EXPLORING LINKED DATA AT WEB SCALE

Andreas Harth

under the supervision of Prof. Dr. Stefan Decker
of the Digital Enterprise Research Institute.

A dissertation submitted to the National University of Ireland, Galway
for the degree of Doctor of Philosophy.

April 2010

Abstract

Universities, government agencies, companies, and individuals are increasingly releasing torrents of data encoded in structured formats. At the same time, the web is evolving from a medium comprising documents to a medium comprising *data* – structured descriptions of objects with defined attributes and connections to other objects. Traditional search models do not fully exploit the potential that structured datasets – some of them collaboratively-edited – offer for searching and browsing.

This dissertation investigates challenges that arise during collection, integration, analysis, and exploration of large structured datasets. We experiment with several publicly available datasets, one of which originates from tens of thousands of disparate and autonomous web sources, and develop a universal method for interactive exploration of such datasets independent of domain or schema. The overall architecture of the system includes many of the traditional search engine components, but the unique characteristics of collectively-edited structured datasets require application of significantly refined techniques inside each component. The dissertation covers work on the overall architecture and implementation of the system, including a crawler for data acquisition, optimised index structures and query processing for fast response times, a ranking component to prioritise data items and data sources for display, and a user interface to allow for incremental construction of complex queries which return structured data to the user. Users can subsequently process results data in desktop applications or online services.

With the rising volume of data available to the public, we foresee broad applications of the methods investigated and hope to raise transparency in data published on the web by putting powerful analysis methods into the hands of everyone.

Contents

Chapter 1

Introduction

Man in the electronic age has no possible environment except the globe and no possible occupation except information-gathering.

Herbert Marshall McLuhan

1.1 The Age of Networked Knowledge

Technological advances have made large quantities of electronic data available to man. Universities and research laboratories collect and release vast amounts of data derived from observations of natural phenomena or experiments. Governmental organisations assemble and publish detailed statistical information about a plethora of indicators. Even ordinary people share personal information via social networking sites and blogs. The challenge is to turn the deluge of online data into actionable knowledge useful for research, policy-making, or personal insight.

Curating, maintaining, publishing, and preserving high-quality databases requires significant resources in terms of time and money [Nat08]. The lack of a common set of standards and best practices for publishing structured data means that each data producer has to craft domain-specific schemas and methods. As a result, data published by different sources differ considerably in structure; often, data analysts have to manually decipher the meaning of entries in data tables. Given the vastly different methods for encoding and protocols for publishing, integration of data from multiple sources is a laborious task. Automatic interoperation between data sources has only been achieved in narrowly specified domains, without the ability to easily re-use and re-purpose data for applications other than the ones intended.

Often, the data publishers also provide domain-specific processing and displaying tools for interacting with structured data in a given schema, aggravating the cost and effort for data publishing. Biologists can choose from a multitude of custom-tailored systems for exploring and analysing protein-protein interaction data, economists employ specialised software to interrogate, analyse and visualise statistical information, while different social networking sites offer different software to browse acquaintance networks. The multitude of tools for interacting with structured data – one for every domain or even dataset – requires users to learn a new interaction model and software package for each domain or dataset they want to explore.

The situation is different on the web. Current web technologies allow for collective creation and interlinkage of a massive collection of documents and media files, spreading the cost of creation and maintenance over a large population. The web has evolved from the contributions of many people and organisations, while keeping coordination to a minimum. One of the key distinctions to previous hypertext technologies was the decision to sacrifice global consistency to minimise the amount of coordination between contributors; as a result, the network of documents could grow rapidly in a decentralised fashion.

The primary mode for information seeking on the web is via search engines, which allow for locating documents that contain user-specified keywords. All major search engines employ a simple interaction model [SBC98]: users first type a few search terms into an input field, and the engine returns a list with hyperlinks to relevant documents. Users can then view the result list in sections of typically ten hits per page. When users click on a link to a document in the result list, they leave the search engine site for the target site and can inspect the document that might contain the sought-after information.

The principal modus operandi of search engines has remained unchanged in the last years. Web search engines have been designed and optimised for locating documents online, and they are doing an excellent job at returning links to documents that contain specified keywords, and often that is enough for simple information retrieval tasks. However, current state of the art search engines (such as Google, Yahoo, Bing, Ask, AltaVista etc.) are not delivering data directly but only links to documents. Such search engines fail when an answer requires aggregation of information – combining information from several sources is still mostly a tasks done by humans. Search engines also lack more sophisticated means of data analysis, mining, or visualisation – tasks required when dealing with structured data.

A recent development are domain-specific search engines which operate over structured data in confined domains. For example, Wolfram Alpha[1] provides search and query capabilities over a set of manually integrated datasets. Microsoft Academic Search [ZN09] and DBLife [DSC$^+$07] provide navigation over structured data in the domain of computer science research. However, all these sites are domain-specific, i.e., the components assume a predefined schema which is manually maintained. General-purpose search engines, on the other hand, are domain-independent; with the explosion of datasets in various domains (and hence various schemas) a general-purpose search and navigation solution for structured data is missing.

[1]`http://www.wolframalpha.com/`

1.2 Example Scenarios

To illustrate the wealth and variety of structured data available today, we list here several publicly available datasets. The various datasets are a (non-representative) selection of publicly available structured datasets, and will serve as examples throughout the thesis (see [CJ] for a more comprehensive listing of available datasets).

- **ACM Digital Library** The digital library of the Association of Computing Machinery (ACM) contains descriptions about research papers, authors, and publishing venues such as journals, conferences, or workshops.
- **Antarcti.ca** Antarcti.ca is a social networking site where people can manage their contacts. A common feature of social networking site is the ability to navigate from one user to another, allowing people in your social network to inspect your acquaintances (and their acquaintances, and so on). Data from social networking sites typically contains personal information such as names, email addresses, workplace, homepages, and connections to other people.
- **BBC** The British Broadcasting Corporation (BBC) maintains a database of musicians, albums, and songs. The information is similar to social networking sites: musicians have attributes such as name, homepages. In addition, artists are connected to descriptions of the songs and albums they released, which in turn have associated release date and links to reviews.
- **CrunchBase** Associated with the technology blog TechCrunch.com is CrunchBase, a user-maintained database of startup companies worldwide. Data include names and addresses of startups, their location, links to the products of the company, links to competitors, links to people involved in management, links to financial organisations that invested in the startups, and so on.
- **Eurostat** The Statistical Office of the European Commission publishes detailed economic indicators and statistics. Data includes Gross Domestic Product figures for members of the European Union, trade figures, air passenger transport data, and much more. The data contains mostly time-series figures tied to geographical locations.
- **FriendFeed** FriendFeed.com is a micro-blogging site where users can post short messages; people can follow other people's micro-blogging stream, essentially forming a social network between people who are interested in each other. Such micro-blogging sites enable rapid communication and information spreading between its participants.
- **Web** The Web dataset is created by crawling structured data available on the web. The dataset starts with the identifier denoting Tim Berners-Lee, the inventor of the web, and

performs breadth-first search up to six degrees in Tim's neighbourhood network. The Timbl dataset contains diverse data from a multitude of sources, containing descriptions of people, links to their homepages and blogs (and items from the blogs), but also data about locations and structured descriptions of encyclopedia entries, and much more. The dataset represents an amalgamation of segments of larger datasets and also low volume "independent" publishers.

The listed datasets are publicly available; all datasets except the Timbl dataset are hosted on websites which have a dedicated interface for searching and query, and all of the datasets are amenable for indexing by search engines. However, mere keyword search is not enough to leverage the wealth of the individual datasets, let alone enable the analysis of the data in combination. Since current search engines perform poorly at handling precise queries over structured information aggregated from multiple sources, completing the following selected tasks would involve significant manual effort using today's search engine technology:

- **Collecting and integrating information from multiple sources** For example, compiling a report with all publicly available information about a given researcher, including address, phone number, employment history, social network (from multiple sites), and publication history is labour-intense given current technologies. Another task requiring manual effort is to aggregate information about companies in a specified sector, the products or services the companies offer, financial indicators such as profit and taxes paid, the name and function of the executives of the companies, including their social network.

- **Selecting and displaying parts of a dataset according to multiple criteria** For example, tasks in this category include the creation of a document with all researchers in a particular scientific field in a particular geographical region, or with financial organisations that have invested in startups with competing products.

- **Generating reports from collected and integrated data** For example, such tasks could involve generating a spreadsheet contrasting the Gross Domestic Product with the research and development spending over time to evaluate the effectiveness of spending compared to other regions.

The goal of this dissertation is to develop and study new methods of collecting, integrating, and exploring web data – methods that empower casual users to easily interact with web-scale datasets and derive insights into the structure, organisation and patterns contained in the dataset. We assume data is published using web standards published by the World Wide Web Consortium (W3C). In particular, we assume that data sources adhere to the Linked Data principles [BL06], which provide a common access mechanism based on web architecture (HTTP) as well as a common data format (the Resource Description Framework (RDF)).

1.3 Research Challenges

Providing advanced search, query, and navigation capabilities over loosely integrated large datasets to end-users poses a number of new research issues. In the following, we briefly introduce many of the challenges, and then highlight the specific contributions in Section 1.4.

1. **How to discover, extract and transform structured information?** In contrast to search engines over documents, advanced search and navigation requires structured information. Thus, we either exploit the structured part of the web, or extract and transform structured data from documents or media files. In the second case, the system depends on means to convert non-structured sources and extract metadata from a range of file formats. To minimise the amount of data transfered, the system ideally performs lookups only on those data sources that contain structured information.

2. **How to manage very large datasets?** Anticipating the growth of electronic data, a centralised repository aggregating and indexing structured data has to scale competently. Even if the system only uses a fraction of the available information, the resulting data set may contain in the order of billions of entries. Thus, scaling up all components of the system represents a significant challenge. The architecture needs to be performance-oriented in order to handle data on web-scale and to keep up with the increased availability of structured data. A system operating over large datasets should scale up by adding new hardware – without requiring a fundamental redesign. Ideally, the system should also leverage the functionality offered by the web infrastructure such as query endpoints.

3. **How to deal with a large number of schemas?** Typical data integration scenarios involve the creation of a mediated schema which covers the integrated datasets. The system uses manually created mappings to provide an integrated view over disparate data sources. On the web, even a small set of data sources (in the order of tens of thousands) may contain thousands of schema elements, which renders the manual creation of mappings at a central location unrealistic. In addition, data published on the web contains already mappings relating data sources to each other, which have been created by the maintainer of a data source. Hence, a web data integration system should cater for schema flexibility, i.e., the schema of the integrated data is unknown at design time of the system.

4. **How to handle noisy data stemming from millions of sources?** In contrast to enterprise databases which underlie a strict curation process, anybody can say anything about anything on the web. The contributions of millions of people self-organise to some extent (in the sense that identifiers for classes and instances are shared across some sources), however, the resulting information space is only loosely integrated and therefore imprecise and chaotic. For example, searching for a researcher over loosely integrated data

might yield multiple class identifiers for "Person", and several different identifiers denoting the same real-world entity. The properties of collaboratively created information spaces require adaptations to established methods and the development of bespoke methods.

5. **How to search and navigate large, chaotic datasets?** Providing an input field to specify keywords and presenting the top ten results in a list is insufficient for supporting complex information-seeking operations. To allow end users to interact with large amounts of poorly structured information, we need new interaction models that take into account both the scale and chaos that characterises data aggregated from a large number of sources. Ideally, users could search and navigate datasets that dramatically vary in scale and structure using a single, universal interaction method. In addition, advanced visualisation techniques are required to enable users to derive insights based on query results over the integrated dataset.

1.4 Contributions

We study the problem of nurturing, collecting, and analysing web-scale datasets created in a decentralised and loosely-coordinated fashion. We describe the design, implementation, and evaluation of an end-to-end architecture for searching, querying, and exploring massive datasets. Due to the characteristics of data integrated from a multiplicity of sources, existing and well-understood search techniques are not directly applicable to structured data created by a large number of contributors. In other words, to be able to answer queries which exploit the semantics of data from a myriad of sources, the components of traditional search engine technology have to be materially changed. Taking into account the issues identified in Section 1.3, we summarise the research contributions found in this thesis.

- We identify various data integration problems arising in the web environment, and present algorithms to deal with web data of varying levels of integration.

- To provide better integration of data from a large number of sources, means to consolidate data from different sources are required. On the Semantic Web, reasoning is used to derive new facts from existing statements. In this thesis we focus on equality reasoning on the instance level and provide scalable algorithms for object consolidation.

- Aggregating data from thousands of sources may lead to thousands of statements describing one single entity. Thus, means of reducing the amount of information available per entity are necessary to reduce the cognitive load placed on the users. To this end, we present a method to rank objects, attributes, links, and data sources in large datasets to prioritise display of more pertinent fragments of data.

- We develop scalable algorithms and data structures for indexing and querying graph-structured data from the web to be able to handle datasets in the order of billions of statements aggregated from millions of sources.
- We study and evaluate distributed indexing methods for graph-structured data and parallel query evaluation methods, both on a cluster of computers connected in a LAN, and over live web sources.
- We evaluate the storage system on a dataset with millions of statements collected from the web, and provide scale-up experiments on billions of synthetically generated statements. We compare our indexing scheme to relational databases.
- We devise a user interaction model over structured datasets that allows for searching and navigating large datasets while striving for a balance between ease-of-use and expressiveness of the query operations.

Please note that all components are schema-agnostic: data sources provide schema information themselves, and the various algorithms applied to data only contain minimal hard-coded knowledge about schemas used (i.e., the core W3C vocabularies required for reasoning, and a minimal set of properties such as labels and dates for display purposes).

1.5 Outline

In this dissertation, we first introduce the data model based on objects and their properties, discuss the process of collaborative editing of such datasets, and define a set of operations that the users can apply to restrict the entire set of objects to the ones matching specified search criteria. Our general system architecture is an extension of the architecture of web search engines; the thesis document is structured accordingly and reflects the main components of a search and navigation system for data. A typical web search engine consists of a crawler, which traverses the hyperlink graph and downloads documents; an indexing component, which constructs an inverted index over the downloaded documents for quick lookup; a query processing component over the inverted index that returns documents matching search terms; a ranking component that prioritises a list of documents matching a search term; and a user interface to allow users to interact with the system. While our system contains components also found in search engines, the functionality of the described components has been significantly enhanced to fit the requirements of search and navigation over structured datasets.

Chapter 2: Collaboratively-Edited Object-Relation Datasets We start by introducing graphs as a knowledge representation format, followed by a description of our assumed object-oriented data model. We introduce multiple abstraction levels over data, from linked data to complex visualisations.

Chapter 3: Architecture for Web Data Exploration Systems In this chapter we present a general architecture for a search and navigation system for data, extending a data warehousing approach with runtime lookups on live data sources. We cover related work in the area of web search engines and data warehousing systems known from database research.

Chapter 4: Web Crawling and Data Extraction Here, we explain the implementation of a crawling algorithm to traverse the Web in order to discover structured data sources. An important requirement is to minimise the amount of non-structured (HTML) sources to visit in order to find links to structured content. The chapter also discusses our implementation of extracting metadata from various media types such as application files, text, messages, audio, and video.

Chapter 5: Ranking Data and Data Sources Users of systems aggregating information from a large number of sources can be overwhelmed with the amount of structured data returned for a query, and may have to manually sift through large result sets in order to find the relevant information. In this chapter we present a set of algorithmic tools for prioritising all elements of semantic graphs based on the inherent network structure. We show how to compute the metric for large graphs, provide a formalisation of the concept of naming authority to derive importance of data sources, and demonstrate how to apply the method to searching and exploring large datasets.

Chapter 6: Object Consolidation An important aspect in data integration systems is the issue of identity and uniquely identifying resources. Currently, there is poor agreement on the use of common URIs for the same instances across sources and as a result a naï vely integrated dataset might miss associations between resources. To solve the problem, we present a method for performing large-scale object consolidation to merge identifiers of equivalent instances occurring across data sources.

Chapter 7: Indexing Graph-Structured Data The central component of our system is the information repository. We describe the design and implementation of our architecture geared towards exhibiting linear scale-up to keep up with anticipated fast growth in data volume. In addition, we detail the distributed architecture which is imperative to meet scale requirements. To allow for good price/benefit ratio, we deploy the system on commodity hardware through use of a shared-nothing architecture [Sto86]. To achieve adequate response times over large amounts of data, the indexing provides constant lookup times with respect to scale.

Chapter 8: Interaction Model for Visual Data Exploration Next, we review the interaction primitives of a range of systems from an end-user perspective, and select and describe several core navigation primitives useful for exploring structured datasets.

Chapter 9: Query Language and Query Processing Visually exploring structured data results in complex queries which have to be evaluated below a certain time threshold for interactive systems. To this end, we discuss and evaluate how to process queries over data sets that exceed the storage capacity of one machine, based on the indexing framework. We describe and evaluate an optimised query processor operating over both static and live data sources which performs top-k processing and therefore returns ranked results that are required for the user interface.

Chapter 10: User Interface In this chapter we extend menu-based browsing paradigms to operate over large interlinked information spaces. In particular, our interaction model allows users to construct complex queries using a point-and-click interface. We argue that a balance has to be found between the complexity of the user interface and the expressiveness of the queries posed. We present an interaction model that allows users to restrict the result set according to facets and to perform a focus change relative to the current result set.

Chapter 11: Conclusion We conclude with an outlook on potential further developments of the web, and discuss how the system described in this dissertation could have scientific and economic impact.

Chapter 2

Collaboratively-Edited Object-Relation Datasets

2.1 Introduction

In the following we introduce the foundations of our conceptual data model. We give a high-level view on the data we use as a basis for studying the design and implementation of the components of a search and navigation system. We introduce graphs as foundational data model, describe how to use global resource identifiers to denote graph nodes and edges, introduce a model for tracking the provenance of data, and finally arrive at an object-oriented view of data aggregated from many sources upon which we base our study and experimentation.

2.2 Graphs as Knowledge Representation Format

Graphs are a universal way of encoding data and describing systems in science and technology, and graph-structured data models are increasingly used on the web to publish semi-structured data.

The power of graphs as a means to formalise and represent knowledge has long been recognised. Leonhard Euler's paper about the Königsberg bridge problem [Eul41] is the one of the earliest papers using graphs to formalise a mathematical problem. More than a century later Charles Peirce presented the notion of "existential graphs", a diagrammatic syntax for an algebra of dyadic relations [Pei09].

Figure 2.1 depicts a graph describing the acquaintance network between three people, expressing the fact that Tim knows Dan and Yolanda. Please note that the network is incomplete – Yolanda may know Tim as well, for example. The Figure shows a small subset of the network from only one data source to illustrate the core concept.

Early applications of "semantic nets" on computers were in the area of machine translation

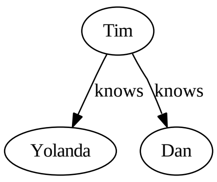


Figure 2.1: An acquaintance network between three people.

[Ric58] and question answering [Sim66]. Later, [Woo75] investigated the meaning of links in semantic networks, and [Bra78] notes the epistemological, or knowledge-structuring, aspect of previously known network-structured representations. Database modelling techniques such as the Entity-Relationship Model (ER) [Che76] and Object Role Modelling (NIAM/ORM) [TMV88] use a graph-based syntax for modelling data.

Consens and Mendelzon applied GraphLog, a graph-based query language, to hypertext [CM90]. Papakonstantinou et al. [PGMW95] describe OEM, the Object Exchange Model, a syntax and data model for exchanging semi-structured data across object-oriented databases.

A wide variety of data can be encoded as a graph (or as a set of binary relations). In addition, modelling data as graphs enables the distributed and loosely coordinated modelling of data which is not organised in a predictable structure. Thus, graph-structured data is particularly useful for integrating heterogeneous data from a large number of sources. The ability to aggregate data on world-wide scale, however, requires global agreement on identifiers for nodes and edges in a graph. For example, the resources “Tim”, “Dan”, and “Yolanda”, or the link “knows”, have to be identified unambiguously.

2.3 Global Resource Identifiers

World-wide data creation requires a system to name things unambiguously. To avoid situations in which multiple data sources create identical identifiers for denoting different things, coordination between the participants is required. The largest publicly accessible electronic system is the internet, and for creating a naming scheme on a world-wide scale it makes sense to re-use the already established infrastructure. To coordinate the distributed minting of world-wide identifiers, we utilise the currently most widely deployed global naming system, based on the Domain Name System (DNS)[1]. The DNS is a hierarchical name service to map cryptic addresses to human-readable host names. For example, the host name `www.w3.org` maps to `128.30.52.166`. A set of registrars maintain a database with these mappings and ensure that assigned names are globally unique. For example, the domain name `w3.org` is registered to the World Wide Web Consortium.

Building atop the DNS facility, organisations or individuals who have registered domain

[1] `http://www.ietf.org/rfc/rfc1034.txt`

names are able to define identifiers for arbitrary resources in the form of Universal Resource Identifiers (URIs)[2] – and subsequently Internationalised Resource Identifiers (IRIs)[3].

IRIs identify resources, where a resource can be either some sort of file (such as a hypertext document, a digital image, or a digital video), a real-world entity (such as a researcher, a building, or a country), or an abstract concept (such as mathematical operations or types of relationships). For example, the IRI `http://www.w3.org/People/Berners-Lee/` can denote a hypertext document describing Tim Berners-Lee. The IRI `http://www.w3.org/People/Berners-Lee/card#i`, on the other hand, can denote the person Tim Berners-Lee himself. Finally, the IRI `http://xmlns.com/foaf/0.1/knows` identifies a relationship.

IRIs provide a basic mechanism to globally identify resources, which are either accessible and can be retrieved via a computer, or just provide identification without the ability to retrieve its content. For the purpose of this thesis, we are mainly concerned about IRIs which denote resources described in the Hypertext Transfer Protocol (HTTP) scheme, regardless of whether they denote retrievable content or not. For a treatment of content retrieval, please see Section 2.5. IRIs in the HTTP naming scheme contain a hostname which is partly managed via the DNS, and a local part which can be freely managed by the organisation or person who registered the domain.

With respect to more recent trends on IRI usage, data publishers are starting to use resource identifiers to denote certain non-information resources (such as locations, people), and incrementally other publishers start to re-use the same resource identifier to denote the same thing, which leads to interconnections and interdependencies between digital artifacts. Please observe that what exactly an IRI denotes is a process guided by social interactions. It is possible, then, that exactly what is meant by an IRI changes over time, depending on the usage of the IRI, much as the meaning of words in natural language may change over time. Resource identifiers form the basic mechanism that can be used to associate otherwise disjoint pieces of data.

2.4 Resource Description Framework

A graph-based knowledge representation formalism, in conjunction with global resource identifiers, leads to a mechanism which facilitates data integration on a world-wide scale. Consider the graph in Figure 2.2, which encodes the same facts as Figure 2.1, but this time using IRIs to identify resources in the graph. Now, consider the graph in Figure 2.3. By using the same IRI to denote Dan, it is possible to combine the graph by just merging resources with the same identifier (Figure 2.4).

Computerised processing of graph-based data requires a syntax to encode the graphs. The Resource Description Format (RDF)[4] is a general method for encoding graph-based data which

[2]`http://www.ietf.org/rfc/rfc3986.txt`
[3]`http://www.ietf.org/rfc/rfc3987.txt`
[4]`http://www.w3.org/TR/rdf-syntax-grammar/`

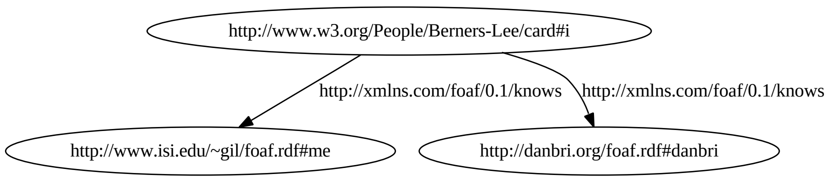

http://www.w3.org/People/Berners-Lee/card#i
http://xmlns.com/foaf/0.1/knows
http://xmlns.com/foaf/0.1/knows
http://www.isi.edu/~gil/foaf.rdf#me
http://danbri.org/foaf.rdf#danbri

Figure 2.2: An acquaintance network between three people using IRIs for identification.

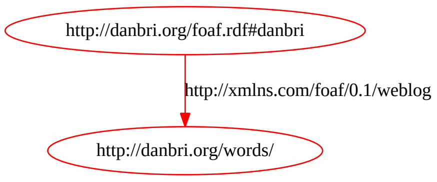

http://danbri.org/foaf.rdf#danbri
http://xmlns.com/foaf/0.1/weblog
http://danbri.org/words/

Figure 2.3: A graph stating Dan's weblog.

does not follow a predictable structure. RDF is a graph-structured data format which is schema-less and self-describing, meaning that the labels of the graph within the graph describe the data itself. The atomic constructs in RDF are subject-predicate-object statements, so-called triples.

Definition 2.4.1 (RDF Triple) *Given a set of IRI references $\mathcal{R}$, a set of blank nodes $\mathcal{B}$, and a set of literals $\mathcal{L}$, a triple (s, p, o) $\in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L})$ is called an RDF triple.*

In a triple (s, p, o), s is called subject, p predicate or property, and o object. An example for a triple in Notation3 syntax[5] stating Dan's name is shown in Figure 2.5. Please observe that RDF allows using so-called literals – datatype values such as strings, numbers, or dates.

We make a distinction here between triples containing IRIs and triples containing literals in the object position. This distinction is made more explicit when using object-oriented terminology (see Section 2.6), and becomes apparent in Chapter 8 which covers the search and navigation model. The concepts of an object property and a datatype property are defined in a W3C Recommendation[6]. A thorough description of RDF can be found in [DMvH$^+$00].

Definition 2.4.2 (Datatype Property, Object Property) *A property p_d in a triple (s, p_d, o) $\in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times \mathcal{L}$ is called a datatype property; a property p_o in a triple (s, p_o, o) $\in (\mathcal{R} \cup \mathcal{B}) \times \mathcal{R} \times (\mathcal{R} \cup \mathcal{B})$ is called an object property.*

Namespaces can be used as abbreviations to improve readability of IRIs. Some commonly used namespaces are `foaf`, `rdf`, `rdfs`, `dc`, `sioc`. These local prefixes map to full IRIs. For example, the IRI `http://xmlns.com/foaf/0.1/knows` can be abbreviated in a suitable RDF syntax as `foaf:knows`. Please note that in practice some properties are used both as datatype properties and object properties (e.g. `dc:creator`).

[5]`http://www.w3.org/DesignIssues/Notation3.html`
[6]`http://www.w3.org/TR/owl-ref/`

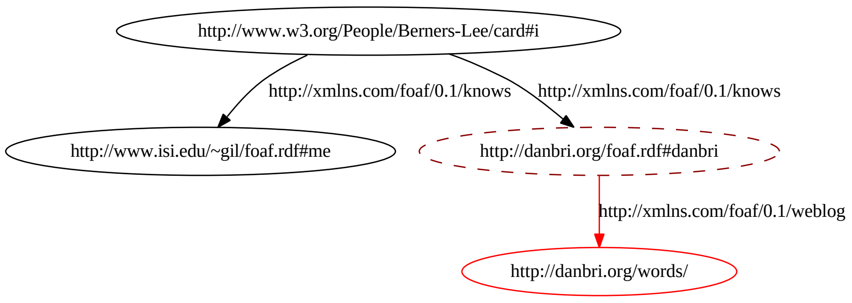

http://www.w3.org/People/Berners-Lee/card#i
http://xmlns.com/foaf/0.1/knows
http://xmlns.com/foaf/0.1/knows
http://www.isi.edu/~gil/foaf.rdf#me
http://danbri.org/foaf.rdf#danbri
http://xmlns.com/foaf/0.1/weblog
http://danbri.org/words/

Figure 2.4: The merge of graphs in Figure 2.2 and 2.3.

```
<http://danbri.org/foaf#danbri> <http://xmlns.com/foaf/0.1/name> "Dan Brickley" .
```

Figure 2.5: An RDF triple in Notation3 syntax.

2.5 Context, Provenance, and Linked Data

Although the RDF specification itself does not define the notion of context, usually applications require context [GMF04] to store additional data for a set of triples. The interpretation of context depends on the application. To this end we extend the classic RDF data model with the notion of context [HD05a]. RDF provides a vocabulary for describing statements [MM04] (so-called "reification"), however, staying inside the triple model means that each description requires four additional triples, leading to an explosion of data.

Context can be used to associate data with a given sets of triples, for example, enhancing a given triple with spatial or temporal information [MK03]. Given our assumption that we deal with data that is collaboratively-edited, and therefore can originate from several millions of sources, we require the ability to keep track of the provenance of a triple in the aggregated graph.

In the following, we adopt the more restrictive form of context for tracking provenance, however, please observe that the notion of provenance can be lifted from a physical level (talking about identifiers of data sources) to referring to people or organisations as the source of sections of information [HPD07]. A special case of the general notion of context [HD05a] is that of named graphs [CBHS05], used solely to track the source of RDF triples. Capturing provenance is one of the fundamental necessities in open distributed environments, where the quality of data has to be judged by its origin.

Definition 2.5.1 (Triple in Context) *A pair (t, c) with t be a triple and $c \in (\mathcal{R} \cup \mathcal{B})$ is called a triple in context c.*

Please note that we refer to a triple $((s, p, o), c)$ in context c as a quadruple or quad (s, p, o, c).

Definition 2.5.2 (Quadruple) *A pair (t, c) with a triple t and* $c \in (\mathcal{R})$ *is called a quadruple or quad (s, p, o, c).*

We use the term statement to refer to a triple or a quadruple.The context of a quad denotes the URI of the data source from hence the contained triple originated.

For example, consider the graphs in Figures 2.6. The context of the graphs are `http://www.w3.org/People/Berners-Lee/card` and `http://danbri.org/foaf.rdf`. Please note that the same triple can occur in multiple contexts, for example, a third data source could assert the triple stating Dan's weblog. When aggregating all of these graphs, our extension of the RDF model allows for tracking the data sources for subgraphs.

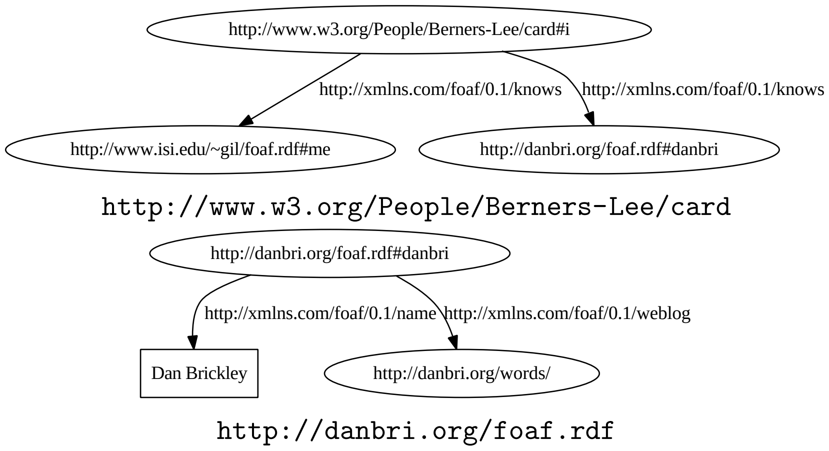

http://www.w3.org/People/Berners-Lee/card#i
http://xmlns.com/foaf/0.1/knows
http://xmlns.com/foaf/0.1/knows
http://www.isi.edu/~gil/foaf.rdf#me
http://danbri.org/foaf.rdf#danbri
http://www.w3.org/People/Berners-Lee/card
http://danbri.org/foaf.rdf#danbri
http://xmlns.com/foaf/0.1/name
http://xmlns.com/foaf/0.1/weblog
Dan Brickley
http://danbri.org/words/
http://danbri.org/foaf.rdf

Figure 2.6: Two RDF graphs with associated data source IRI.

There is one more convention to take into account. We define context as the data source that is available online and can be dereferenced; i.e., its content can be retrieved by performing a network lookup. In our description, we restrict ourselves to discussing HTTP for performing lookups; however, the general mechanisms hold for other data transfer protocols as well.

Dereferencing an IRI involves sending a request to the target web server which processes the request and sends the appropriate response: either the requested file together with a status code (eg., "200 OK"), or a status message indicating the issue (eg., "303 Moved Temporarily" or "404 Not Found").

Please observe that there is a difference between the identifier of a data source and the identifier of an abstract thing. For example, there is a difference between Dan's machine-readable homepage at `http://danbri.org/foaf.rdf` and the IRI `http:danbri.org/foaf.rdf#me` (note the `#me`) which identifies Dan itself. The distinction between data sources and identifiers for things is maintained either via syntactic measures (i.e. a hash sign), or via protocol (i.e. HTTP redirects).

The described mechanism for publishing data based on RDF and web architecture can be summarised in the four Linked Data principles [BL06], postulated by Tim Berners-Lee in 2006:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
4. Include links to other URIs, so that they can discover more things.

These principles ensure a common access protocol (HTTP) and a common data format (RDF) for data on the web, and enable distributed discovery of new data via links to other URIs.

2.6 Object Orientation

The model presented thus far allows for the integration of graph-based data which have been created in a distributed and loosely-coordinated manner. Our second requirement is that the data we deal with is object-oriented, following the prevailing programming and modelling paradigm in computer science.

RDF can be viewed as graph; taking into account the `rdf:type` predicate, it is possible to arrive at an object-orientated view and talk about instances and classes. Informally, the subject of a triple with an `rdf:type` predicate is an instance of the class denoted by the URI specified in the object position. Thus we achieve an object oriented data model, with classes and instances, and connections between them. For example, consider the graph in Figure 2.7, which now states that the three nodes in the graph are instances of type `http://xmlns.com/foaf/0.1/Person`.

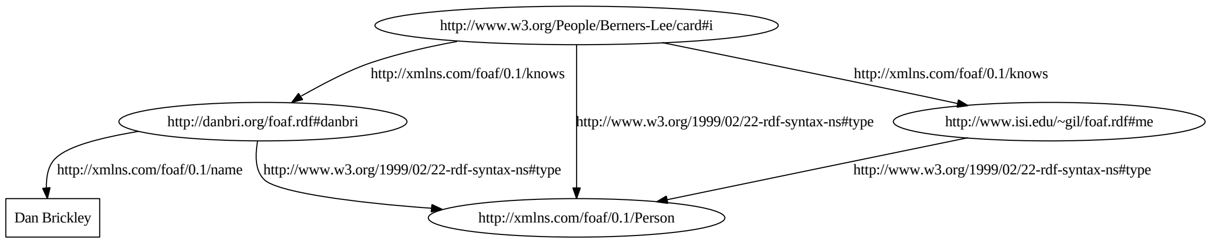

http://www.w3.org/People/Berners-Lee/card#i
http://xmlns.com/foaf/0.1/knows
http://xmlns.com/foaf/0.1/knows
http://danbri.org/foaf.rdf#danbri
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.isi.edu/~gil/foaf.rdf#me
http://xmlns.com/foaf/0.1/name
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Dan Brickley
http://xmlns.com/foaf/0.1/Person

Figure 2.7: The acquaintance graph adorned with type information.

Definition 2.6.1 (Instance) *A IRI reference $e \in \mathcal{U}$ or blank node $e \in \mathcal{B}$ describes an instance e if there exists a triple (e, rdf:type, $u \in U$).*

For ease of exposition we exclude in our definition the case where a class can be denoted by a blank node.

Definition 2.6.2 (Class) *A IRI reference $e \in \mathcal{U}$ or blank node $e \in \mathcal{B}$ describes a class e if there exists a triple (e, rdf:type, rdfs:Class) or (e, rdf:type, owl:Class).*

Consequently, our object model consists of

- objects, which may be divided into classes and instances;
- attributes of objects (such as `rdfs:label`);
- links to other objects (such as `foaf:knows`).

An object-oriented perspective is beneficial, e.g., for conceptual modelling or for viewing the information space in a user interface – where it is convenient for users to operate on an object (or entity) abstraction. Object-orientation includes a number of concepts, most importantly (structural and behavioural) encapsulation, but also classification and inheritance. Figure 2.8 shows an object-oriented view on the acquaintance graph. We use the UML 2 notation to denote instances of classes. Please observe that in our data model an instance can belong to more than one class which are not necessarily directly linked in the subclass hierarchy, i.e., RDF allows for multiple inheritance which is not supported by all object-oriented paradigms.

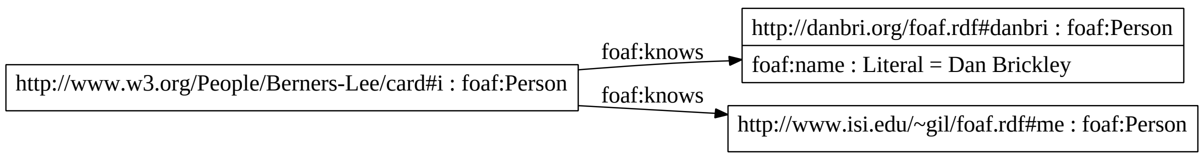

http://danbri.org/foaf.rdf#danbri : foaf:Person
foaf:name : Literal = Dan Brickley
foaf:knows
http://www.w3.org/People/Berners-Lee/card#i : foaf:Person
foaf:knows
http://www.isi.edu/~gil/foaf.rdf#me : foaf:Person

Figure 2.8: An object-oriented view on the acquaintance graph (shown in UML 2 notation).

Object-orientation includes more complex features as well, the most notable being inheritance, meaning that it is possible to specify a so-called subclass which extends or overrides the specification of the superclass. Inheritance is closely related to the notion of RDF Schema's `rdfs:subClassOf` and `rdfs:subPropertyOf`, which allow to model hierarchical relationships between classes and properties. Specifically, RDF Schema adopts a semantics of classes and instances based on the open world assumption and description logics, while traditional object-oriented systems assume a closed world and constraints [ODG+07].

2.7 Vocabulary Descriptions

Ultimately, the semantics has to be embedded in a social context; i.e.. the meaning of an identifier emerges from the consensual reuse of identifiers (where people mean the same thing). To establish a shared understanding between data providers and consumers of what certain schema-level identifiers constitute, we require a method to describe schema-level constructs.

Vocabulary descriptions list these schema-level constructs – classes and properties – and annotate them with certain data, such as human-readable labels and descriptions. In addition, there are certain knowledge representation constructs rooted in logic which allow for mathematical specification of meaning and allow for mechanical inference (such as subclass and subproperty relations). Relating vocabulary identifiers with each other (e.g., via a subclass relation) results in mappings between vocabularies and thus facilitates data integration.

The main vocabularies used in conjunction with RDF are RDF Schema[7] and the Web Ontology Language[8] which are itself encoded in RDF, amended with textual descriptions that specify the meaning of certain identifiers. Thus, RDF, RDF Schema, and OWL are on a different conceptual level than the vocabularies previously described – i.e., they are meta-vocabularies.

Domain-specific vocabularies include Friend-of-a-Friend (FOAF)[9] for describing people, RDF Site Summary (RSS 1.0)[10] for specifying news and blog syndication, Dublin Core (DC) defines a schema for describing library resources, Semantically Interlinked Online Communities (SIOC)[11] for defining data exchange between online community sites, and Simple Knowledge Organisation System[12] (SKOS) for encoding classification schemes and thesauri.

Please note that constraints encoded in vocabulary descriptions are not enforced. One difference between the description logics view towards object orientation (which we take here) and the view that object-orientated languages take is the way how domain and range of properties are treated. Domain and range do not restrict classes in the description logics view, i.e., they are not seen as constraints which have to be adhered to, but rather imply classes and types [ODG$^+$07]. This distinction is similar to the notion of schema in databases vs. vocabulary descriptions on the web. Schemas are to be enforced, e.g., via integrity constraints, whereas vocabulary descriptions just give a recommendation as to how to model things (which also allows for inferencing).

In general, not enforcing domain and range descriptions supports the view of the web as open world, where global consistency and unique identifiers for the same object cannot be enforced due to the diverse and possibly uncoordinated contributions of vast amounts of data publishers. Enforcing constraints on a global scale is extremely difficult. Similarly, the hypertext model of the web does not enforce constraints either - the ability to allow for broken links (the infamous "404 not found" error) reduces the burden of distributed publishing since publishers do not have to maintain the integrity of links. Despite the perceived problem with dangling links[13], the web has enjoyed great success.

[7]http://www.w3.org/TR/rdf-schema/
[8]http://www.w3.org/TR/owl-features/
[9]http://www.foaf-project.org/
[10]http://web.resource.org/rss/1.0/
[11]http://www.sioc-project.org/
[12]http://www.w3.org/2004/02/skos/
[13]http://www7.scu.edu.au/1942/com1942.htm

2.8 Distributed, Collaboratively-Edited Datasets

The power of graph-structured data models for data integration has been noted previously (e.g., [PGMW95]). On the web, we currently see data publishers using graph-structured data models (in particular RDF) to make large amounts of data available. The resulting patchwork of data from autonomous sources is only loosely consistent [HHP$^+$10]. However, sacrificing consistency lowers the burden for data publishers and encourages an incremental approach to data integration (e.g., [DSDH08]). As a result, aggregated data from the web is – to a varying degree depending on the data sources integrated – heterogeneous, inconsistent, and incomplete. Schema-level information is often disparate and not mapped to other schemas.

Chaotic web data complicates data processing and requires flexible systems robust in the face of imperfect data, but also poses a tremendous opportunity when taking into account the combined contributions of millions of people for ranking. Reuse of identifiers from other sources establishes an implicit approval for the data source with the authority to coin the identifier. Kleinberg [Kle99] and Brin and Page [PBMW98] use the hypertext structure to measure importance of pages which is crucial for ranking vast result sets matching keywords, and use anchor text pointing to a document (basically annotations made by people other than the document author) as a description of the content of the page, which is often more accurate than the document itself. The notion of using link structure for search was noted earlier by [Mar97].

Web search engines are inspired by the information retrieval tradition. There is one notable difference between web search and traditional information retrieval systems: the underlying dataset is not curated by a single person or a team of people, but rather consists of the loosely coordinated contributions of a vast number of people. Recall in web search engines is important (engines should ideally retrieve all documents that are relevant), but not necessarily the traditional notion of precision: users are satisfied if the first ten results contain relevant information.

2.9 Syntax, Semantics, Semiotics

Taking a more philosophical stance, we can separate large information spaces, such as the Semantic Web, into syntactic, semantic, and semiotic layers.

- Syntax describes the structure of data in a system; as such, syntax offers a shared understanding of structure useful for data transmission; e.g., in the crawler, indexing, and query processing components.

- Semantics covers the meaning of the symbols in a system (shared understanding of symbols). Semantics is useful for data integration; for example in the reasoning, and ranking components.

- Semiotics is a branch of philosophy that is concerned with the concept of signs occurring in a system. A shared understanding of signs is useful for data display in the user interface.

2.10 Conclusion

In this chapter we have introduced graph-based data models, which will be useful for collecting, integrating, and storing data. We have described how the graph-based data model can be used to encode object-relation data. The object-relation viewpoint is not only common in computer science, but also in architecture [Kum07] or sociology [Emi97].

While the data model is applicable to a wide range of data integration scenarios, data from the web is the most diverse dataset available, in terms of domain coverage as well as structure. If it is possible to handle web data (which is created with only loose coordination and almost no quality control, other than social standards), then the system can handle other, less diverse and cleaner datasets as well.

The conceptual model of collaboratively-edited object-oriented datasets forms the basis of the user interaction model.

Chapter 3

Architecture for Web Data Exploration Systems

3.1 Introduction

In the following, we sketch the general architecture of a system for searching and navigating structured data aggregated from a large number of sources. The aim is to specify requirements and derive a foundational framework upon which to build in subsequent chapters. We assess commonalities with the architecture of traditional large-scale search engine systems and point out the major differences. We will touch on key aspects of each component of the system in the following, while deferring detailed technical description, discussion and evaluation to later chapters. The related work section surveys literature covering the architecture of other search and navigation systems over structured data, including work from information retrieval and relational databases.

3.2 Requirements

The architecture of systems for integrating, searching, and exploring web data has the following requirements:

- Response time: to enable a continuous flow in the user interaction and keep the user's attention focused on the interface, users expect response times below ten seconds [Nie94]. Thus, the system architecture has to be able to deliver these response times.
- Response relevance: when posing precise queries to a database system, all of the answers are correct by definition since they satisfy the rigorous query conditions. However, a user cannot realistically process tens of thousands of results that are returned in response to a search term. Rather, users expect the most relevant results in the first few result elements.

- Coverage: ideally the system has in-depth coverage on any topic imaginable. Since that is an elusive goal, the system should at least have coverage for the intended audience. For example, web researchers might expect to have themselves described in the system – including affiliation and publication record – if the system is targeted towards them.

- Ease of use: the system has to be usable by the targeted user population. The more complex the system is, the more training time is required and the less people will likely use the system. As a result, the system has to strike a balance between complexity (i.e. what type of queries can be posed, and what type of operations are enabled) and ease-of-use. The ideal situation would be to keep simple common tasks easy to execute, while still allowing the user to pose more complex queries. In addition, an easy-to-use system requires not only understandable interaction design, but also good graphical design.

- Universality and domain-independence: the less assumptions the systems makes about the schema of the data processed, the less manual labour is involved in preparing and integrating the data. This is also related to coverage: if the system has the ability to only process specified vocabularies, large amounts of possibly valuable data might never be processed and shown to the user.

- Collaborative organisation and analysis: given that the system should be able to handle arbitrary data, the users of the system likely do not have a complete understanding of what the dataset contains. Hence, the users should be allowed to incrementally learn more about the underlying dataset, and should be enabled to bookmark interesting data elements they encountered and to communicate these insights to their peers.

- Integration and mapping: given that data comes from a multiplicity of sources, mechanisms for integrating and mapping disparate datasets are required. Ideally, the mappings are provided by the data sources themselves; however, there needs to be infrastructure in place to process and carry out the integration based on these mappings.

Some of the requirements are mutually exclusive: for example, delivering fast response times requires the system to pre-process data, while the requirement for data freshness implies that ideally the original data sources are processed only when a user poses a query.

3.3 Features

Based on the requirements, we define the features of a data search and exploration system. Table 3.1 matches the listed requirements to the features addressing them.

- Distribution: the system is distributed. Distribution is a key requirement to scale up the size of the data capacity while retaining reasonable response times.

- Caching: to speed-up lookups, the system includes a caching layer. Given that all communication is carried out via HTTP, the system can comfortably re-use proven and optimised caching systems such as Squid[1] and Apache's mod_cache[2].
- Crawling: the system supports resource discovery, i.e. follows links and discovers new and previously unknown datasets.
- Self-describing data: the algorithms assumes only minimal schema knowledge at design time; data sources provide descriptions and mappings that are leveraged during processing time.
- Integration and warehousing: the system supports warehousing of data; having data collected at one place facilitates integration since reasoning can be carried out over the dataset in its entirety. Also, global ranking is possible when the entire dataset is present at one location.
- Ranking: the system supports ranking based on the popularity of data sources and the reuse of identifiers among data sources. Ranking is required in chaotic web datasets and gives the user interface measures to prioritise the display of data elements.
- Reasoning: the system includes a reasoning engine that fuses object identifiers and performs mappings between data. Reasoning is important to unite fractured data sources and provide a unified view over heterogeneous data.
- Various parsers: to allow for the integration of a multitude of data formats, the system supports a wide variety of data formats and provides parsers for metadata embedded in these file formats into an RDF representation.
- Various export formats: similar to the ability to import data from a large number of formats, the system exports data for further processing in application programs in formats such as RDF, CSV, iCal, and KML.
- Search and browsing: the system supports keyword search and browsing as known from the traditional hypertext web. More complex query operations are supported via drag-and-drop, but not immediately visible to the novice user. Hence, as the users get accustomed to the system and learn more features, they are able to gradually pose more powerful queries.
- Result visualisations: the system supports a wide range of result visualisations: from traditional detail and list views, to table views and displays for spatial and temporal aspects of the data.

[1] `http://www.squid-cache.org/`
[2] `http://httpd.apache.org/docs/2.1/caching.html`

Requirement	Feature
Response time	Distribution, Caching, Integration and warehousing
Response relevance	Ranking
Coverage	Self-describing data, Various parsers
Ease of use	Search and browsing, Result visualisations
Universality and domain-independence	Self-describing data, Various parsers, Reasoning
Collaborative organisation and analysis	Search and browsing, Result visualisations, Various export formats
Integration and mapping	Reasoning

Table 3.1: Requirements and features.

3.4 Assumptions

In the following, we make explicit the assumptions taken to delineate the intended feature set of the system. These assumptions greatly influence design decisions and the final architecture of the system.

- The system integrates data from external sources and does not contain means to enter or modify data. Similarly, mappings are provided externally by the web community.
- Users have at least some understanding of object-oriented design, and posses domain knowledge about the tasks they want to solve, or are capable of learning the domain knowledge while exploring the data.
- Build on web standards: it is important to adhere to web standards to be able to process the growing amount of data in these standards, to facilitate the data gathering, and to be able to re-use existing implementations and infrastructure. Since we build on HTTP, we assume a pull model where requests are sent to the data providers, rather than a push model where the data providers push data to our system.
- Incompleteness: there may be new or unknown data sources not available to the system. Given the scale of the web, it is impossible to gain an up-to-date, complete view over the web in its entirety.
- Data is chaotic: we expect data from possibly millions of data providers – without centralised control or curation – to be noisy and contain erroneous, duplicate, and out-dated data.

3.5 Web Search Engine Architecture

Here we discuss the high-level architecture of web search engines. Web search engines grew out of applying information retrieval techniques to hypertext. The First World-Wide Web Conference in 1994 included a number of search systems [McB94, Kos94, McK94, NH94, Eic94, Pin94] utilising crawlers to collect data. A few commercial systems evolved from that early work, most notably Lycos [Mau97] and AltaVista [SMHM99].

The basic mode of operation of a web search engine can be described as "crawl, index, serve" [Bre98]:

- The first step is to "crawl" the documents, basically traversing the hyperlink graph and downloading the documents. The output of the process are collections of documents, typically stored into single archive files. The crawler keeps track of the visited pages to avoid loops, and has policies in place for revisiting pages.
- The indexer is responsible for parsing and interpreting document collection archives. The output is a portion of the static dataset – called a chunk – reflecting scoring and normalisation of the documents. The aim is to move work from the user-facing servers to the index, thus minimising the amount of work required during query time. For example, the index performs word scoring and document analysis such as detecting the language of a page, checking for spam, and tracking incoming and outgoing links for scoring. Also, the indexer tracks the anchor text used from other documents to point to the current one, essentially performing a simple form of data integration.
- The server executes queries against a collection of chunks, performing query parsing, rewriting, optimisation, and execution. Because the only update operation s a replacement of chunks, there is no need for locking, isolation, and concurrency control in the query processor.

For web search systems implementing such an architecture see [BP98, MRYGM01, HRGMP00]. [Cha03] gives a general overview of techniques used in such systems. A notable exception to centralised systems is Harvest [BDH$^+$95], a distributed indexing infrastructure aimed at providing decentralised caching facilities and keyword search over a set of object brokers.

In typical web search engines, a dedicated ranking component prioritises the results of a search; fundamentally, there are two ways of ranking documents: query-independent (global) ranking methods which do not take the user query into account, and query-dependent (local) ranking methods which operate only on a selected query-specific portion of the dataset. PageRank [PBMW98] is the most well-known global ranking method, now used in combination with tf*idf and various other methods to combine query-dependent and query-independent methods. HITS [Kle99] first selects a subgraph from a search engine index and calculates two sets of rank values: one set for so-called hubs, and one set for so-called authorities. Bharat and Henzinger [BH98] present an improved version of link-based ranking algorithms that take into account the content of a page.

We now discuss similarities and point out the major differences between web search engines and our data search and navigation system; we do so along seven key dimensions:

- **Data Model** In contrast to web search engines which primarily deal with HTML documents, our system assumes RDF, a graph-based data model which can be lifted to represent objects and relations. The different data model requires fundamental changes in almost all aspects of the system.

- **Data Provisioning** Similarly to web search engines, we employ a crawling component that traverses the data source graph. Handling RDF data is straightforward, but other data formats require extraction of data; for example, images might contain embedded information about camera type or geocoordinates, and documents might contain embedded information about authors or date of last change.

- **Data Integration** Web search engines do a very limited form of data integration via link anchor text [BP98]. Link anchor text from external sites is used to describe the content of the link target site, and multiple anchor labels are combined to achieve a description of the link target page. In our system, the data integration component is based on vocabulary descriptions (or ontologies) with powerful logical constructs. One of the important constructs is `owl:sameAs`, which defines equivalence between identifiers, allowing for the integration of disjoint datasets.

- **Ranking Scheme** As in traditional search engines, our input data is extremely diverse in terms of topic and structure; hence, we require ranking to order data elements. As opposed to web search engines which compute rankings over the hyperlink graph, we require a method to calculate ranks over all elements in a directed labelled graph with context. Analogous to web search engines, our ranking method has to be domain-independent and has to work without labour-intensive manual input.

- **Index Structure** Similar to the hypertext web, the amount of structured data available on the web is vast. Hence, the index structure has to be distributed to be able to scale to the required amount of data. However, in contrast to web search engines which perform keyword queries, we require an index structure that allows for both keyword and structural queries.

- **Query Processing** Typical search engine queries are on average 2.4 words long [Hen07]. Aside from processing basic keyword queries posed by human users in the user interface, our system also caters for agents who may pose queries involving keyword and structural components through our application programming interfaces. The query processing component has to evaluate queries over indices spanning multiple machines, which can involve complex joins.

- **User Interaction** Unlike web search engines which allow people to locate documents containing keywords, users of data search and navigation systems are able to additionally pose more complex navigational queries. Thus, the user interface has to provide some means for a user to specify these complex queries. Not unlike search engines, the user interaction should be domain-independent and universal. To this end, we identify a set of atomic operations that can be assembled via a point-and-click interface to form complex queries. Rather than merely returning a list of documents, the data search and navigation system should include application programming interfaces and functionality to export the resulting data to other programs for further processing.

3.6 System Architecture Overview

We present the architecture of a system for searching and navigating collaboratively-edited datasets. Figure 3.1 illustrates the components of the system and their interplay. Boxes denote components of the system, barrels represent stored data, and lines indicate data flow. The system broadly consists of modules concerned with pre-processing data (crawler and extractor, reasoner, ranker, index builders), and modules concerned with query-time processing (index managers, query processor, user interface). The pre-processing yields index and rank files which are used by the query-time components to answer queries posed by the users. Please note that we include two sets of people in the diagram: data publishers and data consumers. A community publishes linked data, and another community – possibly overlapping with the data publishers – consumes the data, with the ability to export results to office productivity applications, for example.

Having introduced the overall architecture, we now give an overview of each component; later chapters discuss components in detail.

- **Crawler and Extractor** To harvest web data we use MultiCrawler [HUD06], a pipelined crawling architecture which is able to syntactically transform data from a variety of sources (e.g., HTML, XML) into RDF to provide an integrated view over the data for search and querying. The crawler gathers data from the web by traversing the link graph and the extractor transforms metadata from HTML documents (e.g. RDFa, GRDDL, or Microformats) and metadata embedded in various file formats (e.g. GIF, PNG, PDF, MS Office) into RDF. The extractor also translates RSS 2.0 and Atom feeds into RDF. In addition, the component also cleans the data syntactically: for example, normalising URIs and parsing datatype literals.

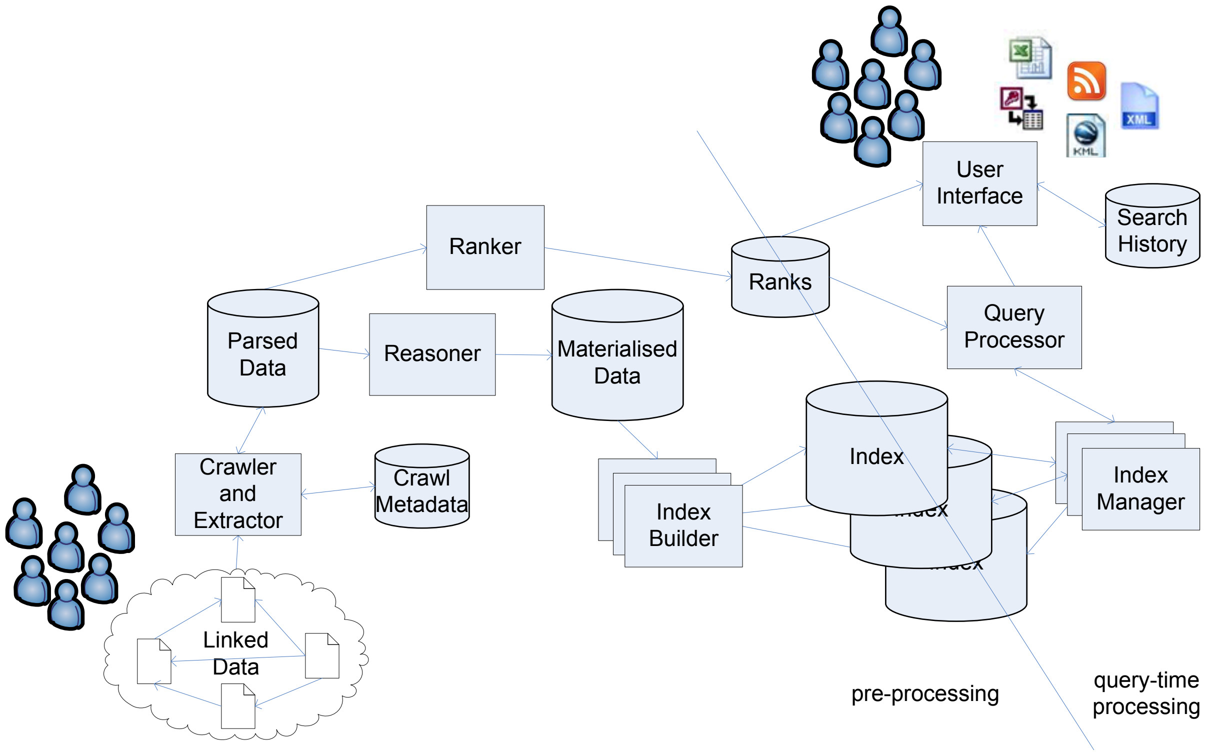


Figure 3.1: Data search and navigation system architecture overview.

- **Object Consolidation** Object consolidation is implemented to improve the quality of data by merging data from multiple sources and schemas into a consolidated dataset. Within RDF, URIs are used to uniquely identify entities. However, on the web, URIs may not be provided or may conflict for the same entities. We can improve the linkage of the data graph by resolving equivalent entities. For example, we can merge equivalent entities representing a particular person.

- **Ranker** To score importance and relevance of results during interactive exploration, we use IdRank [HKD09]. IDRank is a link analysis technique which is used to simultaneously derive ranks for all RDF elements and data sources. Ranking is a vital component in search and query interfaces operating on web data and is used to prioritise presentation of more pertinent search results. Ranking is also useful for prioritising prospective choices users can take to build up their queries.

- **Index Builder** The index builder constructs index files over the materialised dataset. The indexer provides a general framework for locally creating and managing inverted keyword indices and statement indices; we see these two index types as the fundamental building blocks of a more complex RDF index. The index structure comprises of a complete index on quadruples [HD05b] with keyword search functionality based on a standard inverted index. The keyword index functionality maps keywords – which have been extracted from literals – to subject nodes. Each statement index offers prefix lookups over a set of statements of fixed length in a certain order. Multiple statement indices can provide complete support for all possible lookups on statements of any length and can be used to develop optimised methods of servicing otherwise problematic queries. Our framework, with combinations of keyword and statement indices, can be used to implement specialised systems for indexing RDF.

- **Index Manager** The index manager is responsible for providing access to the index files via a network. The index managers can be distributed across a number of machines. The Index Manager provides network access to the local indices, offering atomic lookup functionality over the local indices. Local indices can include keyword indices on text and statement indices such as quad indices on the graph structure, and join indices on recurring combinations of data values.
- **Query Processor** The query processor has to support queries arriving via the user and application programming interfaces. The query processor supports top-k processing of queries posed via the user interface.The first step in the query evaluation is to create and optimise a logical plan, which is then executed over the network in a parallel multi-threaded fashion, accessing the interfaces provided by the local index managers resident on the network.
- **User Interface** To provide user-friendly search and browsing over the data, we provide a user interface which acts as the human access point to the engine. Users typically start by specifying keyword searches and then incrementally build queries to browse and navigate the object graph – through relationships (paths) – and retrieve information about objects.

Please note that the components can be separated into those that perform pre-processing on the data, and those which are involved during query time. In the remainder of the thesis, we first discuss the pre-processing components, followed by the query-time components.

3.7 Related Work

In the following, we review and contrast the architectural characteristics of information retrieval systems [SM84, BYRN99, WMB99], relational database systems [GMWU99, RG03], data warehousing systems, and semantic web systems. The aim is to show that the various system designs have evolved due to application-specific requirements and are tailored towards these requirements. We argue that searching and navigating structured collaboratively-edited data presents unique requirements to a system and thus leads to a system architecture with a novel combination of components and characteristics (as argued by e.g. [ACMD05]).

Table 3.2 shows the typical feature combination of established large-scale architectures of information retrieval systems, relational database systems, and data warehouse systems. Please note that variations of the architecture are possible depending on the use case scenario. In general, information retrieval systems perform best-effort query processing over large document corpora and employ the measure of precision and recall to measure the quality and comprehensiveness of results. Database systems, on the other hand, require exact queries but can deliver exact answers, mainly due to manually provisioned or curated – and thus clean – data. Data warehousing systems typically collect data from multiple relational databases and provide data aggregation, report generation and data analysis, often for narrowly specified domains and scenarios. Semantic web systems still differ more widely in their characteristics and are discussed later.

Dimension	Information Retrieval	Relational Database	Data Warehouse
Data Model	Documents	Relations	Relations
Data Provisioning	-	Transactions	Extract/Transform/Load
Data Integration	-	-	Merge Rules
Index Structure	Inverted Index	B+-Tree	B+-Tree, Bitmap Index
Query Processing	Boolean Keyword	Complex Query	Complex Query
Ranking Scheme	-	-	-
User Interaction	Keyword-Result	-	Reports

Table 3.2: Contrasting the characteristics of information retrieval, relational databases, and data warehouse system architectures.

Information Retrieval Information retrieval systems allow users to pose keyword queries over document collections [BYRN99]. Typical systems operate on document corpora; lookup operations are enabled by creating inverted indices: the documents in the collection are tokenised and an index is built which maps words to the documents in which they occur. Queries are keyword-based, with optionally boolean modifiers (AND, OR, NOT) to specify which documents to include or exclude in the result set. While join processing in the information retrieval area is usually done by intersecting so-called postings lists, join processing in the database area is more intricate. Systems in this class include library catalogue systems (which sometimes allow more complex query operations on data consisting of name/value pairs) and site-wide search on websites. Typically, the user interaction model in these systems is relatively simple [SBC98].

Relational Database The relational model is the prevalent paradigm in the data management area [Sto08]. The typical application domain is online transaction processing inside organisations, storing all kinds of business-related data. Typical index structures employed are B+-Trees [BM72, Com79] which allow for lookups, inserts and deletes while keeping the index structure balanced. The ACID properties (Atomicity, Consistency, Integrity, Durability) ensure reliable data storage, at the expense of performance: the various measures used (such as two-phase commit) represent an overhead in storage and processing. The interface to relational databases is SQL (Structured Query Language), which requires that end users know the schema of the underlying database and are able to write queries in SQL syntax.

While relational databases are dominant in the data management arena, very large scale systems typically do not use off-the-shelf database products. Web search engines mostly employ specialised systems which are only loosely rooted in database research; similarly, large-scale data warehouses – which are close in spirit to our solution – stress read performance over writes and updates. Also, these systems typically employ some form of compression to cope with the high data volume, and run in a distributed environment over a number of machines.

Data Warehouse Data warehouses are also built once-off.The idea is to pre-compute as much as possible to reduce the amount of computation necessary during query processing to keep the effort during query-time to a minimum.

Usually, a large effort is spent in data integration systems to write wrappers that import data. In contrast, we focus on an "information hub" scenario which includes the inverse: namely modules to export data in a number of formats.

Two popular mechanisms exist in the area of relational data integration systems: data warehousing, where data is being replicated from the source databases and pre-processed, and the so-called virtual integration approach in which a query planner determines sources and performs the integration at query time.

Virtual integration has the benefit of providing more timely information; however, a potentially large dataset has to be processed during run-time which negatively impacts performance. On the other hand, a virtual integration approach requires data sources which are able to answer and evaluate queries: an assumption which does not always hold. Even if the data source could answer queries, the combined system would take at least as much time as the slowest data source. In addition, intermediate results during query processing may get very big (several tens of MB of data) which would incur high network costs alongside the time it takes to transfer the data.

In very large data warehousing applications, specialised appliances are often employed – e.g., Netezza and other database machines – to be able to handle the computational expense and the complexities of query processing.

Semantic Web The need to deal with structured data rather than documents affects each component in the architecture of a semantic web search engine over a traditional one. Early prototypes using the concepts of ontologies and semantics on the web include Ontobroker [DEFS98] and SHOE [HHL99], which can be seen as predecessors to standardisation efforts such as RDF and OWL.

Swoogle [DFJ$^+$04] utilises data from the Semantic Web, but is restricted to indexing natively RDF data. Moreover, Swoogle views the Semantic Web as a collection of documents or files and enables keyword search facilities over the files. In contrast, we leverage data in both XML and RDF formats, and view the Web of Data as a massive directed graph with labels on both nodes and arcs. The main distinction between web search engines (and Swoogle) and semantic search engines is the ability to perform queries over structured data involving joins. Therefore, we utilise methods from databases in the system, especially indexing and query processing, but also object consolidation: techniques known from data warehousing and data integration systems.

The recently developed Falcon Search[3] offers entity-centric searching over RDF instances and classes based on Lucene and MySQL. Sindice[4] is a registry and lookup service for RDF files based on Lucene. Watson[5] is an effort to provide natural language search facilities over Semantic Web data.

Parts of this chapter have been published as [HHD$^+$07a].

3.8 Conclusion

Systems integrating data in the broadest sense can be divided into two approaches: systems trying to aggregate data during query-time, and systems relying on central collection and pre-processing. Judging from the deployed systems for searching and querying, the pre-processing flavour of systems is the more prevalent.

We have presented a system architecture for a search and navigation system acting on collaboratively-edited datasets using a modus operandi where data is aggregated at a central location and pre-processed. We have discussed requirements and derived a novel system architecture optimised for the task at hand using an amalgamation of techniques from relational databases, information retrieval, and web search systems, along the lines of argumentation as laid out in [Sto08] and [Bre98]. It has been noted that web search engine systems can benefit from database technologies [Bre98], and we combine techniques from both fields in our work. In the rest of this thesis, we discuss each of the components in detail, starting with the components that carry out pre-processing followed by the components involved during query-time. We present design and implementation details and evaluate the individual components.

[3]`http://iws.seu.edu.cn/services/falcons/`
[4]`http://sindice.com/`
[5]`http://watson.kmi.open.ac.uk/WatsonWUI/`

Chapter 4

Web Crawling and Data Extraction

4.1 Introduction

The quality and utility of any search system depends on the quality and utility of the underlying dataset. The system can only return the required information to the user if the corpus contains the information a user is seeking. Since we aim at offering search and exploration of domain-independent information spaces, simply offering access to certain specialised dataset (such as IMDB, DBLP, DBpedia) is not enough. A general-purpose search system requires access to a broad selection of topics on the web to be able to serve as a general and broad search solution.

The architecture of the web does not directly support searching and querying for pages, since the web standards themselves do not include any global directory of web pages and topics. The only supported lookups by the web architecture are dereferencing URIs according to a set of underlying best practices and foundational standards of the web – the Hypertext Transfer Protocol (HTTP) being one of the more important standards. Dereferencing the URI of a resource via HTTP typically yields content describing the resource. To offer search and query functionality over web resources, automated programs – so-called crawlers – traverse the hyperlink graph and download and store pages along the way. In a later step, the fetched pages are processed and indexed. Please note that a crawler cannot download all available pages because of the decentralised nature of the web and its massive scale.

Crawling for HTML pages has been extensively studied; in this chapter, we show how to extend the crawling procedure to the case of structured data. The key challenges for dealing with heterogeneous semantic data sources are as follows:

1. **Information Extraction** Since a large fraction of the Web consists of HTML, images, or videos, a system that aims at broad coverage on web information has to leverage data from these unstructured sources. A large number of file formats specify means of embedding metadata into the file. The embedded metadata can serve as a rich source of information that should be exploited for a structured data search system.

2. **Linkage** HTML crawlers extract `a href` links from HTML pages in order to find additional sources to crawl. Files containing structured data are rarely linked among each other; often, resources containing structured data are referenced from within HTML documents. Therefore, a crawler for structured data requires some means to extract links from a variety of file formats. The system has to be able to detect the file format of content returned when dereferencing URIs without the file content itself, to be able to decide on the priority of lookups.

With the advent of Linked Data[1], the interlinkage between dataset should increase, however, RDF data on the web is still sparsely linked compared to HTML documents.

Since there are myriads of structured data files available on the web, the diversity of data formats and data models employed makes integration labour-intense. To be able to scale to the web, the methods described in this chapter require minimal manual intervention when deriving RDF from various structured and unstructured file formats. We employ scripts to extract metadata from file formats such as MP3 audio or MPEG video. Also, scripts and wrapper programs can be employed to manually convert relational or XML-based corpora into RDF, or derive more detailed information from documents: e.g., via information extraction techniques. However, these techniques are typically labour- or CPU-intense; hence, we focus in this chapter on automated ways of deriving structured content.

Both data collection and on-demand lookup steps have to take into account requirements applying to all crawlers independent of the content retrieved:

- **Politeness** Crawling puts a strain on the infrastructure of both the organisation that hosts the crawling system and the web sites that are the source of the data. A standard called Robots Exclusion Protocol[2] gives a site owner means of controlling the behaviour of crawlers, and thus helps to avoid denial-of-service attacks. The protocol allows site owners to exclude certain crawlers from downloading pages. In addition, there is a convention that well-behaved crawlers should wait a time period (in the order of seconds) between subsequent requests to the same site.

- **Bandwidth and Caching** Another set of agreements relates to the use of compression (e.g. Accept-Encoding: gzip) whenever possible to reduce bandwidth, and the use of caching as specified in the HTTP standard to avoid repeated transmission of data that has not been changed.

- **Tolerance to Errors** Due to the chaotic nature of the web (for example, servers might be unavailable and documents may contain a broad range of peculiarities and errors), practical crawler architectures typically employ checkpoints and defined stages that can

[1]`http://www.w3.org/DesignIssues/LinkedData`
[2]`http://robotstxt.net/`

be restarted independently. Crawlers have to be robust and error tolerant to be useful and scale to large amounts of data.

- **Performance** A web-scale crawler has to be able to process billions of documents and to run for many weeks. The scale of the web mandates that crawlers employ highly scalable data structures and do not require expensive steps. Thus, crawlers have to be carefully optimised and distributed over multiple machines in order to be able to process even modest fractions of the web.

In the remainder of this chapter, we describe our lookup architecture – which is based on a pipelined architecture and can be used for either data collection or on-demand lookup steps, describe how we incorporate data extraction into the lookup process, and present a scheduling heuristic that maximises the amount of structured data retrieved.

4.2 Crawler Architecture

In the following we describe our general approach to crawling, and discuss the various stages of the crawling algorithm in detail. We assume a pipelined infrastructure similar to the one defined and analysed in [MRYGM01]. However, we have to adapt the approach for web data due to the variety of stages and different time and space behaviour.

The high-level algorithm is shown in Algorithm 1. The process of crawling web data can be logically split into five phases. We refer to these phases as *fetch*, *detect*, *transform*, *extract*, and *schedule*.

```
$uris$: input list of URIs
for $u$ in $uris$ do
  if $u$ not yet visited then
    $c$ = fetch($u$)
    store return code and date
    $t$ = detect($u$, $c$)
    $d$ = transform($c$, $t$)
    store $d$
    $l$ = extract($d$)
  end if
end for
$l$ = schedule($l$)
crawl($l$)
```

Algorithm 1: crawl(*uris*): high-level crawling algorithm

During the fetch phase, the files are downloaded from the web. The detect phase subsequently decides on the type of the content, e.g., `application/rdf+xml` or `image/gif`. The transform phase translates the content into the common RDF data format. The extract phase derives links to more data sources from the downloaded content. The final schedule step prioritises the URIs according to a heuristic that assesses the probability that a URI contains structured data.

We now provide a rationale for some of the different phases in more detail.

Detect A challenge in dealing with multiple data formats is to be able to accurately detect the content type and format of documents. Most of the data formats can be detected by using the file extension or the content-type returned with the header part of an HTTP request. In the case of XML files, the MIME type and the file extension give indication for XML content, but do not give any information about whether the content is well-formed, or which schema is used. Sometimes this information is important: therefore the content itself has to be investigated.

Transform Since we are aiming at a general indexing and querying infrastructure we need mechanisms to extract information from the files and transform them to a structured representation. Ideally, we would like to use a declarative transformation language so that users can define transformations without the need to write code in a procedural language. However, the system should be also able to use procedural language code to extract data from binary data or natural language text, ultimately arriving at a representation of the metadata.

Extract For extracting URIs, we decided to perform a lookup against the final cleaned and structured dataset. We perform URI extraction at the end of the document-processing pipeline, since at that stage the data is represented in a uniform data format and we are able to extract URIs cheaply. Depending on the required crawling strategy (only crawl one site, perform shallow crawling and only take external links into account, etc), we can adapt the URI extraction process. We need to extract links also from HTML pages, otherwise we will not discover the URI of some structured pages, since some files containing structured data are currently only sparsely interlinked from other structured data sources. URIs to structured sources appear mainly in `a href` and `link href` tags within HTML documents. In addition, HTML pages may contain content in microformats or RDF/a[3].

Schedule To be able to better utilise bandwidth, CPU and storage resources, a crawler for our system ideally only downloads and transforms files which contain a certain amount of structured data. During the schedule phase we employ heuristics that allow us to decide which URIs the crawler should download first in order to achieve a high quality dataset early on during the crawl. We also specify a cut-off point to be able to focus our crawling process on certain file formats while ignoring others.

To be able to scale, we need to parallelise and distribute the system. Fetching the data takes much less CPU time than processing. Thus, we want to perform steps in parallel, which means we have to use multiple threads that fetch data and multiple threads that process data etc. Communication between the steps is done via queues. If we want to scale up the process even further, we can replace threads with multiple computers, queues with remote/persistent

[3]`http://www.w3.org/TR/xhtml-rdfa-primer/`

queues, and pipes with network data transfer. As a side note, in the distributed setup it is easy to identify bottlenecks and resolve them by adding new machines to a phase. Another benefit of a distributed architecture is that it facilitates the integration of external components (i.e., web services) into the process.

Our goal is to analyse the complexity of the single tasks and to find the right balance in server ratios to keep the average utilisation of the servers as high as possible. In the next section we describe each processing step, investigate the complexity thereof and present experimental measurements.

4.3 Processing Pipeline

In this section, we describe each step in the processing pipeline in detail. The processing pipeline is composed of five different modules, each of which is capable of running the task in a multi-threaded fashion. First, the fetching module downloads the content and header information of a web page. Second, the detecting module determines the file type of the web page. Third, based on the file type, the transformation module converts the original content into RDF. Fourth, the indexing module constructs an index over the RDF data to enable URI extraction. Fifth, the extraction module poses a query over the index to extract URIs and feeds the resulting URIs back into the pipeline.

To be able to pass parameters between different phases, the system needs to store information associated with the URIs. We put the metadata associated with a URI as RDF triples in a metadata store which runs on a separate machine.

Each phase has an associated queue which contains URIs of pages to be processed. Each phase takes a URI from the queue, retrieves the content from a previous phase if necessary, processes the content, stores the content on disk, and puts the URI into the queue corresponding to the next step in the pipeline. Please observe that storing intermediate results to disk is a purely pragmatic decision to increase stability. Content is passed to successive steps via an Apache HTTP server.

In the following sections, we include complexity analysis and experimental results for each step. We carried out the experiments using a random sample of 100k URIs obtained from the Open Directory Project[4]. We performed all experiments on nodes with a single Opteron 2.2 GHz CPU, 4 GB of main memory and two SATA 160GB disks. The machines were interconnected via a 1GBbp network adapter on a switched Ethernet network.

[4]`http://dmoz.org/`

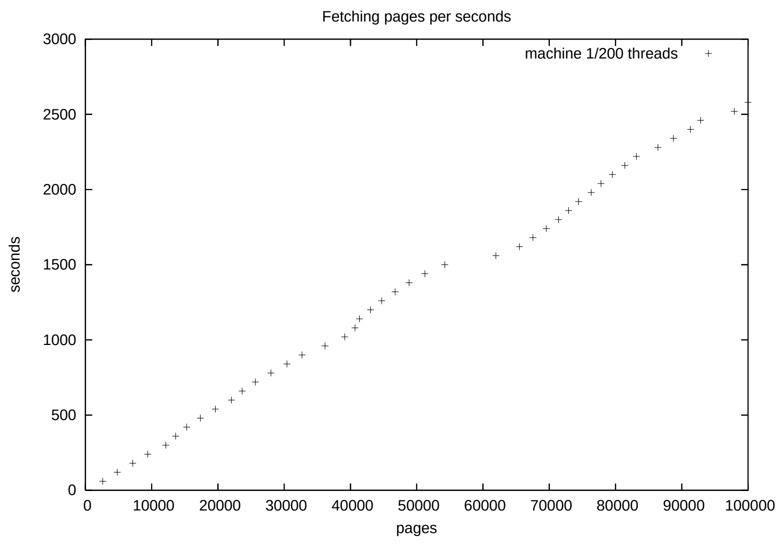


Figure 4.1: Experimental results derived from crawling 100k randomly selected URIs.

4.3.1 Fetching Data

The functionality of the fetching module includes obtaining a new URI from the queue, checking for a `robots.txt` file to adhere to the Robots Exclusion Protocol[5], and fetching and storing header information and content.

After obtaining the next URI from the queue, we retrieve the `robots.txt` information for the host either from the metadata store or directly from the host. Then we determine if the fetcher is allowed to crawl the page or not. If the URI passes of the check, we look at the content length provided by the header information. To avoid downloading very large files, we compare the content-length from the header-field with a given file size threshold.

If the URI passes all these checks, we connect to the web server and download the content of the page. Then we store or update the header information in the metadata store. Finally, we send the URI to the next module in the pipeline and return to the beginning to poll the next URI from the queue.

To provide an estimate of the complexity of the step, let N be the size of the documents fetched, including header information. The fetch step needs to transfer N bytes from the Internet, which takes linear time in the size of the content, O(N).

We verified the complexity analysis experimentally. We chose randomly 100k URIs from ODP's collection of over 5M sites. Figure 4.1 shows the experimental results for the crawling component resulting in 78,038 downloaded pages (1.4 GB of data). The fetching component achieved an average download rate of around 600 KB/sec.

[5]`http://www.robotstxt.org/wc/exclusion.html`

4.3.2 Detecting File Types

The detection module tries to determine the exact content type of the data, which is later used in the transformation phase to choose the most appropriate transformation module. The type detection is based on the information we are able to derive from the URI, the header fields and the content of the page itself.

In the first step of the file type detection process, we try to detect the content type based on the file extension of the URI. The second step retrieves the content-type header field from the metadata store and compares the header field to a list of content types. Table 4.1 lists all supported content types and the information the system needs to detect them. If one of these checks successfully detects a type, we can stop the process and store the type on the metadata repository. In case of XML content, we perform another check to figure out the schema of this XML file. In this case, we must parse the content itself.

TypeID	RFC	MIME media type	File extension	Root element
HTML	2854	`text/html`	`.html .htm`	`html`
XHMTL	3236	`application/xhtml+xml`	`.xht .xhtml .html`	`xhtml:html`
XML	3023	`text/xml application/xml`	`.xml`	`-`
RSS2.0	-	`application/rss+xml`	`.rss`	`rss`
Atom	4287	`application/atom+xml`	`.atom`	`atom:feed`
RDF	3870	`application/rdf+xml`	`.rdf`	`rdf:RDF`

Table 4.1: File types the system is currently able to handle.

If we detect XML content, we try to find out the specific type of the XML content – that is, we retrieve the content data from the file system and parse it with a SAX XML parser. We try to extract the namespaces and root element of the XML file and compare the values to the known content types. If all checks fail, we assume an unknown or unsupported content type. Finally, we store the type on the metadata store and forward the URI to the next pipeline module.

During the complexity analysis, we do not consider the simplest case where we can detect the file type based on file extension or header information. Let N be the size of the XML document whose content type we want to detect. Parsing the XML content – in this case utilising SAX to retrieve the root element – has a time complexity of O(N).

Figure 4.2 shows the experimental results for the file type detection phase.

4.3.3 Transforming to RDF

For transforming the content into the common data format RDF, the system applies different transformation modules depending on the type of the content. The transformation phase can be split into two steps: (i) conversion from non-XML data – such as HTML – into XML by using

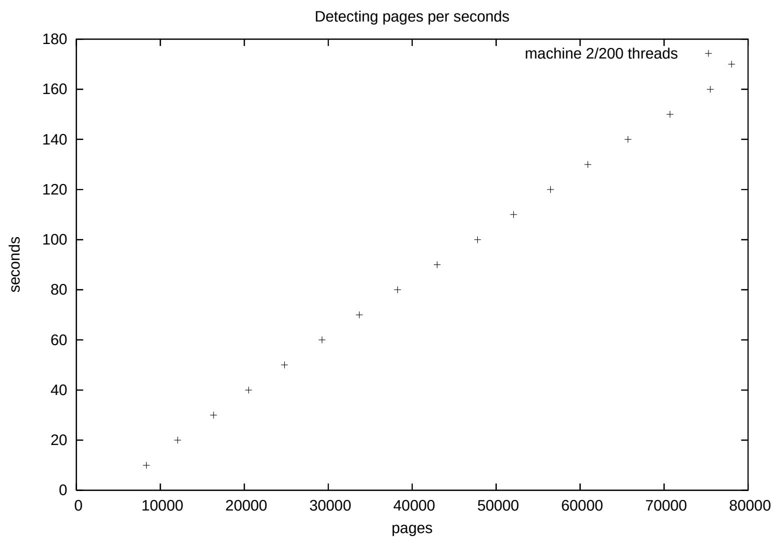


Figure 4.2: Experimental results for detecting file types.

user specified transformation tools and (ii) transformation of XML data to RDF via XSLTs and xsltproc[6]. For a given URI we retrieve the content type, which has been added in the detect phase, from the metadata store. Depending on the result of the query, we execute different transformation modules on the content. Naturally, if the data format is already RDF, we can skip the transforming step.

To transform non-XML data, we can invoke external services which convert the data directly into XML or RDF. At the moment, our support for non-XML data consists only of cleaning up HTML using the tool Tidy[7] running as a cgi-bin on a HTTP server, but various external services for extracting metadata – e.g., from image or video files – can be easily plugged in.

To transform XML data, we use xsltproc with an XSLT from the file system depending on the type identifier of the page. We use an XSLT that transforms RSS 2.0 and Atom to RDF[8]. We also developed an XSLT[9] which transforms XHTML pages into an RDF representation based on RDFS, DC, and FOAF vocabularies. In this stylesheet, we extract from a HTML document the following information: title, email addresses, images, and relative and absolute links and their anchor labels.

After the URI passes successfully all content transformation steps, we pass it to next step of the pipeline. In an optional step, we are able to run named_entity_extraction($result$) to extract named entities, and add the resulting entity descriptions to $result$.

The worst case scenario when performing the transforming step is in dealing with HTML documents, because we must first pass the content to Tidy and then perform the XSLT trans-

[6]`http://xmlsoft.org/XSLT/`
[7]`http://tidy.sourceforge.net/`
[8]`http://www.guha.com/rss2rdf/`
[9]`http://sw.deri.org/2006/05/transform/xhtml2rdf.xsl`

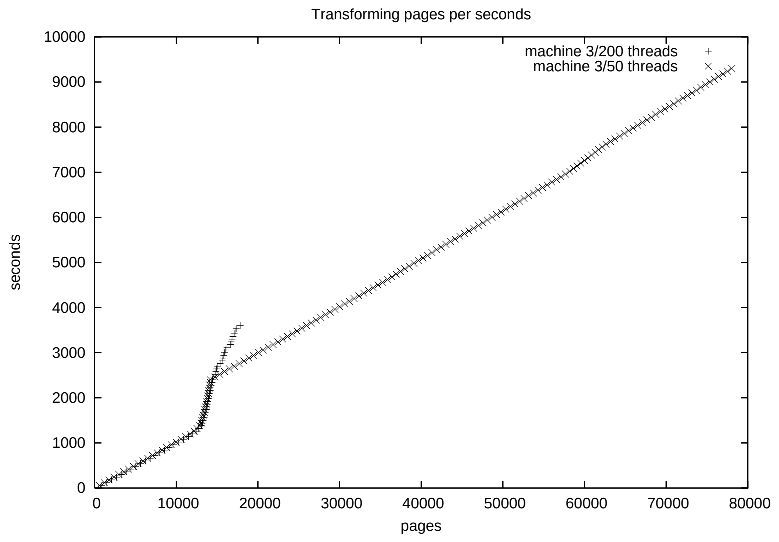


Figure 4.3: Experimental transformation performance using 50 and 200 threads.

formation. Imagine a document of size N and an XSLT stylesheet of size M. We assume Tidy takes time linear to the size of the content O(N). The worst-case complexity for XPATH has been shown to be $O(N^4 * M^2)$ [GKPS05]; however, for a large fragment called Core XPath the complexity is O(N*M). Our XHTML XSLT uses only simple Core XPath queries: therefore, the worst-case complexity for the step is O(N*M).

Figure 4.3 shows the experimental results for the transformation component utilising the `xhtml2rdf.xsl` and `rss2rdf.xsl` stylesheets. Using 200 threads as in all other tests, the transformation performance decreased rapidly after around 13k pages because the machine was assigned with too many transformation tasks and had to swap. Therefore we plotted only the first 60 minutes of running time. We repeated the tests with only 50 threads to not overload the machine. In the end, the transformation step yields 907 MB of XHTML resulting in 385 MB RDF/XML. The discontinuity of the performance can be traced back to a single large file which took a long time to process.

4.3.4 Extracting URIs

To feed the processing pipeline with new URIs, we have to extract URIs from the indexed content, which is done in the extraction module. The process can be divided into two steps: the first step is to extract URIs from the data and the second step is to filter the URIs according to specified criteria.

To extract new URIs, we execute a query on the index for typical link predicates such as `rdfs:seeAlso` and `rss:link`.

If an extracted URI is to be added to the queue, we pass this URI through the installed

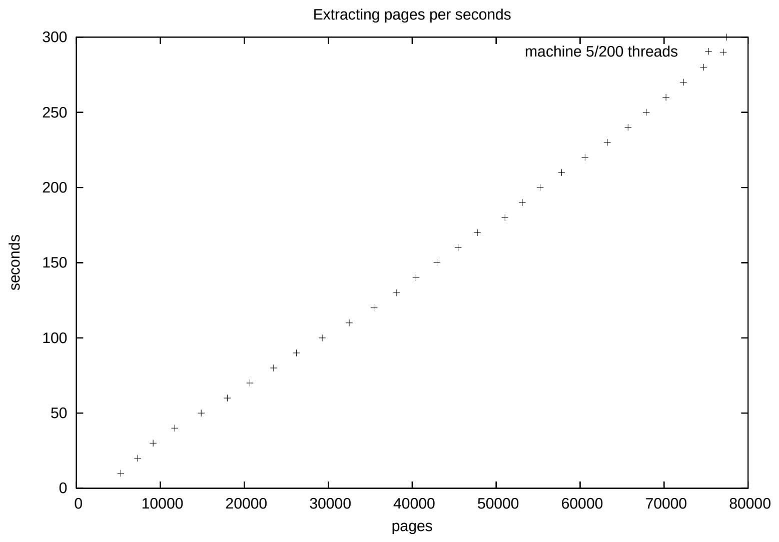


Figure 4.4: Experimental link extraction performance.

filter. In this filter we can restrict which URIs should be sent to the fetching module. If we want to crawl only a specific domain or a set of domains, we can filter the addition of URIs using regular expressions stored in memory; note that it is also possible to add new expressions to the filter at runtime.

Figure 4.4 shows the experimental results for the extraction component. We discuss the results of all phases in the next section.

4.3.5 Scheduling and Reordering Heuristics

Search engines focusing on particular media types face a difficulty in discovering suitable URIs on the Web. Since the engines are only interested in a small fraction of the Web, a crawler should use heuristics to concentrate on that fraction. To devise such a heuristic, we postulate four hypotheses based on RFCs and W3C recommendations to find cues for certain content types. Tests on a corpus of 22m files (793GB content size) and 630m URIs show that for the content types text, image, and application, the recommendations are almost being followed, while results for audio and video are much less consistent. Our findings and recommendations can be implemented as heuristics for efficient discovery of structured content on the Web.

Finally, we employ a heuristic to prune the extracted links and order them according to the probability of a URI containing structured information.

4.4 Analysis and Tradeoffs

In the following, we analyse the performance results for the five phases of the pipeline process and discuss two questions: i) how to distribute the individual phases to remove potential bottlenecks and fully utilise the processing power of each machine and ii) how to run multiple pipelines in parallel to achieve a throughput of the total system which can be calculated by: *number of pipelines * pipeline throughput.*

Currently, the transform phase represents the bottleneck in the pipeline and can only process a fraction of the pages delivered by the fetch and detect phase. The random sample of URIs are biased towards HTML data, which means that during the transform phase almost every page has to be processed. If we are able to reduce the amount of HTML and XML sources and increase the amount of RDF sources, the transform phase has to process less pages and as a result the throughput (in terms of time per page) increases. However, given the fact that the majority of content on the web is in HTML format, we have to distribute the transform component to achieve acceptable performance.

Assuming a crawling architecture as described in this chapter, we can distribute phases by just adding more machines. Pages are assigned to nodes using a hash function. In initial experiments we observed that we can scale up the fetch step by a constant factor if we add more fetcher machines and all fetcher nodes take URIs concurrently from the queue. The case is a bit different for the transform step; here, we employ one thread pool with individual threads which retrieve a URI from the previous step in the pipeline and invoke Tidy and XSLT operations on cgi-bins running on a web server. In other words, while the other phases employ a pull model, transform component tasks are pushed to external processors. We choose the push model to allow for external transformation services.

Figure 4.5 shows the performance results where: i) all steps and external processors run on one node, ii) one node was used for the steps and two nodes for external processors (1+2), and iii) four external processors (1+4) were used. Please note that the scale-up is not constant in the number of machines added. The reason is that the hash function assigns the pages equally to the external processors. If a single page takes a remarkably long time to process, the external processor node cannot keep up with the assigned operations and at some point in time needs to swap, which leads to a decrease in performance. The discontinuity in the measured overall performance in the distributed transform setup can be explained by a large file which took a long time to process (see also Figure 4.3).

Apart from the issues described for the transform steps, we claim that all other steps can be scaled by a constant factor (w.r.t. the number of machines added) using a hash function to distribute URIs to nodes since URIs in those phases can be processed independently. Table 4.2 shows the throughput in pages per second for each phase, and a ratio that determines which fraction of the stream (assuming that the fastest component determines the throughput) one node can process.

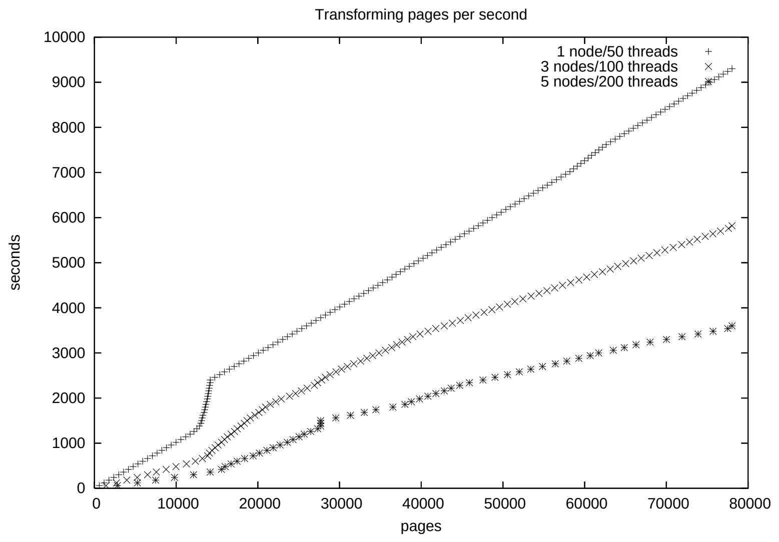


Figure 4.5: Performance measurement for the transform phase with 1, 3 (1+2), and 5 (1+4) transform nodes.

Phase	Servers	Pages/sec	Ratio.
fetch	1	38	0.082
detect	1	460	1.0
transform	1	5	0.011
transform	1+2	13	0.028
transform	1+4	21	0.045
extract	1	260	0.565

Table 4.2: The number of servers and the achieved performance. Ratio is calculated based on the fastest phase (1 = 460 pages/sec).

To be able to scale up the system even further, we can employ more pipelines and achieve a total throughput which can be calculated by multiplying the number of pipelines with the throughput achieved on one pipeline. The limit is then only determined by how many resources (Internet bandwidth and number of machines) are available.

4.5 Related Work

Most of the papers concerned with web content analysis contain statistics about the distribution of content size, top-k media types, top level domains, and HTTP status codes. The data set presented in [NH02] consists of 815m documents while [HN99] analysed a corpus of 75m URIs. [GS05] characterises the web community of people in Portugal based on 3.2m pages in 2005. [LG00] analyses the accessibility of information indexed by search engines and presents estimates such as the number of online servers on the web and the number of files and links per domain.

Surveys focusing on content metrics exist for media types `application/xml` or `application/rdf+xml`. E.g. [Ebe02] analysed the RDF Web in 2002, [WPH06] published a survey of the OWL ontology landscape in 2006, [DF06] analysed RDF and OWL documents based on 1.7m sources in 2006, and [BSLF06] studied the usage of web services in 2006. These studies do not present information about indicators for determining the content type of a given URI.

There are services that try to perform resource discovery in a distributed manner. The idea is to outfit browsers with plugins that detect autodiscovery links to RDF data on the pages a user is browsing, and "ping" a central server with the URI. The central server maintains a repository of URIs, and provides lists of these URIs to crawlers[10].

Crawler frameworks such as UbiCrawler [BCSV04] or Mercator [HN99] are focused on the performance of the crawling step only. Similarly, IRLbot [LLWL08] is a system that uses optimised data structures and heuristics for scheduling URI retrieval. Google [BP98] handles HTML and some associated link structure. We focus less on crawling but on detecting Semantic Web data and the transformation of XHTML and XML to RDF.

A few efforts have been undertaken to extract structured content from web pages, but these efforts differ considerably in scale. Fetch Technologies' wrapper generation framework[11] and Lixto [BFG01] are examples of commercially available information extraction tools. Lixto defines a full-fledged visual editor for creating transformation programs in their own transformation language, whereas we use XSLT as transformation language and focus on large-scale processing of data. Fetch (similarly [MAT$^+$04]) combine wrapper generation and a virtual integration approach, whereas we use a data warehousing approach and therefore need scalable index structures.

SemTag (Semantic Annotations with TAP) [DEG$^+$03] perform mostly text analysis on documents, albeit on a very large scale. In contrast, we extract structured information from documents and XML sources, and combine the information with RDF files available on the web.

This chapter has been published as [HUD06].

4.6 Conclusion

We have presented a distributed system for syntactically integrating a large amount of web content. The method involved accessing web pages, transforming the content into an RDF representation, extracting links, and prioritising URIs according to their probability of containing structured information. We have given both the theoretical complexity and experimental performance of the pipeline. The outcome of the fetching and extraction process for a given

[10] `http://pingthesemanticweb.com`
[11] `http://www.fetch.com/`

web document is an RDF file that contains a structured representation of the content and its metadata, together with detected media type and additional header information. The system is capable of invoking external services for content transformation. The outcome of the crawling process forms the basic corpus used in subsequent steps.

Chapter 5

Ranking Data and Data Sources

5.1 Introduction

More and more structured and interlinked data is appearing on the web, in the form of microformats, XML, and RDF (Resource Description Format). A common feature of these formats is that they take a loosely object-oriented view, describing objects such as people, events, or locations. Given that the information published in these formats exhibits more structure and a fine-grained description of objects, typical keyword-based search engines would not exploit the full potential that structured data offers. The established methods for information retrieval are not directly applicable to structured data, since i) the basic result units of search moves from documents to objects which may be associated with several sources and ii) the users are able, in addition to keyword searches, to more accurately state their information needs via precise queries.

The problem of search in hypertext collections has been extensively studied, with the web as the premier example. Given the large number of data providers compared to traditional database scenarios, an information retrieval system for hypertext on the web must be able to handle data with the following properties:

- Domain variety: the web contains data about many topics (e.g., from social networks to protein pathways to entertainment).
- Structural variety: aggregating data from many autonomous web sources – with no data curation or quality control of any sort – results in datasets with overlapping, redundant, and possibly contradictory information.
- Noise: as the number of data contributors increases, so too does the amount of errors (e.g., syntax errors, spelling mistakes, wrong identifiers).
- Spam: when everybody can say anything about anything with little effort or cost, malicious activity emerges.

- Scale: identifiers and documents on the web number in the trillions.

We expect the properties identified above to also hold for structured information sources on the web, though they are typically not taken into account for classical relational query processing or data warehousing where the number of autonomous sources is orders of magnitude lower. In traditional data integration systems, the schema of the integrated data is known in advance and is hard-coded into applications to, for example, determine the order in which data elements are displayed. In this chapter, we focus on the problem of ranking in structured datasets which have been integrated from a multitude of disparate sources without a-priori knowledge of the vocabulary used. We assume a basic interlinked data model, enabling the definition of objects and relationships between those objects. Further, we assume the possibility of global identifiers which enable the reuse of identifiers across sources and thus the interlinkage of data. RDF, and to a limited extent XML and microformats, fulfil these assumptions.

Based on the above scenario the contributions of this chapter are as follows:

- We introduce the notion of naming authority which establishes a connection between an identifier (in our case a URI) and the source which has authority to assign that identifier (in our case a web source, also identified via a URI). The notion of naming authority can be generalised to other identifier schemes (e.g., trademarks) for which it is possible to establish a connection to the provenance of the identifier, such as a person or an organisation (Section 5.4).

- We present a method to derive a naming authority matrix from a dataset, and use the PageRank algorithm to determine rankings for sources. In a second step, an algorithm ranks individual identifiers based on the values assigned to their sources (Section 5.5).

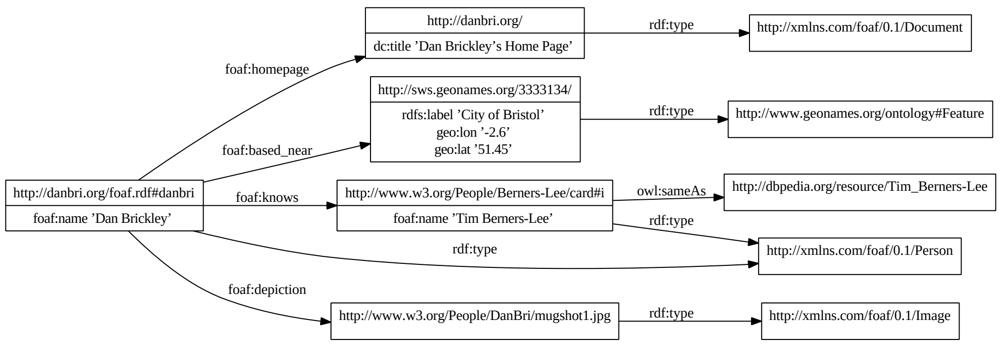
- We provide an experimental evaluation on real-world web datasets containing up to 1.1bn data items from 6.5m web sources, and provide evidence for the quality of the rankings with a user study of 36 participants (Section 5.6).

We present an example scenario in Section 5.2, and provide an overview in Section 5.3. Section 5.7 outlines related approaches and Section 5.8 concludes.

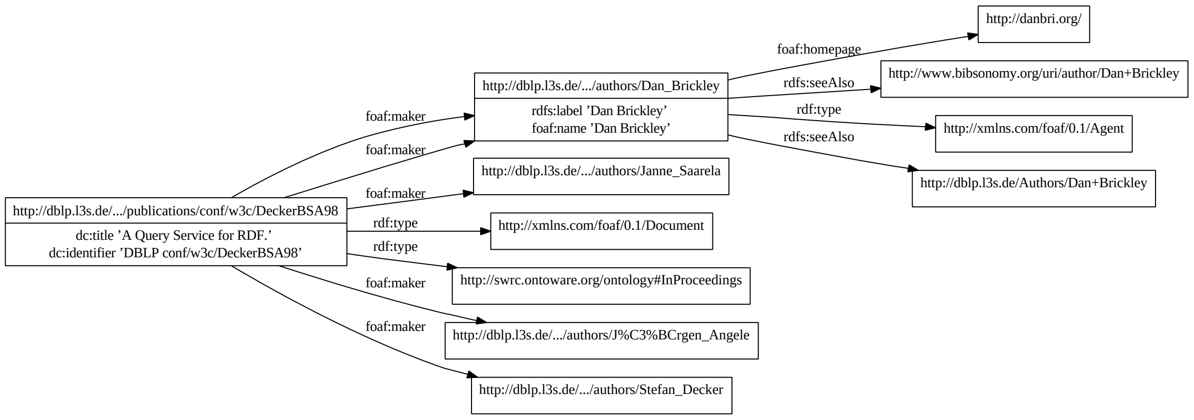
5.2 Motivating Example

For the motivating example, we use social network information describing people – who they know and what they do – as there is large amounts of this data available. Data sources typically express person-related information in the Friend-of-a-Friend (FOAF) vocabulary, but also use their own schemas (i.e., sub-classing the general Person class with classes such as Professor or Ph.D. student). The personal data available on the web exhibits the properties we expect from any large-scale loosely coordinated distributed knowledge base.

In our running example URIs are used to identify both the objects (e.g. `http://danbri.org/foaf.rdf#danbri`) and the data sources (e.g. `http://danbri.org/foaf.rdf`). Identifiers might be reused across sources, for example, the source `http://danbri.org/foaf.rdf` uses the URI `http://www.w3.org/People/Berners-Lee/card#i` to denote the Person Tim Berners-Lee. The reuse of identifiers provides a direct means to consolidate information about objects from different sources. In our experimental dataset, Dan's URI is referenced in at least 70 sources. Representing only a small subset of the available data, the graphs in Figure 5.1 depict object descriptions from two sources.


http://danbri.org/
dc:title 'Dan Brickley's Home Page'
rdf:type
http://xmlns.com/foaf/0.1/Document
foaf:homepage
http://sws.geonames.org/3333134/
rdfs:label 'City of Bristol'
geo:lon '-2.6'
geo:lat '51.45'
rdf:type
http://www.geonames.org/ontology#Feature
foaf:based_near
http://danbri.org/foaf.rdf#danbri
foaf:name 'Dan Brickley'
foaf:knows
http://www.w3.org/People/Berners-Lee/card#i
foaf:name 'Tim Berners-Lee'
owl:sameAs
http://dbpedia.org/resource/Tim_Berners-Lee
rdf:type
rdf:type
http://xmlns.com/foaf/0.1/Person
foaf:depiction
http://www.w3.org/People/DanBri/mugshot1.jpg
rdf:type
http://xmlns.com/foaf/0.1/Image

`http://danbri.org/foaf.rdf`


http://danbri.org/
foaf:homepage
http://www.bibsonomy.org/uri/author/Dan+Brickley
http://dblp.l3s.de/.../authors/Dan_Brickley
rdfs:label 'Dan Brickley'
foaf:name 'Dan Brickley'
rdfs:seeAlso
rdf:type
http://xmlns.com/foaf/0.1/Agent
rdfs:seeAlso
http://dblp.l3s.de/Authors/Dan+Brickley
foaf:maker
foaf:maker
http://dblp.l3s.de/.../authors/Janne_Saarela
foaf:maker
http://dblp.l3s.de/.../publications/conf/w3c/DeckerBSA98
dc:title 'A Query Service for RDF.'
dc:identifier 'DBLP conf/w3c/DeckerBSA98'
rdf:type
http://xmlns.com/foaf/0.1/Document
rdf:type
http://swrc.ontoware.org/ontology#InProceedings
foaf:maker
http://dblp.l3s.de/.../authors/J%C3%BCrgen_Angele
foaf:maker
http://dblp.l3s.de/.../authors/Stefan_Decker

`http://dblp.l3s.de/d2r/resource/publications/conf/w3c/DeckerBSA98`

Figure 5.1: A subset of RDF descriptions about Dan, taken from two sources.

Several challenges arise when displaying data aggregated from the web:

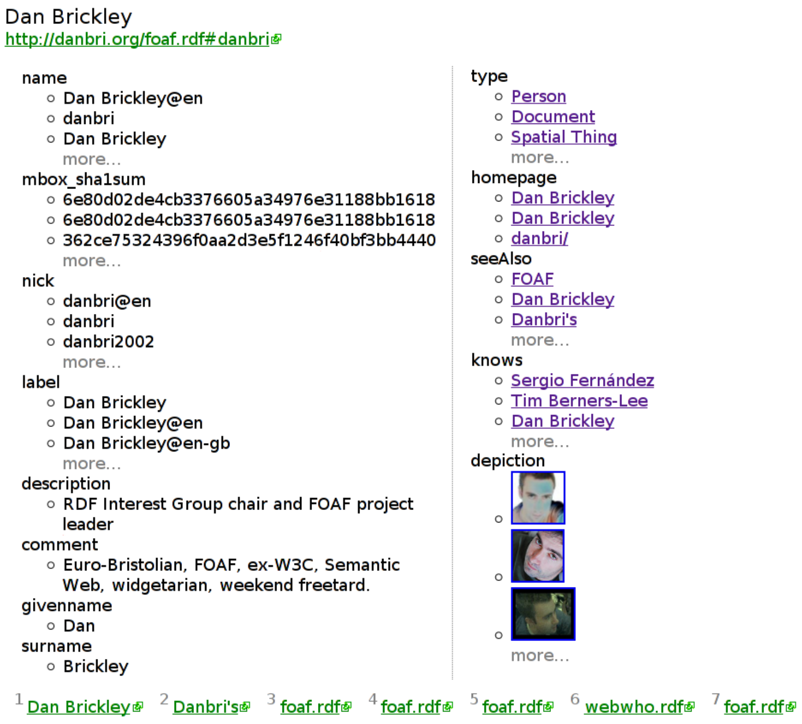
1. How to prioritise results from keyword searches (i.e., in which order to show the resulting objects)?

2. How to prioritise predicates (i.e., which predicate/value pairs to show first)?

3. How to prioritise objects from multi-valued attributes (i.e., which image to show first if there are multiple depictions of a person)?

4. How to prioritise sources that contribute data to an object (i.e., in which order to present the sources, from the most important to the least)?

Question 1) surfaces mainly in a web search scenario, where a search engine returns a list of identifiers matching the user-specified query – keywords or structured query such as “return all instance of type `owl:Class`” or “return all instances of type `foaf:Person` that have `foaf:workplaceHomepage` `http://www.deri.org/`”. Questions 2) to 4) are also relevant for Linked Data browsers such as Disco[1], Data Explorer[2], or Tabulator[3] where the vocabularies used in the data are unknown a-priori to the browsers. Figure 5.2 shows the rendering of information about an object using rankings derived with the method presented here in VisiNav[4].



Dan Brickley
http://danbri.org/foaf.rdf#danbri

- name
  - Dan Brickley@en
  - danbri
  - Dan Brickley
  - more...
- mbox_sha1sum
  - 6e80d02de4cb3376605a34976e31188bb1618
  - 6e80d02de4cb3376605a34976e31188bb1618
  - 362ce75324396f0aa2d3e5f1246f40bf3bb4440
  - more...
- nick
  - danbri@en
  - danbri
  - danbri2002
  - more...
- label
  - Dan Brickley
  - Dan Brickley@en
  - Dan Brickley@en-gb
  - more...
- description
  - RDF Interest Group chair and FOAF project leader
- comment
  - Euro-Bristolian, FOAF, ex-W3C, Semantic Web, widgetarian, weekend freetard.
- givenname
  - Dan
- surname
  - Brickley

- type
  - Person
  - Document
  - Spatial Thing
  - more...
- homepage
  - Dan Brickley
  - Dan Brickley
  - danbri/
- seeAlso
  - FOAF
  - Dan Brickley
  - Danbri's
  - more...
- knows
  - Sergio Fernández
  - Tim Berners-Lee
  - Dan Brickley
  - more...
- depiction
  - more...

1 Dan Brickley 2 Danbri's 3 foaf.rdf 4 foaf.rdf 5 foaf.rdf 6 webwho.rdf 7 foaf.rdf

Figure 5.2: Rendering of information pertaining to Dan Brickley (datatype properties and values on the left, object properties and objects on the right, and data sources on the bottom). The decision on ordering of all data elements on the page has been taken based on rankings alone, without a-priori schema knowledge.

[1]http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/
[2]http://demo.openlinksw.com/rdfbrowser2/
[3]http://www.w3.org/2005/ajar/tab
[4]http://visinav.deri.org/

5.3 Method Overview

Herein, we introduce the data model, present the requirements for a ranking method on web data and outline our procedure for ranking data sources and data items.

5.3.1 Data Model

In general, our ranking method can be applied to datasets with i) global, reused identifiers, ii) tracking of provenance, and iii) correspondence between object identifiers and source identifiers. Specifically, we assume the following:

- a set of identifiers I, encompassing a set of global identifiers U, a set of local identifiers B, and a set of strings L (we omit datatypes such as integer or date for brevity);
- a set of data sources $S \subseteq U$;
- a function *ids* which maps sets of global and local identifiers and literals $i \in I$ to the sources $s \in S$ in which they occur.

This generic model applies to a wide variety of data models, such as hypertext, graph-structured data, and relational data.

5.3.2 Requirements

A ranking procedure operating on collaboratively-edited datasets should exhibit the following properties:

- The use of an identifier owned by source s_a by a source s_b indicates an acknowledgement of the authority of s_a and should benefit from the ranking of s_a
- Data providers who reuse identifiers from other sources should not be penalised, i.e. their data sources should not lose any rank value.
- A data provider who simply links to other important identifiers (requiring no external effort) should not gain any rank from doing so. Using only the node-link graph without taking into account the source (e.g. [BHP04]) makes the ranking method receptive for spam: by adding a triple pointing from a popular URI to a spam URI, the spam URI gains rank from the popular URI.
- We cannot make any assumptions of directionality of links between objects, since link direction is arbitrary (in a triple with u_a on the subject position and u_b on the object position, is u_a related to u_b or u_b related to u_a?). Thus we cannot use links occurring in the data graph as a vote.

5.3.3 Algorithm Outline

Our method for deriving a rank value for all data elements in a dataset consists of the following steps:

1. Based on the occurrence of identifiers $u \in S$, construct the naming authority graph $S \times S$ that serves as input to a fixpoint calculation.

2. The naming authority graph is used to derive PageRank scores for the data sources S.

3. The source ranks are used to derive a rank value for both global identifiers $u \in U$ and data elements with local scope $b \in B$, $l \in L$.

5.4 Naming Authority

The traditional goal of ranking on the web is to rate popular pages higher than unpopular ones. Indeed, PageRank [PBMW98] interprets hyperlinks to other pages as votes. A possible adaptation for structured data sources would rank popular data sources higher than unpopular ones. However, data models such as RDF do not specify explicit links to other web sites or data sources. Therefore, a straightforward adaptation of PageRank for structured data is not possible, since the notion of a hyperlink (interpreted as a vote for a particular page) is missing.

However, a closer examination of the data model leads to the following observation: a crucial feature of structured data sources is the use of global identifiers. Typically, these global identifiers – URIs in case of the web – have a specified syntax, and exploit naming mechanisms such as the domain name system.

Consider Dan's identifier `http://www.danbri.org/foaf.rdf#danbri`. Clearly the owner of the `danbri.org` domain can claim authority for creating this URI. Thus if the URI is used on `danbri.org` to denote Dan, the usage of the URI on other sites can be seen as a vote for the authority of the data source `danbri.org`.

To generalise this idea, one needs to define the notion of "naming authority" for identifiers. A naming authority is a data source with the power to define identifiers of a certain structure. Naming authority is an abstract term which could be applied to the provenance of a piece of information, be that a document, host, person, organisation or other entity. Data items which are denoted by unique identifiers may be reused by sources other than the naming authority.

We now define the general notion of naming authority:

Definition 5.4.1 (Naming Authority) *The naming authority of a global identifier $u \in U$ is the data source $s \in S$ which has the authority to mint the globally unique identifier u.*

5.4.1 Naming Authority for URIs

For naming authority in the RDF case, we assume a relation between $u \in U$ and $s \in S$ in the following way:

- if e contains a `#`, we assume the string before the `#` as the naming authority of the element, e.g. the naming authority for `http://danbri.org/foaf.rdf#danbri` is `http://www.danbri.org/foaf.rdf`.
- if e does not contain a `#`, we take the full element URI as the naming authority, e.g. the naming authority for `http://xmlns.com/foaf/0.1/maker` is `http://xmlns.com/foaf/0.1/maker`.

The Hypertext Transfer Protocol (HTTP)[5], which is used to retrieve content on the web, allows the redirection of URIs, possibly multiple times, resulting in redirect chains. Redirect chains are unidirectional, i.e. the redirect relation does not follow the logical properties of the equivalence relation. To derive the correct naming authority we have to take HTTP redirects into account. Thus, for each naming authority, we check if the naming authority URI is redirected to another URI. If there is a redirect, we assume the redirected URI as the naming authority. E.g. `http://xmlns.com/foaf/0.1/maker` redirects to `http://xmlns.com/foaf/spec/`, hence the naming authority becomes `http://xmlns.com/foaf/spec/`. This is in-line with Linked Data principles[6] for publishing RDF on the web.

5.4.2 Deriving the Naming Authority Matrix

As a first step, we derive the naming authority graph from the input dataset: that is, we construct a graph encoding links between data sources based on the implicit connections via identifiers.

Definition 5.4.2 (Naming Authority Matrix) *Given a data source $s_i \in S$ we define the naming authority matrix A as a square matrix defined as:*

$$a_{i,j} = \begin{cases} 1 & \text{if } s_i \text{ uses identifiers for which } s_j \text{ has naming authority} \\ 0 & \text{otherwise} \end{cases}$$

A naming authority graph for the example in Figure 5.1 is shown in Figure 5.3. In this example we assume the naming authority on a pay-level domain (PLD) level as described in the following.

[5] `http://www.ietf.org/rfc/rfc2616.txt`
[6] `http://www.w3.org/DesignIssues/LinkedData`

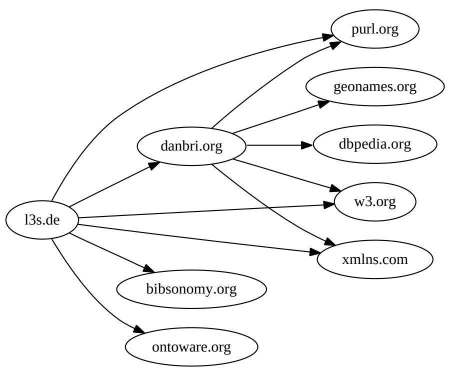


Figure 5.3: Naming authority graph for data in Figure 5.1 on a pay-level domain granularity.

5.4.3 Pay-Level Domains

In a variation of our algorithm ,we use the notion of pay-level domains (PLDs) as defined in [LLWL08]. A top-level domain (TLD) is a domain one level below the root in the DNS tree and appears as the label after the final dot in a domain name (e.g., `.com`, `.ie`). A pay-level domain (PLD) is a domain that must be paid for at a TLD registrar. PLDs may be one level below the corresponding TLD (e.g., `livejournal.com`), but there are many exceptions where they are lower in the hierarchy (e.g., `cam.ac.uk`).

Bharat and Henzinger [BH98] suggest using a host rather than a web page as the unit of voting power in their HITS-based [Kle99] ranking approach. PLD-level granularity is preferable to domain or host-level granularity because some sites like Livejournal assign subdomains to each user, which would result in large tightly-knit communities if domains were used as naming authorities. The use of PLDs reduces the size of the input graph to the PageRank calculation.

Previous work has performed PageRank on levels other than the page level, for example at the more coarse granularity of directories, hosts and domains [JXS+04], and at a finer granularity such as logical blocks of text [CHWM04] within a page.

5.4.4 Internal vs External Links

Regardless of the level of granularity for naming authorities, there exist internal references occurring within a single naming authority, and external references occurring between different naming authorities. An internal reference is when a data source refers to an identifier which it has the authority to mint. An external reference is when a data source refers to an identifier which is under the authority of a different data source. Since our authority algorithm considers a reference as analogous to a vote for a source, it may be desirable to treat external references (from another source) differently to internal links (from the same source).

Similarly, in the HTML world, the HITS algorithm [Kle99] considers only links which exist between different domains (which they call transverse links) and not links which occur within a single domain (which they call intrinsic links). The reason why only cross-domain links are taken into account in the HITS algorithm is that many intra-domain links are navigational and do not give much information about the authority of the target page. In variations of our algorithm, we use either all links or take only external links into account.

5.5 Calculating Ranks

Having constructed the naming authority matrix, we now can compute scores for data sources, and in another step propagate data source scores to individual RDF terms. The algorithm can be implemented using a single scan over the dataset, which derives both the naming authority matrix and a data structure that records the use of terms in data sources. As such, the method can be applied to streaming data. Subsequent calculations can then be carried out on the intermediate data structures without the use of the original data.

5.5.1 Calculating Source Ranks

For computing ranking scores, we calculate PageRank over the naming authority graph. Essentially, we calculate the dominant eigenvector of the naming authority graph using the Power iteration while taking into account a damping factor.

In the input graph there may be sources which have no outlinks, referred to by the inventors of PageRank as dangling nodes [PBMW98]. The rank of these dangling nodes is split and distributed evenly across all remaining nodes. Conversely, there might be sources which have no inlinks, where nobody uses identifiers the source speaks authoritatively for; these sources only receive the damping factor plus the rank of the dangling nodes.

5.5.2 Calculating Identifier Ranks

Based on the rank values for the data sources, we now calculate the ranks for individual identifiers. The rank value of the individual identifier $u \in U$ depends on the rank values of the data sources $s \in S$ where the identifier occurs. The identifier rank of a global identifier $u \in U$ is defined as the sum of the ranks of the sources $s \in S$ in which u occurs.

$$identifierrank(u) = \sum_{s \in \{s | u \in s; s \in S\}} sourcerank(s) \tag{5.1}$$

The identifier rank of local identifiers $b \in B$ and $l \in L$ are defined as the source rank of the source in which b or l occurs.

Symbol	Small Dataset	Large Dataset
S	14k	6.5m
U	500k	74.3m
tuple	2.5m	1.1bn
Redirects	77k	4.5m

Table 5.1: Dataset properties.

5.6 Experiments and Evaluation

In the following, we evaluate our method on two real-world web datasets. We first introduce the datasets (one small and one large), then present runtime performance results, and finally present and analyse results of a quality evaluation of several variants of our method.

5.6.1 Datasets

We constructed two datasets:

- a small-scale RDF dataset crawled from a seed URI[7]. All unique URIs were extracted and their content downloaded in seven crawling rounds. The uncompressed dataset occupies 622 MB of disk space.
- a large-scale RDF dataset derived from the 2008 Billion Triple Challenge datasets[8]. From these seed sets (around 450m RDF statements) we extracted all unique URIs and downloaded their contents during June and July 2008. The uncompressed dataset occupies 160 GB of disk space.

Table 5.1 lists the properties of the datasets. In our experiments we followed one redirect, however, it is possible to take longer redirect chains into account.

5.6.2 Evaluation Variants

The experiments were carried out on five different algorithms which are enumerated in in Table 5.2. The methods evaluated include four variants of our algorithm which differ according to the level of the naming authority (URI or PLD), and the references which we took into consideration for the authority calculation (all references, or external references only). We compared our method to a naive version of PageRank operating directly on the node-link graph without taking sources into account. We did not compare to ObjectRank [BHP04] because ObjectRank requires manual assignment of weights to each of the thousands of properties in the dataset, which is infeasible.

[7]`http://www.w3.org/People/Berners-Lee/card`

[8]`http://challenge.semanticweb.org/`; we used the Falcon, Swoogle, Watson, SWSE-1, SWSE-2 and DBpedia datasets

On the small dataset, we compared performance of all the methods listed in Table 5.2, and carried out a quality evaluation on the results. On the large dataset, we conducted a scale-up experiment on the EU variant to demonstrate the scalability of our algorithm. We performed all experiments on a quad-core Xeon 2.33GHz machine with 8GB of main memory and a 500GB SATA drive.

Method	Description
AU	All links contribute authority URI-level naming authorities
EU	External links exclusively contribute authority URI-level naming authorities
AP	All links contribute authority PLD-level naming authorities
EP	External links exclusively contribute authority PLD-level naming authorities
PR	PageRank over the object graph

Table 5.2: Ranking methods evaluated.

The implementation assures the uniqueness of links in the naming authority graph via sorting, and aggregates rank scores by writing the rank fractions to a file, sorting the file to group common identifiers together, and summing up the values via file scan. For the PageRank calculation, we fixed the number of iterations to ten rather than having the method converge in specified error bounds, which means we have to maintain only the current version of the rank vector rather than also maintaining the version from the previous iteration for comparison.

5.6.3 Performance Evaluation

The runtime of the five tested variants on the small dataset is plotted in Figure 5.4. Each processing step is plotted separately. The relatively large amounts of time spent on the AP and EP variants is due to the reduction of full URIs to pay-level domains. Given the coarser granularity of pay-level domains, the derived naming authority graph is quite small and thus accounts for the short running time of the PageRank calculation. In all approaches, a significant time is spent on the pre-processing of the data: processing steps which could be either carried out in-memory for small datasets or could be distributed across machines if required.

5.6.4 Scale-Up Experiment

We also conducted a scale-up experiment in which the algorithm performed ranking on a dataset with 1.1bn statements. In the scale-up experiment we used the Linux command `sort` with 6GB

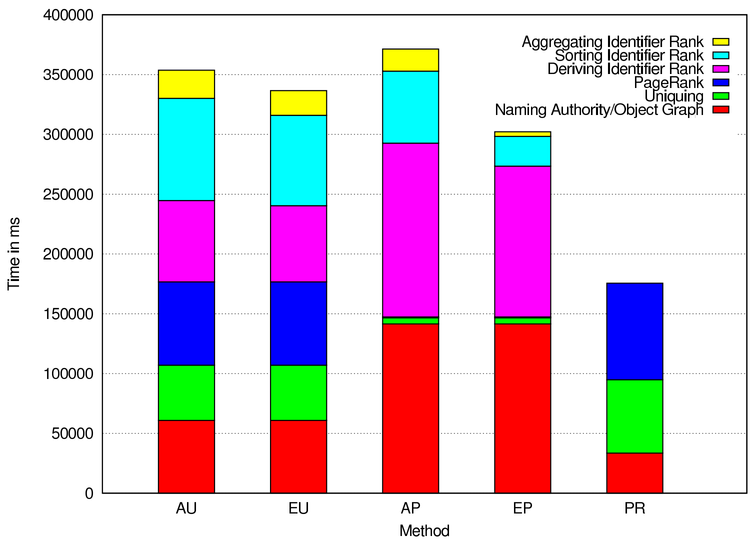


Figure 5.4: Runtime of algorithm variations.

Processing Step	Duration
Deriving Naming Authority Graph	55h36m22s
Uniquing Naming Authority Graph	40h17m52s
PageRank Calculation	12h30m24s
Deriving Identifier Ranks	39h7m13s
Sorting Identifier Ranks	14h36m27s
Aggregating Identifier Ranks	1h56m43s

Table 5.3: Runtime of processing steps for the large dataset.

of main memory for sorting and uniquing instead of our standard Java implementation. The runtime of each processing step is listed in Table 5.3.

5.6.5 Quality Evaluation

In the information retrieval community, there are clearly defined processes and well-established metrics for evaluating how well a system performs in meeting the information requirements of its users. Standard test collections consisting of documents, queries, and relevance judgements are available and are widely used. Search over Semantic Web data is a relatively new area however, and equivalent labelled collections do not yet exist. Therefore, given the lack of a labelled dataset, we use an alternative approach to quality assessment.

To compare the quality of the methods, we conducted a study in which we asked participants to manually rate results of queries for each algorithm. For every query, we presented the evaluators with five different ranked lists, each corresponding to one of the ranking methods. The result lists consisted of the top ten results returned by the respective method.

The evaluators were asked to order these lists from 1 to 5, according to which lists they deemed to represent the best results for each query. Our analysis covered the five scenarios listed in Table 5.4. For each scenario there were between 11 and 15 evaluators. Scenarios 1 and 2 were evaluated by the participants of a Semantic Web related workshop held in November 2008. During this evaluation, we presented each participant with results to a general query (S1) and with results of a query for their own name (S2), which was possible since all eleven participants have FOAF files and thus satisfactory data coverage. Scenarios 3 - 5 were evaluated by Ph.D. students and post-docs in the university. These final three scenarios were queries for people with whom the evaluators are familiar.

Scenario	Request	N
S1	Query for all persons in the dataset	11
S2	Keyword search: evaluator's own name	11
S3	Keyword search: "Tim Berners-Lee"	15
S4	Keyword search: "Dan Brickley"	15
S5	Keyword search: "John Breslin"	15

Table 5.4: Scenarios used for evaluation. N is the number of evaluators.

For each scenario, we plotted the mean rank assigned by evaluators for each method. Error bars represent one standard deviation. We determine the significance of the results using the Wilcoxon test with $p < .05$.

Figure 5.5 shows the average ranks which resulted for S1: a query for all people in the dataset. For this query, the top ten results of the methods AU and EU were identical. The differences between methods in this table are all statistically significant. The evaluators were almost unanimous in ranking this particular query, which is why for three of the points there is no standard deviation marked.

Figure 5.6 shows the average ranks which were assigned by evaluators to a query for their own name (S2). In this table, the differences between each variant of our method and the PageRank method are all statistically significant. However, the differences between the variants of our method are not statistically significant.

Figures 5.7, 5.8 and 5.9 show the average ranks which resulted for queries for three people involved in the Semantic Web community: scenarios S3, S4 and S5. The differences between each variant of our method and the PageRank method are statistically significant for all queries, with one exception (scenario S4, methods AP and OR). For S5, every evaluator rated PR as the worst method so for the corresponding point there is no standard deviation marked. The differences between the variants of our method are generally not statistically significant.

For scenarios 2 - 5, the average ranks follow similar patterns, showing that the evaluator's assessments of the methods were consistent over different queries.

We can conclude from the quality evaluation that our algorithm gives significantly better results than simply implementing PageRank on the object graph.

We cannot determine the best variant of our algorithm with statistical significance. However

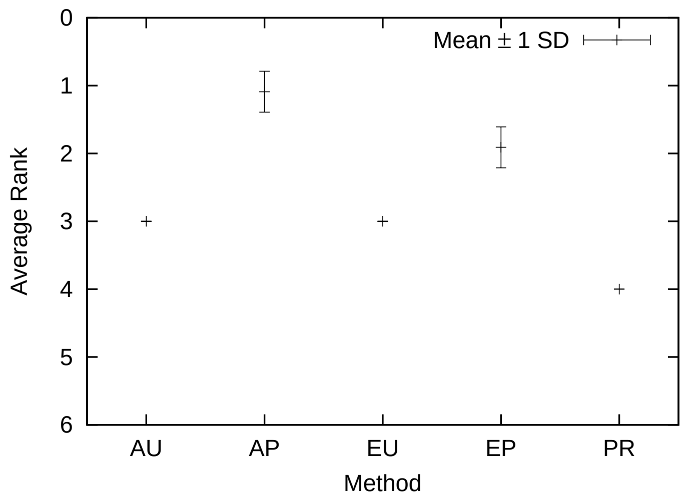


Figure 5.5: Average ranks for scenario S1: Query for all persons in the dataset.

with the exception of the query for all people in the dataset, the best method is always EU (where naming authorities are URIs, and only external links contribute to authority).

5.7 Related Work

Citation-based algorithms have been investigated in sociology in the area of social network analysis [Sco88], which discusses the mismatch between two-dimensional graph theory and multi-dimensional social networks. We have extended links-based connectivity algorithms such as PageRank [PBMW98] and HITS [Kle99] to operate on data graphs. PageRank scales very well but only operates on two-dimensional matrices: the graph derived from the hyperlink structure.

An extension of PageRank and HITS to the multi-dimensional case is TOPHITS [KBK05]: a ranking procedure rooted in multilinear algebra which encodes the hypertext link structure (including link labels) as a three-way tensor. Rather, our approach operates on a general model for semistructured data, and is centred around the notion of trustworthiness of data sources.

ObjectRank [BHP04] is an approach to rank a directed labelled graph using PageRank. The work includes a concept called authority transfer schema graphs, which defines weightings for the transfer of propagation through different types of links. ObjectRank relies on user input to weight the connections between nodes, so that the three-way representation can be collapsed into a two-way matrix on which a PageRank-style algorithm is subsequently applied. Our approach does not require any manual input, a key aspect given the scale and heterogeneity of the input. In addition, omitting the provenance of data as in ObjectRank opens up the method to abuse - anyone could maliciously link to their own identifiers from well-known, highly ranked identifiers and therefore gain reputation by association. Using our notion of naming authority, reusing popular identifiers only results in a propagation of reputation from the containing sources to

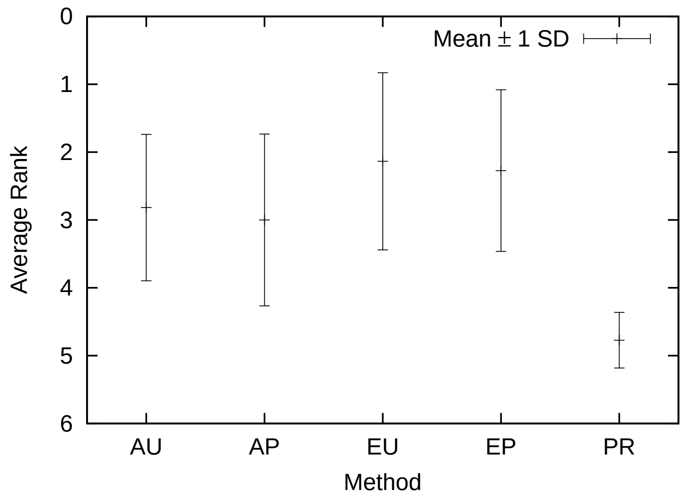


Figure 5.6: Average ranks for scenario S2: Keyword search for the evaluator's own name.

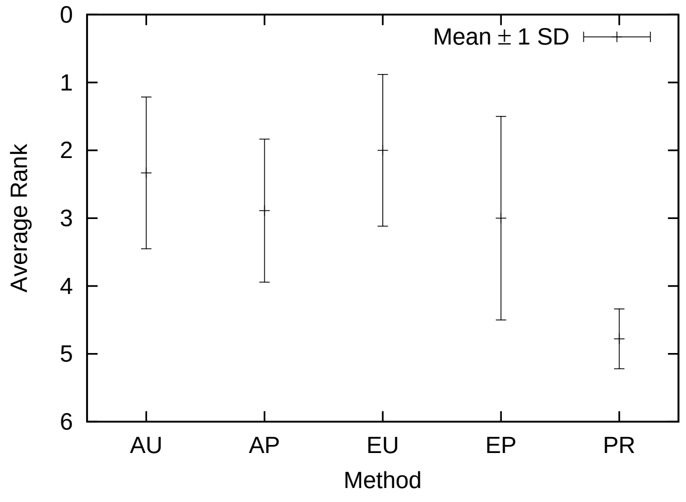


Figure 5.7: Average ranks for scenario S3: Keyword search for "Tim Berners-Lee".

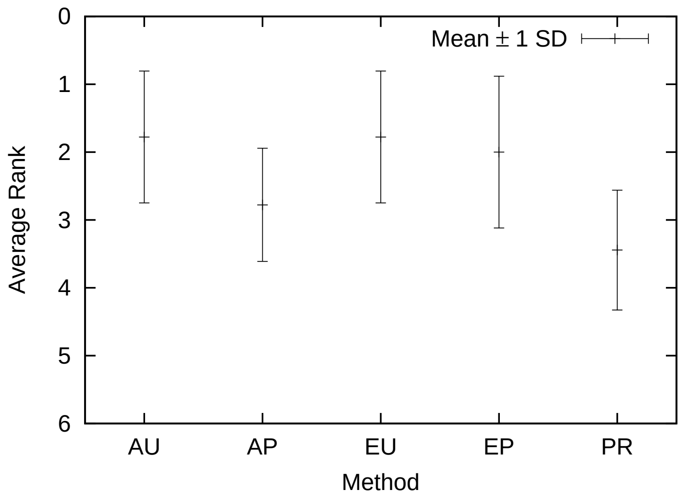


Figure 5.8: Average ranks for scenario S4: Keyword search for "Dan Brickley".

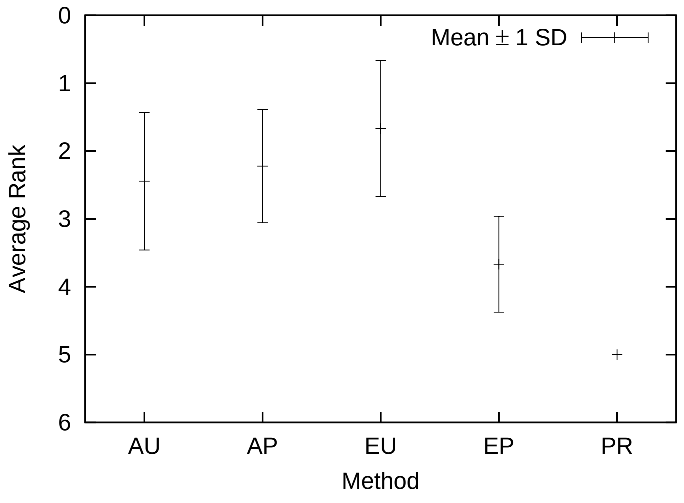


Figure 5.9: Average ranks for scenario S5: Keyword search for "John Breslin".

the popular source. As an added benefit, taking into account the naming authority results in a much smaller graph for PageRank calculation.

ReConRank [HHD06] applies a PageRank-type algorithm to a graph which unifies the documents and resources in a dataset. The method generates scores for the documents and entities in a collection, but not for the properties. ReConRank does take data provenance into account, however because it simultaneously operates on the object graph, it is still susceptible to spamming.

SemRank [AMS05] ranks relations and paths in Semantic Web data using information-theoretic measures. In contrast, we assign a rank value to all identifiers occurring in the data sources, based on a fixpoint calculation on the naming authority graph.

Swoogle [DPF+05] ranks documents using the OntoRank method, a variation on PageRank which iteratively calculates ranks for documents based on references to terms (classes and properties) defined in other documents. We extend the method described in [DPF+05] in several important ways: i) we generalise the notion of term-use to naming authority which establishes a connection between identifier and source; ii) we include the PLD abstraction layer which has been found to be advantageous for ranking in the web environment; and iii) we extend our ranking scheme to not only cover vocabulary terms but instance identifiers as well.

The notion of naming authority is related to that of authoritative sources as considered by the SAOR reasoning system [HHP09]. SAOR uses authoritative sources to determine whether a source has authority to extend a class or property, while we use naming authority to rank sources and identifiers.

AKTiveRank [ABS06] is a system for ranking ontologies based on how well they cover specified search terms. AKTiveRank combines the results of multiple analytic methods to rank each ontology. Individual instances and vocabulary terms are not ranked. Ontocopi [ADOS03] provides a way of locating instances in a knowledge base which are most closely related to a target instance. The Ontocopi tool uses a spreading activation algorithm and allows both manual and automatic tuning. However, the source of data is not taken into consideration. Similarly, the SemSearch system [TBF+08] ranks entities according to how well they match the user query, but does not consider the source of data.

This chapter has been published as [HKD09].

5.8 Conclusion

Ranking provides an important mechanism to prioritise data elements and assuage the noise inherent in datasets which have been aggregated from disparate sources or have been created in a decentralised way. We have demonstrated a set of scalable algorithms for ranking over a general model of structured data collected from an open, distributed environment, based on the notion of naming authority. We adapted the general model to the case of RDF, taking the intricacies of RDF data from the web into account.

In comparison to using plain PageRank on a node-link graph representation of RDF, our methods exhibit similar runtime properties while improving on the quality of the calculated rankings. Contrary to methods which require manual input of a domain expert to specify schema weights, our method derives rankings for all identifiers in the dataset automatically.

With regards to the integration pipeline presented in Chapter 3, ranking could be carried out on the unconsolidated data or performed on the consolidated data. We chose to perform ranking on the unconsolidated data as object consolidation collates data from multiple sources and would therefore impact the results of the ranking procedure which uses the data source to calculate identifier ranks. In addition, we would like to use calculated ranks in the object consolidation step to determine the most appropriate identifier in an equality chain which requires to compute rankings before object consolidation. However, the impact of object consolidation for ranking is subject to further studies.

We expect our ranking method having applications in search, query processing, reasoning, and user interfaces over integrated datasets from a large number of sources, an environment where assessing trustworthiness of sources and prioritising data items without a priori schema knowledge is vital.

Chapter 6

Object Consolidation

6.1 Introduction

An important aspect of Semantic Web technologies is the issue of identity and uniquely identifying resources, which is essential for integrating data across sources. Currently, there is poor agreement on the use of common URIs for the same instances across sources. In fact, since the assignment of URIs to instances is optional within the Resource Description Framework (RDF), many entities are described anonymously without use of a URI. Where agreement on URIs for resources cannot be reached, multiple URIs may exist for the one resource. As a result, a naïvely integrated dataset might miss associations between resources.

The problem of object consolidation has received significant attention in the database community under the names of record linkage, instance fusion, and duplicate identification. Due to the lack of formal specification for determining equivalences, these approaches are mostly concerned with probabilistic methods. In more formal approaches, properties unique to an entity can be used to determine its identity; examples are personal public services numbers or email addresses which can be used to identify people. On the Semantic Web, these "inverse functional properties" are specified in Web Ontology Language (OWL) descriptions. For example, from the FOAF specification one can determine that `foaf:mbox` (a person's email address) is a property unique to a person; hence, two instances sharing the same value for `foaf:mbox` properties must be the same real-world entity.

There is much more agreement on values that are established in real-world areas or widely used applications such as email addresses or instant messaging usernames than on URIs as the identifying factor of entities. Thus, equivalence of identifiers based on inverse functional properties has to be taken into account when merging RDF graphs, since infrequent reuse of instance identifiers across sources leads to problematic data integration: the total knowledge contribution for an entity may be fragmented over multiple instances. One desired outcome of the Semantic Web effort is a massive data graph spanning the web, and agreement on identifiers is crucial to achieving a well-connected graph where associations between entities are recorded correctly. Fusing identifiers is especially important for entity-centric applications relying on RDF data, such as search and query engines, interactive browsing tools and data mining systems.

There are a number of challenges related to performing object consolidation on Semantic Web data:

- The web is massive in size, therefore methods operating on considerable fractions of the web require scalable algorithms. In particular, we require an algorithm that scales object consolidation to the web.
- Data gathered from the web has been created by a large number of people and exhibits a lot of variance in terms of quality and completeness. Therefore, we require our algorithm to be robust in the face of potentially problematic data.
- We wish to reduce the number of URIs denoting the same entity to a single canonical URI. We require a method to make such a decision through various analyses on the use of the candidate identifiers in the data.

In this chapter, we describe an object consolidation algorithm which analyses inverse functional properties and is used to identify and merge equivalent instances in an RDF dataset with more than 400m statements obtained from over 3m sources. Our approach consists of the following steps: firstly, we determine a list of inverse functional properties from ontologies that are used to describe instances in the dataset. Secondly, we determine the instances that are equal based on the values of the inverse functional properties. Thirdly, we store the transitive closure of the equivalences in a data structure for efficient lookup. Finally, we determine a canonical URI for each equivalence chain, scan the dataset and rewrite identifiers.

The contributions of this chapter are as follows:

- We describe a method to perform object consolidation on a large web dataset, and determine multiple alternatives to selecting a canonical URI from a list of choices.
- We provide an empirical analysis on the RDF data currently available online, and measure the effects of object consolidation on the dataset.

The structure of the chapter is as such: Section 6.2 introduces a running example used in the chapter. Section 6.3 provides necessary definitions. Section 6.4 describes our object consolidation algorithm for identifying and merging equivalent instances. Section 6.5 provides a before/after characterisation of our large Semantic Web dataset accompanied by a runtime performance evaluation of the algorithm. Section 6.6 discusses related work, and Section 6.7 concludes.

`http://www.harth.org/~andreas/foaf.rdf#ah` = `genidA` = `genidE`
`http://sw.deri.org/~aidanh/...#Aidan_Hogan` = `http://sw.deri.org/~aharth/foaf.rdf#ahog` = `genidD`
`genidB` = `genidC`

Table 6.1: Identifiers determined as being equal based on matching `owl:IFP` values.

6.2 Running Example

To motivate and illustrate the problem, we introduce in the following some example data that will be referred to throughout the chapter. Note that for brevity we use local prefixes to abbreviate schema URIs for RDF, OWL, Friend of a Friend, and Dublin Core vocabularies; i.e., please interpret `foaf:Person` as the URL `http://xmlns.com/foaf/0.1/Person`. In addition, we introduce a new namespace `pub`[1]. Also, please note that we may abbreviate the concept `owl:InverseFunctionalProperty` to `owl:IFP`.

The example graphs in Figure 6.1 show partial RDF descriptions for this chapter and its two authors. The three data graphs are obtained from three distinct sources, namely `http://sw.deri.org/~aidanh/foaf/foaf.rdf`, `http://sw.deri.org/~aharth/foaf.rdf` and also `http://example.org/index.rdf`. Instances with different identifiers but referencing the same real-world entity are denoted in dotted, dashed, and bold line patterns; instances representing the same entity are denoted using the same pattern.

In addition to the instance data graph, we assume that we have knowledge about which properties are declared as inverse functional in their respective ontology specification document. For the example, our supposition is that we know of the following inverse functional properties: `pub:key` relates a document to a unique key; `foaf:mbox`, `foaf:homepage` and `foaf:icqChatID` relate a person to their personal email address, ICQ username, and homepage; and finally, the property `pub:primaryAuthorOf` relates a primary author to the respective paper. Please observe that Figure 6.1 adds highlighting (line style and asterisk *) for illustration purposes only; information about equivalences and schema cannot be found in the original instance data.

Given the instance information together with information obtained from the ontology specification, we can deduce a number of identifier equalities. Table 6.1 shows the inferred equalities.

Ideally, we would like to fuse the identifiers of the equal instances to arrive at a graph where associations between real world entities are expressed correctly. Figure 6.2 shows the output of applying our object consolidation algorithm to the data in Figure 6.1.

The rest of the chapter is concerned with describing an algorithm which starts with a large-scale instance graph of data as exemplified in Figure 6.1, deduces equalities and manages them efficiently, replaces equivalent identifiers with a canonical one and arrives at a consolidated data

[1]`http://sw.deri.org/2006/11/research/publications.rdfs\#`

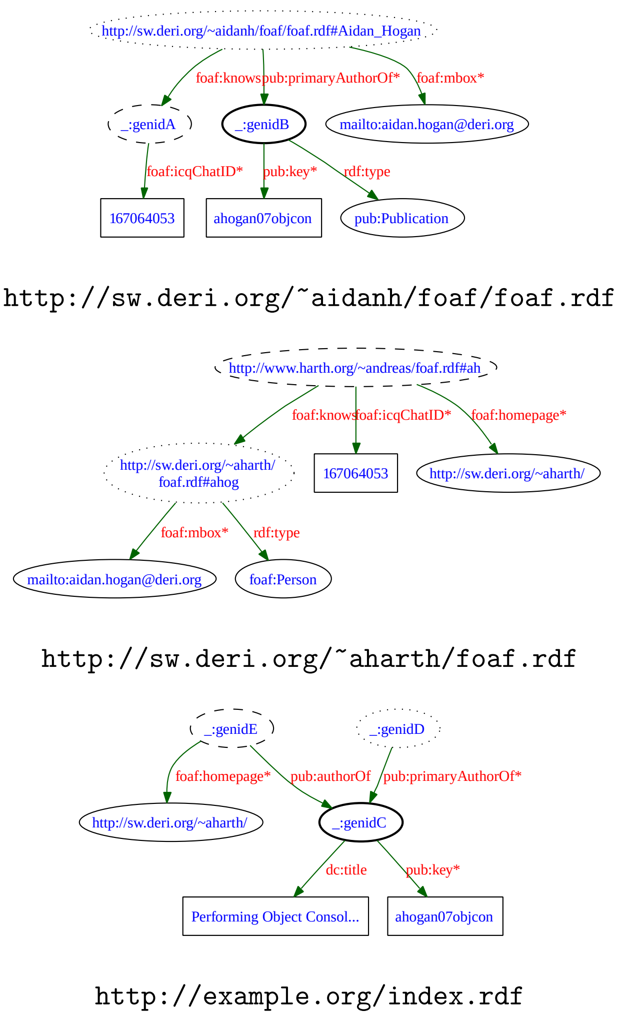

http://sw.deri.org/~aidanh/foaf/foaf.rdf#Aidan_Hogan
foaf:knows
pub:primaryAuthorOf*
foaf:mbox*
_:genidA
_:genidB
mailto:aidan.hogan@deri.org
foaf:icqChatID*
pub:key*
rdf:type
167064053
ahogan07objcon
pub:Publication
http://sw.deri.org/˜aidanh/foaf/foaf.rdf
http://www.harth.org/~andreas/foaf.rdf#ah
foaf:knows
foaf:icqChatID*
foaf:homepage*
http://sw.deri.org/~aharth/foaf.rdf#ahog
167064053
http://sw.deri.org/~aharth/
foaf:mbox*
rdf:type
mailto:aidan.hogan@deri.org
foaf:Person
http://sw.deri.org/˜aharth/foaf.rdf
_:genidE
_:genidD
foaf:homepage*
pub:authorOf
pub:primaryAuthorOf*
http://sw.deri.org/~aharth/
_:genidC
dc:title
pub:key*
Performing Object Consol...
ahogan07objcon
http://example.org/index.rdf

Figure 6.1: Three sample data graphs from three different sources. Inverse functional properties are depicted with an asterisk (*). Identifiers denoting the same entity are denoted by a common line style.

graph as illustrated in Figure 6.2.

6.3 Prerequisites

In the following section, we briefly describe the data indexing we employ but firstly, we introduce the data format used for representing the data graph.

We assume the reader is familiar with the basic notions of RDF [MM04]. We use an extension of the RDF N-Triples format for the description of our data. Namely, we extend the classical triple model (subject, predicate, object) with context. We use context to encode the URL of the data source from hence a triple originated. Whereas individual statements in RDF N-Triples

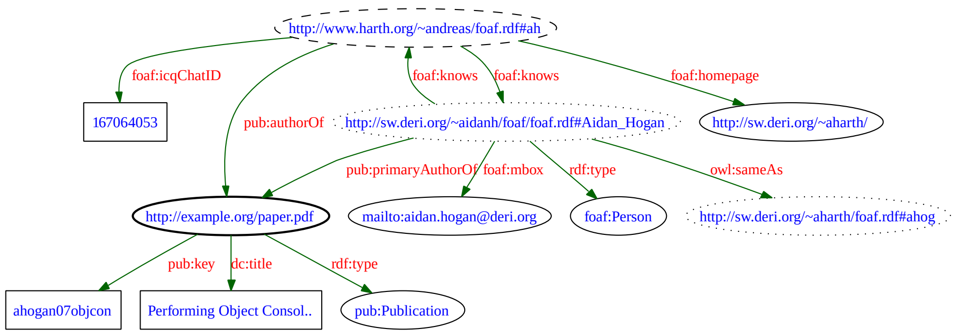

http://www.harth.org/~andreas/foaf.rdf#ah
foaf:icqChatID
167064053
pub:authorOf
foaf:knows
foaf:knows
foaf:homepage
http://sw.deri.org/~aidanh/foaf/foaf.rdf#Aidan_Hogan
http://sw.deri.org/~aharth/
pub:primaryAuthorOf
foaf:mbox
rdf:type
owl:sameAs
http://example.org/paper.pdf
mailto:aidan.hogan@deri.org
foaf:Person
http://sw.deri.org/~aharth/foaf.rdf#ahog
pub:key
dc:title
rdf:type
ahogan07objcon
Performing Object Consol..
pub:Publication

Figure 6.2: Output of the object consolidation algorithm applied to the three graphs from Figure 6.1.

are called triples, we call triples with context, "quadruples" or "quads". We call the data format used "N-Quads"[2]. Context is used as a means of tracking the provenance of data, which is an important aspect in evaluating object consolidation and how data is integrated from different sources. Context can also be analysed in the decision of which canonical URI to use, as discussed in Section 6.4.4.

For the purposes of indexing, we store quadruples or quads in sorted, blocked, compressed files on-disk. Lookups can be carried out as index scans or via binary search. For the purpose of our object consolidation algorithm, we re-configure quads from their natural ordering of subject, predicate, object, context (SPOC) to predicate, object, context, subject (POCS). We create an on-disk POCS index over these re-ordered quads and sort them respective to the POCS order. The POCS index created for the purpose of this work is distributed according to a hash function over the predicate of each quad. Further details of the distributed index are outside the scope of this chapter; the index can be abstracted as an on-disk data structure containing a list of sorted quads in POCS ordering.

6.4 Algorithm

The following section discusses the operation of the object consolidation algorithm used to merge equivalent instances. The algorithm is run pre-query-time and the results are materialised in the index. Therefore, we avoid possible issues of the object consolidation algorithm affecting query response times. Also, we can guarantee complete consolidation through multiple iterations which could be too expensive for a query-time algorithm.

We describe a once-off object consolidation algorithm which operates on the data present in an index.

Specifically, in this section we describe:

[2]`http://sw.deri.org/2008/07/n-quads/`

- our method for achieving a list of inverse functional properties;
- our algorithm for finding equivalences;
- our data structure for storing equivalences in memory;
- the process of picking the identifiers to consolidate to;
- our method of rewriting the index (materialising equivalences);
- the possible iterative nature of the algorithm.

6.4.1 Obtaining Ontologies

Object consolidation for RDF data involves analysis of inverse functional properties and their values. Properties are defined as being inverse functional in their respective ontologies. Thus, prior to object consolidation, to achieve a list of properties which are inverse functional, we must acquire the available ontologies which describe properties in the dataset.

To achieve the ontology data applicable to the instance data we have, we assume that the location of the ontology describing a property or class (if available) is consistent with the namespace URI of that property or class. To obtain the list of namespaces, we perform an index scan and for every property (resource in the predicate position) or class (resource in the object position of a triple/quad with predicate `rdf:type`) encountered we trim after the last hash or slash to achieve that concept's namespace URI. For instance, we expect the ontology information for predicate `http://xmlns.com/foaf/0.1/mbox` at the URI `http://xmlns.com/foaf/0.1/`. During the index scan we preserve a unique list of namespaces URIs, and upon completion of the scan, we attempt to retrieve the data from these URLs, using HTTP accept headers set to `application/rdf+xml`.

Having retrieved the ontologies applicable to the dataset, we now parse and scan these ontologies for properties which are defined with value `owl:IFP` for `rdfs:subPropertyOf` or `rdf:type`. All properties defined as such are inverse functional and the full list of these properties' URIs is written to a file.

With a list of IFPs in hand, we can now begin detecting equivalence of instances present in the dataset.

6.4.2 Detecting Equivalence

We begin with the formal definition of an instance and instance equivalence, and then discuss in detail our method of detecting equivalence between instances.

Definition 6.4.1 *(Instance) Given our dataset a set of quads $\mathcal{Q}$, an instance I with identifier $i \in (\mathcal{R} \cup \mathcal{B})$ has a set of quads $\mathcal{I} \subset \mathcal{Q}$ and optionally a set of in-linking quads $\mathcal{J} \subset \mathcal{Q}$ where for each quad $q \in \mathcal{I}$, $s(q) = i$ and for each quad $q \in \mathcal{J}$, $o(q) = i$.*

Definition 6.4.2 *(Instance Equivalence) Given an instance I_0 and an instance I_1, and a set of inverse function properties $\mathcal{IFP} \subset \mathcal{R}$, instance I_0 and I_1 are equivalent if there exists a quad $q_0 \in \mathcal{I}_0$ and a quad $q_1 \in \mathcal{I}_1$ such that $p(q_0) = p(q_1) \in \mathcal{IFP}$ and $o(q_0) = o(q_1)$. Instance equivalence is reflexive, symmetric and transitive.*

We use the term instance equivalence to refer to the case whereby multiple instances with differing identifiers describe the same resource or entity. Table 6.1 shows the equivalent instances from Figure 6.1.

To find instances matching the criteria for equivalence specified in Definition 6.4.2 we perform a sequential index scan of the POCS index. Since the POCS index is sorted respective of predicate, then object, then context, then subject; the POCS index contains groups of quads with the same predicate and object values. Thus we require minimal overhead to find quads q_0 and q_1 where $p(q_0) = p(q_1)$ and $o(q_0) = o(q_1)$. With the additional constraint of $q_0 \& q_1 \in \mathcal{IFP}$ we can say that $s(q_0)$ and $s(q_1)$ are different identifiers for equivalent instances.

Algorithm 2 summarises the process.

```
for quads $q_k$ in $POCS$ do
  if $p(q_{k-1}) = p(q_k)$ and $p(q_k) \in \mathcal{IFP}$ then
    if $o(q_{k-1}) = o(q_k)$ then
      store equivs($s(q_{k-1})$,$s(q_k)$)
    end if
  end if
end for
return $eq.list$
```

Algorithm 2: find equivs($POCS$, $\mathcal{IFP}$)

Now that we can detect equivalence, we require a data structure for the algorithm to store these equivalences efficiently.

6.4.3 Storing Equivalence Data

We design and implement an in-memory data structure to maintain listings of equivalent instances. Thus we use a list of lists to store the URIs of instances which meet the constraints for equivalence; this data structure can be thought of as a list of rows of variable length with each row containing instance identifiers. The identifiers in each row are for equivalent instances. Thus we implement the reflexive, symmetric and transitive properties of instance equivalence.

We also use a hashtable to implement an inverted index which maps identifiers in the equivalent instance list to the rows in which they are found. Thus, given an instance identifier i, we can find the row i appears in and from this we can find all of the identifiers of the equivalent instances of i. An identifier may only appear in one row at a time, specifically to satisfy transitivity. Equivalent identifiers are added in pairs. If one identifier is already present

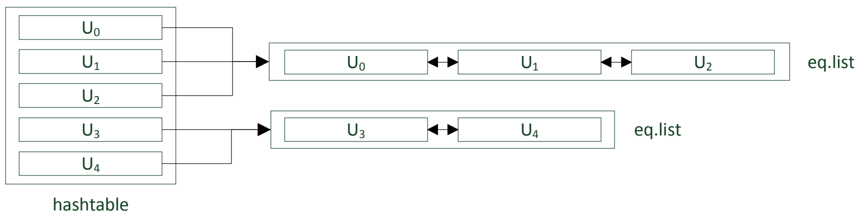


Figure 6.3: Illustration of hash table and list data structures for storing equivalences.

in a row of the list (as dictated by the inverted index) the other identifier is added to that row. If both are already present in different rows, both rows are merged. If neither is present a new row is created. We call this data structure an equivalence list or table. Figure 6.3 illustrates the hash table and list data structures. Algorithm 3 outlines the process of adding two equivalents to the equivalence list.

```
$r_A$ = find $eq.list$ row A is in
$r_B$ = find $eq.list$ row B is in
**if** !$r_A$ and !$r_B$ **then**
  new row $r_{AB}$
**else if** $r_A$ and $r_B$ **then**
  merge $r_A$ and $r_B$
**else if** $r_A$ **then**
  add B to $r_A$
**else if** $r_B$ **then**
  add A to $r_B$
**end if**
**return**
```

Algorithm 3: store equivs(A,B)

The equivalence list is filled as the sequential scan is performed on the POCS index and equivalences are detected. The worst case memory requirements for the equivalence list would occur for an index where each instance is equivalent to at least one other instance. In this scenario, for an index with N instances, the equivalence list would have to store a list of lists with N entries and a hashtable with N buckets.

6.4.4 Merging Instances

Having filled the equivalence list by scanning the POCS index, we must now begin the task of merging instances to one consolidated instance. We begin by defining the process of merging instances.

Definition 6.4.3 *(Merging Instances) To merge an instance I_0 with identifier i_0 and an instance I_1 with identifier i_1, we replace identifier i_1 with i_0. Thus, $\mathcal{I}_0 = \mathcal{I}_0 \cup \mathcal{I}_1$ and $\mathcal{J}_0 = \mathcal{J}_0 \cup \mathcal{J}_1$.*

We wish to merge instances under a consolidated identifier. We call this consolidated or canonical identifier the pivot identifier or pivot element. There must exist a pivot element for each row in the equivalence list; we see the identifiers in the row as candidates. There are no formal guidelines for such selection of pivot elements for RDF data and there are multiple alternatives to selecting a pivot identifier from an equivalence chain or row.

Decisively, we can choose URIs over blank node identifiers as URIs can be recycled for future extensions of the resource description. However, if all of the candidates are blank node IDs, we must decide whether or not to create and assign a URI for the consolidated instance. Currently, we do not create a URI and instead pick a blank node pivot ID from the candidates.

We are faced with numerous choices if there are multiple URI candidates. The choice of URI makes no difference to a machine interpretation of the consolidated instance. However, humans may desire the instance URI to be somehow informative; being perhaps a web resource with information linked to the instance or being human interpretable with some keywords contained within the string. Also, it would be courteous to choose URIs which people assign to themselves or their property. Choosing suitable URIs in this respect encourages better future URI agreement.

Thus, in choosing between URIs, there are multiple possible approaches (or combination of approaches).

1. **Random or lexical ordering:** a URI is chosen from the candidates at random or alphabetically. This is the simplest method to implement.

2. **Most agreed upon across data sources:** a URI is chosen which appears in the most distinct data sources and so is the most agreed upon.

3. **Count of occurrence:** the URI with the highest degree is chosen (i.e., the URI appearing in the largest number of statements).

4. **Links analysis:** a URI is chosen according to a links analysis technique. ReConRank [HHD06] or IDRank presented in Chapter 5 could be used as it is suited to RDF ranking and incorporates context; thus it would also take into account the agreement across data sources. However, links analysis would be quite expensive to implement for large datasets.

We currently choose method 3 to pick pivot identifiers. If the occurrence count still yields multiple candidates (i.e., multiple identifiers occur the same number of times), we pick one arbitrarily by alphabetical order. We perform the occurrence count of all the identifiers in the equivalence list together by means of another sequential scan of the index. Once we have acquired the count, we can pick the pivot elements. The restriction of selecting URIs over blank node identifiers supersedes the restriction of selecting those with more occurrences. Algorithm 4 outlines the process.

```
for row r $\in$ eq.list do
   temp = null
   for entry e $\in$ r do
      if temp is null then
         temp = e
      else if $temp \in \mathcal{R}$ & $e \in \mathcal{B}$ then
         continue
      else if $temp \in \mathcal{B}$ & $e \in \mathcal{R}$ then
         temp = e
      else if $count_e > count_{temp}$ then
         temp = e
      end if
   end for
   $pivot_r$ = temp
end for
return pivot
```

Algorithm 4: pick pivots(*count*, *eq.list*)

In the example data depicted in Figure 6.1, `http://sw.deri.org/~aidanh/foaf/foaf.rdf#Aidan_Hogan` is picked as the pivot element before `_:genidE` which is a blank node ID, and picked before `http://sw.deri.org/~aharth/foaf.rdf#ahog` which occurs in less statements (2 vs. 3).

We now can merge the instances in each row under the row's pivot element. For each quad q in the POCS index, if $s(q)$ is found to appear in row a of the equivalence list, $s(q)$ is rewritten as the pivot element of row a – unless of course, the pivot element happens to be $s(q)$ itself. The same applies for $o(q)$. We also store the equivalences we found through object consolidation as `owl:sameAs` triples, so that references to the old identifiers are maintained in the index.

One pass of the object consolidation process is now complete. The process of rewriting the index with the consolidated identifiers is summarised in Algorithm 5.

6.4.5 Multiple Iterations

It is quite possible that multiple iterations of the object consolidation process are required to be run. Specifically, if $o(q)$ gets rewritten and $p(q) \in \mathcal{IFP}$, another iteration may be required. Algorithm 5 returns true if another iteration is required.

To illustrate, in Figure 6.1, the data requires two iterations of object consolidation for completeness. In the first iteration, instances `_:genidB` and `_:genidC` are consolidated through value `ahogan07objcon` for inverse functional property `pub:key`. In the rewrite stage, `_:genidC` is replaced with `_genidB`. Now that instances `http://sw.deri.org/~aidanh/foaf/foaf.rdf#Aidan_Hogan` and `_:genidE` share value `_:genidB` for IFP `pub:primaryAuthorOf`, they are consolidated on the second pass.

```
rerun = false
for quads q $\in$ $POCS$ do
  if o(q) $\in$ $eq.list$ then
    o(q) = $pivot_{o(q)}$
    if p(q) $\in \mathcal{IFP}$ then
      rerun = true
    end if
  end if
  if s(q) $\in$ $eq.list$ then
    s(q) = $pivot_{s(q)}$
  end if
  write quad q to $newPOCS$
end for
for row r $\in$ eq. list do
  for entry e $\in$ r do
    write $pivot_r$ owl:sameAs e to $newPOCS$
    write e owl:sameAs $pivot_r$ to $newPOCS$
  end for
end for
return rerun
```

Algorithm 5: rewrite index(*POCS*, *pivot*, *eq.list*)

A summary of the entire object consolidation process is given by Algorithm 6

```
scan $POCS$ for namespaces
crawl namespace URLs
retrieve $\mathcal{IFP}$ from data
done = false
while !done do
  $eq.list$ = find equivs($POCS$,$\mathcal{IFP}$)
  for entry e in $eq.list$ do
    get count($POCS$)
  end for
  pivots = pick pivots($counts$,$eq.list$)
  done = rewrite index($POCS$,$pivot$,$eq.list$)
end while
```

Algorithm 6: objcon(*POCS*)

6.5 Evaluation and Discussion

The following section evaluates the object consolidation algorithm described in Section 6.4. We begin by discussing our process of acquiring the evaluation dataset and then discuss various properties of the dataset. We highlight various issues we uncovered whilst performing object consolidation on our dataset. Finally, we give insights into the results achieved by performing object consolidation on the evaluation dataset.

6.5.1 Data Acquisition

For the purpose of evaluation, we employ MultiCrawler [HUD06] which follows `rdfs:seeAlso` links from data source to data source. MultiCrawler is a distributed architecture for crawling web data and converting such data to RDF. For the purpose of this work, we only retrieve data which is natively RDF.

The data, collected during 2006, consists of 472,602,998 unique quads describing 72,274,051 instances (unique subjects) equating to an average of 6.539 statements per instance.

6.5.2 Results

The following section presents some insights into the operation of object consolidation algorithm over the data described in the previous subsection. Firstly we discuss various issues we uncovered whilst performing object consolidation on our dataset. We then present some statistics compiled while running the algorithm.

One major issue we discovered involved `foaf:weblog`, which is defined as being an `owl:IFP` in the FOAF ontology. Thus its semantics mandates that the property uniquely defines the `foaf:Person` instance for which it is described. However, this is not the case for sites such as LiveJournal which use the property to define communal weblogs which multiple people share. By interpreting `foaf:weblog` as an inverse functional property, we incorrectly consolidate members of the same communal weblogs together. Thus we omit `foaf:weblog` from the list of inverse functional properties we utilise.

Another obstacle we encountered involved invalid values being defined for properties relating to instant messaging usernames. For example, many users of Livejournal had entered common invalid values such as ”`ask`”, ”`none`” or simply ”`?`” for various instant messaging usernames which they either did not have, or wished not to reveal. This issue led to erroneous consolidation of 85,803 entities (0.12% of total, 2.9% of those consolidated) to one consolidated entity; the 85,803 entities were linked through a web of shared invalid values. A possible solution to this problem would involve the creation of a manual black-list of values which the algorithm ignores or possibly the provision of a more strict definition of the datatypes of values of datatype properties which can be cross-checked against values observed. Thus, literals with spaces, invalid characters or literals not meeting length requirements for various properties would be ignored during object consolidation.

Overall, object consolidation saw 2,443,939 instances merged to 401,385 pivot elements. Of the pivot identifiers, 400,892 were blank nodes IDs which were merging 2,425,175 instances (6.05:1), 493 were URIs merging 18,764 instances (38.06:1). Besides the invalid 85,803 length equivalence chain, the longest chain found was of length 32,390 and contained equivalent instances referring to a user of Vox[3] (a weblog hosting site) named ”Team Vox”, which had `foaf:knows` inlinks from all other users on the site. Figure 6.4 shows the distribution of equivalence row lengths.

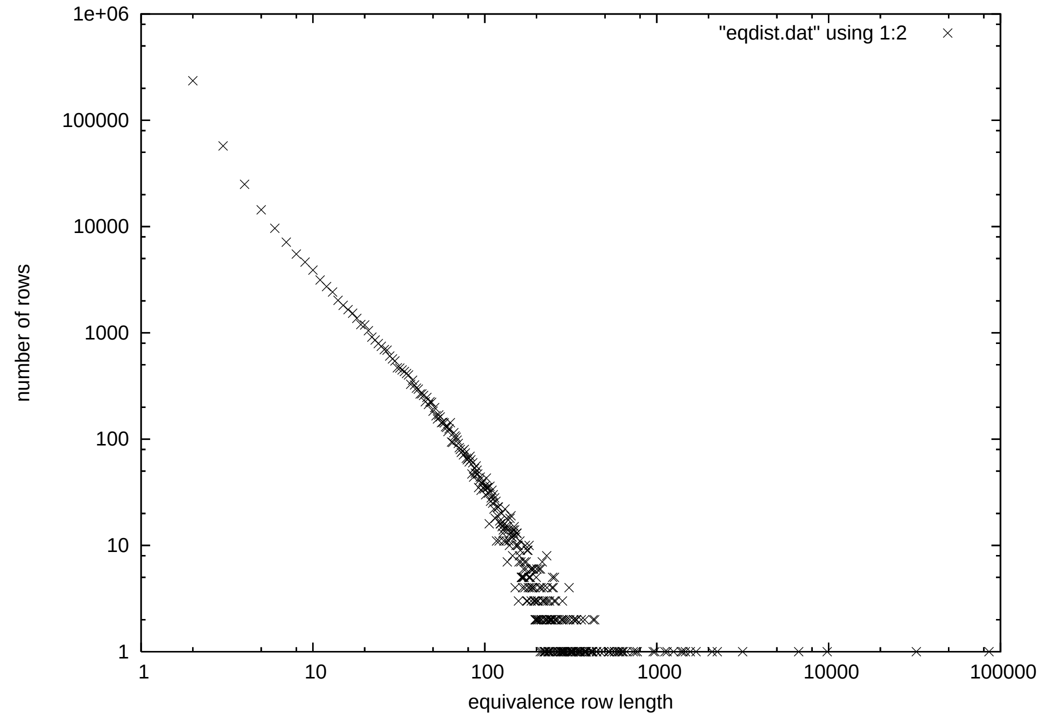


Figure 6.4: Equivalence row distribution (log/log scale)

Table 6.2 shows the list of IFPs and the amount of atomic equivalences found through them (where non-zero).

The consolidated dataset contains 70,231,472 instances, a reduction of the number present in the dataset before object consolidation (since object consolidation has merged many instances). Overall, the average degree in the datagraph improves slightly to 1.53 (+0.05). 54,552,267 of the instances in the data are of type `foaf:Person`, a decrease of 2.04M; almost all of the object consolidation was performed on foaf:Person instances.

6.5.3 Functional Properties

It is interesting to note here that `owl:FunctionalProperties` may also be used for object consolidation. An instance can only have one value for a functional property. Therefore, if an instance is seemingly attributed multiple values for a functional property, these values may

[3]`http://vox.com/`

Inverse Functional Property	Equivalences
`http://xmlns.com/foaf/0.1/mbox_sha1sum`	1927168
`http://xmlns.com/foaf/0.1/homepage`	1397528
`http://xmlns.com/foaf/0.1/aimChatID`	54868
`http://xmlns.com/foaf/0.1/msnChatID`	18154
`http://xmlns.com/foaf/0.1/jabberID`	9634
`http://xmlns.com/foaf/0.1/icqChatID`	9530
`http://xmlns.com/foaf/0.1/yahooChatID`	6392
`http://xmlns.com/foaf/0.1/mbox`	2856
`http://usefulinc.com/ns/doap#homepage`	5

Table 6.2: List of IFPs which result in successful consolidation and the number of *atomic* equivalences they find.

be considered equivalent. An example is `foaf:primaryTopic`; an instance can only have one primary topic.

We performed some evaluation on object consolidation which used an algorithm similar to that described in Section 6.4 using a SPOC ordered index and analysing functional properties; we found that very few instances were merged. We observed six equivalences found through `foaf:primaryTopic` and one through `wot:identity`[4]. Thus we only mention our functional property analysis here and omit it from the main text.

6.5.4 Schema Mapping

This chapter deals specifically with consolidation of data on the instance level. We make no special efforts to adapt the algorithm for consolidation of similar concepts for different schemas. Where inverse functional properties are shared by the same concept under different URIs, consolidation will be performed. However, inverse functional properties are not normally defined for schema concepts. Instead, equivalences are often explicitly defined between concepts through use of properties such as `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty`. Thus, schema mapping can be performed by populating the equivalence list structure through analysis of these properties, picking the pivot identifier as described in Section 6.4.3 and additionally rewriting the predicate elements in the index as well as the subject and object elements.

6.6 Related Work

Record linkage has a long history, starting from seminal work by Newcombe et al. in 1959 [NKAJ59]. Chen et al. use a graph-based data model for representing data, and use inter-object relationships to detect clusters of similar items. Michalowski et al. [MTK03] utilise

[4]`http://xmlns.com/wot/0.1/`

secondary sources to determine equivalences. A recent survey summarising many of duplicate record detection methods can be found in [EIV07].

The SemTag effort as described in [DEG$^+$03] assigns identifiers to web pages. The authors describe a simple algorithm to disambiguate entities and provide a large-scale effort at tagging and assigning topic URIs to web pages. In contrast, we utilise `owl:InverseFunctionalProperty` information obtained from ontologies to determining equivalences.

The TAP project [GM03] tries to overcome the problem of two applications using different names for the same concept by a process called "Semantic Negotiation". Applications can bootstrap from a small agreed vocabulary to a shared understanding in a larger domain. The identifiers used in TAP are centrally created, whereas in our approach, URIs denoting concepts stem from a large number of sources on the web. However, it would still be interesting to combine the Semantic Negotiation approach with our dataset to help applications and data creators decide on URIs for instances.

Bouquet et al. [BSMG06] motivate the problem of (re)using common identifiers as one of the pillars of the Semantic Web, and provide a framework and fuzzy matching algorithms to fuse identifiers. We perform object consolidation on a larger scale and avoid use of probabilistic methods.

The idea of performing object consolidation on FOAF data from the web based on values of inverse functional properties has been coined "smushing"[5]. Brickley discusses in [Bri02] implementation strategies for merging identifiers in RDF based on RDF reasoning engines. However, the technique has not yet been applied to large datasets. Our approach handles data from a large number of sources obtained from the web; thus to cater for potentially long equality chains derived from many sources, efficient data structures for storing an equality table are required. In addition, we provide detailed statistics on the current usage of data on the Semantic Web.

The Florid system [FHK$^+$97] is an F-logic inspired reasoning system that maintains an equality relation data structure to be able to deal with ground equalities. In the Hyperion project [KAM03], the issue of relating identifiers amongst peers is solved using so-called "mapping tables". Finally, Park and Durusau [PD06] use the notion of subject-centric merging of ontologies based on a Topic Map approach to denote a fine-grained approach to subject identity.

This chapter has been published as [HHD07b].

6.7 Conclusion

We have presented a method to consolidate identifiers from web data to improve accuracy of applications operating on the dataset. The method includes fetching of ontology information, incremental build-up of an inverted equality table, selecting a canonical identifier, rewriting

[5]http://lists.w3.org/Archives/Public/www-rdf-interest/2000Dec/0191.html

the dataset and maintaining the footprint of object consolidation through use of `owl:sameAs` statements added to the index. The experiments we conducted show that the algorithms scale to large datasets and thus are applicable for web data.

Chapter 7

Indexing Graph-Structured Data

7.1 Introduction

Storing and querying RDF data is one of the basic tasks within any Semantic Web application. Many applications that deal with RDF have the need to store the data persistently and perform queries on the data set. For our system in particular, optimised indexing provides the necessary foundation for returning queries in an acceptable amount of time. Index structures applied to the web data integration scenario have to be able to scale to large amounts of relatively static data. In this chapter, we investigate data structures that allow for quick lookups over graph-structured RDF data.

A number of storage systems provide assistance for storing and retrieving RDF. However, after investigating the index structures of existing systems, most of the systems use an index structure which do not support typical query scenarios for data from the web: some queries cannot be answered, or query answering performance deteriorates. For example, some RDF storage systems lack text indexing which would allow end users to pose free-text queries. In addition, the notion of context (useful for provenance tracking) is missing in some current RDF storage systems. Furthermore, a crucial feature of any large-scale system is the ability to run in a distributed setup (Chapter 9).

Existing systems targeting large-scale datasets can be broadly classified into information retrieval and relational database approaches. Information retrieval indices focus on text, and are thus not optimised for queries over graphs. Relational database systems assume a fixed schema for transactional relational data. In contrast, RDF from the web does not follow a fixed schema. Relational systems focus on safe data storage, implementing indices that adhere to the so-called ACID properties (atomicity, consistency, integrity, durability) which impede performance. In contrast, data integration systems do not act as the primary storage for transactional data but collect and integrate data from primary sources. Thus, we relax ACID properties in our scenario to simplify data structures and achieve increased read performance.

Our system requires lookups on both the textual portion of the graph (RDF literals) and the graph structure itself. Hence, we adapt information retrieval and database techniques to RDF data storage and indexing that satisfy the requirements for performance and functionality for a system dealing with data from the web. Specifically, this chapter makes the following contributions:

- We define and realise a complete index structure for RDF triples with context.
- We present a general indexing framework for RDF, instantiated by a read-optimised index structure with near-constant access times with respect to index size.
- We provide alternatives for compressing the index, and evaluate time and space behaviour of compression.
- We investigate different data placement techniques for distributing the index structure.

In the following, we first discuss atomic lookup operations, next describe our implementation of the keyword index, and then discuss and evaluate the structure index. After covering our index distribution method, we present ideas for lookups over linked data. Finally, we discuss related work and conclude the chapter.

7.2 Atomic Lookup Operations

Our proposed index structure enables two types of atomic lookups: keyword lookups and structure lookups. These atomic lookup operations return a mapping that binds constants (in $\mathcal{R} \cup \mathcal{B} \cup \mathcal{L}$) to variables.

The keyword lookup operation requires a set of keyword terms and returns a variable binding with the RDF subjects which match the specified keyword terms.

For the structure lookup, the atomic lookup construct posed to our index is a quadruple pattern.

Definition 7.2.1 *(Variable, Quadruple Pattern) Let $\mathcal{V}$ be the set of variables. A quadruple (s, p, o, c) $\in (\mathcal{R} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{R} \cup \mathcal{V}) \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \times (\mathcal{R} \cup \mathcal{B} \cup \mathcal{V})$ is called a quadruple pattern.*

The aim of the structure index is to quickly return bindings for the variables of all possible quadruple patterns.

7.3 Keyword Index

The prevalent type of web search queries are specified via keywords; keyword searches are a universal method where users do not need knowledge about the schema or structure of the knowledge base to be able to pose entry-level queries.

Typically, keyword search engines use so-called inverted indices [SM84], which map keyword terms to documents in which the keyword terms occur. Each document is tokenised into words. Each word represents a key in the index, with a sorted list of document identifiers as occurrences. In contrast, we use an inverted index which maps keyword terms in the graph to related RDF terms.

In our implementation we index keywords appearing in literals directly connected to a subject against that subject (Figure 7.1). There is also the possibility of a hop-2 text index which allows for broader matching of keywords (Figure 7.2).

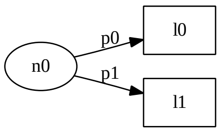


Figure 7.1: The text index contains all literals (l0 and l1) directly connected to the subject (n0).

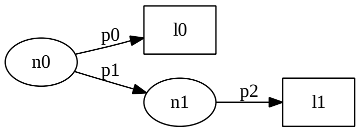


Figure 7.2: The text index contains all literals connected to a subject with distance 2; the contents of literal 0 and literal 1 are indexed for node n0.

For our experiments we use Apache Lucene[1], a state-of-the-art implementation of inverted indices. Constructing an inverted index for RDF data consists of two steps:

- Group all subjects together so that we can determine all text directly associated to an RDF subject. To group required RDF literals together the system sorts the input data on RDF subjects of datatype triples.
- Scan the grouped file, generate a "document" containing all text directly linked to an RDF subject, and add the "document" to the Lucene index.

Lookups are specified via search terms, which have to match the terms in the inverted index. The query evaluation process returns as result a list of instances (in $\mathcal{R} \cup \mathcal{B}$) that are relevant to the specified keywords. Several variations for matching can be used: phrase match (where the specified keywords have to appear exactly as specified in the text), conjunctive match (where all specified words have to appear in a matched document), and disjunctive match (where one or more specified words have to appear in documents to yield a match). To prioritise results, the standard TF*IDF formula [SM84] can be used.

[1]`http://lucene.apache.org/java/docs/fileformats.html`

7.4 Graph Structure Index

The structure index covers lookups on quad patterns. We require index support to provide acceptable performance for evaluating queries. The indices include:

- a keyword index to enable keyword lookups;
- quad indices to perform atomic lookup operations on the graph structure;
- join indices to speed up queries containing certain combinations of values, or paths in the graph.

We implement the quad index using a generic indexing framework using (key, value) pairs distributed over a set of machines. In the following, we illustrate the indexing framework using the quad index; join indices can be deployed analogously.

Relational databases are marketed as a one-size-fits-all solution to any data management problem [Sto08]. We did not use a relational database for implementing our indexing and query processing infrastructure for a number of reasons, mainly because the following required feature set would not match a single database product:

- compression needed for very large datasets;
- distribution based on flooding and hashing to potentially hundreds of machines;
- free-text keyword search;
- frequent scans over index files;
- batch-processing does not require two-phase commit and we sacrifice ACID for performance;
- price;
- overhead in using a full-fledged RDBMs to just store (key, value) pairs.

In the evaluation we compare state-of-the-art indices with our read-optimised indices for indexing quads.

7.4.1 Complete Index on Quadruples

A naïve index structure for RDF graph data with context would require four indices: on subject, predicate, object, and context. For a single quad pattern lookup containing more than one constant, such a naïve index structure needs to execute a join over up to four indices to derive the answer. Performing joins on the quad pattern level would severely hamper performance.

Index	Quad patterns
SPOC	(?,?,?,?),(s,?,?,?),(s,p,?,?),(s,p,o,?),(s,p,o,c)
POC(S)	(?,p,?,?),(?,p,o,?),(?,p,o,c)
OCS(P)	(?,?,o,?),(?,?,o,c),(s,?,o,c)
CP(SO)	(?,?,?,c),(?,p,?,c)
CSP(O)	(s,?,?,c),(s,p,?,c)
OS(PC)	(s,?,o,?)

Table 7.1: Six index groups and the corresponding 16 query patterns.

Instead, we implement a complete index on quads [HD05a] which allows for direct lookups on multiple dimensions without requiring joins. If we abstract each of the four elements of a quad pattern as being either a variable $\mathcal{V}$ or a constant $\mathcal{C} = \mathcal{R} \cup \mathcal{B} \cup \mathcal{L}$, we can determine that there are $2^4 = 16$ different quad lookup patterns for quadruples. Naïvely, we can state that 16 complete quad indices are required to service all possible quad patterns; however, assuming that prefix lookups are supported by the index, all 16 patterns can be covered by six alternately ordered indices. Prefix lookups allow for the execution of a lookup with a partial key; in our case an incomplete quad.

The six index orderings and the patterns they service are listed in Table 7.1: a question mark within a pattern denotes a variable, a letter denotes a constant. Please note that there are multiple indices which can answer some patterns: e.g., all indices can service quad patterns (?,?,?,?) and (s,p,o,c); patterns are not duplicated in the table for clarity.

We continue by examining three candidate data structures for providing complete coverage of the quad patterns. In examining possible implementations, we must also take into account the unique data distribution inherent in RDF. The most noteworthy example of skewed distribution of RDF data elements is that of `rdf:type` predicate; almost all entities described in RDF are typed. Also, specific schema properties can appear regularly in the data. Without special consideration for such data skew, performance of the index would be impacted.

7.4.2 Index Structure Candidates

Index structures that allow direct multidimensional access are kd-trees [Ben75] and similar multidimensional access structures. However, due to their complexity, these structures are not in widespread use today. Also, multidimensional access structures may require fetching more data than is really needed from disk and then in a later step have to filter out data. Since we also require lookups according to only one dimension, the increased lookup overhead precludes such approaches from applicability in our scenario.

For implementing a complete index on quadruples, we consider three index structures: B-tree, hash table, and sparse index [GMWU99].

- A **B-tree** index structure provides prefix lookups which would allow us to implement a complete index on quads with only six indices as justified in Section 7.4.1; one index can cover multiple access patterns. However, assuming a relatively large number of entries ($10^6 - 10^9$), the logarithmic search complexity of $O(log_m N))$ may require prohibitively many disk I/O operations (20 - 30 for m=2) given that we are limited as to the portion of the B-tree we can fit into main memory. Even with a higher fanout of m=100, the number of disk I/Os is in the range of 3 to 4. In addition, as B-trees are data structures which offer insert and delete operations, implementations often store additional transaction data with each record which impacts read performance.

- **Hash-tables** enable search operations in constant time; however, a hash-table implementation does not allow for prefix lookups. A complete index on quads implemented using hash tables would thus require maintaining all 16 indices. The distribution of RDF data elements is inherently skewed; elements such as `rdf:type` would result in over-sized hash buckets. If the hash value of a key collides with such an oversized bucket, a linear scan over all entries in the hash bucket is prohibitively expensive.

- A **sparse index** is an in-memory data structure that refers to an on-disk sorted and blocked data file. The sparse index holds the first entry of each block of the data file with a pointer to the on-disk location of the respective block. To perform a lookup, we perform binary search on the sparse index in memory to determine the position of the block in the data file where the entry is located, if present. With the sparse index structure, we are guaranteed to use a minimum number of on-disk block accesses, and thus achieve constant lookup times similar to hash tables. Since the sparse index allows for prefix lookups, we can use concatenated keys for implementing the complete index structure on quads.

Table 7.2 gives an overview of the the different index structures and their properties. The sparse index provides lookup times linear in the size of the result set. Please note that there are alternative index structures for implementing the index, such as tries, and optimisations to existing index structures, such as compression. For our experiments we chose the conceptually simplest index structure – the sparse index – to be able to experiment with various aspects of the index (such as block size, different compression techniques) while minimising side effects from components that are difficult to control. Optimisation techniques presented here may be nevertheless applicable to index structures conceptually similar to the sparse index such as B-trees.

7.4.3 Implementing a Complete Index on Quads

The overall index we implement comprises of six individual blocked and sorted data files containing quads in six different combinations. For the sparse indices over the data files, we only

Index model	Index creation		Lookup process (on-disk)	
	indices	complexity	complexity	values
B-tree (m=2)	6	$O(Nlog_2(N))$	$O(log_2 N)$	$log_2(N) \times 5ms$
B-tree (m=100)	6	$O(Nlog_{100}(N))$	$O(log_{100} N)$	$log_{100}(N) \times 5ms$
Hash table	16	$O(N)$	$O(1)$	5ms
Sparse index	6	$O(Nlog_2(\frac{N}{2}))$	$O(1)$	5ms

Table 7.2: Comparison of index structures.

store the first two elements of the first quad of each block to save memory at the expense of more data transfers for lookups keys with more than two dimensions.

With sparse entries of length two, when a lookup key with more than three constants is specified, the search space on-disk is only narrowed down to a certain range – even so, in the majority of cases, the correct block is actually identified. In practice, only one specific lookup would be measurably affected: a lookup of quad pattern (?,P,O,C) with predicate defined as `rdf:type` and object defined as a common resource type. In fact, lookup keys of length greater than two are quite rare in practice. With this in mind, we accept this limitation and implement this optimisation of abbreviated sparse entries.

More generally, the sparse index represents a trade-off decision: by using a smaller block size and thus more sparse index entries, we can speed up the lookup performance. By using a larger block size and thus less sparse index entries, we can store more entries in the data file relative to main memory at the expense of performance. The performance cost of larger block sizes is attributable to the increase of disk I/O for reading the larger blocks.

We can vary the main memory requirements for an index over data of a given size by choosing smaller and large block sizes at the expense of slower response times. The minimum block size we can implement is 1kB; any quads greater than the block size cannot be indexed and some quads may be above even larger block sizes.

The main memory requirements of a sparse index given the block size and raw data file size can be roughly estimated as follows:

$$s = \frac{rC}{b} \tag{7.1}$$

where s is the estimated memory requirements of the sparse index to be calculated in bytes, b is the chosen block size of the index in bytes, r is the size of the raw data file to be indexed in bytes and C is a constant we measure empirically as having the approximate value of 630 bytes. This constant is an approximation of the average in-memory size of a sparse index entry.

To save disk space for the on-disk indices, we compress the individual blocks using Huffman coding or LZW compression. Depending on the data values and the sorting order of the index, we achieve a compression rate of $\approx$ 90 %. Although compression has a marginal impact on performance, we deem that the benefits of saved disk space for large index files outweighs the slight performance dip. In addition to compression, we employ an encoding scheme where repeated constants in a block are not stored directly but via a numeric reference. We evaluate the effect of various compression and encoding schemes in Section 7.4.6.

7.4.4 Index Construction

The Indexer component handles the local creation of the keyword and sparse indices for the given data.

The Indexer

- sorts raw index data via multi-way merge-sort,
- stores the sorted data using blocks of a constant size and optionally compresses the blocks,
- creates a sparse index which is serialised and can be loaded into memory at runtime, and
- creates all necessary statement indices through the above processes.

For our specific complete quad index, we require building six distinctly ordered, sorted and compressed files from the raw data. The following outlines the process for local index creation orchestrated by the Indexer component:

1. Block (and optionally compress) the raw data into a data file ordered in subject, predicate, object, context order (SPOC).
2. Sort the SPOC data file using a multi-way merge-sort algorithm.
3. Reorder SPOC to POCS and sort the POCS data file.
4. Complete step 3 for the other four index files.

7.4.5 Evaluation Setup

We devised different variants of the basic on-disk blocked data file. In our experiments we evaluated the effect of the following variations:

- No compression: the data file is only blocked and stored to disk as-is.
- GZIP: blocks are compressed using the GZIP format[2].
- ZLIB: blocks are compressed using the ZLIB format[3].
- RLE: for repeated entries in a block an integer pointer to the original entry is stored.

[2] `http://www.ietf.org/rfc/rfc1952.txt`
[3] `http://www.ietf.org/rfc/rfc1950.txt`

We experimented with combinations of the above variants as well as with various blocks sizes (from 512 to 32768 bytes).

In addition, we considered the following commercial-grade implementations for evaluation:

- Oracle BerkeleyDB JE[4]: a B+-Tree implementation in Java which allows for easy embedding into programs. BerkeleyDB requires – on top of the space for storing a row – around 60 bytes for each row for auxiliary information, which inflates the index size considerably. In addition, BDB hits a performance wall on index sizes of around 3 GB, far too small for our anticipated application scenario.

- PostgreSQL (Version 8.3): a relational database using B-trees for indexing. In our first experiments, PostgreSQL terminated with an OutOfMemory exception when scanning the 5m triples dataset. In addition, lookups took twice as long as with MySQL. Hence, we excluded PostgreSQL in further experiments.

- MySQL (Version 5.0.51a): a relational database using B-trees for indexing. We used MySQL as comparison in our experiments (labelled "relational database" in the figures).

The benchmark was implemented in Java (using Sun's JVM 1.6) and used JDBC to interface with the relational databases. We conducted the experiments on machines with an AMD Opteron 2.2GHz CPU and 4 GB of main memory running Debian Linux.

7.4.6 Experimental Results

As dataset we constructed graphs using the random graph model proposed by Erdos-Renyi [ER59]. Lookup keys were generated randomly.

We first benchmarked the effect of varying block sizes (block sizes determined pre-compression) on the performance of the index (writing, scanning, sorting, and lookups) on a dataset with 5m triples. Figures 7.3, 7.4, 7.5, 7.6 show the performance for creating, scanning, sorting, and performing lookups on 5m synthetically generated statements on a range of block sizes using several compression and encoding schemes.

Writing was generally fastest without compression, as data has just to be stored on disk without any further processing. Scanning, however, was fastest with RLE compression, as less disk I/O has to be performed due to smaller index sizes. For larger block sizes, the other compression schemes became competitive compared to RLE, as better compression further reduces the index size and thus required disk I/O. Sort performance is basically a combination of scan and write performance, as the in-memory sort phase is the same independent of the index scheme used. For lookups, the scheme with RLE compression performed best; for larger block sizes the scheme storing data uncompressed and those with the best compression ratio became competitive. In general, compression strongly impacted the performance for all tasks.

[4]`http://www.oracle.com/database/berkeley-db/je/index.html`

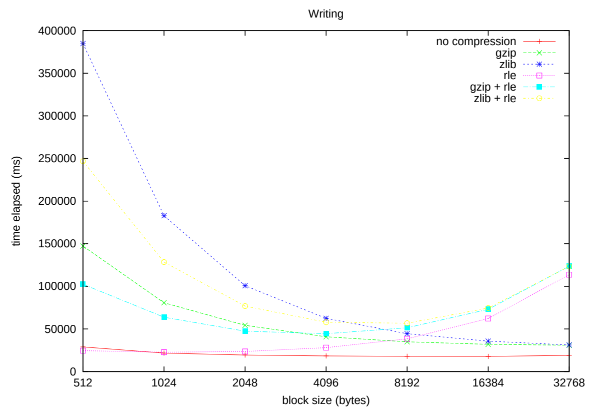


Figure 7.3: Write performance for 5m statements, on indices with varying block sizes.

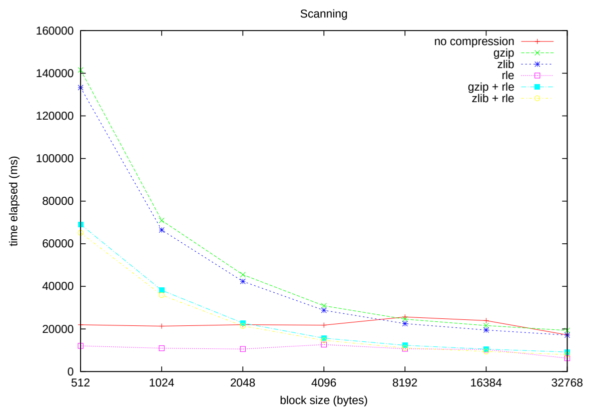


Figure 7.4: Scan performance for 5m statements, on indices with varying block sizes.

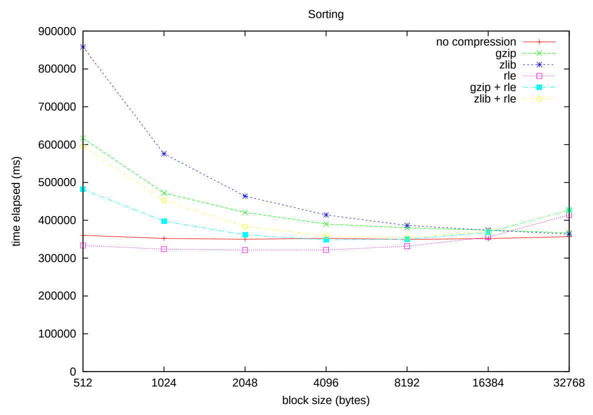


Figure 7.5: Sort performance for 5m statements, on indices with varying block sizes.

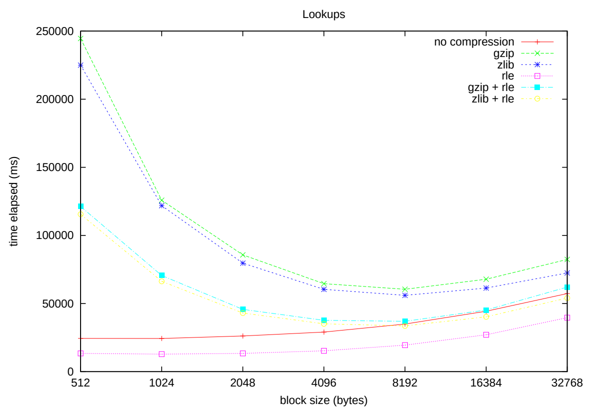


Figure 7.6: Lookup performance for 5m statements, on indices with varying block sizes.

From these experiments, we have determined 8k as a competitive block size, and conducted scale-up experiments on 8k blocks with 5m to 125m statements.

Next, we benchmarked the scale-up performance on indices with 8k block sizes. In the scale-up experiment, we compared our index structure against an open source database, MySQL (labelled "relational database" in the figures) with datasets of increasing numbers of statements, ranging from 5m triples to 125m triples. Figures 7.7 shows the performance for index creation. Please note that the index creation for MySQL includes the time for sorting (as when B-trees are constructed sorting is done implicitly). Index creation for our index structure scales linearly with the size of the dataset. Figure 7.8 shows the scan performance. The uncompressed variant and the variants with high compression ratios perform best. There is little difference in the sort performance, shown in Figure 7.9, as most of the time is spend on the in-memory sorting operation rather than on disk I/O.

The lookup performance is shown in Figure 7.10. The MySQL indexing scheme is slower than the uncompressed index variant at 80m triples, and thus we excluded tests on larger datasets. We can observe a decrease of performance for the schemes that do not use RLE encoding at around 100m statements. At that point, the index files become too large and do not fit into the buffer cache of the operating system, and lookups have to go directly to disk. The index files using RLE are smaller and thus the lookups can be serviced from the buffer cache.

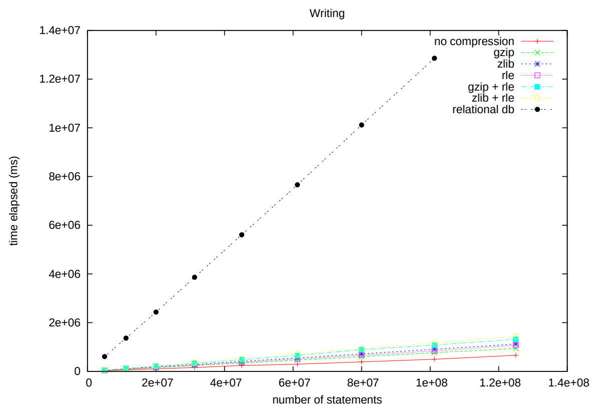


Figure 7.7: Write performance for 5m to 125m statements for various configurations of the index.

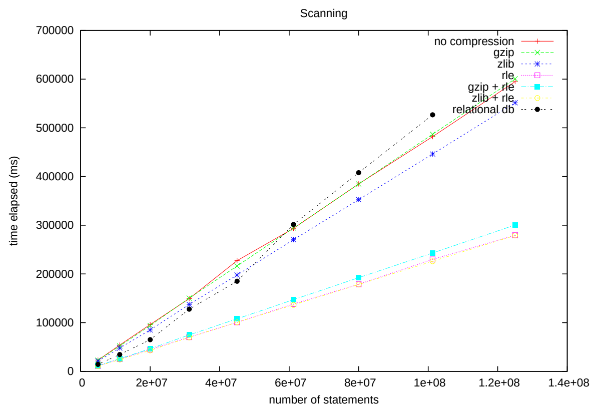


Figure 7.8: Scan performance for 5m to 125m statements for various configurations of the index.

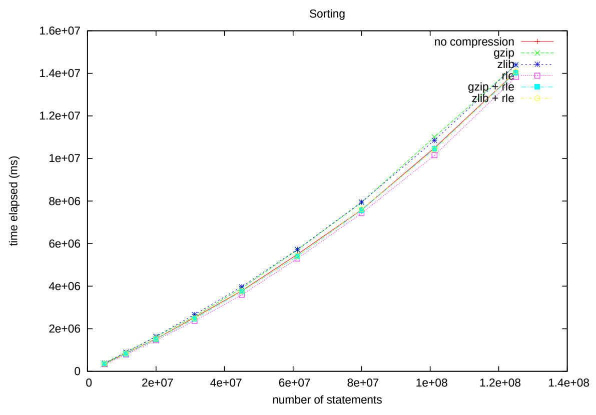


Figure 7.9: Sort performance for 5m to 125m statements for various configurations of the index.

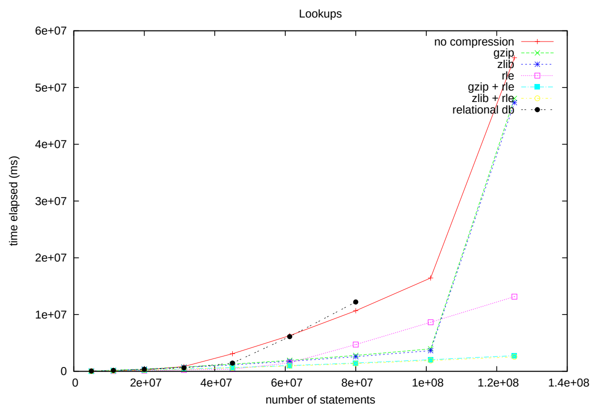


Figure 7.10: Lookup performance for 5m to 125m statements for various configurations of the index.

7.5 Related Work

Systems such as Jena2 [WSKR03], Sesame [BKvH02], rdfDB [Guh], Redland [Bec02], Kowari [WGA05], FORTH RDF Suite [ACK$^+$01], 3Store [HG03], RDF3X [NW09], Hexastore [WKB08] and others provide a storage infrastructure for RDF data.

We employ variations of well-understood techniques from the fields of information retrieval, databases, and distributed systems. Inverted indices are discussed in Salton and McGill [SM84]. Our sparse index implementation for quads and supporting indices can be seen as a BTree index [BM72] with height 2, where the first level is entirely kept in memory. We optionally use compression, whose importance is well motivated by [WMB99]. In contrast to traditional database-style indexing – which uses key/value pairs, meaning that parts of the data are not available in an index – we employ a key-only index structure with prefix lookup capability. In effect, since we only store the first two elements of an index in-memory, we employ a structure where we have a full index on the first two elements which allows direct lookups, and use binary search on the sorted last two elements of a quad key.

Indexing RDF in relational databases has been studied in [WSKR03, MSP$^+$04, CDES05, BKvH02]; native index structures for RDF have been studied in [Bec02, BKvH02].

The idea of using multiple sorting orders for keys to allow multidimensional lookups stems from [Lum70]. Kowari [WGA05] uses a similar complete quadruple index implemented using a hybrid of AVL trees and B-Trees.

Swoogle [DFJ$^+$04] uses information retrieval methods to provide keyword searches over RDF documents and ontology terms on a single machine. In contrast, we provide a structure index embedded into a distributed architecture. Our keyword index matches objects (URIs) containing specified keywords. For keyword queries matching subgraphs, see e.g., [TWRC09].

Some recent results have been obtained in the area of main memory databases. [CBHR06] describe a system that performs parallel query processing on a large shared memory system and describe the performance characteristics of hash joins and index nested loops join. We suggest a model with defined space-time tradeoffs in terms of main memory.

Key-value stores (or so-called NoSQL databases) modelled after BigTable [CDG$^+$06] have recently gained prominence in the research community, as useable systems such as Apache Cassandra [LM10] become available. Initial tests show that Cassandra's indexing (which is write-optimised) is an order of magnitude slower than our read-optimised index, however, in some applications where update-able indices are required a key-value store may be a viable alternative.

Parts of this chapter have been published in [HD05c] and [HUHD07].

7.6 Conclusion

Storage and retrieval of RDF data is an important issue for Semantic Web applications and we have determined a set of indices required for efficient RDF query processing. We utilise indices on string representations and on the RDF graph structure to efficiently answer queries that are characteristic for data from the web. Our approach provides indices for all access patterns on RDF with context and has been adopted by a wide range of RDF repositories.

Our implementation of a local sparse index data structure exhibits constant seek time and linear throughput, and serves as basis for a distributed indexing and query processing system.

The indexing chapter concludes the description of pre-processing steps carried out on the data. Based on the generated indices, a query processor can evaluate queries that have been generated via the user interface.

Chapter 8

Interaction Model for Visual Data Exploration

Diagrammatic reasoning is the only really fertile reasoning.

Charles Sanders Peirce

8.1 Introduction

An object-oriented model facilitates access to data; however, typically datasets contain too much information for users to readily digest and process. Thus, user interfaces are required to allow for

- selecting only parts of the data that will answer a question;
- browsing the dataset to get a sense of what the dataset is about and help users to pose the right questions;
- performing further processing, such as statistical analysis or visualisation, on selected parts of the data.

Keyword search over hypertext documents is an established method and is used by a large majority of web users [Hen07]. Search engines operate over millions of documents which have been collected automatically; however, the functionality is limited: the engine returns only links to sites but not directly the actual answer or data items sought. Typical keyword phrases used for search are insufficient to specify a complex information need since they consist mostly of only a few words [Hen07]; moreover, information expressed in documents in natural language is ambiguous and thus hard to process automatically.

The main topic of this chapter revolves around the open question of how end users should express complex queries over datasets collected from the web. A promising approach is to use a menu-based dialogue system in which users construct the query incrementally [TRTS83, YSLH03]. Offering only valid choices ensures that the user can only pose queries which can be satisfied by the available data, preventing empty result sets. Designing an interaction model and developing a useable system for interrogating collaboratively-edited datasets poses several requirements:

1. Intuitive use: both occasional users and subject-matter experts should be able to interact with the data immediately. The user interface should be consistent and allow users to quickly derive results with a few clicks.

2. Universality: previous attempts at using structured information have been restricted to manually crafted domain-specific datasets since the data on the web lacked quantity (no general-domain information available) and quality (no shared identifiers, no interlinkage). Users should be constrained as little as possible in constructing queries while keeping the system easy to use.

3. Zero configuration: data on the web comes in an abundance of formats and vocabularies. Consequently, manual intervention is a labour intensive task, rendering manual intervention infeasible.

4. Tolerance: web data is often chaotic and may contain duplicates, erroneous items, malformed syntax and incorrect formatting, so the user interface should be able to deal with irregular data gracefully. The system has to be tolerant enough to deal with diverse content, both in terms of schema and noisy data.

5. Scalability: since we target the web as a data source, the system has to scale competently, which has implications on the architecture and implementation of the system. Also, quick response times enhance the user experience.

6. User satisfaction: the system should be visually appealing and users should be able to import the results of their information seeking task into application programs to get an immediate sense of achievement.

The initial step in the proposed interaction model is typically a keyword search to locate objects, leveraging existing familiarity of users with search engines. In subsequent steps, users refine their query based on navigation primitives; as such, the interaction model leads to an explorable system that can be learned through experimentation. Since the system calculates the possible next steps based on the current state, the user can only compose queries which the system can answer.

The main contribution of this chapter is identifying, defining, and formalising a set of atomic query operations on object-oriented data models which can be combined to form complex queries, and introduce the notion of result trees which extend single-set results to multiple result sets containing result paths. The implementation of the user interaction model (called VisiNav) is covered in Chapter 10.

8.2 Preliminaries

8.2.1 Interaction Overview

To leverage existing familiarity of users with search engines, the first step in our interaction model is a keyword search to locate objects. We introduce a point-and-click extension to the traditional web search interaction model, which represents an incremental path towards higher-quality search while retaining the ease of use of traditional web search. In subsequent steps, users refine their query based on the navigation primitives; as such, the interaction model leads to an explorable system that can be learned through experimentation. Since the system computes further possible steps relative to the current result set, only valid choices are offered.

The user interaction model of web search engines is illustrated in Figure 8.1 (modelled after [SBC98]). Our search and navigation model extends the web search workflow with restriction and navigation operations on objects, illustrated in Figure 8.2. Rather than sets of documents, our model returns sets of objects, which can be connected to form a tree structure.

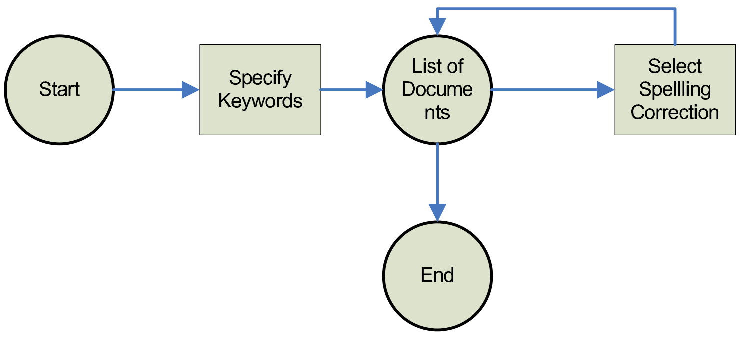


Figure 8.1: Interaction model for web search: the user specifies keywords which are matched with document contents, and may change the query based on spelling suggestions.

In summary, our search and interaction model aims at empowering casual users to freely navigate the information space that spans all available objects, and have the ability to ask queries that go beyond the precision and expressiveness of simple keyword searches.

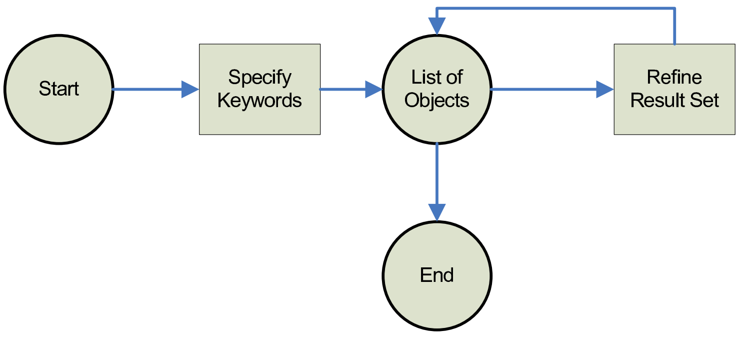


Figure 8.2: Interaction model for searching and navigating data: the user specifies keywords that are matched with objects, and may apply various operations to incrementally modify the query and result set.

8.2.2 Conceptual Model

Norman [Nor02] argues that the conceptual model of a system has to fit the user's own conceptual model about it. We therefore introduce here the assumed conceptual model; i.e., what the users have to know about the system before interacting with the system.

Our conceptual model for navigation assumes an object-oriented view, describing objects, their attributes and links to other objects. Attributes of objects are expressed using datatype properties, and links to other objects are specified using object properties[1]. Objects and properties are identified via identifiers (e.g. URIs). Attributes can have datatypes such as integer or date. Please note that there is no clear distinction between instance-level objects and schema-level ones – classes and properties can be instances themselves.

Users perceive and act on objects, in-line with early graphical user interfaces [LENW82]. In general, there is a 1:1 correspondence between the objects in the dataset and the objects displayed to the user, loosely following the "naked objects" approach [Con02]. Users are able to search and navigate the objects in the dataset; a user query yields objects as a result. Users can choose to display the result set in detail, list, or table view; optionally, a timeline or map visualisation is available if the result objects contain suitable information. In addition, users are able to export the results to application programs or services.

8.2.3 Tasks

In the following, we introduce a set of example queries that are typical queries a user can pose to the system. Given the wide availability of information about people and communities, we use the social network scenario to study user interfaces on collaboratively-edited datasets. However, the interaction model and the implemented system are domain independent and thus applicable

[1]as specified in OWL, Web Ontology Language, `http://www.w3.org/2004/OWL/`

to any object-structured dataset. We list a number of example queries – that can be answered with currently available web data – with increasing complexity in Table 8.1[2]

Query	Description
1	objects matching the keyword phrase "tim berners-lee"
2	information available about `timbl:i`
3	objects `foaf:made` by `timbl:i`
4	`sioc:Posts` `foaf:made` by `timbl:i`
5	people that `timbl:i` `foaf:knows`
6	objects `foaf:made` by people that `timbl:i` `foaf:knows`
7	locations where people that `timbl:i` `foaf:knows` are `foaf:based_near`
8	objects of `rdf:type` `foaf:Person`

Table 8.1: Example queries. Identifiers in `typewriter` are part of the query.

8.3 Query Operations

In the following we introduce atomic query operations which users can pose against the dataset.

To construct queries, users select operations that are common to existing faceted browsing and navigation systems. We list the feature matrix of these systems in Section 8.4; we argue that by using a set of features found in existing systems, we capture the community consensus of operations that are deemed necessary and useful for interacting with object-structured datasets. We describe each operation using textual descriptions and a graphical example using the notation in Figure 8.3.

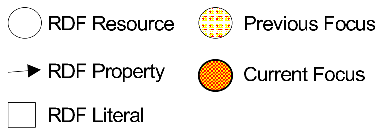


Figure 8.3: Graphical notation.

- **Keyword Search** Users may specify keywords to pinpoint objects of interest. The operation leads to an initial set of results based on a broad matching of string literals connected to objects. We perform matching on keywords without manually extending the query for synonyms or other natural language processing techniques. Rather, we leverage the noise in web data, i.e. the fact that the same resource might be annotated using different spellings or different languages. Keyword search has the interesting property that the users do not need to know the schema of the data, enabling users to pose queries without previous domain knowledge. To illustrate, Figure 8.4 shows a partial data graph with coloured focus nodes matching specified keywords.

[2]`timbl:i` expands to `http://www.w3.org/People/Berners-Lee/card\#i`

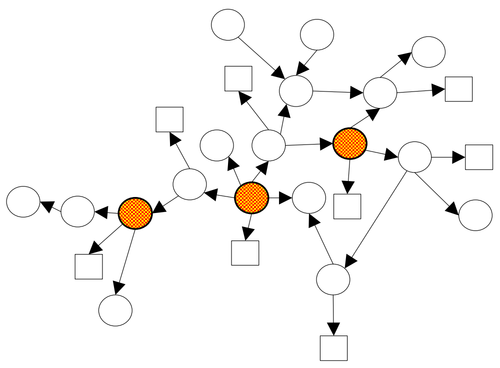


Figure 8.4: Focus nodes that match user-specified keywords.

- **Object Focus** The operation is similar to following a hypertext link in a web browser. From a set of results or a single result, the user selects an object which is used to create a new query and returns a result set containing the object. The result of an object focus operation is always a result set with a single object. Figure 8.5 illustrates the object focus operation.

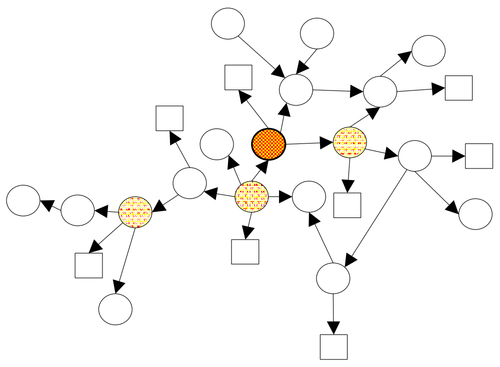


Figure 8.5: Focus node after teleporting.

- **Path Traversal** Rather than arriving at a single result by selecting a focus object, users are also able to navigate a path along an object property to establish a new set of results. Users can select an object property which allows them to perform a set-based focus change – i.e., they follow a certain path – either from a single result or a set of results. Figure 8.6 shows the focus nodes after traversing a path. Please observe that the figure depicts two result sets, one at step t and one at step $t-1$.

- **Facet Selection** Another way of restricting the result set is via selecting facets. A facet is a combination of a property and a literal value or an object (distinguishing between datatype and object properties). Facets are calculated relative to the current result set. Based on derived facets, the user can reformulate the query and obtain increasingly specific

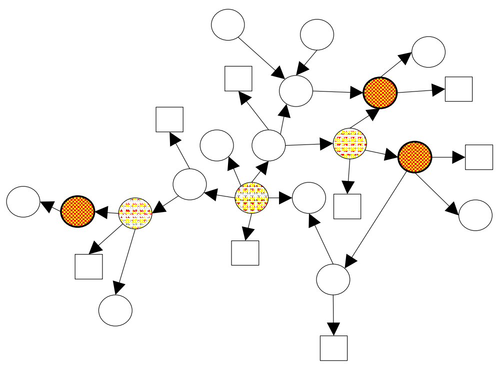


Figure 8.6: Users can move around in the information space by traversing paths from the current result set to a new result set; light orange indicates source nodes whereas dark orange indicates destination nodes.

result sets. Figure 8.7 shows two focus nodes after selecting a facet; the light focus node in the middle of the figure has been eliminated by adding a facet to the query with the effect of making the query more specific.

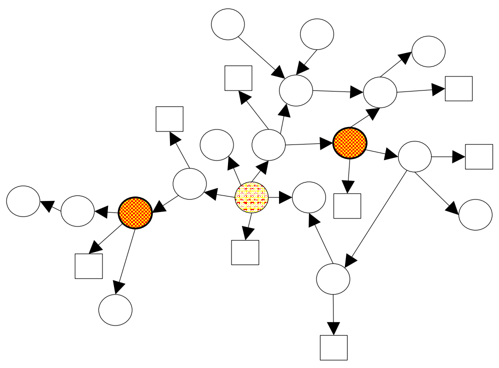


Figure 8.7: Adding facets to the query restricts the number of focus nodes in the result set.

8.4 Feature Survey

Faceted browsing [YSLH03] has become popular as an interaction form for sites which contain structured data, such as Ebay.com and Yelp.com [Hea08]. In addition, the faceted browsing model has been adopted in systems which operate over RDF datasets.

Our choice of a few core operations represents a trade-off decision between complexity of queries and ease of use. Table 8.2 compares the core features of browsing systems taking an object-oriented perspective. The interaction model chosen uses the core set of query primitives common to a range of established browsing and navigation systems for semi-structured data, providing evidence that the selection of features in our system represents a consensus in the community. This suggests that a sizeable user community is able to conceptually grasp the query operations offered by the system.

System	Keyword Search	Object Focus	Path Traversal	Facet Selection	Results
Magnet [SK05]	x	x	-	x	set
MuseumFinland [HMS+05]	x	x	-	x	set
GRQL [ACK04]	-	x	x	o	set
SWSE [HHD+07a]	x	x	-	-	set
/facet [HvOH06]	x	x	-	x	set
BrowseRDF [ODD06]	x	x	-	x	set
ESTER [BCSW07]	x	x	-	x	set
TcruziKB [MMSK08]	x	x	x	-	set
Tabulator [BLCC+06]	-	x	x	-	tree
Falcons [CGQ08]	x	x	-	o	set
Humboldt [KD08]	x	x	x	x	sets
Parallax [HK]	x	x	x	x	sets
ECSSE [CCT09]	x	x	-	-	set
VisiNav	x	x	x	x	trees

Table 8.2: Comparison of query operations of systems operating over object-structured data (x = yes, - = no, o = only `rdf:type` facets).

8.5 Related Work

NLMenu [TRTS83] is an early system advocating the use of multi-step query construction based on menus. Faceted browsing [YSLH03] – while less expressive in terms of the complexity of queries – has become popular and is used on e-commerce sites such as Ebay.com. Polaris [STH02] provides complex query and aggregation operations, but operates over relational data and thus requires a-priori knowledge about the schema used.

The systems most closely related to ours in terms of features are GRQL [ACK04], Humboldt [KD08] and Parallax [HK]. GRQL relies on schema information rather than automatically deriving the schema from the data itself, a feature required for web data which does not necessarily adhere to the vocabulary definitions. GRQL lacks keyword search, a useful feature when operating on arbitrary data since keywords are independent of any schema. Rather than permitting arbitrary facets, GRQL allows for restrictions based on the `rdf:type` predicate. GRQL is, to our knowledge, the earliest system that supports set-based navigation. Parallax [HK] is a recent system which exhibits browsing features similar to ours. However, Parallax operates over the Freebase dataset which is manually curated; our system operates over RDF data collected from the web. Parallax lacks ranking, a crucial feature when operating on web data. Our system prioritises facets, navigation axes and results based on global ranks. Although Parallax uses multiple result sets, the connections between the result sets are not propagated to the level of the user interface; our system maintains result paths in the results trees. Finally, we provide a set of export plug-ins which allows to directly load result sets into application programs and online services for display or further processing.

Regarding methodology, our system can be described in terms of the Semantic Hypermedia Design Method [dMS04]. We describe our abstract interface – the information exchange between users and system – in terms of user operations, formalised in EBNF, and result trees.

Tools and methods developed in the field of semiotics [Gog93] can help to formalise user interaction models and signs, which supports consistency and thus understandability in the user interface. Work on display of information [Tuf01] is related to the visual appearance of the user interface and influences the structure of the user interaction; similarly, design plays a role as well [Nor02].

8.6 Conclusion

In this chapter, we have identified several desiderata for user interaction over integrated structured datasets and we have introduced a set of atomic search, browsing, and navigation operations which can be combined to form complex queries.

Chapter 9

Query Language and Query Processing

9.1 Introduction

The query processor has to provide the user interface component with the data necessary to render results for user queries. Requirements for the query processing component are:

1. Expressiveness: the query language should balance the expressiveness with the ease of use; in particular, the query language should support primitives that allow for users constructing queries in a graphical user interface.

2. Performance: wait time diminishes the user's satisfaction with a software system. The system thus has to return results in a time-span that does not impede the users' attention.

3. Prioritised results: given the potentially huge query results, the data returned to the user has to be prioritised. The system thus has to return high quality, relevant results.

Regarding 1), we have shown in Chapter 8 how the individual query operations can be combined via end-user interactions to lead to more complex queries.

From 2) follows that the system has to be distributed and multi-threaded in order to deliver acceptable performance (ideally results should be returned as fast as possible, but no longer than 10 seconds to keep the user's attention focused on the dialogue [Nie94, Mil68]). To be able to deliver acceptable performance, pre-processing, parallel execution, and caching are required. Since the user builds queries iteratively via the user interface – expanding the query by adding new restrictions – results from earlier queries should be reused to avoid re-computing previously computed parts of the query. The query processor thus employs a number of optimisations to be able to satisfy the performance requirements which include

- reduce/minimise network communication;

- parallelise query execution; and
- employ caching to avoid repeated lookups.

From 3) follows that the system has to include ranking and top-k processing in order to swiftly deliver high-quality results. To be able to rank data, the system needs to have a ranking function and possibly pre-computed ranks; the query processing component has to take these ranking measures into account when evaluating a query.

In this chapter we formally define the query language $\mathcal{KOPF}$ which is amenable for graphical query construction. We show the semantics and hence the expressive power of the query language by translation to relational algebra operations. We provide algorithms for evaluating $\mathcal{KOPF}$ queries over distributed data sources, and show how ranking can be incorporated into the query evaluation process. Given that the query operations supported by the user interface follow a predefined pattern, we are able to devise a specialised and optimised query processor.

9.2 Query Processing Overview

The index structures introduced in the previous chapter allow for quickly performing atomic lookups (keywords and quad patterns). End-user queries for interactive data exploration in our system are more complex and are composed of four query primitives which amount to conjunctions of basic lookup operations, i.e., joins. In other words, the query processor has to compile down the declarative query to a low-level execution plan consisting of index lookups and join processing operations. The high-level process of query evaluation is as follows:

- Parse the query into an internal parse tree. In the case considered here, the query may consist of four basic end user operations: keyword search, object focus, path traversal, and facet selection.
- Translate the internal parse tree into an operator tree that constitutes the logical query plan containing information on the join conditions and the ordering of operations to be executed. The logical query representation is expressed in relational algebra operations.
- Translate the logical query plan into a tree which represents the executable query plan. For distributed query evaluation, care has to be taken to send the lookup operations to the appropriate index machine.
- Evaluate the executable query plan via multi-threaded lookups to the data sources, resulting in a set of topical subgraphs enriched with auxiliary information encoding connections between focus nodes.

The query processor starts with a declarative query as input, decomposes the query into atomic operations, and then decides on a suitable plan for execution, taking into account estimates of the size of intermediate results which affects the choice of the join algorithm employed.

Please note that we do not change join order since queries are constructed iteratively in the user interface, and previous results are reused in subsequent query evaluations.

We assume that the data sources (or index managers) contain the ranks which are necessary for deciding on the top-k items to return. Queries are evaluated against a set of index managers residing on multiple machines as illustrated in Figure 9.1.

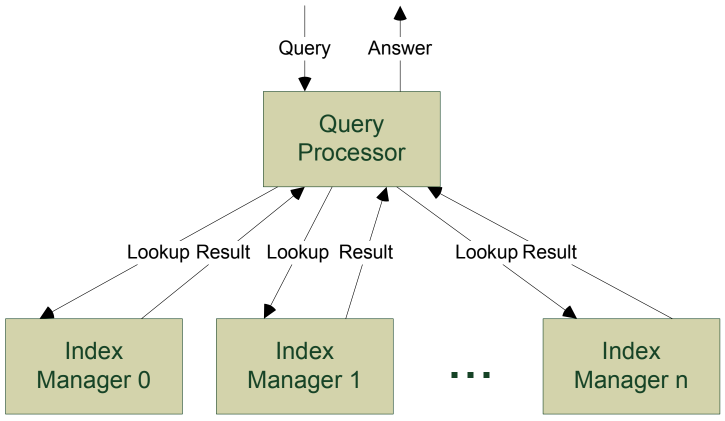


Figure 9.1: Schematic view of query processor and index managers. Data transport is via HTTP.

Queries posed via the user interface consist of two parts: i) search, navigation, and restriction operations to derive the objects that satisfy the criteria specified by the user and ii) requests for the topical subgraphs of the current result set to be able to render results. Thus, the query processor evaluates the query in two phases:

- derive the result sets with identifiers for objects in the result tree, and
- retrieve the topical subgraphs for the identifiers in the current result set.

9.3 Generalised SPC Algebra

Before we discuss the query language we introduce the SPC algebra comprising the primitive operators selection, projection, and cross-product. The generalised SPC algebra contains additional operators which can be simulated using the primitive ones [AHV95]. The SPC algebra performs over unnamed relations, i.e., the rows in a relation are identified via integers. We use the relational algebra to describe the semantics of our query language.

The SPC algebra consists of the following primitive operators (I and J are relations):

- Selection: the operator takes as input a relation and a selection condition and returns as output a relation which contains only the tuples satisfying the selection condition. The operator is written as $\sigma_\phi I$.

- Projection: the operator takes as input a relation and a deletes or permutates rows of the relation according to a specified sequence of integers (which represent the rows of a relation). The operator is written as $\pi_{i_0 \ldots i_n} I$.
- Cross-product (or Cartesian product): the operator takes as input two relations and combines the two relations by calculating the cross product. The operator is written as $I \times J$.

The extended SPC algebra adds the following operations:

- Conjunctive selection: instead of single selection conditions, the conjunctive selection allows for the specification of multiple, conjunctive selection conditions, written as $\sigma_{\phi_1 \wedge \phi_2} I$.
- Equi-Join: the equi-join, denoted as $I \bowtie_\phi J$, is shorthand for $\sigma_\phi(I \times J)$.

9.4 Query Language

We now introduce the syntax of the query language – called $\mathcal{KOPF}$ – and describe the semantics by mapping the query operations into relational algebra. We also devise an algorithm for evaluating $\mathcal{KOPF}$ queries.

9.4.1 Overview

We assume a collection of objects U identified via URIs or blank nodes. Objects are described using datatype properties P_D with literal attributes L and object properties P_O denoting links to other objects U. Properties can be objects as well: $P_D \in U$ and $P_O \in U$. We assume one relation $Quad(s, p, o, c) = (U \times P_D \cup P_O \times L \cup U \times U)$, modelling the source of subject/predicate/object triples with a fourth field. We also assume a relation $Text(t, s) = (T \times U)$ which models an inverted index over tokens T derived from the literals L using a suitable string tokenisation method. Please note that we use integers rather than names to denote a row in a relation, as the SPC algebra represents the unnamed perspective on relations.

A query $\mathcal{Q}$ consists of i) a set of search and navigation operations and ii) request-specific data such as preferred language. The search and navigation operations map to atomic lookup operations, which are:

- Keyword search ($\mathcal{K}$): return objects matching specified keywords.
- Object focus ($\mathcal{O}$): return all information pertaining to a specified object.
- Path traversal ($\mathcal{P}$): given a set of objects, traverse a specified predicate leading to a new set of objects.

- Facet selection ($\mathcal{F}$): given a set of objects, restrict the current result set according to specified facets (predicate/object pairs).

Here we define $\mathcal{KOPF}$, a query language that encompasses the above operations. In addition to the basic navigation primitives, we require an operation which, given a set of object identifiers, return the subgraphs describing these objects (described in Section 9.5).

In our query processing system, a query may consist either of i) one atomic lookup or of ii) multiple lookup patterns where each pattern satisfies the join condition with at least one other pattern. Using a single result set is sufficient for keyword searches, object focus (specifying an initial result set), and facet selection (reducing the size of the result set). The path traversal operation is different: traversing a path restricts the old result set (to the objects with the specified property) and generates a new result set (with the objects that are connected via the specified property). Iterative application of the restriction and navigation operations leads to sets of focus nodes $Result_{0...n}$, where n is the number of path traversal operations.

9.4.2 Example

Consider, for example, the query "objects `foaf:made` by people that `timbl:i foaf:knows`" (Query 6 from Chapter 8), which yields three result sets: $Result_0, Result_1, Result_2$. We assume that the query was constructed in the following way: the user specifies the object URI of Tim Berners-Lee $Result_0 =$ `timbl:i` via the object focus operation, from there performs path traversal along the `foaf:knows` property $Result_1 =$ people that Tim knows, and from there performs another path traversal along the `foaf:made` property yielding $Result_2 =$ things made by people Tim knows. An example result tree is shown in Figure 9.2.

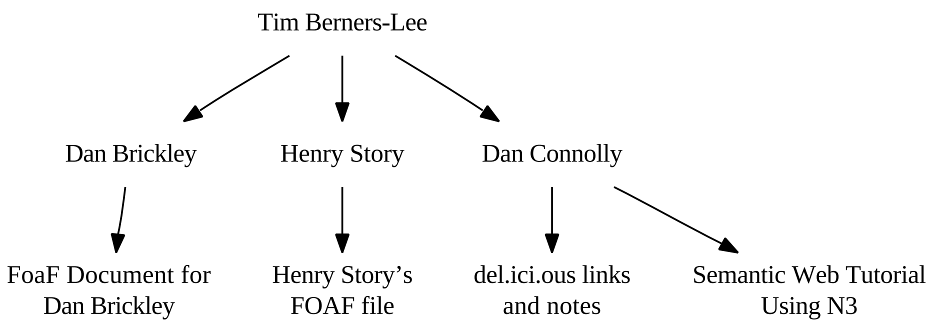

Tim Berners-Lee
Dan Brickley
Henry Story
Dan Connolly
FoaF Document for Dan Brickley
Henry Story's FOAF file
del.ici.ous links and notes
Semantic Web Tutorial Using N3

Figure 9.2: Partial result tree for query "objects `foaf:made` by people that `timbl:i foaf:knows`" (Query 6). Labels displayed instead of URIs for clarity.

9.4.3 Syntax

The Extended Backus-Naur Form grammar in Figure 9.3 describes how the individual operations can be combined – via interaction with the user interface – to form complex queries. Terminals in the grammar are parameters that the users specify, one at a time, during a search session. Thus, a query consists of steps $t_0 \ldots t_n$. Keyword search and object focus operations start a new query. In addition, users can remove parts of the query; removing a path traversal deletes all steps specified after the path traversal, while removal of other operations yields a query without the selected operation.

```
<query>          ::= <init> { <refine> | <new query> } ;
<init>           ::= keyword | object focus ;
<refine>         ::= <facet> | path traversal ;
<new query>      ::= keyword search | object focus | <facet> | path traversal ;
<facet>          ::= datatype facet | object facet;
keyword search   ::= specify keywords ;
focus object     ::= specify focus object U ;
path traversal   ::= specify P_O for path traversal ;
datatype facet   ::= specify P_D, L facet ;
object facet     ::= specify P_O, U facet ;
```

Figure 9.3: EBNF grammar describing queries. Terminals describe end user actions.

9.4.4 Semantics

We define the semantics of the query language by mapping to relational algebra [AHV95]. The individual atomic operations are defined in terms of relational algebra as follows:

- Keyword search ($\mathcal{K}$): the kw operation returns a set of objects which satisfy the keyword selection criteria, and an empty result if there is no match. The operation is defined as:
  $kw(k_{0..m}) = \pi_0(\sigma_{0=k_0} Text) \cap \ldots \cap \pi_s(\sigma_{0=k_m} Text)$

- Object focus ($\mathcal{O}$): the of operation returns an object with the specified URI, if the object exists in the dataset, and an empty result otherwise. The operation is defined as:
  $of(s) = \pi_0(\sigma_{0=s} Quad)$

- Path traversal ($\mathcal{P}$): the pt operation returns a set of objects which satisfy the path selection criteria. The pt operation always returns a result, since the user can only choose valid paths. The operation is defined as:
  $pt(p) = \pi_{0,2}(\sigma_{1=p} Quad)$

- Facet selection ($\mathcal{F}$): the fs operation returns a set of objects which satisfy the selection criteria. The fs operation always returns a result, since the user can only choose valid facets. The operation is defined as:
  $fs(s,p) = \pi_0(\sigma_{0=s \wedge 1=p} Quad)$

The final query is composed of atomic operations combined with the join operator. For example, consider the query "things `foaf:made` by people that `timbl:i` `foaf:knows`". A user first focuses on the object Tim Berners-Lee (of(`timbl:i`)), then traverses the `foaf:knows` relation to arrive at Tim's acquaintances (pt(`foaf:knows`)), and finally traverses the `foaf:made` relation to retrieve the final result set (pt(`foaf:made`)).

9.5 Topical Subgraphs

To be able to establish the connections between objects from $Result_{0...n}$, and to be able to render properties of objects in the result sets, the system requires more information than just object identifiers in R. Getting additional information for object identifiers involves a number of challenges:

- The dataset contains an RDF graph. For pre-processing and displaying the objects we require Java objects.
- To minimise the amount of data fetched and decrease response time, the system should only fetch additional information about objects that are ultimately rendered. The result sets may contain thousands of object identifiers: however, typically only a few results are displayed at once.
- Depending on view or visualisation, only partial sections of the result object are required. For example, the list view only requires a few properties of an object.
- Since the data is very diverse, and the amount of information available for objects varies enormously, we need to pick the best data for display. For example, displaying thousands of objects in a results list will likely overwhelm the users. Similarly, when showing detailed data about an object, the system requires data items to be ordered, showing the most important values first.

The fundamental question is how much of the vicinity of the focus nodes to return. We solve the so-called "Decker-Problem"[1] – deciding where to stop describing an object interlinked to other objects in a graph – by introducing the notion of "topical subgraph", which we use to specify how much information to return for a focus node, including information relevant for further navigation. Figure 9.4 depicts a topical subgraph. A related approach is used in the HITS method [Kle99] for selecting input to a link analysis ranking algorithm.

In our approach, a topical subgraph refers to a directed labelled graph of data with context derived for a particular query. We refer to the nodes directly matching the query criteria as focus nodes. The topical subgraph contains the focus nodes and their outgoing links up to a specified

[1] `http://lists.w3.org/Archives/Public/public-sws-ig/2004Feb/0048.html`

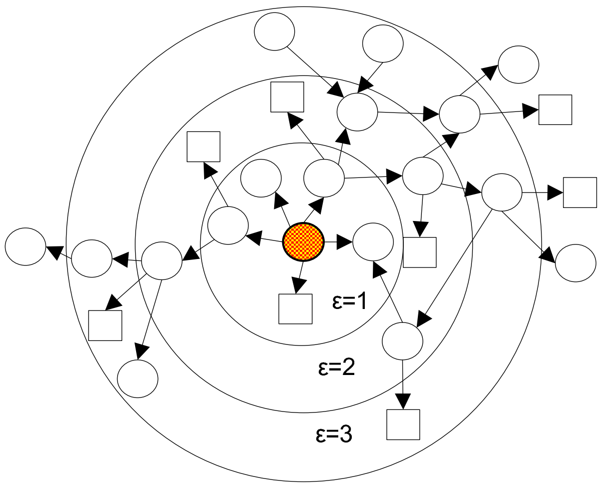


Figure 9.4: Topical subgraph for one focus node with $\epsilon = 1, 2, 3$.

distance ϵ. We define a parameter ϵ to denote how much information in the neighbourhood of the focus nodes should be returned. Some applications might require larger portions of the graph to operate. Extracting topical subgraph can be arbitrarily costly, especially considering the small-world property of networks.

The operation to retrieve topical subgraphs is defined as follows:

- Topical Subgraph: the ts operation returns a set of statements which describe the selected object identifier s with paths of maximum length ϵ starting from s. The operation is defined as:
  $ts(s, \epsilon) = \sigma_{0=s}\, Quad ⟕ \ldots ⟕_{2=0}\, Quad$

To be able to query topical subgraphs with $\epsilon >= 2$ we use a left outer-join, denotes as $I ⟕_{\phi} J$.

Please observe that our notion of topical subgraph does not contain inlinks to nodes. Given that the notion of directionality of links in semantic graphs is both somewhat arbitrary and difficult to communicate to end users, we just assume outlinks from nodes. In case browsing is required both from and to a node, we assume that a property is either specified as symmetric or has its inverse property defined. Upon reasoning, a link will be inferred in both directions.

Having computed the results tree $Result_{0\ldots n}$, a number of topical subgraphs are retrieved depending on the selected view and paging. The result tree and respective topical subgraphs are handed over to the user interface which then renders the data. For the results in the range of the active set, the query processor obtains the topical subgraphs which contain data required for display. The final result returned to the user-facing components is an RDF graph consisting of the union of the topical subgraphs in the selected range of the active set, including their position in the results tree and the ranks of all elements in the graph.

9.6 Logical Query Plan

Here we give an overview of the query translation step from the query operations to a logical query plan. A query Q, expressed over the object set U and described by the $Quad$ and $Text$ relations, returns a result in the form of a number of sets $Result_{0 \dots n}$ where n is the number of path traversal operations. The result of individual operators are combined with the join operator $\bowtie$ as described in Algorithm 7.

```
Require: Q: set of query operations
  $n = 0$
  for all operation in Q do
    T = execute operation
    if operation is path traversal then
      {deal with path traversal}
      if n == 0 then
        $Result_n = \pi_0(T)$
      else
        $Result_n = \pi_0(Result_n \bowtie_{0=0} T)$
      end if
      $Result_{n+1} = \pi_1(Result_n \bowtie_{0=0} T)$
      $n = n + 1$
    else if n == 0 then
      {new query (any operation)}
      $Result_n = T$
    else
      {additional facet selection}
      $Result_n = Result_n \bowtie_{0=0} T$
    end if
  end for
  return $Result_{0 \dots n}$ containing object identifiers
```

Algorithm 7: Algorithm for constructing a logical query plan from a set of query operations.

In addition to the atomic lookup operations the logical query plan includes the following operations:

- Join: atomic lookup operations are combined via joins. The logical query plan does not specify the actual implementation of the join operation; only the fact that a join has to be carried out is noted.
- Sort: to be able to retrieve top-k results, a sort operation prioritises the variable bindings derived from evaluating the query according to global rank values. Similar to the join operation, exactly how the sort is being performed is left open in the logical query plan representation.
- Range: the user interface typically only shows a fraction of the entire result set in a given

results page. Hence, we introduce an additional paginate operator which selects variable bindings only in the specified range (e.g., from result 11 to result 20).

Figure 9.5 shows the logical query plan for deriving the result sets $Result_{0...2}$ for the example query.

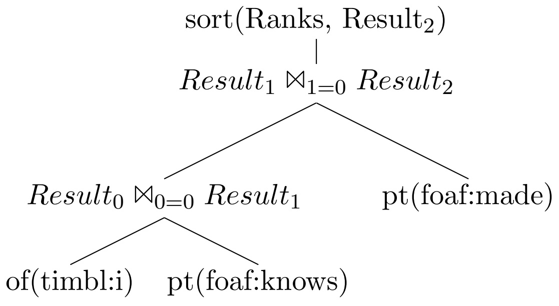

sort(Ranks, Result$_2$)
$Result_1 \bowtie_{1=0} Result_2$
$Result_0 \bowtie_{0=0} Result_1$
pt(foaf:made)
of(timbl:i)
pt(foaf:knows)

Figure 9.5: Logical query plan for retrieving result bindings for “things `foaf:made` by people that `timbl:i foaf:knows`”.

The final step is to paginate the results in $Result_2$ and retrieve the topical subgraphs. Figure 9.6 shows the logical query plan for deriving the topical subgraphs for the example query. Please observe that to keep the size of the topical subgraph manageable, the query plan can optionally contain sort and range operations to trim the size of the topical subgraph.

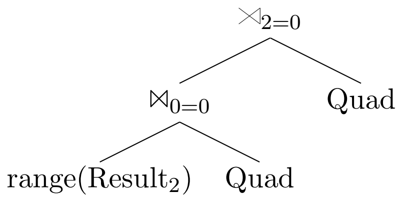


Figure 9.6: Logical query plan for retrieving topical subgraphs with $\epsilon = 2$.

9.7 Distribution and Parallelisation

The operator plan contains information necessary to obtain the topical subgraphs which constitute the result to the query. Before covering execution of the plan, however, we examine data placement strategies to deploy the index across multiple machines. Data sources are queried in parallel and joins are computed on the query processing machine.

9.7.1 Data Placement Strategies

To be able to scale-up the indexing structures and query processing, the index has to be distributed across a number of storage nodes. Conceptually, there are at least two solutions to distributing data: structured networks, where nodes are connected based on some globally known function or data structure, and unstructured networks, where connections among nodes do not follow any fixed structure and data is distributed along, for example, social or organisational ties. There are a number of possible data placement policies:

- Random placement with flooding of queries to all machines. During index construction, the statements are distributed either in round-robin fashion or randomly. During the lookup phase, the queries are sent to all nodes in parallel since we do not keep information about the distribution of statements: i.e., we do not know where a statement has been placed. The benefits are simple distribution policy and lookup implementation. In addition, with a random placement we can trivially achieve a balanced distribution of storage and query load, and data can easily and flexibly be moved if machines are added or removed. However, the approach might be inefficient since all queries have to be posed to and answered by all machines, which incurs a lot of traffic in the network and potentially many lookup misses (since queries are distributed to all machines, no matter whether the machine contains results or not) which are expensive to evaluate.

- Placement based on a hash function with lookup to machines where quads are located. A hash function that is globally known to all nodes decides where to place a statement. During query evaluation, the hash function can be used to route lookup requests to the appropriate node. Any distributed hashtable substrate can be used. Benefits are simple implementation, efficient lookups because of deterministic data distribution, and possible reuse of DHT technologies such as replication. Drawbacks are a possibly uneven distribution of data and query load ("data skew"): machines holding popular data can become a bottleneck. In addition, data ownership is lost, i.e., the distribution policy decides where data is stored.

- Range-based placement with directed lookups via a global data structure. Rather than a hash function, a globally known data structure is used to decide on data placement in the range-based partitioning approach. A range of values is assigned to each node in the network; data items in a certain range are stored on that node. During lookup, the range data structure is used to route the lookup request to the appropriate node. The benefits are even distribution and load balancing and the possibility of redistribution to avoid skew. The main drawback is complex implementation: redistribution requires a change in the global data structure.

For our implementation, we decided to use random placement of data which means flexibility in how data is being distributed and a relatively straightforward implementation. The possibility of repeated lookups to data sources which do not contribute an answer is mitigated by the use of caching.

9.7.2 Ranking

For ranked retrieval, the current result set is ordered based on a combination of query-time and global ranks. Query-time ranks include TF*IDF ranks in case of keyword queries.

The query processor has access to the ranks to sort intermediate results according to their importance. To avoid a bottleneck on one machine which has to store all the ranks, the system stores ranks directly in the index structures on the individual index nodes. The query processor receives query answers and associated ranks; based on data and ranks, the intermediate results are sorted and only top-k answers are retained.

9.8 Query Plan Execution

In the final step during query evaluation, the query processor creates an executable query plan from the logical query plan. We distinguish between the part of the plan which derives the result sets and the part of the plan which derives topical subgraphs for the requested objects in the result set.

9.8.1 Result Sets

To derive result sets, the results of atomic lookup operations have to be combined using the join operation. There are two processing methods for joins:

- with hash joins, the results of both operators are loaded into memory, and then both result sets are intersected on the join condition;
- with index nested loops joins, the results of the left operator in the query tree are combined with the lookup operation on the right part of the query tree, and these sub-queries are used to probe the index to satisfy the join condition.

The choice of the join method is the main optimisation step carried out during evaluation of user-interface queries. Using hash joins, the query evaluation process can be easily parallelised: all operators can be evaluated in parallel since none of the operators depend on input from each other, and results are joined in a subsequent step. However, retrieving all results for an operator might be too costly, especially in cases where only a few results are involved on the left operator in the query tree, and many are involved on the right operator. In the case of variable bindings smaller than a constant n, an index nested loops join offers better performance.

The threshold n is determined by the time a single probe takes on the *quad* relation. Sequential scans are fast in comparison to a large number of random lookups. In our experiments, we found that index scans are preferable to index lookups if more than 5% of the index is involved. Thus, our query processor uses index-nested loops joins in cases where the number of bindings n is less than 5% of the number of statements in the *quad* relation.

When hash joins are used, the query processor performs atomic lookup operations on all index machines in parallel, and joins the results in memory once the bindings have been retrieved. For index-nested loops joins, we utilise the iterator notion during query execution to start evaluating the right side of the execution plan as early as possible and thus increase parallelism and reduce overall execution time.

Once the bindings and ranks for the object identifiers in the result set are retrieved, sorting and pagination operations are applied which yield a set of object identifiers to display.

9.8.2 Topical Subgraphs

The second step during query evaluation retrieves the topical subgraphs which describe the object identifiers in the result set. Topical subgraphs are evaluated in parallel in a multi-threaded fashion.

The evaluation of topical subgraphs might also involve a join if $\epsilon > 1$. In that case, the query processor performs an index-nested loops join where the bindings between operators are passed via queues to gain the benefit of multi-threaded retrieval.

The query processor also performs top-k retrieval for the topical subgraph lookups to minimise the amount of data processed as early as possible. The idea is similar to pushing joins, where irrelevant results are eliminated early on in the query evaluation process and not considered in subsequent operations; cutting results early reduces the amount of unnecessary processing being performed and thus yields better response time.

The process of topical subgraph retrieval can be viewed as a breadth-first expansion with a cut-off heuristic. First, only the top-k statements are considered for the retrieval of the subgraph with $\epsilon = 1$. Similarly, for subgraphs retrieved that are $\epsilon = 2$ from the focus node, only the top-k statements are retrieved (where k_2 is typically smaller than k_1). Topical subgraph joins equate to outer joins – results from the left side of the query tree are not reduced, but only new (optional) bindings are added.

The query for topical subgraphs could include additional restrictions: for example, a parameter specifying the language used for topical subgraph retrieval, mandating the existence of certain predicates in the subgraph irrespective of rank (such as `rdfs:label` or `geo:lat/geo:lon`), or expanding `owl:sameAs` to enable equality reasoning.

9.9 Related Work

We synthesise techniques from the fields of information retrieval, databases, and distributed systems for use in a specialised query processor. The query processor for a data exploration system has, compared to general-purpose query processing, unique requirements which our approach satisfies.

A number of techniques are related to our query processor, and several related systems provide general-purpose query processing – however, none are quite equipped with the functionality required for our scenario. Semijoins, a method for performing joins in distributed databases, has been introduced by Bernstein and Goodman [BG81]. Selinger et al. [SAC$^+$79] introduced dynamic programming as a method for deriving query plans. A large body of work exists for the evaluation of top-k queries. For a survey see [IBS08]. A number of top-k algorithms do not apply to our case given different characteristics of queries and data. Given the variety of data (and thus the variety of ranks), we cannot specify a fixed threshold value which helps to determine when to stop query evaluation. Also, given the ability to retrieve paginated results pages, the ranking method has to be deterministic for pagination to function correctly. Many top-k algorithms assume a pipelined query processing model, which we didn't employ in favour of parallel execution. Investigating optimisations for top-k query processing in our scenario is thus future work.

Sesame [BKvH02] is one of the early RDF stores operating on one machine. Cai and Frank [CF04] propose a method for providing RDF storage over a distributed hash table substrate. Stuckenschmidt et al. [SVHB04] investigate theoretically the use of global indices for distributed query processing of RDF.

A hash placement method can utilise established distributed hash table substrates to add replication and fail safety, such as Chord [SMLN$^+$03] or Pastry [RD01], or distributed index structures supporting prefix lookups such as P-Grid [ACMD$^+$03].

We have explored a hash-based distribution strategy for use in a general-purpose SPARQL query processor in [HUHD07]. A treatment of RDF from a graph database perspective can be found in [AG05]. For a comprehensive survey of distributed query processing techniques, we refer the interested reader to [Kos00]. The query processing presented in this chapter is optimised for end-user queries in a data exploration engine. We have made a step towards unifying query processing with web search; adding reasoning functionality to the mix [FvH07] is the next step.

Some recent results have been obtained in the area of main memory databases. [CBHR06] describe a system that performs parallel query processing on a large shared memory system and describe the performance characteristics of hash joins and index nested loops join. [HFV96] discuss open issues in distributed query processing. Related visual query languages are covered in [ACS89, CMW87].

Parts of this chapter have been published in [HUHD07] and [Har09b].

9.10 Conclusion

We have defined the query language $\mathcal{KOPF}$ which is amenable for graphical query construction, and devised methods for distributed evaluation of such queries. We have presented the distributed architecture of a query evaluation system for use in a data exploration engine, which offers parallelisation and top-k processing.

Chapter 10

User Interface

10.1 Introduction

In this chapter we describe VisiNav, a fully implemented system[1] based on the visual query construction paradigm introduced in Chapter 8. The users of the system can construct complex queries from several atomic operations. Our system is unique in that it is the first system which offers these features in combination over datasets collected and automatically integrated from the open web.

The contributions of this chapter are as follows:

- We develop a visual vocabulary for representing queries (and parts thereof), result trees of objects, and individual objects that can be composed to render objects in a schema-agnostic manner (Section 10.3).
- We present an interaction model for searching and navigating object spaces, and define a mapping of user actions (via keyboard input, point-and-click, and drag-and-drop actions) on signs to query operations (Section 10.4).

We provide an overview of the architecture and implementation in Section 10.2 and discuss experiments and evaluation of the system on a web dataset with 18.5m statements collected from 70k sources in Section 10.5. Section 10.6 covers related work, and Section 10.7 concludes.

10.2 Architecture and Implementation

To verify our ideas, we implemented a prototype system as a Java web application. We present first design principles, next an architecture based on the Model-View-Controller (MVC) pattern, and finally describe the rendering pipeline.

[1]`http://visinav.deri.org/`

10.2.1 Design Principles

Designing the presentation of information is a mix of art and science [Jac99]. Tufte [Tuf01] postulated the following rules for good graphic design (which we strived to adhere to when designing the visual elements of the system):

- above all else show the data;
- maximise the data-ink ratio;
- erase non-data ink;
- erase redundant data ink;
- revise and edit.

We adopted an iterative approach to come up with the design, oscillating between prototyping and user testing and expert reviews as advocated by [Nie94]. Since an important aspect of usability of user interfaces is consistency and predictability [Nie94][Nor02], we formalise query operations, visual elements, and user actions, and the mapping between them, to ensure the system behaves consistently and predictably.

10.2.2 System Architecture

The VisiNav system architecture is based on the Model-View-Controller (MVC) paradigm. The Controller, implemented as Servlets, receives queries from the users and retrieves the topical subgraph from the database – we use top-k processing over our specialised index structures. The Model classes parse the statements comprising the topical subgraph for the query into Java objects. Finally, in the View, the results are rendered and returned to the web browser (Section 10.2.3). Optionally a set of files for import in external services or applications is generated.

10.2.3 Result Rendering Pipeline

Having processed and ranked the dataset, the View components prepare the display of information to the user. The system can present the results using different visualisation views, ie. detail view, list view, table view. The table view is similar to a spreadsheet program, where users are allowed to specify projections to show only selected properties of the returned objects. Depending on the types of objects returned, a map view (for geocoordinates), timeline view (for objects with associated date), or slide-show view (for images) can be selected.

There are three ways of rendering results:

- Results display in the web browser: the web application renders the view in XML; the browser then applies XSL and CSS to render the final view to the user.

- XML-based data export: the web application renders the view in XML and returns the file to the requester.
- text-based data export: the web application renders the results in plain text returns the file to the requester.

In addition, the system can generate certain files in matching data formats for subsequent processing by the user via software programs. For example, geocoordinates can be exported to KML, objects with associated dates to RSS, or objects with a postal address to an address book application.

Displaying and exporting based on the result types requires export plug-ins to process and convert the objects to the target file format. This is the inverse to data integration systems where wrappers are used to convert the data to a common data format. With Semantic Web data, the objects are already described in the common data format RDF, so export plug-ins are becoming important. The system currently allows to export objects containing `dc:date` properties in RSS and Timeline[2] formats, `geo:lat` and `geo:long` in KML[3], and result trees in JSViz[4]. We provide rendering views of these formats in Timeline, Google Maps, and JSViz widgets directly in the user interface.

10.3 Visual Display of Result and State

In the following we introduce an alphabet of atomic signs that comprise the user interface. Specifying a set of signs in a formal manner is useful for maintaining consistency in the user interface across views and visualisations.

Composing individual signs leads to more complex visualisations of sets of objects. In the following, we first show how to display individual objects and sets of objects in generic detail, list, and table views. Second, we show how to visualise temporal and spatial properties of objects. Third, we introduce a visual representation of queries. Fourth, we list and describe controls for selecting views and paging of results.

From the object model – built from the topical subgraphs expressed in RDF – we derive higher-level visualisations on characteristics common to a large fraction of objects. To decide how to display properties of an object, we use the following information:

- Protocol level: HTTP[5] `content-type` header
- RDF level: datatypes, language tag, `rdf:type`, datatype and object properties based on usage in RDF versus the (potentially absent) specification in OWL

[2]`http://simile.mit.edu/timeline/`
[3]Keyhole Markup Language, `http://code.google.com/apis/kml/documentation/`
[4]JavaScript graph visualisation, `http://www.jsviz.org/`
[5]Hypertext Transfer Protocol, `http://www.ietf.org/rfc/rfc2616.txt`

- Vocabulary level: `geo:lat`, `geo:long`

Ideally, the system uses lower-level, more general cues over higher-level ones for display. For example, using the more general `content-type` header to decide which objects to display as images, rather than choosing only instances of `rdf:type foaf:Image` which would require hard-coding schema-level elements into the system. Similarly, we prefer the datatype of a literal as `xsd:date` over depending on schema-level properties such as `dc:date`.

Each view receives as input the query together with user-specified parameters sent via HTTP headers by the browser. Header information includes IP address (which can be used to implement geo-tracking), the time and date (including timezone) of the requester, and optionally information such as preferred language specified by the browser. Views can potentially use the additional information about the user to customise the display.

10.3.1 Generic Views

We now describe how to visualise objects, while assuming only minimal schema knowledge (that `geo:lat` and `geo:long` denote spatial properties). Rather than displaying the RDF triples directly, we derive higher-level visualisations on characteristics common to a large fraction of objects. To consistently use signs across views, we introduce a set of visual representations of properties of objects in Table 10.1.

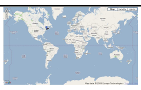
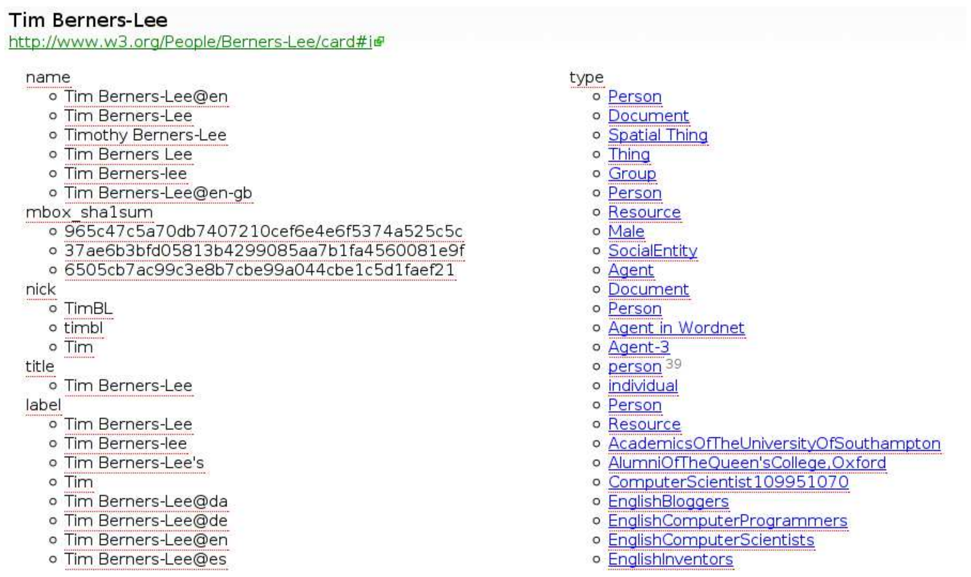
What	How	Vocab
External URI	http://www.w3.org/People/Berners-Lee/card#i	-
Object in Dataset	Tim Berners-Lee	rdfs:label, dc:title
Object of type Image		content-type image/*
Object of type Point		geo:lat/ geo:long
Date Literal	APR 5 2007	^^xsd:date
Time Literal	14:23 38	^^xsd:time
DateTime Literal	APR 5 2007 14:23 38	^^xsd:dateTime

Table 10.1: Graphical representation of individual objects and attributes that are common across views (time is shown in Coordinated Universal Time, UTC). In addition, we assume `rdf:type`, `rdfs:label`, `dc:description` for the list view.

VisiNav supports three generic views: detail, list, and table. The detail view shows all properties pertinent to a single object in a two column-format (Figure 10.1). The list view shows a few selected properties in a list of objects of typically ten objects at a time, similar to search engine results (Figure 10.2). In the list view, users can choose to download the data in RSS format. The table view shows all datatype properties of the current result in a tabular view – typically up to hundred at a time – similar to spreadsheets. In the table view, the users can download a version of the data displayed in comma separated value (CSV) format for further processing in Excel.


Tim Berners-Lee
http://www.w3.org/People/Berners-Lee/card#i

name
- Tim Berners-Lee@en
- Tim Berners-Lee
- Timothy Berners-Lee
- Tim Berners Lee
- Tim Berners-lee
- Tim Berners-Lee@en-gb

mbox_sha1sum
- 965c47c5a70db7407210cef6e4e6f5374a525c5c
- 37ae6b3bfd05813b4299085aa7b1fa4560081e9f
- 6505cb7ac99c3e8b7cbe99a044cbe1c5d1faef21

nick
- TimBL
- timbl
- Tim

title
- Tim Berners-Lee

label
- Tim Berners-Lee
- Tim Berners-lee
- Tim Berners-Lee's
- Tim
- Tim Berners-Lee@da
- Tim Berners-Lee@de
- Tim Berners-Lee@en
- Tim Berners-Lee@es

type
- Person
- Document
- Spatial Thing
- Thing
- Group
- Person
- Resource
- Male
- SocialEntity
- Agent
- Document
- Person
- Agent in Wordnet
- Agent-3
- person 39
- individual
- Person
- Resource
- AcademicsOfTheUniversityOfSouthampton
- AlumniOfTheQueen'sCollege,Oxford
- ComputerScientist109951070
- EnglishBloggers
- EnglishComputerProgrammers
- EnglishComputerScientists
- EnglishInventors

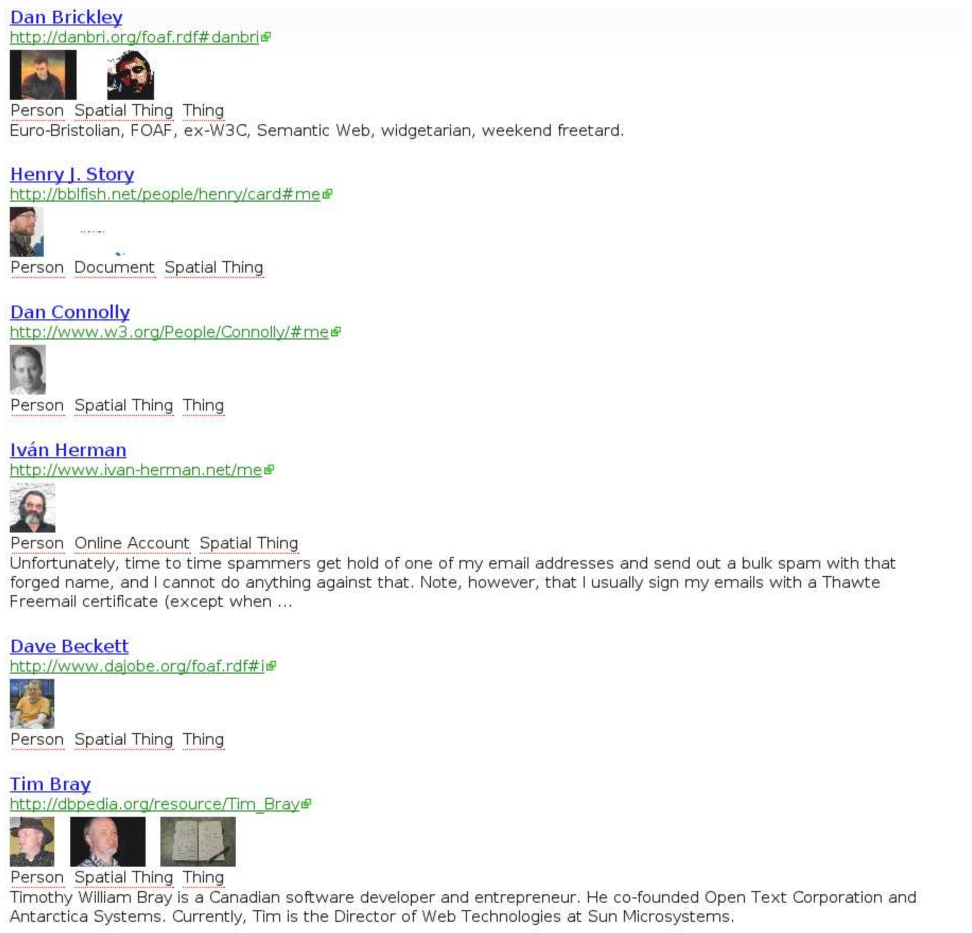
Figure 10.1: Detail view of the query "information available about `timbl:i`" (Query 2).

10.3.2 Visualisations

VisiNav currently supports visualisations for temporal and spatial aspects of objects. The system renders temporal attributes (i.e. those with datatype `xsd:date` or `xsd:dateTime`) in a timeline view (Figure 10.3) using the SIMILE Timeline widget[6]. An individual object is displayed as a dot in the timeline view, however, when clicking on the dot, the object is rendered similar to the display in the list view. In the timeline view, users can choose to download the data in iCal format and thus import the data into calendaring applications.

Spatial properties (i.e. objects with `geo:lat` and `geo:long` properties) are rendered in a map view (Figure 10.4) based on Google Maps[7]. The map view displays all objects in a result set that have both `geo:lat` and `geo:long` properties attached to them, and also uses objects linked from the current result set with these properties. Users can choose to download data in

[6] `http://www.simile-widgets.org/timeline/`
[7] `http://maps.google.com/`



Dan Brickley
http://danbri.org/foaf.rdf#danbri
Person Spatial Thing Thing
Euro-Bristolian, FOAF, ex-W3C, Semantic Web, widgetarian, weekend freetard.

Henry J. Story
http://bblfish.net/people/henry/card#me
Person Document Spatial Thing

Dan Connolly
http://www.w3.org/People/Connolly/#me
Person Spatial Thing Thing

Iván Herman
http://www.ivan-herman.net/me
Person Online Account Spatial Thing
Unfortunately, time to time spammers get hold of one of my email addresses and send out a bulk spam with that forged name, and I cannot do anything against that. Note, however, that I usually sign my emails with a Thawte Freemail certificate (except when ...

Dave Beckett
http://www.dajobe.org/foaf.rdf#i
Person Spatial Thing Thing

Tim Bray
http://dbpedia.org/resource/Tim_Bray
Person Spatial Thing Thing
Timothy William Bray is a Canadian software developer and entrepreneur. He co-founded Open Text Corporation and Antarctica Systems. Currently, Tim is the Director of Web Technologies at Sun Microsystems.

Figure 10.2: List view of the query "people that `timbl:i` knows" (Query 5).

KML format and thus import the data into mapping applications such as Google Earth.

10.3.3 Putting All Together

The system provides the users a graphical representation of the accumulated query operations (a "breadcrumb trail"). Table 10.2 lists the visual representations of individual query constituents.

A status area contains selections for specifying one of the three views or, optionally if data is available, the temporal and spatial visualisations. The status area also contains the number of total results for the query, and buttons to go to the next (→) or previous (←) page depending on the view and number of results.

Figure 10.5 shows the composition of the result display. The "Query" area displays the current query; the "New query" area contains an input field where users can enter new queries. The "Status and paging" area shows the count of results and allows users to select different views; and finally the "Content" area displays the result objects in the selected view or visualisation. In case the content area does not fit on the screen, users can scroll using the browser

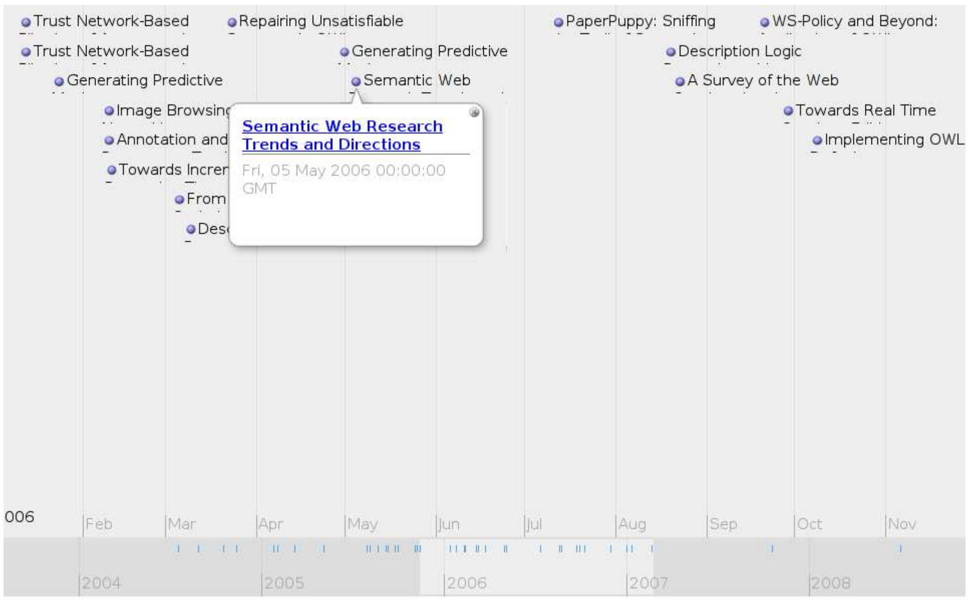


Figure 10.3: Timeline visualisation of results to the query “objects `foaf:made` by people that `timbl:i foaf:knows`” (Query 6).

Operation	Sign
Keyword Search	tim berners lee
Link Traversal	Tim Berners-Lee
Path Traversal	Knows
Facet Selection	type Person
Remove	☒

Table 10.2: Graphical representation of operations (query elements).

scrollbar.

10.4 Actions: Mapping Sign Manipulation to Operations

In this section, we first give an overview over the interaction cycle: i.e., in what sequence the user builds a query. We then introduce three user interactions (keyboard input, point-and-click, drag-and-drop) and map these actions to query operations.

10.4.1 Interaction Cycle

Users start a query building process via specifying a keyword, and then iteratively refine the query, or start a new query based on the available choices offered by the result set. Figure 10.6 illustrates the high-level interaction cycle. During each interaction step, the users select one of

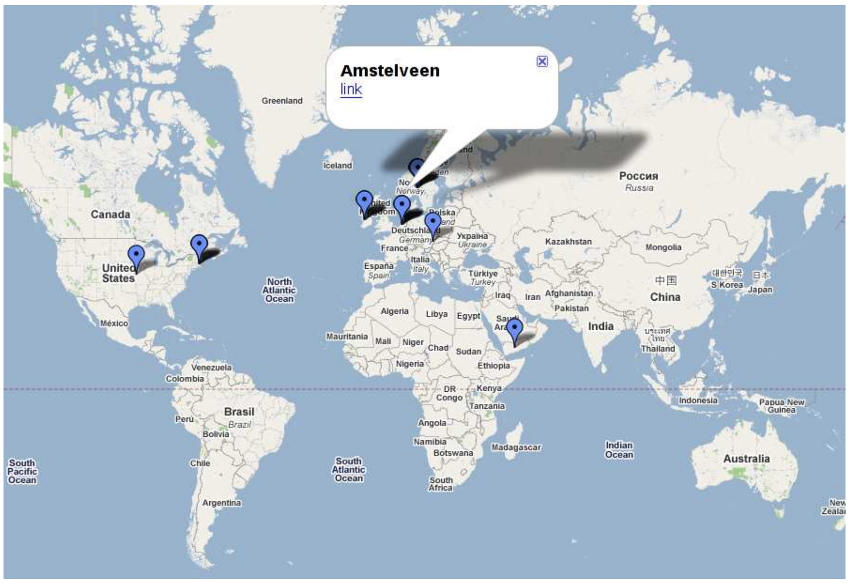


Figure 10.4: Map visualisation of results to the query "locations where people that `timbl:i` `foaf:knows` are `foaf:based_near`" (Query 7).

the operations, and the system returns a page containing a visual representation of the query, state, and results. The query results are first displayed in a familiar list view, but users can choose other views or visualisations depending on the properties of the objects in the result set.

10.4.2 Elementary Actions

Users have three fundamental actions at their disposal: keyboard input, point-and-click, drag-and-drop. Keyboard input and point-and-click are the two typical operations in web site interaction. We motivate our choice for using drag-and-drop after introducing the mapping of actions to operations in Table 10.3.

Operation	Action
specify keywords $k_{0\ldots m}$	type in keywords
specify focus object $u \in U$	point-and-click on $u \in U$ (or type in focus node URI $u \in U$)
specify $p \in P_D$, $l \in L$ facet	drag-and-drop $l \in L$ to query area - choice of $p \in P_D$ is implicit
specify $p \in P_O$, $u \in U$ facet	drag-and-drop $u \in U$ to query area - choice of $p \in P_O$ is implicit
specify $p \in P_O$ for path traversal	drag-and-drop $p \in P_O$ to query area

Table 10.3: Mapping operations to user actions.

Whenever users perform keyword search or specify a focus object they start a new query. Facets and path traversal operations can be either added to the current query by dropping the

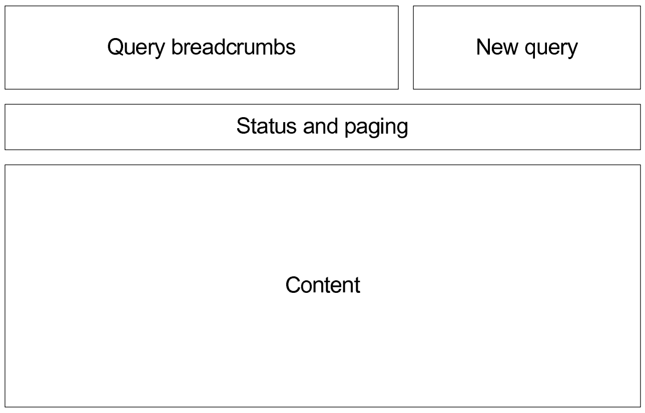


Figure 10.5: Main areas of the VisiNav result page.

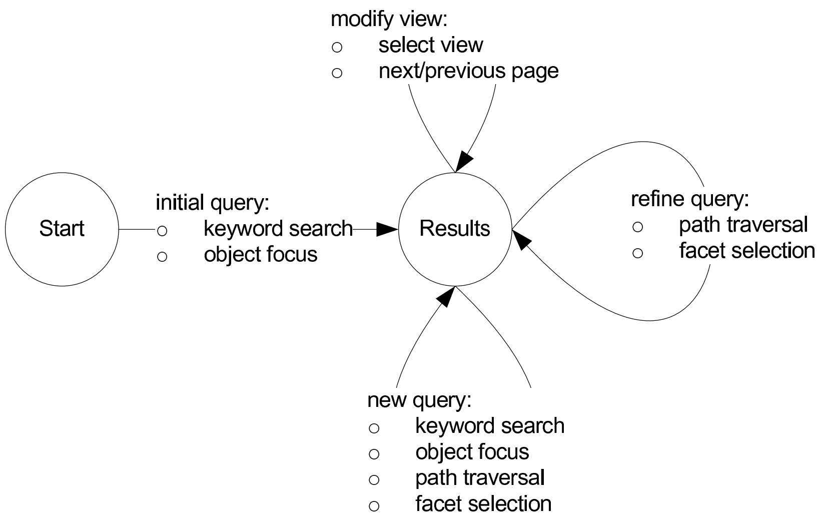


Figure 10.6: Interaction flow diagram.

facet over the current query, or can be used to start a new query by dropping the facet over the keyword search box ("New query" area).

We decided to include a drag-and-drop action mainly because some symbols in the user interface serve as input for multiple operations. For example, the object focus operation requires as a parameter an object $u \in U$, and the facet selection operation requires as parameters $p \in P_O$ and $u \in U$. In addition to refining a query, a user could also choose to select a facet for a new query. Using only a point-and-click model would clutter the user interface with three locations to click on; we claim that this would be confusing for the user and difficult to visualise. By introducing drag-and-drop, the user can bring an object into focus by point-and-click, add a facet to the current query by dropping the facet over the query breadcrumbs area, and start a new query by dropping the facet over the new query area.

The choice of p for facet selection is implicit: in the detail view, literal values and objects are grouped under the predicates in which they occur, and by dragging either the literal l or object u, the system adds the corresponding predicate p. In the list view and the visualisations, an abbreviated set of properties is shown due to limited screen space. Here, we treat classes (objects in the range of the `rdf:type` property) as a special case - although no predicate is shown, the system automatically adds an `rdf:type` predicate in case the class is dragged. By assigning drag-and-drop to more advanced features (path traversal and facet selection), novice users can begin to use the system with the traditional operations (keyword search and object focus) known from other web sites, and discover more advanced features later.

One issue of drag-and-drop is the lack of affordance - i.e. the users does not realise which elements can be dragged. As a visual cue for the possibility of dragging a sign, we underline the signs which could serve as input to operations with a dotted line.

We decided to depart from the aggregate view of prospective choices typical for faceted browsing, and instead embedded the facet selection into the content area. Doing so saves screen space and avoids displaying duplicate content. In addition, user studies revealed that the condensed view of prospective choices was too dense – the facet selector for a typical result set contained tens of predicates with tens of values and objects each.

While users are able to construct queries based on the attributes of prototypical objects shown in the current result, the system currently lacks a method for specifying arbitrary query operations. Allowing users to add arbitrary query operations, possibly via keyboard input assisted by autocompletion, is subject to further research.

10.5 Experiments and Evaluation

We implemented a series of prototypes operating on a number of datasets to validate and refine our design ideas. Our methodology was iterative: once we received feedback on a version of the implementation, we incorporated the user feedback into the next prototype. The user tests provide anectodal evidence of the utility of our system. We also provide an evaluation of the effort involved in creating queries based on a cost model. Finally, we present measurements evaluating query processing and view rendering performance.

10.5.1 Iterative Design and Continuous Feedback

During the design of VisiNav, we tested various prototypes on a number of datasets: the Mondial database[8] consisting of information about countries, an RDF version of CrunchBase[9] containing information about technology startups, and an RDF dataset retrieved by means

[8]`http://www.dbis.informatik.uni-goettingen.de/Mondial/`
[9]`http://cb.semsol.org/`

of a seven hops crawl from a seed URI[10]. After crawling, the web data was consolidated by materialising inferences [HHP09].

We initiated the design process using the Mondial dataset and several queries (e.g. "islands in calabria", "gdp of countries bordering italy"). We asked ten participants (colleagues who where familiar with web data and RDF) to evaluate early versions of the interface design based on several user tasks and a questionnaire. Each session took on average around 20 minutes; in order to simulate visitors to the VisiNav web site, we asked users to interact with the system immediately without a training or introduction phase.

The setup was the participants' laptop together with a projector so that the evaluator could track the user actions. We utilised the "thinking aloud" method [Nie94] to gain insight into what the users would expect from the system. While some users picked up quickly the conceptual model behind the user interface and were able to complete all queries, the majority were able to retrieve the right answers only for about half of the queries. Comments were taken into account for subsequent versions of the interface.

The suggestions and comments of the first round of evaluations were taken into account, and a second round of evaluations were conducted on a new user interface using a different dataset. This time, we used CrunchBase as the dataset, and performed only a few tests to verify that the changes made were actually benefitting the users (which the small study confirmed).

We also performed a series of user tests on the web dataset with five participants. All participants were able to find the correct answers to queries.

10.5.2 Query Construction Times

One of the key aspects of an exploratory search interface is how fast the users arrive at the desired result set or visualisation. In this section we assess the number of user actions required to arrive at the sought-after information. Table 10.4 lists the number of keyboard input, point-and-click, and drag-and-drop actions required to arrive at a query result for the example queries. The table lists only the time taken to arrive at a result, and does not include the single point-and-click operation it would take to change the view or visualisation. However, the times include selecting the detail view when necessary to reach the input for path traversal or facet selection. To be able to translate the number of actions into time, we assume constants derived from the literature [MSB91][KHHK99]: keyboard entry: 1.71 s per word – these constants assume use of English keyboards – 0.67 s for point-and-click, and 0.92 s for drag-and-drop and mouse as input devices.

[10]`http://www.w3.org/People/Berners-Lee/card`

Query	Actions	Time (s)
1	kb(kw)	5.13
2	Q_1 + pac(fo)	5.80
3	Q_2 + dnd(pt)	6.72
4	Q_2 + dnd(pt) + pac(v) + dnd(fs)	8.31
5	Q_2 + dnd(pt)	6.72
6	Q_5 + pac(v) + dnd(pt)	8.31
7	Q_5 + pac(v) + dnd(pt)	8.31
8	kb(fo) + dnd(fs)	11.18

Table 10.4: Interaction time and keyboard input (kb), point-and-click (pnc), and drag-and-drop (dnd) actions required to arrive at a query result for the example queries from Table 8.1. View change denoted with "v".

10.6 Related Work

NLMenu [TRTS83] is an early system advocating the use of multi-step query construction based on menus. Faceted browsing [YSLH03], while less expressive in terms of the complexity of queries, has become popular and is used on e-commerce sites such as Ebay.com. Polaris [STH02] provides complex query and aggregation operations, however, operates over relational data and thus requires a priori knowledge about the schema used.

Principles from user interface design [Nor02, Nie94] influenced the look-and-feel of and the interaction model employed in the application.

Parts of this chapter have been published in [Har09b], [Har09a]. VisiNav was awarded 2nd prize at the 2009 Semantic Web Challenge.

10.7 Conclusion

The ability to integrate hitherto disparate pieces of data – and thus enable applications to re-purpose data in unanticipated ways – enables the creation of novel applications, but pose new challenges to the design of user interfaces. In this chapter, we introduce a general, formal model for searching and browsing objects, and present a prototype system implementing these ideas. The interaction model provides operations that allow users to explore and visualise data without requiring knowledge about the schema of the data. Users are able to learn the structure of the domain of interest while interacting with the system.

The system provides a universal way of interacting with any RDF dataset, without manual curation of the data or domain-specific adaptation of the interface, and thus can be directly applied to data from the open web. In addition, the system allows to rapidly develop and employ data integration systems in more confined environments such as intranets. Here, the data is typically domain-specific and the number of data sources confined, which opens the door for cost-effective manual curation of the data. In such environments, providing domain-specific widgets for visualising popular types of objects (similar to the current timeline and map visualisations) is a way to further increase the overall utility of the system.

Potential areas of future work include investigation of the possibility of mapping the generic query operations to other modalities, such as natural language or speech input. Also, we would like to test our system on new hardware such as touch screens, and investigate how groups of users can use VisiNav to collectively explore and analyse data integrated from vast amounts of sources.

Chapter 11

Conclusion

Let my dataset change your mindset.

Hans Rosling

In traditional media – such as newspapers, radio, or television – information is being purified, processed, and integrated from a fixed number of sources; analysis and interpretation of information is carried out by a closed team and the results are then finally broadcast to the target audience. In contrast, web technologies allow for performing integration steps automatically and rapidly from a huge number of sources; users are encouraged to contribute and become part of the process, resulting in a participatory medium. Collaborative data creation models, in conjunction with data integration systems such as the one presented in this dissertation, hold the potential to change how information from a large number of domains is collected, interpreted, analysed, and presented.

We have presented the architecture and implementation of a system that syntactically and semantically integrates information from a large number of data sources on the web, demonstrating the feasibility of bringing together data and corroborating evidence from multiple sources. Making web data available for querying and navigation has significant scientific and commercial potential. Firstly, the web – the largest digital artifact of human knowledge – becomes subject to scientific analysis [BLHH$^+$06]. Understanding the implicit connections and structure in the web data graph can help to reveal new understandings of collaboration patterns and the processes by which networks form and evolve. Secondly, making sense out of data published on the web can help scientists to gain insights for their research. The ability to navigate and search effectively through a store of knowledge integrated from thousands of sources can broaden the pool of information and ideas available to a scientific community. Thirdly, making the web data graph available for interactive querying, browsing, and navigation has applications in areas such as e-commerce and e-health, and has the potential to drastically improve the quality and utility of web search in general.

While this dissertation has made the first steps towards realising that vision, a number of issues remain to be solved before that vision can become a reality:

- Data publishing: The more that people publish data and participate in a network, the more useful the overall network becomes. Thus, as more people and organisations provide open data, the resulting web of data becomes more useful. To completely achieve the vision of open data, a number of developments which take place at communities of early adopters have to become mainstream; for example, computer scientists typically publish their research papers on their home page. That change has to permeate to other, less web-savvy research communities.
- Privacy: With the ubiquity of computers and mobile devices in modern life, the amount of personal data recorded and shared increases dramatically. An area for future work is to design mechanisms and algorithms that empower users to decide which data to share with whom, and to allow data providers to track the usage of their personal information.
- Data quality: The amount of data on the web has grown considerably in the past years; however, the quality of the data (e.g., interlinkage and re-use of identifiers for objects and schema elements) has to improve to enable more complex application scenarios. An additional open issue is that of logical consistency: while individual pieces of information might be logically consistent, combining data from independent sources might introduce inconsistencies. While individual users and small groups publish well-interlinked data (most notably the linked open data effort), larger groups of users need to converge to re-using URIs for instances and vocabularies. The convergence is happening through social processes – for example, the growth of community-driven publishing – which take time. How to support this evolutionary process with technology is an area for future work.
- Expressive knowledge representation: Currently, data available on the web is varying in terms of the underlying expressiveness of knowledge representation. While most of the data describes basic facts, little coordinated action went into representing more complex dimensions of data, such as spatial (geometrical objects, shapes) or temporal aspects, or units and measurements. While the data currently available exhibits enough structure for basic applications, more complex use cases can only satisfied if data sources employ more complex knowledge representation aspects. The shift toward more elaborate data should happen in parallel with the effort to improve data quality.
- Comprehensive visualisations: While basic object-centred navigation and queries are enabled through the currently available data, more complex visualisations require better modelled data as well as more advanced visualisation components. Anticipating more and better modelled data, next-generation visualisation systems could include the display of time series, shapes, and statistical information through easy-to-understand visualisations.

Bibliography

[ABS06] Harith Alani, Christopher Brewster, and Nigel Shadbolt. Ranking ontologies with AKTiveRank. In *5th International Semantic Web Conference*, pages 1–15, 2006.

[ACK$^+$01] Sofia Alexaki, Vassilis Christophides, Greg Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The RDFSuite: Managing voluminous RDF description bases. In *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with WWW10, Hongkong*, May 2001.

[ACK04] Nikos Athanasis, Vassilis Christophides, and Dimitris Kotzinos. Generating on the fly queries for the semantic web: The ICS-FORTH graphical RQL interface (GRQL). In *3rd International Semantic Web Conference*, pages 486–501, November 2004.

[ACMD$^+$03] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Record*, 32(3):29–33, 2003.

[ACMD05] Karl Aberer, Philippe Cudré-Mauroux, and Zoran Despotovic. On the convergence of structured search, information retrieval and trust management in distributed systems. In *Multiagent System Technologies, Third German Conference, MATES 2005*, pages 1–14, 2005.

[ACS89] Michele Angelaccio, Tiziana Catarci, and Giuseppe Santucci. Qbd*: a graphical query language with recursion. In *Proceedings of the third international conference on human-computer interaction on Designing and using human-computer interfaces and knowledge based systems (2nd ed.)*, pages 384–397, New York, NY, USA, 1989. Elsevier Science Inc.

[ADOS03] Harith Alani, Srinandan Dasmahapatra, Kieron O'Hara, and Nigel Shadbolt. Identifying communities of practice through ontology network analysis. *IEEE Intelligent Systems*, 18(2):18–25, 2003.

[AG05] Renzo Angles and Claudio Gutiérrez. Querying RDF data from a graph database perspective. In *Proceedings of the Second European Semantic Web Conference*, pages 346–360, 2005.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AMS05] Kemafor Anyanwu, Angela Maduko, and Amit Sheth. Semrank: ranking complex relationship search results on the semantic web. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 117–127, New York, NY, USA, 2005. ACM Press.

[BCSV04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. UbiCrawler: a Scalable Fully Distributed Web Crawler. *Software: Practice and Experience*, 34(8):711–726, 2004.

[BCSW07] Holger Bast, Alexandru Chitea, Fabian Suchanek, and Ingmar Weber. ESTER: efficient search on text, entities, and relations. In *30th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 671–678, 2007.

[BDH$^+$95] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The harvest information discovery and access system. *Comput. Netw. ISDN Syst.*, 28(1-2):119–125, 1995.

[Bec02] David Beckett. The design and implementation of the Redland RDF application framework. *Computer Networks*, 39(5):577–588, 2002.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, September 1975.

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 119–128, September 2001.

[BG81] Philip A. Bernstein and Nathan Goodman. Power of natural semijoins. *SIAM Journal on Computing*, 10(4):751–771, 1981.

[BH98] Krishna Bharat and Monika R. Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 104–111, Melbourne, AU, 1998.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: authority-based keyword search in databases. In *VLDB '04: Proceedings of the*

Thirtieth international conference on Very large data bases, pages 564–575. VLDB Endowment, 2004.

[BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proceedings of the 2nd International Semantic Web Conference, Sardinia*, pages 54–68. Springer, 2002.

[BL06] Tim Berners-Lee. Linked data - design issues, 2006. http://www.w3.org/DesignIssues/LinkedData.

[BLCC$^+$06] Tim Berners-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 Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06*, page 06, 2006.

[BLHH$^+$06] Tim Berners-Lee, Wendy Hall, James Hendler, Nigel Shadbolt, and Daniel J. Weitzner. Creating a science of the web. *Science*, 313(11), 2006.

[BM72] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, 30(1-7):107–117, 1998.

[Bra78] Ronald J. Brachman. On the epistemological status of semantic networks, 1978.

[Bre98] Eric A. Brewer. Combining Systems and Databases: A Search Engine Retrospective . *Readings in Database Systems, 4th. Edition*, 1998.

[Bri02] Dan Brickley. RDFWeb notebook: aggregation strategies, January 2002. http://rdfweb.org/2001/01/design/smush.html.

[BSLF06] Daniel Bachlechner, Katharina Siorpaes, Holger Lausen, and Dieter Fensel. Web service discovery - A reality check. In *Proceedings of the 1st Workshop: SemWiki2006 - From Wiki to S emantics*, Budva, Montenegro, June 2006.

[BSMG06] Paolo Bouquet, Heiko Stoermer, Michele Mancioppi, and Daniel Giacomuzzi. OkkaM: Towards a solution to the "identity crisis" on the semantic web. In *Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop*, volume 201 of *CEUR Workshop Proceedings*, December 2006.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, May 1999.

[CBHR06] John Cieslewicz, Jonathan Berry, Bruce Hendrickson, and Kenneth A. Ross. Realizing parallelism in database operations: insights from a massively multithreaded architecture. In *DaMoN '06: Proceedings of the 2nd international workshop on Data management on new hardware*, page 4, New York, NY, USA, 2006. ACM Press.

[CBHS05] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick 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 Press.

[CCT09] Richard Cyganiak, Michele Catasta, and Giovanni Tummarello. Towards ecsse: live web of data search and integration. In *Semantic Search 2009 Workshop*, 2009.

[CDES05] Eugene Inseok Chong, Souripriya Das, George Eadon, and Jagannathan Srinivasan. An efficient SQL-based RDF querying scheme. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1216–1227. VLDB Endowment, 2005.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, pages 15–15. USENIX Association, 2006.

[CF04] Min Cai and Martin Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In *Proceedings of the 13th International World Wide Web Conference*, pages 650–657. ACM Press, 2004.

[CGQ08] Gong Cheng, Weiyi Ge, and Yuzhong Qu. Falcons: searching and browsing entities on the semantic web. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 1101–1102. ACM, 2008.

[Cha03] Soumen Chakrabarti. *Mining the Web*. Morgan Kaufmann Publishers, Inc., 2003.

[Che76] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CHWM04] Deng Cai, Xiaofei He, Ji-Rong Wen, and Wei-Ying Ma. Block-level link analysis. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 440–447. ACM New York, NY, USA, 2004.

[CJ] Richard Cyganiak and Anja Jentzsch. Linking open data cloud diagram. http://lod-cloud.net/.

[CM90] Mariano P. Consens and Alberto O. Mendelzon. GraphLog: A Visual Formalism for Real Life Recursion. In *PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 404–416, New York, NY, USA, 1990. ACM Press.

[CMW87] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. *SIGMOD Rec.*, 16(3):323–330, 1987.

[Com79] Douglas Comer. The Ubiquitous B-Tree. *ACM Computing Surveys*, 11:121–137, 1979.

[Con02] Larry L. Constantine. The emperor has no clothes: Naked objects meet the interface, 2002.

[DEFS98] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *DS-8: Procs of 8th Working Conference on Database Semantics*, 1998.

[DEG$^+$03] Stephen Dill, Nadav Eiron, David Gibson, Daniel Gruhl, Ramanathan Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John A. Tomlin, and Jason Y. Zien. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *Proceedings of the Twelfth International World Wide Web Conference*, pages 178–186, May 2003.

[DF06] Li Ding and Tim Finin. Characterizing the semantic web on the web. In *Proceedings of the 5th International Semantic Web Conference*, Athens, Georgia, November 2006.

[DFJ$^+$04] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal C Doshi, , and Joel Sachs. Swoogle: A Search and Metadata Engine for the Semantic Web. In *Proceedings of the 13th ACM Conference on Information and Knowledge Management*, November 2004.

[dMS04] Sabrina Silva de Moura and Daniel Schwabe. Interface development for hypermedia applications in the semantic web. In *Joint Conference 10th Brazilian Symposium on Multimedia and the Web & 2nd Latin American Web Congress*, pages 106–113, 2004.

[DMvH$^+$00] Stefan Decker, Sergey Melnik, Frank v. van Harmelen, Dieter Fensel, Michel C. A. Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. The semantic web: The roles of xml and rdf. *IEEE Internet Computing*, 4(5):63–74, 2000.

[DPF^{+}05] Li Ding, Rong Pan, Tim Finin, Anupam Joshi, Yun Peng, and Pranam Kolari. Finding and ranking knowledge on the semantic web. In *4th International Semantic Web Conference*, pages 156–170, 2005.

[DSC^{+}07] Pedro DeRose, Warren Shen, Fei Chen, AnHai Doan, and Raghu Ramakrishnan. Building structured web community portals: a top-down, compositional, and incremental approach. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 399–410. VLDB Endowment, 2007.

[DSDH08] Anish Das Sarma, Xin Dong, and Alon Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 861–874, New York, NY, USA, 2008. ACM.

[Ebe02] Andreas Eberhart. Survey of RDF data on the web. Technical report, International University in Germany, 2002.

[Eic94] David Eichmann. The RBSE spider - balancing effective search against web load. In *First International Conference on the World-Wide Web*, May 1994.

[EIV07] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[Emi97] Mustafa Emirbayer. Manifesto for a relational sociology. *American Journal of Sociology*, 103(2):281–317, September 1997.

[ER59] Paul Erdös and Alfréd Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[Eul41] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[FHK^{+}97] Jürgen Frohn, Rainer Himmeröder, Paul-Thomas Kandzia, Georg Lausen, and Christian Schlepphorst. FLORID: A prototype for F-logic. In W. A. Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, page 583. IEEE Computer Society, 1997.

[FvH07] Dieter Fensel and Frank van Harmelen. Unifying reasoning and search to web scale. *IEEE Internet Computing*, 11(2):94–95, 2007.

[GKPS05] Georg Gottlob, Christoph Koch, Reinhard Pichler, and Luc Segoufin. The Complexity of XPath Query Evaluation and XML Typing. *Journal of the ACM*, 52(2):284–335, 2005.

[GM03] Ramanathan V. Guha and Rob McCool. TAP: a semantic web platform. *Computer Networks*, 42(5):557–577, 2003.

[GMF04] Ramanathan V. Guha, Rob McCool, and Richard Fikes. Contexts for the Semantic Web. In *Proceedings of the 3rd International Semantic Web Conference, Hiroshima*, November 2004.

[GMWU99] Hector Garcia-Molina, Jennifer Widom, and Jeffrey D. Ullman. *Database System Implementation.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[Gog93] Joseph A. Goguen. Keynote: On notation. In *10th International Conference on Technology of Object-Oriented Languages and Systems*, pages 5–10, 1993.

[GS05] Daniel Gomes and Mario J. Silva. Characterizing a national community web. *ACM Transactions on Internet Technology*, 5(3):508–531, 2005.

[Guh] Ramanathan V. Guha. rdfDB : An RDF Database. http://www.guha.com/rdfdb/.

[Har09a] Andreas Harth. Interactive exploration of web datasets with visinav. In *Proceedings of the 2009 Semantic Web Challenge*, 2009.

[Har09b] Andreas Harth. Visinav: Visual web data search and navigation. In *DEXA '09: Proceedings of the 20th International Conference on Database and Expert Systems Applications*, pages 214–228, Berlin, Heidelberg, 2009. Springer-Verlag.

[HD05a] Andreas Harth and Stefan Decker. Optimized index structures for querying RDF from the web. In *Proceedings of the 3rd Latin American Web Congress*, pages 71–80. IEEE Press, 2005.

[HD05b] Andreas Harth and Stefan Decker. Optimized index structures for querying rdf from the web. In *Proceedings of the 3rd Latin American Web Congress*, pages 71–80. IEEE Press, 2005.

[HD05c] Andreas Harth and Stefan Decker. Optimized index structures for querying RDF from the web. In *Proceedings of the 3rd Latin American Web Congress*. IEEE Press, 2005.

[Hea08] Marti A. Hearst. Uis for faceted navigation: Recent advances and remaining open problems. In *HCIR08 Second Workshop on Human-Computer Interaction and Information Retrieval*, October 2008.

[Hen07] Monika Henzinger. Search Technologies for the Internet. *Science*, 317(5837):468–471, 2007.

[HFV96] Waqar Hasan, Daniela Florescu, and Patrick Valduriez. Open issues in parallel query optimization. *SIGMOD Record*, 25(3):28–33, 1996.

[HG03] Stephen Harris and Nicholas Gibbins. 3store: Efficient bulk rdf storage. In *Proceedings of the First International Workshop on Practical and Scalable Semantic Systems*, 2003.

[HHD06] Aidan Hogan, Andreas Harth, and Stefan Decker. ReConRank: A scalable ranking method for semantic web data with context. In *2nd Workshop on Scalable Semantic Web Knowledge Base Systems*, 2006.

[HHD$^+$07a] Andreas Harth, Aidan Hogan, Renaud Delbru, Jürgen Umbrich, Sean O'Riain, and Stefan Decker. SWSE: Answers before links! In *Semantic Web Challenge, 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*, 2007.

[HHL99] Jeff Heflin, James Hendler, and Sean Luke. SHOE: A Knowledge Representation Language for Internet Applications. Technical Report CS-TR-4078, Dept. of Computer Science, University of Maryland, 1999.

[HHP09] Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable authoritative owl reasoning for the web. *International Journal of Semantic Web Information Systems*, 5(2):49–90, 2009.

[HHP$^+$10] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the pedantic web. In *Proceedings of the Linked Data on the Web WWW2010 Workshop (LDOW 2010)*, 2010.

[HK] David F. Huynh and David Karger. Parallax and companion: Set-based browsing for the data web. Available online (2008-12-15) http://davidhuynh.net/media/papers/2009/www2009-parallax.pdf.

[HKD09] Andreas Harth, Sheila Kinsella, and Stefan Decker. Using naming authority to rank data and ontologies for web search. In *Proceedings of the 8th International Semantic Web Conference*, 2009.

[HMS$^+$05] Eero Hyvnen, Eetu Mkel, Mirva Salminen, Arttu Valo, Kim Viljanen, Samppa Saarela, Miikka Junnila, and Suvi Kettula. Museumfinland – finnish museums on the semantic web. *Journal of Web Semantics*, 3(2):25, 2005.

[HN99] Allan Heydon and Marc Najork. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web*, 2(4):219–229, 1999.

[HPD07] Andreas Harth, Axel Pollres, and Stefan Decker. Towards a social provenance model for the web. In *Workshop on Principles of Provenance (PrOPr)*, November 2007.

[HRGMP00] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. WebBase: a repository of Web pages. *Computer Networks*, 33(1–6):277–293, 2000.

[HUD06] Andreas Harth, Jürgen Umbrich, and Stefan Decker. Multicrawler: A pipelined architecture for crawling and indexing semantic web data. In *Proceedings of the 5th International Semantic Web Conference*, pages 258–271, 2006.

[HUHD07] Andreas Harth, Juergen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A Federated Repository for Searching and Querying Graph-Structured Data. In *Proceedings of the 6th International Semantic Web Conference*, November 2007.

[HvOH06] Michiel Hildebrand, Jacco van Ossenbruggen, and Lynda Hardman. /facet: A browser for heterogeneous semantic web repositories. In *5th International Semantic Web Conference*, pages 272–285, November 2006.

[IBS08] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):1–58, 2008.

[Jac99] Robert Jacobson, editor. *Information Design.* MIT Press, 1999.

[JXS$^+$04] Xue-Mei Jiang, Gui-Rong Xue, Wen-Guan Song, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Exploiting PageRank at Different Block Level . In *Proceedings of Web Information Systems WISE 2004*, pages 241–252. Springer, 2004.

[KAM03] Anastasios Kementsietsidis, Marcelo Arenas, and Renee J. Miller. Managing data mappings in the hyperion project. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, pages 732–734, 2003.

[KBK05] Tamara G. Kolda, Brett W. Bader, and Joseph P. Kenny. Higher-order web link analysis using multilinear algebra. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 242–249. IEEE Computer Society Washington, DC, USA, 2005.

[KD08] Georgi Kobilarov and Ian Dickinson. Humboldt: Exploring linked data. In *Linked Data on the Web Workshop*, 2008.

[KHHK99] Clare-Marie Karat, Christine Halverson, Daniel Horn, and John Karat. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 568–575, New York, NY, USA, 1999. ACM.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[Kos94] Martijn Koster. Aliweb - archie-like indexing in the web. In *First International Conference on the World-Wide Web*, May 1994.

[Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[Kum07] Kengo Kuma. *Anti-Object: The Dissolution and Disintegration of Architecture*. AA Publications, 2007.

[LENW82] Daniel E. Lipkie, Steven R. Evans, John K. Newlin, and Robert L. Weissman. Star graphics: An object-oriented implementation. In *SIGGRAPH '82: Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 115–124. ACM, 1982.

[LG00] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Intelligence*, 11(1):32–39, 2000.

[LLWL08] Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. Irlbot: scaling to 6 billion pages and beyond. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 427–436. ACM, 2008.

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010.

[Lum70] Vincent Y. Lum. Multi-attribute retrieval with combined indexes. *Communications of the ACM*, 13(11):660–665, 1970.

[Mar97] Massimo Marchiori. The quest for correct information on the Web: Hyper search engines. *Computer Networks and ISDN Systems*, 29(8–13):1225–1236, 1997.

[MAT$^+$04] Martin Michalowski, Jose Luis Ambite, Snehal Thakkar, Rattapoom Tuchinda, Craig A. Knoblock, and Steve Minton. Retrieving and Semantically Integrating Heterogeneous Data from the Web. *IEEE Intelligent Systems*, 19(3):72–79, 2004.

[Mau97] Michael I. Mauldin. Lycos: design choices in an internet search service. *IEEE Expert [see also IEEE Intelligent Systems and Their Applications]*, 12(1):8–11, January/February 1997.

[McB94] Oliver A. McBryan. GENVL and WWWW: Tools for taming the web. In *First International Conference on the World-Wide Web*, May 1994.

[McK94] Doug McKee. Towards better integration of dynamic search technology and the world-wide web. In *First International Conference on the World-Wide Web*, May 1994.

[Mil68] Robert B. Miller. Response time in man-computer conversational transactions. In *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, 1968.

[MK03] Robert M. MacGregor and In-Young Ko. Representing Contextualized Data using Semantic Web Tools. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems, Sanibel Island, Florida*, October 2003.

[MM04] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-primer/.

[MMSK08] Pablo N. Mendes, Bobby McKnight, Amit P. Sheth, and Jessica C. Kissinger. TcruziKB: Enabling complex queries for genomic data exploration. *IEEE International Conference on Semantic Computing*, pages 432–439, August 2008.

[MRYGM01] Sergey Melnik, Sriram Raghavan, Beverly Yang, and Hector Garcia-Molina. Building a Distributed Full-Text Index for the Web. In *Proceedings of the 10th International World Wide Web Conference, Hong Kong*, pages 396–406, 2001.

[MSB91] I. Scott MacKenzie, Abigail Sellen, and William A. S. Buxton. A comparison of input devices in element pointing and dragging tasks. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 161–166. ACM, 1991.

[MSP$^+$04] Li Ma, Zhong Su, Yue Pan, Li Zhang, and Tao Liu. RStar: an RDF storage and query system for enterprise resource management. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 484–491, New York, NY, USA, 2004. ACM Press.

[MTK03] Martin Michalowski, Snehal Thakkar, and Craig A. Knoblock. Exploiting secondary sources for automatic object consolidation. In *Proceeding of 2003 KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[Nat08] Nature. Community cleverness required. *Nature - Special Issue on Big Data*, 455(7209):1, 2008.

[NH94] Christian Neuss and Stefanie H"ofling. Lost in hyperspace? free text searches in the web. In *First International Conference on the World-Wide Web*, May 1994.

[NH02] Marc Najork and Allan Heydon. *High-performance web crawling*, pages 25–45. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Nie94] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994.

[NKAJ59] Howard B. Newcombe, James M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records: Computers can be used to extract "follow-up" statistics of families from files of routine records. *Science*, 130:954–959, October 1959.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, September 2002.

[NW09] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. Research Report MPI-I-2009-5-003, Max-Planck-Institut fr Informatik, Saarbrücken, Germany, March 2009.

[ODD06] Eyal Oren, Renauld Delbru, and Stefan Decker. Extending faceted navigation for rdf data. In *5th International Semantic Web Conference*, Nov 2006.

[ODG$^+$07] Eyal Oren, Renaud Delbru, Sebastian Gerke, Armin Haller, and Stefan Decker. Activerdf: object-oriented semantic web programming. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 817–824, New York, NY, USA, 2007. ACM.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[PD06] Jack Park and Patrick Durusau. Towards subject-centric merging of ontologies. In *Proceedings of the 9th International Protege Conference*, July 2006.

[Pei09] Charles Peirce. Ms 478: Existential graphs, 1909.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260, Washington, DC, USA, 1995. IEEE Computer Society.

[Pin94] Brian Pinkerton. Finding what people want: Experiences with the webcrawler. In *First International Conference on the World-Wide Web*, May 1994.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware 2001 : IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–351. Springer, 2001.

[RG03] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems.* McGraw-Hill Book Co., 3rd edition, 2003.

[Ric58] Richard H. Richens. Interlingual Machine Translation. *The Computer Journal*, 1(3):144–147, 1958.

[SAC$^+$79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 International Conference on Management of Data*, pages 23–34, 1979.

[SBC98] Ben Shneiderman, Donald Byrd, and W. Bruce Croft. Sorting out searching: a user-interface framework for text searches. *Commun. ACM*, 41(4):95–98, 1998.

[Sco88] John Scott. Trend report: Social network analysis. *Sociology*, 22(1):109–27, 1988.

[Sim66] Robert F. Simmons. Storage and retrieval of aspects of meaning in directed graph structures. *Communications of the ACM*, 9(3):211–215, 1966.

[SK05] Vineet Sinha and David R. Karger. Magnet: supporting navigation in semistructured data environments. In *ACM SIGMOD International Conference on Management of Data*, pages 97–106, 2005.

[SM84] Gerard Salton and Michael McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill Book Company, 1984.

[SMHM99] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[SMLN$^+$03] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[STH02] Chris Stolte, Diane Tang, and Pat Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[Sto86] Michael Stonebraker. The case for shared nothing. *Database Engineering Bulletin*, 9(1):4–9, 1986.

[Sto08] Michael Stonebraker. Technical perspective - one size fits all: an idea whose time has come and gone. *Commun. ACM*, 51(12):76, 2008.

[SVHB04] Heiner Stuckenschmidt, Richard Vdovjak, Geert-Jan Houben, and Jeen Broekstra. Index Structures and Algorithms for Querying Distributed RDF Repositories. In *Proceedings of 13th International World Wide Web Conference, New York*, pages 631–639, May 2004.

[TBF$^+$08] Michal Tvarozek, Michal Barla, György Frivolt, Marek Tomśa, and Mária Bieliková. Improving semantic search viaintegrated personalized faceted andvisual graphnavigation. In *SOFSEM 2008: Theory and Practice of Computer Science*, pages 778–789, 2008.

[TMV88] Olga De Troyer, Robert Meersman, and P. Verlinden. Ridl on the cris case: A workbench for niam. In *Computerized Assistance During the Information Systems Life Cycle*, pages 375–459, 1988.

[TRTS83] Craig W. Thompson, Kenneth M. Ross, Harry R. Tennant, and Richard M. Saenz. Building usable menu-based natural language interfaces to databases. In *9th International Conference on Very Large Data Bases*, pages 43–55, 1983.

[Tuf01] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001.

[TWRC09] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *Proceedings of the 25th International Conference on Data Engineering*, pages 405–416, 2009.

[WGA05] David Wood, Paul Gearon, and Tom Adams. Kowari: A platform for semantic web storage and analysis. In *XTech 2005 Conference*, 2005.

[WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1:1008–1019, August 2008.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann, May 1999.

[Woo75] William A. Woods. What's in a link: Foundations for semantic networks, 1975.

[WPH06] Taowei David Wang, Bijang Parsia, and James Hendler. A survey of the web ontology landscape. In *Proceedings of the 5th International Semantic Web Conference*, Athens, Georgia, November 2006.

[WSKR03] Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, and Dave Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proceedings of SWDB'03, 1st International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Berlin*, pages 131–150, September 2003.

[YSLH03] Ka-Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 2003. ACM Press.

[ZN09] Jianfeng Zheng and Zaiqing Nie. Architecture and implementation of object-level vertical search. In *International Conference on New Trends in Information and Service Science*, volume 0, pages 264–268. IEEE Computer Society, 2009.