performed using the AC or PC operators. Join operations over these patterns are also feasible. Intersection, union and difference between two or more quad patterns can be achieved efficiently using set manipulations over tree nodes.

The covered quad patterns are a subset of the quad patterns covered by conventional RDF data management systems [HD05]. In fact, they give the ability to retrieve information about variables that are restricted to be at the subject, object or context position.

It is important to underline however that we are restricting the search of an entity inside a dataset, i.e. SIREn does not allow the use of joins over different contexts, and the intersection of quad patterns within an entity, i.e. SIREn does not allow the use of chains of joins among multiples entities. This limits the query expressiveness to a star-shaped query, e.g. in Fig. 5.1b.

5.4 Experimental Benchmark

We compared the performance of our approach to native quad and triple stores using two large datasets having different characteristics. We assess the space requirements, the performance of compression, the index creation time and the query evaluation speed.

Our first comparison system is the native store backend (based on b+-trees) of Sesame 2.0, a very popular open-source system which is commonly used as baseline for comparing quad store performances (e.g. in [ZLZ⁺07, NW08]). The second system we compare against is the state-of-the-art triple store RDF-3X [NW08].

For our experiments we used two datasets. The first one, called “Real-World” dataset has been obtained by random sampling the content of the Sindice.com semantic search engine. The real world dataset consists of 10M triples (approximately 2Gb in size), and contains a balanced representation of triples coming from the Web of Data, e.g. from RDF and Microformats datasets published online. The second dataset is the MIT Barton dataset that consists of 50M triples (approximately 6Gb in size).

The machine that served for the experiment was equipped with 8GB ram, 2 quad core Intel processors running at 2.23 Ghz, 7200 RPM SATA disks, linux 2.6.24-19, Java Version 1.6.0.06 and GCC 4.2.4. The following benchmarks were performed with cold-cache by using the `/proc/sys/vm/drop_caches` interface to flush the kernel cache and by reloading the application after each query to bypass the application cache.

In Table 5.2 the index creation time for the two datasets is reported. For SIREn we report two cases: SIREn10 is the time it takes to construct the index committing 10.000 triples at a time while SIREn100 is the time it takes when the batch is composed by

	SIREn10	SIREn100	Sesame	RDF-3X
Barton	3	1.5	266	11
Real-World	1	0.5	47	3.6

Table 5.2: Indexing time in minutes per dataset and system

100.000 triples. Concerning RDF-3X, it is important to notice that it does not support context, therefore it indexes triples and not quads and does not support incremental indexing; RDF-3X needs the full dataset beforehand in order to construct the indexes in one batch process, as opposed to Sesame whose data structures support incremental updates. We can see from the results in Table 5.2 that SIREn is 50 to 100 times faster than Sesame and 3 to 6 times faster than RDF-3X.

In the next test, we plot the performance of SIREn and Sesame in an incremental update benchmark. The Fig. 5.4a shows the commit times for an incremental 10.000 triples batch on the two systems. As the time scales are greatly different, the graph is reported in Log scale. While the absolute time is significant, the most important result is the constant time exhibited by SIREn for incremental updates, as compared to the Sesame performance which progressively decreases as the index size increases.

We continue the test, with SIREn only, in Fig. 5.4b, where the commit time is plotted for a synthetic dataset constructed by replicating Barton 20 times so to reach 1 billion triples. The total indexing creation time is 31 minutes. We can notice that SIREn keeps a constant update time during the entire indexing. Outliers are due to periodic merges of the index segments.

Finally, we evaluate SIREn scalability by indexing a dataset composed by 1 billion entities described in approximately 10 billion triples. The dataset is derived from the billion triple challenge dataset²⁹. To avoid hitting the limit of 2 billion entities due to the current implementation, we remove entities which have only one or two triples of description and duplicate the remaining triples to reach 10 billion. The set of queries is provided at <http://siren.sindice.com>. The performance is given in the Table 5.3.

²⁹Semantic Web Challenge: <http://challenge.semanticWeb.org/>

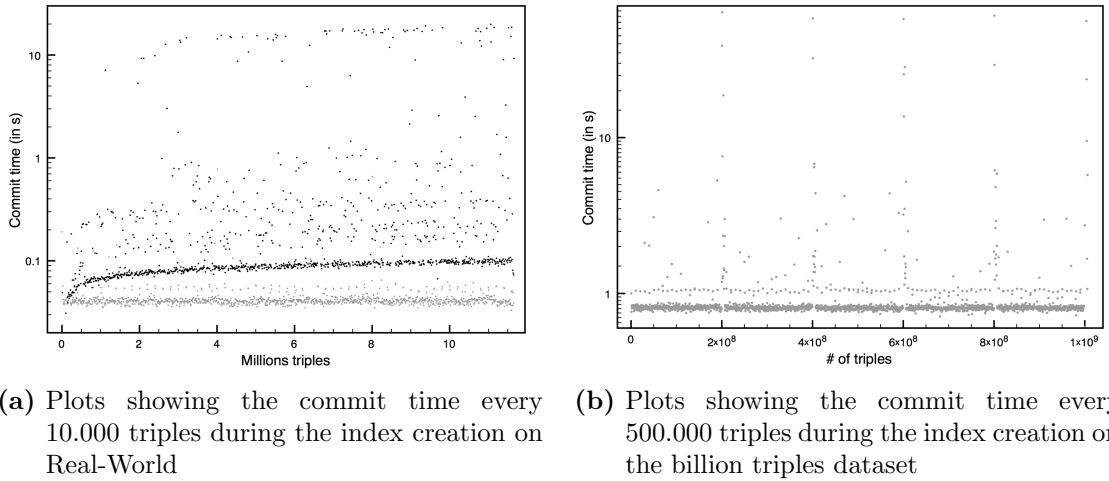


Figure 5.4: Dark dots are Sesame commit time records while gray dots are SIREn commit time records

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Time (s)	0.75	1.3	1.4	0.5	1.5	1.6	4	0.35
Hits	7552	9344	3.5M	57K	448	8.2M	20.7M	672

Table 5.3: Querying time in seconds and number of hits for the 10 billion triples benchmark

5.5 Conclusion

We presented SIREn, a novel indexing scheme for semi-structured data in RDF. SIREn was designed for indexing very large datasets and handling the requirements of indexing and querying the Web of Data: constant time incremental updates and very efficient entity lookup using semi-structural queries (i.e. star shaped) with full text search capabilities. With respect to DBMS and IR systems, SIREn tries to get the best of both worlds as it allows semi-structural queries while retaining many desirable IR features: single inverted index, effective caching, top-k queries and efficient index distribution over shards.

SIREn has been implemented and is in production at the core of the Sindice semantic search engine. At the time of the writing, SIREn serves over 50 million harvested web pages containing RDF or Microformats and answers several tens of thousands queries per day on a single machine.

Chapter 6

DING: A Dataset-Centric Approach to Ranking*

Effective processing of top-k queries is a crucial requirement in an environment that involves massive amount of data. End-users are more interested in the most relevant top-k query answers from the potentially huge answer space. One solution is based on link analysis algorithms for determining the “popularity” of a resource on the Web, and has become a central technique in Web search. A static “popularity score” is computed for each resource, giving the possibility to rank and select the most relevant top-k results. As the Web of Data graph is becoming very large, containing over billions of nodes and edges, it is clear that link analysis algorithms for computing popularity score on Web-scale graph is becoming an important requirement.

6.1 Introduction

Current link analysis algorithms for the Web of Data generally work on a flat link graph of entities. In addition to their high computation complexity, they suffer from biased ranking since they do not take into consideration the structure of the Web of Data. Based on previous observations, we derive a two-layer model that both considers the hierarchical structure and the link structure, and propose a novel ranking algorithm called DING (for Dataset rankING). DING first aggregates entities into datasets before performing link analysis on the dataset graph. Then, for each dataset, a ranking is computed for the local entity collection. As a final step, the popularity of the dataset

*This chapter is (partially) based on [TUD⁺09], [CDTT09]

is propagated to its entities and combined with their local ranks in order to compute a global rank for each entity. We will show that DING has the following properties:

scalable the computation of DING and of the local entity ranks is greatly simplified compared to standard link analysis algorithms and can be easily distributed.

targeted the separation of local and global ranking allows for specialized local ranking, something known to be beneficial.

effective the ranking produced by DING is qualitatively comparable or even better than a link analysis performed on the global flat graph.

6.2 A Two-Layer Model for Web Data

In this section, we introduce the two layer model for the Web of Data that is at the core of the DING algorithm. The two model layer is in fact the model pictured in Figure 2.1, but without the view layer. The top layer (dataset layer) is composed of a collection of interconnected datasets whereas the lower layer (entity layer) is composed of independent graphs of entities. We first define the graph model on which DING performed link analysis. The graph model is a subset of the one defined in Chapter 2. We finally present the two types of links that are used in the two-layer model.

In this section, we see the Web of Data as a set of datasets \mathcal{D} . A dataset $D_i \in \mathcal{D}$ is a named graph where nodes are strictly URIs, i.e. where each statement (s, p, o, c) is an element of URI^4 . We discard deliberately all the other statements, i.e. the ones containing a blank node or a literal, since they are not of interest for DING. In addition, all non-authoritative statements are discarded.

A statement is interpreted as a link between two entities. We consider two types of links. The “intra-dataset” links are links from one authoritative entity to another authoritative entity in a same dataset. The “inter-dataset” links are links from one authoritative entity to a non-authoritative entity. While the later type of links indicates a link between two resources, we interpret this as a link from a dataset D_i named c_i to a dataset D_j named c_j . Furthermore, inter-links are aggregated into *linksets*. A linkset $L_{i \rightarrow j}$ is defined as a subset of D_i and contains the links from D_i to D_j .

Intra-dataset links form a local graph of entities while inter-dataset links form a global graph of datasets. We derive from this final observation the two-layer model for the Web of Data, shown in Figure 2.1 where the top layer (dataset layer) is a collection

of interconnected datasets and the lower layer (entity layer) is composed of independent local graphs of entities.

6.3 DING Algorithm

The DING algorithm is an adaptation of the PageRank’s random surfer to the two-layer graph structure presented in the previous section. Instead of visiting Web pages, the random surfer now browses datasets. The random walk model is as follows:

1. At the beginning of each browsing session, a user randomly selects a dataset node.
2. Then, the user may choose one of the following actions:
 - a) Selecting randomly an entity in the current dataset.
 - b) Jumping to another datasets that is linked by the current dataset.
 - c) Ending the browsing.

According to this hierarchical random walk model, we can apply a two-stage computation. In the first stage, we calculate the importance of the dataset nodes which is explained next. The second stage calculates the importance of entities inside a dataset. We argue that local entity ranking is strongly dependent of the nature (graph structure) of the dataset. The datasets on the Web of data may come in a variety of graph structures, such as generic graphs coming from user input, hierarchical graphs, pattern-based graphs like citation datasets, etc. An approach which takes into account the peculiar properties of each dataset could give better results. We explain in Section 6.3.1 how to combine the importance of datasets and entities.

The dataset surfing behavior is the same as the page surfing behavior in PageRank. We can obtain the importance of the dataset nodes by applying a PageRank algorithm on the dataset-level weighted graph.

The rank score $r(D_i)$ of a dataset is composed of a part $\sum_{j \in O(i)} r(D_j)p_{j \rightarrow i}$ corresponding to the votes from the datasets linking to D_i and of a part corresponding to the probability pr_i of a random jump to D_i from any dataset in the collection. The probability pr_i is defined as:

$$pr_i = \frac{n(D_i)}{\sum_{D_j \in \mathcal{D}} n(D_j)} \tag{6.1}$$

Although the algorithm only examines the dataset level of browsing, we still assume that the browsing is done on a URI level. As a consequence, the probability pr_i of

selecting a dataset during a random jump is proportional to its size, $n(D_i)$. Similarly, the probability $p_{j \rightarrow i}$ of following a linkset from D_j to D_i is proportional to the size of the linkset $L_{j \rightarrow i}$. The final DING rank formula is given below. As in PageRank, the two parts are combined using the teleportation factor α . We set α the recommended [PBMW99] value of 0.85, since we found that this value provides also good results for the DING use case.

$$r(D_i) = (1 - \alpha)pr_i + \alpha \sum_j r(D_j)p_{j \rightarrow i} \quad (6.2)$$

A method used in layered ranking algorithms is to assign to the page the importance of the supernode [EAT04]. In our case this would correspond to assign the DING rank of the dataset to all its entities. In large datasets, such as DBPedia, this approach does not hold. Any query is likely to return many entities having the same importance from one dataset. This unnecessarily pushes part of the ranking problem to query-time. Instead we should assign a score combining both the importance of a dataset and the importance of an entity within its dataset.

6.3.1 Combining DING and Entity Rank

The combination of dataset and local entity ranks is a key part of the DING approach to ranking. One is to adopt a purely probabilistic point of view, interpreting the dataset rank score as the probability of selecting the dataset and the local rank score as the probability of selecting an entity in the dataset. Hence we would have the global score of the entity e from dataset D defined as

$$r(e) = P(e \cap D) = P(e) * P(D) \quad (6.3)$$

This approach has however one practical issue, it favors smaller datasets. Indeed, their local ranking scores will be much higher than the ones in larger datasets, since in the probabilistic model all scores in a dataset sum to 1. As a consequence any small dataset receiving even a single link from a larger one will likely have its top entities score way above many of the ones from the larger dataset, and opens a spamming possibility. One solution is to normalize, based on the dataset size, the local ranks to a same *average*.

	PageRank	DING
1	http://www.w3.org/2000/10/swap/pim/contact	http://www.w3.org/People/Berners-Lee/car...
2	http://www.w3.org/People/Berners-Lee/card	http://www.w3.org/2000/10/swap/pim/contact
3	http://www.w3.org/2009/Talks/0204-ted-tbl	http://www.w3.org/2000/10/swap/data#Cwm
4	http://richard.cyganiak.de/foaf.rdf	http://www.w3.org/People/EM/contact
5	http://www.ivan-herman.net/me	http://danbri.org/foaf.rdf
6	http://www.advogato.org/person/timbl/foa...	http://www.w3.org/2009/Talks/0204-ted-tbl
7	http://heddley.com/edd/foaf.rdf#edd	http://www.w3.org/2000/10/swap/pim/doc
8	http://myopenlink.net/dataspace/kidehen/...	http://www.w3.org/2000/10/swap/pim/contact
9	http://myopenlink.net/dataspace/kidehen/...	http://www.ivan-herman.net/foaf.rdf
10	http://www.advogato.org/person/timbl/foa...	http://dig.csail.mit.edu/2005/ajar/ajaw/data...

Table 6.1: Comparison between PageRank and DING – querying for resources mentioning the URI of Tim Berners-Lee

In our experiments we used the following formula for ranking an entity e in a dataset D :

$$r(e) = P(D) * P(e) * \frac{n(D)}{\sum_{D' \in \mathcal{D}} n(D')}$$

where $n(D)$ is the size of the dataset D in terms of entities and \mathcal{D} the set of datasets.

6.4 Example

Our first experiments with DING gave us encouraging results, showing a behavior comparable to our own implementation of PageRank. As shown in Table 6.1, DING returns a relevant top-k result list, favoring content from the W3C website. Such a behavior is the direct consequences of the normalization factor explained in Section 6.3.1, which takes in consideration both the size and the importance of a dataset.

6.5 Conclusion

We presented DING, a novel two-layer ranking model that both considers the hierarchical structure and the link structure of the Web of Data. DING was designed for computing popularity score of Web resources on Web-scale graph, allowing quasi real-time ranking thanks to simpler computational properties than traditional approaches,. It could also be combined with personalized local entity-rank algorithms, improving its results by using carefully selected algorithms for specific datasets.

Chapter 7

A Real-World Use Case: Sig.ma*

Sig.ma (<http://sig.ma>) is both a service and an end user application to access the Web of Data as an integrated information space. It uses an holistic approach in which large-scale semantic Web indexing, logic reasoning, data-aggregation heuristics, ad-hoc ontology consolidation, external services and responsive user interaction all play together to create rich entity descriptions. These consolidated entity descriptions then form the base for embeddable data mash-ups, machine oriented services as well as data browsing services.

7.1 Introduction

There is a strong need to demonstrate convincing applications that can exploit the capabilities of leveraging multiple, distributed, data sources when solving a task of interest to the user. Sig.ma leverages the Sindice Semantic Web Index [ODC⁺08] as well as other services to automatically integrate and make use of information coming from multiple Web sources. It does so to fulfill the following tasks:

7.1.1 Browsing the Web of Data

Starting from a textual search, the user is presented with a rich aggregate of information about the entity likely identified with the query. Queries can be about people as well as any entity type that is likely to have been described on the Web of Data (e.g. locations, name of documents, products etc). As the user visualizes the aggregate, she can follow

*This chapter is (partially) based on [CCT09]

links to other properties, switching the focus of visualization to that of neighboring entities.

7.1.2 Live views on the Web of Data: rich, embeddable, addressable

At any aggregation page, Sig.ma offers rich interactions tools to *expand* and *refine* the information sources that are currently in use as well as some *data oriented* cleanups functionalities to *hide* and *reorder* values and properties.

As a result, it is possible to interactively create curated “views” on the Web of Data about a given entity which can be then addressed with persistent URLs, therefore passed in IMs or emails, or embedded using a specific markup in external HTML pages. Views are “live” and cannot be spammed: new data will appear on these views exclusively coming from the sources that the mash-up creator has selected at creation time.

7.1.3 Structured property search for multiple entities

A user, but more interestingly an application, can make a request to Sig.ma for a list of properties and a list of entities. For example requesting “affiliation, picture, email, telephone number, [...] @ Giovanni Tummarello, Michele Catasta [...]” Sig.ma will do its best to find the specified properties and return an array (raw JSON or in a rendered page) with the requested values.

7.2 Sig.ma: processing dataflow

Sig.ma revolves around the creation of *Entity Profiles*. An entity profile – which in the Sig.ma dataflow is represented by the “data cache” storage (Figure 7.1) – is a summary of an entity that is presented to the user in a visual interface, or which can be returned by the API as a rich JSON object or a RDF document. Entity profiles usually include information that is aggregated from more than one source. The basic structure of an entity profile is a set of key-value pairs that describe the entity. Entity profiles often refer to other entities, for example the profile of a person might refer to their publications.

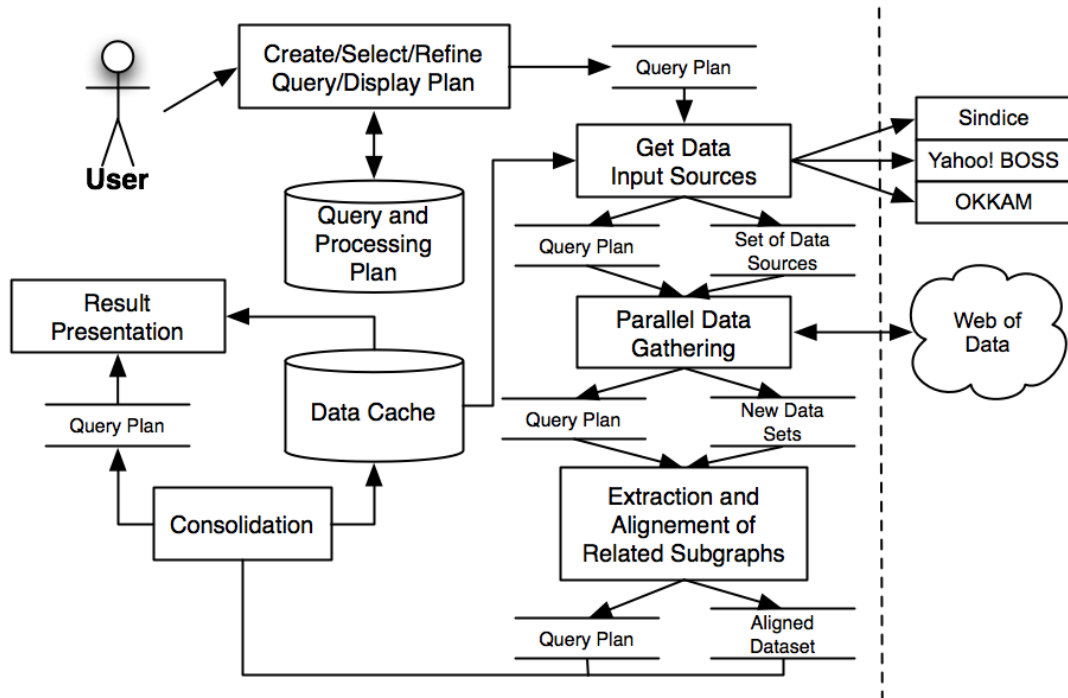


Figure 7.1: Sig.ma dataflow

A *data source*, in our terminology, is a Web document that contains structured data. Examples include RDF/XML documents, HTML pages with embedded RDFa markup, or HTML pages with embedded microformats.

There are several ways how a user can instruct Sig.ma to display an entity profile:

1. Based on a keyword or simple structured query
2. By following a hyperlink from one entity profile to another
3. By accessing a *permalink* to an entity profile (possibly created by another user) or simply by visualizing a Web page where a Javascript tag embeds an entity profile via a permalink.

The process of creating an entity profile involves the following main steps, which are described in detail later in this section:

Creation of the Sig.ma query plan: The keyword and structured queries provided by the user form the initial “Query Plan” in Sig.ma.

Data Sources selection: The query plan is expanded in a set of diverse Search Engine interrogations, which returns a list of related sources

Parallel Data Gathering: All the candidate data sources are retrieved from a Web cache, or directly from the Web for a limited number of cache misses. Structured data is extracted from each source.

Extraction and Alignment of Related Subgraphs: The structured data extracted from each source is broken down into chunks that each describe distinct entities (*resource descriptions*). Then, the set of sources is expanded based on `owl:sameAs` links and inverse functional properties found in the descriptions.

Consolidation: All the descriptions are then merged into a single entity profile, by means of various heuristics.

Result presentation: The result are presented to the user through a Web user interface, or as a JSON object/RDF document.

Source list refinement: After the entity profile is presented to the user, it can be refined by modifying the sources list. This final step generates a closed loop in the Sig.ma dataflow, which actually represent an “expansion-refinement” loop which is driven by the user input.

7.2.1 Creation of a Sig.ma query plan

The process of creating a Sig.ma query plan takes three inputs, each of which is optional:

1. A keyword search phrase
2. A number of source URLs
3. A number of resource identifiers (URIs)

The difference between the last two items is that a source URL names a document, which is accessible on the Web, and might contain descriptions of any number of entities. A resource identifier names a specific entity, but may or may not be resolvable to a Web document.

The initial user interface shown to a Sig.ma user presents an input box that allows entry of either a search phrase, or a single resource identifier. Other combinations of inputs are accessed through hyperlinks either from within Sig.ma or from a permalink.

7.2.2 Data Sources selection

The first challenge is to identify a set of initial sources that describe the entity sought for by the user. This is performed via textual or URI search on the Sindice index and yields a set of source URLs that are added to the input source URL set.

Next, a search for each resource identifier is performed in the Sindice index. The Sindice index does not only allow search for keywords, but also for URIs mentioned in documents. This allows us to find documents that mention a certain identifier, and thus are likely to contribute useful structured information to the description of the entity named by the identifier.

Now we have a list of sources that potentially describe the entity signified by the user query. The list is naturally ranked: sources directly specified in the input come first, and the other sources are ranked as returned by the Sindice index. Then, we interleave these results with the candidate list returned by the Yahoo! BOSS API³⁰, that we process to fit our peculiar scenario.

BOSS supports only text queries, thus we take advantage of it only if the user provided a search phrase. The result list we get from this service reflects the standard Yahoo! search results, so we have to filter for entries which contains some embedded metadata. To do this filtering on-the-fly, we leverage the additional functionalities provided by the integration with the Searchmonkey³¹ service: basically, we consider the given URL to be interesting if and only if their metadata extraction layer detected semi-structured content in the page.

If the source list is long, it is trimmed. The desired length is still subject to experimentation, but 25 sources seems to be a good compromise of response time, data variety, and it is still manageable in the user interface. If there is a large number of sources from a single domain, then these are dropped with preference. This ensures a larger data variety and produces what appears to be a more interesting default search result. The user interface has then a control for requesting more resources, which repeats the process with a higher source cutoff limit.

7.2.3 Parallel Data Gathering

Sig.ma operates on data collected as part of the Sindice[ODC⁺08] project.

³⁰<http://developer.yahoo.com/search/boss/>

³¹<http://developer.yahoo.com/searchmonkey/>

The Sindice infrastructure uses the RDF data model as a lingua franca that hides the syntactic differences between source formats. A set of parsers, which we are currently publishing as open source under the name *any23*³², is used to extract RDF data from those different formats. Different RDF serialization formats (RDF/XML, Notation 3, Turtle, N-Triples, RDFa) are supported, as well as several microformats (hCard, hEvent, hListing, hResume and others). Conceptually, any format for which a converter or extractor into RDF is available, can provide input for Sig.ma and Sindice.

Sindice collects data from the Web using a number of techniques: Web crawling, RDF dump indexing based on Semantic Sitemaps[CSD⁺08b], and receiving update notifications (*pings*) from sources such as `www.PingTheSemanticWeb.com` and our own ping interface.

After documents have been fetched from the Web and their structured data parts have been extracted, the structured part is stored in a highly specialized index that facilitates keyword queries as well as more advanced query forms based on triple patterns. The index, based on information retrieval technology, as well as part of the infrastructure is described in [ODC⁺08][MT08].

The structured data extracted from Web documents is also stored in the HBase-based *page repository*, which allows subsequent fast access to the documents' contents without incurring the cost of Web retrieval. It is simply a large distributed hash map that contains the content fetched from each URL that Sindice knows about, as well as additional metadata, most notably an inference closure computed over the document's RDF graph using the quarantined reasoning technique described in [DPDT08].

Data fetching and extraction represents one of the main factor which affects Sig.ma performances, so we designed the data gatherer as a parallel three-tier cache system which exploits as much as possible the Sindice infrastructure. First, a memcached server is consulted. Second, the page repository is queried. Third, an HTTP request to the source URL is performed to retrieve its contents. In the third case, the Sindice parsers and extractors are invoked to get the structured content from the source. Whatever the result, it is stored in memcached to speed up subsequent requests.

If a successful request to the Web was performed, then the source URL is sent to the ping manager module of Sindice, which ensures it will be scheduled for later fetching by the main Sindice infrastructure. This will result in adding it to the index and the page repository, including triples inferred during reasoning. Noteworthy is the actual scenario

³²<http://code.google.com/p/any23/>

in which documents are missing from the page repository, especially if users browse from one entity profile to another referred entity. In the end, it is a nice way of discovering new sources that are relevant to user interests, and a low-cost method (compared to undirected Web crawling) of increasing the Web of Data coverage of Sindice.

7.2.4 Extraction and Alignment of related subgraphs

The structured RDF graph extracted from each source is broken down into chunks (called *resource descriptions*) that each describe distinct entities. This is made easy by the use of the RDF data model. A resource description contains the outgoing and incoming RDF triples of a specific resource.

In some cases it would be desirable to include more information into a resource description. An example are geographic locations, which are often attached to a resource via a property such as `foaf:based_near`, which points to another resource, often an RDF blank node, which in turn has properties `geo:lat` and `geo:long` that give the geographical coordinates. Obviously it would be good to have the coordinates included in the resource description, even though they are only indirectly attached to the resource in question. This could be solved either by manually identifying commonly occurring cases such as the one given here, or by using generic heuristics based on graph shape, e.g. include linked blank nodes that have less than a certain number of outgoing triples.

As an example of a decomposition into resource descriptions, consider the case of a typical FOAF³³ file that describes a person. It will be decomposed into one resource description for the file's owner, one small description for each of their friends listed in the profile, and possibly one description for the FOAF document itself, containing statements about its `foaf:maker` and `foaf:primaryTopic`.

Resource descriptions are now ranked. If the resource has one of the resource identifiers from the source acquisition step, then it will immediately receive a large boost, as there is almost total certainty that it described the entity in question.

Each description will be matched and scored against the keyword phrase, considering both RDF literals and (with a lower score) words in URIs. This helps to pick out the correct resource in cases such as FOAF files, which talk about multiple people, but it is easy to select the right one given a name.

³³<http://www.foaf-project.org/>

Very small entities are slightly reduced in score, because experimental results show they are unlikely to contain interesting information, while cluttering up the source list in the user interface.

Resource descriptions below a certain threshold are removed from consideration. We now have a ranked list of descriptions that are hoped to describe the same entity. Of course, since fuzzy keyword matching is used in several places in the process, the result will often contain false positives.

A first cut of our algorithm used only the highest-ranking resource description from each source, discarding all others. This has proven to be problematic, as our ranking sometimes would score the document resource description higher than the description of the person or other entity described in the document, because both might have the same, highly salient, label. By including both, we leave the problem to the user, instead of risking the wrong pick.

If the number of highly-scoring resource descriptions is low at this point, then an attempt is made to discover additional sources, based on the RDF data we have already retrieved and established to likely describe the target entity. We obtain new resource identifiers for the target entity using four methods:

1. If the selected resource descriptions are centered on a URI (not a blank node), then this URI is considered.
2. If the resource descriptions include any `owl:sameAs` links, then the target URIs are considered.
3. If the resource descriptions include any OWL inverse functional properties (IFPs) from a hardcoded list (e.g. `foaf:mbox` and `foaf:homepage`), then a Sindice index search for other resources having the same IFP value is performed – resources having the same value for an IFP are considered equivalent under OWL inference rules.
4. By means of a query to the OKKAM service. OKKAM is an experimental service which assigns names to entities on the Web [BSB08]. OKKAM returns resource identifiers along with a confidence value. Any resource identifiers whose confidence value exceed a certain threshold are added to the set of input resource identifiers. We observe that currently the number of entities that are reliably recognized by the OKKAM service is still low, as not many OKKAM ids can be found out in the Web, so this step will often not produce a result. In the case where it returns results however, it is a high-quality identifier that is likely to contribute relevant results to the next steps.

Any resource identifier discovered using these methods will be added into the *Query plan*, which will be then examined in the *refinement* step.

7.2.5 Consolidation

All selected resource descriptions are merged into a single entity profile. This simply means that all key-value pairs from all resource descriptions are combined into a single description. A reference to the original source is kept for each value.

Often different properties (keys in the key-value pairs that describe the entity) express the same thing. The next step is to consolidate the potentially large and chaotic list of properties into a simpler list that is more meaningful to the user. In RDF, properties are named with URIs; we consider only the last segment (“local part”) of the URI. By convention, this local part is usually a good name for the property, written in CamelCase or with underscores or dashes, which are converted back into a more readable string consisting of space-separated words. In the future, we should also check the definition of the property (obtainable by dereferencing its URI, and often already in the page repository) for an `rdfs:label`.

The next step is to treat both incoming triples (of the shape “other-entity - relationship - our-entity”) and outgoing triples (of the form “our-entity - relationship - value” or “our-entity - relationship - other-entity”) as outgoing triples. This is done simply by flipping the incoming triples around, and adding an *inverse flag* to the relationship. For example, *A creator B* becomes *A is creator of B*.

Next, we apply some simple English-language heuristics on the property names. This is based on observing properties typically used in the wild. The heuristics are:

- remove initial “has” (e.g. in “has title”)
- remove initial “holds” (e.g. in “holds role”)
- remove initial “gives” (e.g. in “gives presentation”)
- remove final “of” and flip property (e.g. in “member of”)
- remove surrounding “is ... of” and flip property (e.g. in “is topic of”)

Next, we apply a manually-compiled list of approximately 50 preferred terms. For example, we replace all of the following property names with the preferred term “Web page”: work info homepage, workplace homepage, page, school homepage, Weblog, Website, public home page, url, Web. Special attention has been given to terms that can

be used in customized ways in the user interface: labels, depictions (images), short descriptions, Web links.

Next, we drop a number of properties that are of little value in an end-user interface, e.g. `foaf:mbox_sha1sum` or `rdfs:seeAlso`.

After consolidation, properties are ranked. We use a simple ranking metric: the number of sources that have values for the property. This will push generic properties such as “label” and “type” to the top. The number of distinct values for the property is also factored in: properties where many sources agree on one or a few values (as observable e.g. with a person’s name or homepage) receive a boost.

Value labeling

For key-value pairs where the value is not a literal value (such as a name or a date), but a reference to another resource (usually by URI), a best-effort attempt is made to retrieve a good label for the resource:

1. The original source RDF graph in which the resource was found is examined for typical label property, such as `foaf:name` or `dc:title` or `rdfs:label`.
2. If nothing is found, and it is a URI, it will be resolved against the page repository or the Web, as described above in *Fetching and Parsing*.
3. If nothing is found, and it is a URI, then the last part of the URI will be used in a manner similar as described above for property names.

A typical entity profile can refer to dozens or hundreds of other entities, so this is an expensive process. Yet it is very important for a decent user experience. We consider showing a quality label much more desirable than showing a URI or some fragment of a URI, and it is practically a requirement to allow a user to make sense of the entity profile. The labels also feed into further Sig.ma requests: when a user wants to follow a link to another entity, then the underlying resource identifier(s) as well as the label are used to submit a new Sig.ma request in order to produce the linked entity’s profile.

To achieve responsiveness despite the large required number of page repository or Web requests, the initial profile displayed to the user will only contain labels produced by method 1 and 3 above. The additional labels are retrieved while the user already sees the profile and will be displayed incrementally using AJAX requests.

Value consolidation

If a property has several values with identical or very similar labels, then they are collapsed into one value to improve the visual presentation. For example, several sources that describe a scientist can state that they have authored a certain paper, using different identifiers for the paper. Without label-based consolidation, the paper would appear several times because the identifiers are not the same. After label consolidation, it appears only once. Both identifiers are retained internally. A click on the paper link will cause a new Sig.ma search that has the label and both of the URIs as input.

Since labels are retrieved and displayed incrementally, the value consolidation has to be performed in the same fashion.

7.2.6 Source list refinement

After the entity profile is presented to the user, they can refine the result by adding or removing sources.

Almost any entity profile initially includes some poor sources that add noise to the results. Mixed into the desired entity profile are other entities that have the same or a similar name, or that for other reason ranked highly in the text search portions. The user interface allows quick removal of these. Widgets for source removal exist in the list of sources, and next to each value that is displayed in the profile. If the profile shows a poor label or unrelated depiction for the entity, a quick click will remove the offending source, and the next-best label or depiction will automatically take its place if present.

Since the profile is based on a fixed number of resources, it will often show only a subset of what is known about the entity. There is a button for including more sources in the source list. After having retrieved more sources and run the usual processing, it will simply mix the results into the profile.

We include also widgets that facilitate retrieval of more information of a specific kind, which show to be useful, for example, when a person's entity profile shows several academic publications that come from sources on a certain domain. Thus, it is likely that fetching more sources from that domain will yield more publications.

7.3 Test driving Sig.ma: example user interactions

In this section we will illustrate how Sig.ma presents itself to the end user and comment on specific results. These comments highlight some of Sig.ma possibilities which are then compared with that of other systems in the next chapter.

7.3.1 Querying Sig.ma: Giovanni Tummarello

In case of researcher “Giovanni Tummarello”, there exist plenty of data sources available: RDF sources such as DBLP, Ontoworld, SemanticWeb.org but also Microformat sources provided by BOSS such as Tummarello’s public Facebook and LinkedIn profiles which, for instance, add more pictures to the mashup. Particularly rich sources such as the RDF coming from the DERI institute team page³⁴ add data such as his work phone number, some of the publications and related projects etc. As ambiguity on the name is low, pressing “Add More Info” button returns many more relevant results which provide social contacts alternative affiliations from previous employers and more.

The result of a 60 sources aggregation is shown partly in Figure 7.2.

Following the Web of Data browser paradigms, most of the results are clickable and lead to further information discovery. For example, clicking on the paper titled “Exposing Large Dataassets with Semantic Sitemaps” reveals a Sig.ma coming from mostly 3 Websites with complementary information such as the coauthors, the tags and alternative locations. Interestingly, a reference to a review of the paper from the Semantic review site (Revyu.com) is also given, see Figure 7.3.

7.3.2 Querying Sig.ma: Eyal Oren

While returning plenty of relevant information, the Sig.ma for researcher Eyal Oren shows the weakness of kickstarting the search with a simple text query but at the same time the importance of user feedback and interaction. A 50 source Sig.ma shows several correct pictures but also some incorrect ones. Similarly it shows a comment saying that he is an “Israely born america actor”, the “birth place” is Tel Aviv and his birthdate on the 11-11-1975.

³⁴<http://www.deri.ie/about/team/>



Figure 7.2: Sig.ma screenshot for the query “Giovanni Tummarello” when expanded to 60 sources – space usage is optimized using the capabilities of the Web interface itself (e.g. property reordering and visualization options)

Exposing Large Datasets with Semantic Sitemaps

alternate:	Exposing Large Datasets with Semantic Sitemaps [9]
accepted by:	ESWC2008 [1,4]
about:	Datasets [1,4] Sitemaps [1,4] Crawling [1,4] Search [1,4] Provenance [1,4]
creator:	Renald Delbru [1,4,6] Richard Cyganiak [1,4,16,17,18] Stefan Decker [1,4,19,20] Holger Stenzhorn [1,4] Giovanni Tummarello [1,4,12,15]
creation date:	2009-04-23T14:36:53+02:00 [1]
identifier:	http://revyu.com/things/eswc-2008-paper-exposing-large-sitemaps [1,4,6,8,10,12,15,16,17,19,20] http://data.semanticweb.org/conference/eswc/2008/papers/356 [1,4,6,8,10,12,15,16,17,18,19,20] http://sw.deri.org/2007/07/sitemapextension/ [8,10] http://semanticweb.org/wiki/Exposing_Large_Datasets_with_Semantic_Sitemaps [9]
label:	Exposing Large Datasets with Semantic Sitemaps [1,4,6,12,15,16,17,18,19,20]
review:	Review of Exposing Large Datasets with Semantic Sitemaps (Article) , by Chris Bizer [8,10]
type:	Paper [1,4] Ontology [1] Subject [1,4,6,12,15,16,17,18,19,20] http://swrc.ontoware.org/ontology#InProceedings [8,10]
web page:	http://semanticweb.org/wiki/Exposing_Large_Datasets_with_Semantic_Sitemaps [1,4,6,12,15,16,17,18,19,20]

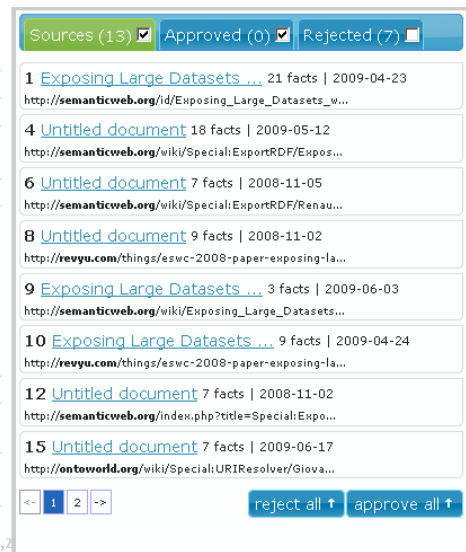


Figure 7.3: Sig.ma screenshot for a paper. Information comes from multiple RDF sources also including a review from Revyu.com

Interestingly, to remove these and many other false statements, the user must only be able to locate a single false value (e.g. notably the picture or the statement about him being an actor) and select the pop-up option “remove source” to remove the source itself and all the metadata associated. With a few clicks (5 approve and 3 reject) we were able to refine a Sig.ma that can be safely expanded and becomes very descriptive³⁵.

7.3.3 Querying Sig.ma: Trento

Querying for the Italian city “Trento” returns data from DBpedia, Geonames, and others. Possibly the most interesting aspect of this query, however, is how multiple sources that describe formally different entities can be put together in the mashup, resulting in a practically useful unified profile.

As an example, the city of Trento is different from the entity “province of Trento”, e.g. in Wikipedia. Yet, using the two DBpedia entries at the same time in Sig.ma yields results which are meaningful to a human user. For example, the final entity profile about Trento would contain also an overall map of Italy with the Trento province highlighted and the name of many neighboring towns (due to the province source), which in certain contexts can be considered relevant pieces of information for the Trento entity.

7.4 Related works and comparative evaluation

In this section we illustrate several works which are related to Sig.ma and use their description to form a comparative evaluation table, to be used in a later qualitative discussion.

7.4.1 Closely related works: Semantic Web Aggregators

So far two notable approaches have been demonstrated which make heavy use of Semantic Web technologies. In 2006, the SWSE Semantic Search engine demonstrated large scale aggregation of Semantic Web data [HHD⁺07a]. Possibly for the first time, thanks to the use of a scalable cluster infrastructure, SWSE could collect and contain a significant part of the data available on the Semantic Web so to be able to aggregate information pages with elements coming from multiple sources talking about the same

³⁵<http://sig.ma/search?pid=8aa3d50637670e46b4647141492e11c7>

entity. To perform such entity information consolidation, SWSE strictly adhered to the theoretical rules of the Semantic Web: consolidation via reuse of the same identifier across different data sources and several forms of lightweight reasoning such as explicit SameAs statements, OWL Inverse Functional Properties etc. [HHD07b].

As a result, the engine displayed two peculiar side effects, as discussed in [HHD⁺07a]. On the one hand, for each textual query indicating an entity, multiple Semantic Web Entities would be displayed, mostly due to the very scarce reuse of URIs across different sources. As an example, a query for “Giovanni Tummarello” returns 44 RDF nodes, each with different informations attached. On the other hand, information could be wrongly aggregated due to errors or different interpretations of semantic properties across different datasets. For example, if a property, e.g `foaf:homepage`, is defined as Inverse Functional Property, then all the entities that had this value set to null would share the same, erroneous aggregated set of statements. Built on top of the base SWSE aggregation and query engine, VisiNav was recently demonstrated³⁶, which adds Faceted Browsing capabilities.

A completely different approach is that of the Tim Berners-Lee initiated Tabulator project[LCC⁺06]. In Tabulator, the idea is to leverage the linked data principle: data published on the Web in RDF should have dereferencable identifiers (URIs). If this is the case, then the identifier doubles as a network location, so it is possible to fetch the description of the entity by resolving, e.g. with an HTTP lookup, the identifier itself.

In Tabulator, once the user enters a starting resolvable URI, the entity description is fetched and the contained statements are displayed, typically in form of a statement tree. The interesting part comes when the user decides to investigate on one of the leaves of said tree. If the leaf is itself a resolvable URI, then the description of the URI is also fetched and the new statements coming from the new location are therefore added to the old ones. This, in practice, creates a live data mash up driven by how the user decides to explore the graph.

This approach, while fascinating in theory, suffers from certain shortcomings. In theory, for tabulator to consistently return information, the resources should always contain backlinks, that is, all the statements that are known to be in common with other dereferencable resources. This is clearly a daunting maintenance task which is much against the nature of the Web and the way people create their dataspaces. In Sig.ma, this role is fulfilled instead by the Sindice index. A further shortcoming is the dependence on identifier reuse. If a dataset doesn't mention explicitly the URI for the

³⁶<http://visinav.deri.org/>

same conceptual resource in another dataset, no browsing and data mash up can be possible.

7.4.2 Other notable entity-based information services

There are a number of further information services which we feel should be somehow compared with Sig.ma, given their ability to fulfill comparable user-driven information gathering tasks.

First of all, already *Google* itself can often return all the information required by the end user. Google is notably good, for example, at showing snippets of text with phone numbers or addresses in them when looking up a name of a person or a business.

Then, many specialized entity based search engines exist. For our comparison we believe that *people search engines* are probably the most interesting. Some of these work mostly by query time Web search and shallow textual mining (E.g. Pipl), while others keep an index and ask for people to confirm profiles (Zoominfo, Spock). Generally the information provided includes links to Web pages and fields like Title, Affiliation or previous employers and possibly one or more pictures.

Very recently released, *Google Squared* allows the creation of tables to compare structured fields describing multiple entities belonging to the same category named by the user. For example a search for “Italian cities” returns some names with pictures, and some additional information. The query can be changed and yet added to the square, e.g. adding “French Cities” to the mix, and a single entity can be added, e.g. typing the name.

Needless to say, *Wikipedia* itself, with its semantically structured derived project *DBPedia*, provide clean structured information about million of notable entities. Due to its encyclopedic scope policy, however, there are limitations on which entities can be listed and which specific information can be added about them.

Based on text-structured relationship mining from textual information, Evri provides information aggregates about entities, e.g. related news, pictures and base structured facts. While Evri seems to operate over live data coming from RSS feeds and other Web sources, it limits its operations on what seem to be the entities listed in Wikipedia.

7.4.3 Comparative evaluation

Sig.ma is a system where the human and the machine capabilities are complementary and augment themselves to solve a task at hand: the user provides iterative indications of what the entity should or should not look like, while the algorithms automate further information finding, revision and consolidation.

In literature these systems are usually described as Mixed Initiative [Hor99]. Meaningful evaluation of mixed initiative systems is in general not a straightforward task. For example, often these systems are composed by machine parts which are simple per se, but are used in strategic ways which are enabling for an end user.

A way that has been employed to evaluate these system distinguishes between:

- The evaluation of the machine part
- The holistic evaluation of the salient characteristics and the value added in the human machine integration

In case of Sig.ma, the innovative machine parts are mostly the capabilities of structured data search of the Sindice semantic index, the evaluation of which is outside the scope of this paper.

It is interesting to notice, however, that Sindice cannot be evaluated strictly in the same way as a normal search engine would, e.g. notably via precision and recall, as it can operate in ways which are much more similar to a DBMS than to an Information Retrieval engine. A query for an Inverse Functional property, for example, will return all and only the entities which precisely match the query. With respect to the other machine parts, such as the consolidation of entities based on property values, Sig.ma currently uses simple text distance algorithms which provide a reasonable level of performance but are subject to errors as expected.

In this section we will proceed to an evaluation of the high level characteristics of Sigma. We do so by self scoring Sigma as well as the previously listed related works along several conceptual dimensions. These are labeled as follows:

Generic: Can the system address any kind of entity? People search get the worse score while systems which are based on predefined lists of entities can be considered only partially good.

Provides Metadata: Can the system be used to have reusable metadata about the entity? Semantic Web based systems and Freebase excel on this as they can provide

RDF metadata directly. Providing metadata is also key to allow direct entity comparison.

External Docs: Does the system make use of or points to relevant of textual documents on the Web? In this case nothing can be compared to using Google directly. Some systems, Sig.ma included, provide relevant Web sources.

External Metadata: Does the system make use of the Web Metadata? This is mostly an exclusive characteristic of Semantic Web tools, with Google having just an initial support for a limited vocabulary and Freebase being updated by manual, centralized imports of external datasets.

User can expand: Can the user expand the entity profile by requesting more? No other tool except Google, partially Google Squared and Sig.ma give this functionality.

Updatability: How easy is it to update? Wikipedia and Freebase allow easy updates. Google, by design, does not allow anyone to easily enter the first pages when an entity is looked up. LOD browsers only visualize information as provided and sources only when directly linked. Sig.ma excels at this as it reflects the status of the Sindice index, which accepts and index new sources within short intervals.

Clean Output : A subjective evaluation of the amount of unrelated or irrelevant information given to the end users. Encyclopedic sites excel at this while Semantic Aggregation tools have the higher risk of “noise”.

Spam immune: Can the system be easily spammed? Somehow dual of *Updatability*: LOD browsers do not typically show information not directly linked by the original source. On the other hand, Semantic Search capabilities means that by using the right properties it would be possible to spam very effectively. Sig.ma can however perform reasonably given that it inherently uses source ranking concepts and Top K results from external services.

Response time: As Sig.ma and LOD browsers are the only systems which perform online live data and service lookups, they are the worse performing in the group.

These results are summarized in table 7.4.

This comparison highlights that approaches based on semantic Web technologies offers several interesting features. Not surprisingly however, these also currently exhibit potential shortcomings (e.g. noise, response time). Yet, among the presented systems, Sig.ma shows what is seemingly a novel mix characteristics.

	Generic	Provides Metadata	Extern docs	Extern metadata	User can expand	Updatability	Clean output	Span immune	Response time
Ppl search	-	+	+	=	-	=	+	+	++
Google	++	-	++	=	++	=	=	+	++
Google SQ	++	+	+	-	+	+	=	=	+
Wikipedia	=	=	-	-	=	++	++	+	++
Freebase	=	++	=	=	=	++	++	+	++
Evri.com	=	+	+	-	=	=	+	+	++
LOD browsers	++	++	-	++	-	-	+	++	=
SWSE	++	++	-	++	=	+	-	-	+
Sigma	++	++	+	++	++	++	-	=	=

Figure 7.4: Sig.ma evaluation table

7.5 Sig.ma implementation and performances

The Sig.ma processing workflow is implemented mainly by two layers:

1. a Java backend, wrapped in a Webapp hosted in a Tomcat application server
2. a full MVC stack built in JavaScript

The backend exposes a mainly RESTful API, which is used by the JavaScript layer through AJAX calls. It also represents a facade for our caching systems (memcached, HBase, etc.) and for other minor services.

The JavaScript layer, instead, follows the Model-View-Controller architectural pattern, taking care of page rendering, of some final consolidation functions and of the Data Selection logic.

Decoupling the system in this way was necessary to smooth the user experience, because it let us render the page incrementally. Yet, it gave us some nice benefits while implementing the JSON/RDF API. Thanks to Rhino³⁷, we are able to run the same logic both on client and server-side.

³⁷<http://www.mozilla.org/rhino/>

Averaged performance tests show that Sig.ma API takes around 1 second per 10 processed sources when serving RDF output, thus skipping the page rendering overhead. Most notably, Sig.ma seems to perform linearly with the number of sources, averaging a response time of 11 seconds per 100 processed sources. This result gives us a certain confidence about possible improvements, mainly in the Rhino integration layer.

7.6 Conclusion

While Sig.ma is by no mean the first data aggregator for the Semantic Web, its contribution is to show that exciting possibilities lie in a holistic approach for data discovery and consolidation. In Sig.ma, elements such as large scale semantic Web indexing, logic reasoning, data aggregation heuristics, ad hoc ontology consolidation and, last but not least, user interaction and refinement, all play together to provide entity descriptions which become live, embeddable data mash ups.

When Sig.ma automatic source selection fails, the user can intuitively recognize this by spotting, for example, a wrong entity type or wrong data statements or by seeing that multiple sources confirm one statement while others come from only one source. When this happens, the user can easily eliminate the wrong datasource, thus providing Sig.ma with more elements for further information cleansing and augmentation by successive queries.

The way by which the user can quickly adapt the mashup possibly for her intended target goal inspires the general thought that a little semantic might in fact go a long way, *at least* in making it easy for a *human* to do the last step (and validate, by accepting, the final list of sources “crystallized” in the permalink).

As a result, we believe that Sig.ma style embeddable mashups could be effective at providing incentives for publishing Semantic Web data: data elements from any Web source could potentially be shown on any Web page, possibly rewarding the data provider with back link and branding opportunity. Similarly, users themselves who would use Sig.ma mashups, e.g. to display CV like information on one’s homepage, would have a reason to ask for the creation of more semantic data, e.g. by a conference Web site about a published paper.

Chapter 8

Conclusions

In this thesis we presented a model for the Web of Data and then illustrated how this is reflected in the building blocks of Sindice, an infrastructure for large-scale processing of semantic data.

While there are already some notable efforts ([HHD⁺07a], [DFJ⁺04], [BCSW07], [DBG⁺07]) in building Semantic search engines, a unified structure and vision about them has still to be consolidated in literature. This encouraged us in exploring several novel tradeoffs while developing Sindice.

For example, with respect to DBMS and IR systems, SIREn tries to get the best of both worlds allowing semi-structural queries while retaining many desirable IR features: single inverted index, effective caching, top-k queries and efficient index distribution over shards.

In SIREn, the tradeoff is query complexity vs. scalability. It might be arguable that an indexing approach limiting itself to relatively simple queries can still be sufficient to solve interesting use cases, not needing necessarily the full expressiveness of SPARQL .

Similarly, the reasoning model that we propose offers sensible inference capabilities for Web published RDF data while is limited to the expressiveness of RDFS plus certain OWL Lite rules.

Sindice is a service which has run continuously over the past two years, proving day after day its reliability and performances. Considering the limited amount of resources we had available (both regarding hardware and networking), this could come as a surprise. However, employing “Cloud-friendly” technology stacks (refer to Appendix A and B) granted us the right mix of agility in development and high-level performances, let-

ting us focus more on our domain-specific challenges rather than mere implementation issues.

At least among the community of Web of Data enthusiasts, adoption of Sindice is remarkable, with a good number of external services blossoming thanks to our APIs (e.g.: <http://sameas.org>) and tens of thousands queries served per day from a commodity server. Yet, developing credible applications on top of these functionalities is probably the only way to prove the validity of the above choices. To do so, and also as an inspiration for others to try, we explored the actual usefulness of our APIs by building demonstrators. One such project, Sig.ma, seems to indicate that in fact simple semantic queries and services, when driven by an appropriate logic, can already bring interesting and useful results.

8.1 Future Works

Among the possible improvements, one that would affect both Sindice and Sig.ma would be a batch data consolidation service. It would not only consolidate the entity profiles contained in the Sindice data repository (thus improving the recall factor), but would also greatly benefit the Sig.ma user experience. That is to say, in case of ambiguities due to an under-specified query, the user would not need to manually clean the profile by means of rejecting resources, but rather she will be able to select from a disambiguation page the profile she was looking for. Speaking about performance, a major bottleneck due to the complex client-side logic which the browser is currently handling would practically disappear. This result could be obtained by serving the content directly from the consolidated index, avoiding all the client-side consolidation at runtime.

Regarding SIREn, future works will concentrate on enabling path-based queries and relational queries, e.g., by employing an additional property index, as well as how to extend the traditional scoring function to take into account RDF structural elements.

Several possible paths of improvement are foreseeable for DING. Much work still has to be done in the area of automation of graph structure recognition. This would allow better match the local ranking algorithm to the graph, vastly improving the performance of the ranking on a heterogeneous Web. Further optimization and fine-tuning could also be done on the combination of local and dataset ranks to achieve better balancing between smaller and larger datasets.

Appendix A

The Apache Hadoop Project*

“Tape is dead, disk is tape, flash is disk, RAM locality is king”

— Jim Gray

Apache Hadoop is a Java software framework that supports data intensive distributed applications. It enables applications to work with thousands of nodes and petabytes of data. Being an open-source effort, it is a collection of related subprojects hosted by the Apache Software Foundation¹, which all share the same goal of building a technology stack currently dubbed “*Apache Cloud Stack*”.

Although Hadoop is best known for *MapReduce* [DG04] and its distributed filesystem (*HDFS*), the other subprojects provide complementary services, or build on the core to add higher-level abstractions. Follows a brief description of the components which are relevant to this thesis:

Core A set of components and interfaces for distributed filesystems and general I/O (serialization, Java RPC, persistent data structures).

MapReduce A distributed data processing model and execution environment that runs on large cluster of commodity machines.

HDFS A distributed filesystem that runs on large clusters of commodity machines.

HBase A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).

*This chapter is (partially) based on [Whi09]

¹<http://hadoop.apache.org>

Pig A dataflow language and execution environment for exploring very large datasets. Pig runs on HDFS and MapReduce clusters.

Hive A distributed data warehouse. Hive manages data store in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to MapReduce jobs) for querying the data.

ZooKeeper A distributed, highly available coordination service. ZooKeeper provides primitives such as distribution locks that can be used for building distributed applications.

Appendix B

Apache Lucene and Solr*

B.1 Apache Lucene

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Among its features:

Scalable, High-Performance Indexing

- over 20MB/minute on Pentium M 1.5GHz
- small RAM requirements – only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

Powerful, Accurate and Efficient Search Algorithms

- ranked searching: best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g., title, author, contents)
- date-range searching
- multiple-index searching with merged results
- allows simultaneous update and searching

Available as Open Source software under the Apache License which lets Lucene to be used both in commercial and Open Source programs

*This chapter is (partially) based on documentation available at <http://lucene.apache.org/>

B.2 Apache Solr

Apache Solr is an open source enterprise search server with a web-services like API. It is based on the Lucene Java search library and runs in Java servlet container such as Tomcat² or Resin³. Among its features:

- optimized for high volume web traffic
- standards based open interfaces - XML and HTTP
- comprehensive HTML administration interfaces
- efficient replication to other Solr search servers
- extensible plugin architecture
- support for dynamic faceted browsing and filtering
- highly configurable and user extensible caching
- fast incremental updates and snapshot distribution
- distributed search with sharded index on multiple hosts
- multiple search indices

²<http://tomcat.apache.org>

³<http://www.caucho.com/resin>

*“It meant the world to hold a bruising faith
But now it’s just a matter of grace. . .”*
— Billy Corgan (*To Sheila*)

Bibliography

- [AFG⁺09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.
- [BCSW07] Holger Bast, Alexandru Chitea, Fabian Suchanek, and Ingmar Weber. ESTER: efficient search on text, entities, and relations. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 671–678, New York, NY, USA, 2007. ACM.
- [BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In I. Horrocks and J. Hendler, editors, *Proceedings of the First International Semantic Web Conference*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer Verlag, July 2002.
- [BL06] T. Berners-Lee. Linked data. W3C Design Issues, July 2006.
- [BSB08] Barbara Bazzanella, Heiko Stoermer, and Paolo Bouquet. An entity name system (ENS) for the semantic web. In *Proceedings of the European Semantic Web Conference*, 2008.
- [BVT⁺02] Kevin Beyer, Stratis D. Viglas, Igor Tatarinov, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of Data*, pages 204–215, New York, NY, USA, 2002. ACM.
- [CBHS05] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM.

- [CCT09] Richard Cyganiak, Michele Catasta, and Giovanni Tummarello. Towards ECSSE: live Web of Data search and integration. In *Proceedings of the Semantic Search 2009 Workshop*, 2009.
- [CDG⁺06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A distributed storage system for structured data (awarded best paper!). In *OSDI*, pages 205–218. USENIX Association, 2006.
- [CDTT09] Michele Catasta, Renaud Delbru, Nickolai Toupikov, and Giovanni Tummarello. *Semantic Web Information Management: A Model-based Perspective.*, chapter Managing Terabytes of Web Semantics Data. Springer, 2009.
- [CSD⁺08a] Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. Semantic Sitemaps: Efficient and Flexible Access to Datasets on the Semantic Web. In *Proceedings of the 5th European Semantic Web Conference*, volume 5021 of *Lecture Notes in Computer Science*, pages 690–704. Springer, June 2008.
- [CSD⁺08b] Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. Semantic sitemaps: Efficient and flexible access to datasets on the semantic web. In *Proceedings of the European Semantic Web Conference*, 2008.
- [dBG⁺07] M. d’Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Characterizing Knowledge on the Semantic Web with Watson. In *EON*, pages 1–10, 2007.
- [Dec02] Stefan Decker. Semantic web methods for knowledge management [online]., 2002.
- [DFJ⁺04] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 652–659, New York, NY, USA, 2004. ACM.
- [DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design and Implementation*, pages 137–150, 2004.
- [DH08] Stefan Decker and Manfred Hauswirth. Enabling networked knowledge. In Matthias Klusch, Michal Pechoucek, and Axel Polleres, editors, *CIA*,

- volume 5180 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.
- [DPDT08] Renaud Delbru, Axel Polleres, Stefan Decker, and Giovanni Tummarello. Context dependent reasoning for semantic documents in sindice. In *In Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2008.
- [DPTD08] Renaud Delbru, Axel Polleres, Giovanni Tummarello, and Stefan Decker. Context dependent reasoning for semantic documents in sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008)*, October 2008.
- [DTCT09] Renaud Delbru, Nikolai Toupikov, Michele Catasta, and Giovanni Tummarello. Siren: a semantic information retrieval engine for the web of data. In *Submitted to the 8th International Semantic Web Conference (ISWC 2009)*, 2009.
- [EAT04] Nadav Eiron, Kevin S. McCurley John A., and Tomlin. Ranking the web frontier. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 309–318, New York, NY, USA, 2004. ACM.
- [EM08] Orri Erling and Ivan Mikhailov. Towards web scale rdf. In *4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008)*, October 2008.
- [GMF04] R. V. Guha, R. McCool, and R. Fikes. Contexts for the Semantic Web. In *International Semantic Web Conference*, pages 32–46, 2004.
- [Guh92] R. V. Guha. *Contexts: a formalization and some applications*. PhD thesis, Stanford, CA, USA, 1992.
- [Hay04] P. Hayes. RDF Semantics. W3C Recommendation, World Wide Web Consortium, February 2004.
- [HD05] Andreas Harth and Stefan Decker. Optimized index structures for querying rdf from the web. In *LA-WEB*, pages 71–80, 2005.
- [HHD⁺07a] Andreas Harth, Aidan Hogan, Renaud Delbru, Juergen Umbrich, Sean O’Riain, and Stefan Decker. Swse: Answers before links! In *Semantic Web Challenge 2007, 6th International Semantic Web Conference*, 2007.
- [HHD07b] Aidan Hogan, Andreas Harth, and Stefan Decker. Performing object consolidation on the semantic web data graph. In *Proceedings of 1st I3: Identity, Identifiers, Identification Workshop*. I3, 2007.

- [Hor99] Eric Horvitz. Principles of mixed-initiative user interfaces. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 159–166, New York, NY, USA, 1999. ACM.
- [HUHD07] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, volume 4825 of *Lecture Notes in Computer Science*, pages 211–224. Springer Verlag, November 2007.
- [LCC⁺06] Tim B. Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *In Proceedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06)*, page 06, 2006.
- [MBS06] A. Miles, T. Baker, and R. Swick. Best Practice Recipes for Publishing RDF Vocabularies. Technical report, 2006.
- [McB02] Brian McBride. Jena: A semantic web toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.
- [MF03] J. Mayfield and T. Finin. Information retrieval on the Semantic Web: Integrating inference and retrieval. In *Proceedings of the SIGIR Workshop on the Semantic Web*, August 2003.
- [MT08] P. Mika and G. Tummarello. Web semantics in the clouds. *Intelligent Systems, IEEE*, 23(5):82–87, 2008.
- [Neu94] B. Clifford Neuman. *Scale in distributed systems*, pages 463–489. IEEE Computer Society, Los Alamitos, CA, 1994.
- [NW08] Thomas Neumann and Gerhard Weikum. RDF-3X - a RISC-style Engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.
- [ODC⁺08] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *Int. J. of Metadata and Semantics and Ontologies*, 3:37–52, November 10 2008.
- [PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.
- [PFH06] A. Polleres, C. Feier, and A. Harth. Rules with contextually scoped negation. In *3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *LNCS*, Budva, Montenegro, June 2006. Springer.

- [SCCS09] Haw Su-Cheng and Lee Chien-Sing. Node Labeling Schemes in XML Query Optimization: A Survey and Trends. *IETE Technical Review*, 26(2):88, 2009.
- [tH05] H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2-3):79–115, 2005.
- [TUD⁺09] Nickolai Toupikov, Juergen Umbrich, Renaud Delbru, Michael Hausenblas, and Giovanni Tummarello. DING! Dataset Ranking using Formal Descriptions. In *Linked Data on the Web Workshop (LDOW09)*,. *18th International World Wide Web Conference (WWW09)*, Madrid, Spain, 2009.
- [Whi09] Tom White. *Hadoop – The Definitive Guide*. O’Reilly Media, Inc., 2009.
- [ZLZ⁺07] Lei Zhang, Qiaoling Liu, Jie Zhang, Haofen Wang, Yue Pan, and Yong Yu. Semplere: An IR Approach to Scalable Hybrid Query of Semantic Web Data. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, volume 4825 of *Lecture Notes in Computer Science*, pages 652–665. Springer Verlag, November 2007.

List of figures

2.1	The three-layer model of the Web of Data	8
3.1	Overview graph	10
5.1	In these graphs, oval nodes represent resources and rectangular ones represent literals. For space consideration, URIs have been replaced by their local names.	26
5.2	The SIREn data model	27
5.3	SPARQL queries and their SIREn interpretation	29
5.4	Dark dots are Sesame commit time records while gray dots are SIREn commit time records	32
7.1	Sig.ma dataflow	41
7.2	Sig.ma screenshot for the query “Giovanni Tummarello” when expanded to 60 sources – space usage is optimized using the capabilities of the Web interface itself (e.g. property reordering and visualization options)	51
7.3	Sig.ma screenshot for a paper. Information comes from multiple RDF sources also including a review from Revyu.com	51
7.4	Sig.ma evaluation table	57

List of tables

5.1	Quad patterns covered by outgoing relations and their interpretation with the SIREn operators. The ? stands for the elements that are retrieved and the * stands for a wildcard element.	29
5.2	Indexing time in minutes per dataset and system	31
5.3	Querying time in seconds and number of hits for the 10 billion triples benchmark	32
6.1	Comparison between PageRank and DING – querying for resources mentioning the URI of Tim Berners-Lee	38