VisiNav: Visual Web Data Search and Navigation

Andreas Harth[⋆]

National University of Ireland, Galway
Digital Enterprise Research Institute

Abstract. Semantic Web technologies facilitate data integration over a large number of sources with decentralised and loose coordination, ideally leading to interlinked datasets which describe objects, their attributes and links to other objects. Such information spaces are amenable to queries that go beyond traditional keyword search over documents. To this end, we present a formal query model comprising six atomic operations over object-structured datasets: keyword search, object navigation, facet selection, path traversal, projection, and sorting. Using these atomic operations, users can incrementally assemble complex queries that yield a set of objects or trees of objects as result. Results can then be either directly displayed or exported to application programs or online services. We report on user experiments carried out during the design phase of the system, and present performance results for a range of queries over 18.5m statements aggregated from 70k sources.

1 Introduction

Keyword search over hypertext documents is an established technology and is used by a large majority of web users [5]. Search engines are popular because i) users are accustomed to the concept of hypertext: documents and links, and ii) search engines employ a simple conceptual model: the engines return those documents that match the specified keywords. Search engines operate over millions of documents which have been collected automatically, however, the functionality is limited: the engine returns only links to web pages 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 [5]; moreover, information expressed in documents in natural language is ambiguous and thus hard to process automatically. Data formats such as RDF[1] provide more structure, however, there is the open question of how end users should express complex queries over such datasets.

Natural language question answering interfaces are judged preferable to other interfaces by users [10], however, are not in common use today because the approach is fraught with usability issues: despite user training with regards to the

[⋆] Current affiliation: Institute AIFB, University of Karlsruhe (TH), Germany
[1] Resource Description Framework, `http://www.w3.org/RDF/`

capabilities and limitations of a natural language system, users quickly develop negative expectations about the system due to the relatively high error rates in parsing and interpreting natural language [16]. Users are unable to understand the limitations of such systems, that is, to distinguish between conceptual coverage (i.e. does the dataset contain the answer?) and linguistic coverage (i.e. is the system capable of parsing the query?).

A promising approach is to use a menu-based dialogue system in which users incrementally construct the query [16] [18]. 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 raises a number of challenges:

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 inter-linkage).
3. Zero Configuration: data on the web comes in an abundance of formats and vocabularies. Consequently, manual intervention is a labour intensive task. In addition, web data is often chaotic and may contain duplicates, erroneous items, malformed syntax and incorrect formatting.
4. 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.
5. 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 a sense of achievement immediately.

In this paper, we describe VisiNav, a fully implemented system[2] based on a visual query construction paradigm. 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 from a large number of web sources. To leverage existing familiarity of users with search engines, the first step in our interaction model is typically a keyword search to locate objects. 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 calculates the possible next steps based on the current state, only legal choices are displayed and thus the user can only compose queries which the system can answer.

Our contributions are as follows:

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

- We define and formalise a set of atomic query operations on object-orientated data models which can be combined to form complex queries.
- We introduce the notion of result trees which extend single-set results to multiple result sets containing result paths.
- We describe the architecture and implementation of a prototype system to investigate the practicality of the interaction model.
- We define the notion of topical subgraphs, subsets of the data which contain both the answer to the query and auxiliary information required to derive prospective choices and render the results.
- We propose a set of indices supporting the atomic operations and a query processing algorithm with top-k processing, and present a performance evaluation of the system on a web dataset with 18.5m statements.

We provide an overview of the user interface and preliminary definitions in Section 2, define and formalise the atomic operations and result trees in Section 3, present architecture and implementation in Section 4 and discuss experiments and evaluation in Section 5. Section 6 covers related work, and Section 7 concludes.

2 Overview and Preliminaries

In the following, we describe the characteristics of the target dataset collected from the web, present example queries, and introduce the conceptual model and the user interface.

2.1 Web Data

Common to data currently found on the web in structured formats (microformats, XML, RDF) is that data publishers take a loosely object-centred view. RDF in particular uses URIs[3] as global identifiers for objects, which, if multiple sources reuse identifiers, leads to an interconnected object space encoded in a graph-structured data format. Currently, reuse of identifiers is particularly common in social networking and social media data, expressed in FOAF[4] for people, SIOC[5] for online community sites, and DC[6] for documents. While a large number of current RDF files use a mix of these vocabularies, data publishers use a plethora of other vocabularies. Our dataset[7] of 18.5m statements from 70k sources contains over 21k different vocabulary URIs.

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

[3] Uniform Resource Identifiers, `http://www.rfc-editor.org/rfc/rfc3305.txt`
[4] Friend-of-a-Friend, `http://foaf-project.org/`
[5] Semantically Interlinked Online Communities, `http://sioc-project.org/`
[6] Dublin Core, `http://dublincore.org/`
[7] Crawled six hops from the seed URI `http://www.w3.org/People/Berners-Lee/card` and with materialised authoritative inferences [7]

domain independent. We list a number of example queries – that can be answered with currently available web data – with increasing complexity in Table 1[8]. We use these queries instantiated with different names and object URIs to conduct performance tests in Section 5.

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	objects that `timbl:i` `foaf:knows`
6	objects `foaf:made` by objects that `timbl:i` `foaf:knows`
7	query 6, results sorted by `dc:date`

Table 1. Example queries. Users typically start with a keyword query ("tim berners-lee") and subsequently select the URI identifying the intended object. Further choices are made from a menu of valid selections.

2.2 Conceptual Model

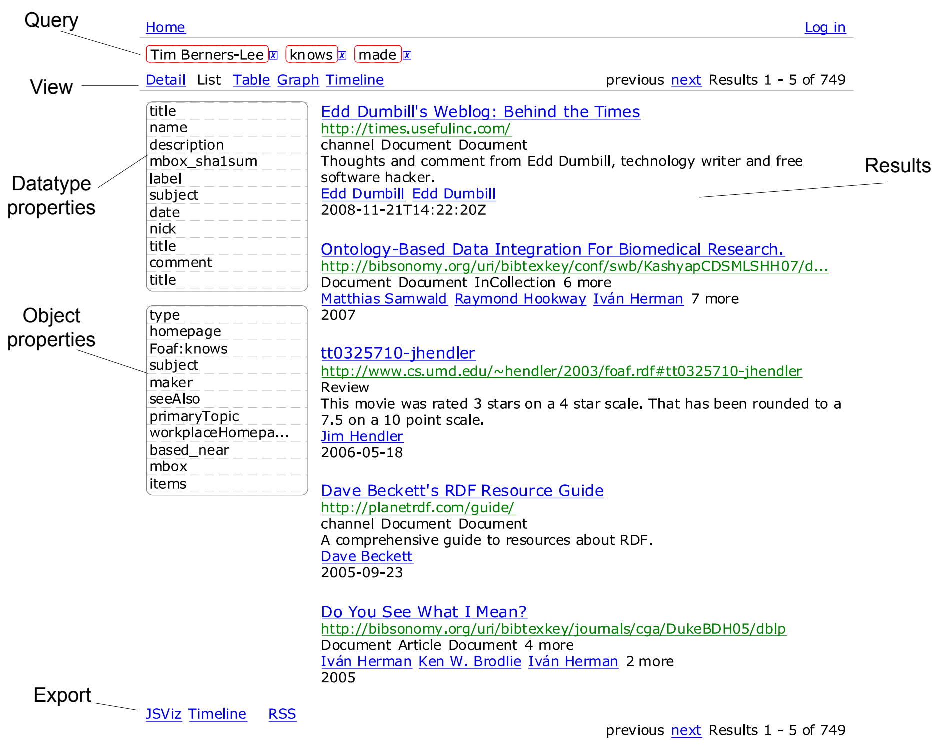
Our conceptual model for navigation assumes an object-oriented view, describing objects (U), their attributes and links to other objects. Attributes of objects are expressed using datatype properties (P_D), and links to other objects are specified using object properties (P_O)[9]. Please note that there is no clear distinction between instance-level objects and schema-level ones – classes and properties can be instances themselves.

Users are able to search over the object space yielding objects as a result set. Users can restrict the result set to objects matching specified facets - combinations of properties and objects or datatype values (L). In addition, query operations can be used to navigate in the result set along object properties, yielding another set of objects. The individual object sets form, in combination, a result tree, and previous result sets can be used later in the search process. The current result set can be modified by projecting out datatype properties and sorting the object result set according to datatype properties. Users can choose to display the result set in detail, list, and table view; optionally, a map, timeline, or graph view are available if the result objects contain suitable information for the view. Users are able to export the results to application programs or services. We discuss each operation in detail in Section 3.

The interface in Figure 1 shows results for the query "objects `foaf:made` by objects that `timbl:i` `foaf:knows`, sorted by `dc:date`" (Query 7). The results displayed have been aggregated and integrated from multiple web sources.

[8] `timbl:i` expands to `http://www.w3.org/People/Berners-Lee/card#i`; we assume the standard namespace prefixes for `foaf`, `sioc` and `dc`

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



Home Log in

Tim Berners-Lee knows made

Detail List Table Graph Timeline previous next Results 1 - 5 of 749

title
name
description
mbox_sha1sum
label
subject
date
nick
title
comment
title

type
homepage
Foaf:knows
subject
maker
seeAlso
primaryTopic
workplaceHomepa...
based_near
mbox
items

Edd Dumbill's Weblog: Behind the Times
http://times.usefulinc.com/
channel Document Document
Thoughts and comment from Edd Dumbill, technology writer and free software hacker.
Edd Dumbill Edd Dumbill
2008-11-21T14:22:20Z

Ontology-Based Data Integration For Biomedical Research.
http://bibsonomy.org/uri/bibtexkey/conf/swb/KashyapCDSMLSHH07/d...
Document Document InCollection 6 more
Matthias Samwald Raymond Hookway Iván Herman 7 more
2007

tt0325710-jhendler
http://www.cs.umd.edu/~hendler/2003/foaf.rdf#tt0325710-jhendler
Review
This movie was rated 3 stars on a 4 star scale. That has been rounded to a 7.5 on a 10 point scale.
Jim Hendler
2006-05-18

Dave Beckett's RDF Resource Guide
http://planetrdf.com/guide/
channel Document Document
A comprehensive guide to resources about RDF.
Dave Beckett
2005-09-23

Do You See What I Mean?
http://bibsonomy.org/uri/bibtexkey/journals/cga/DukeBDH05/dblp
Document Article Document 4 more
Iván Herman Ken W. Brodlie Iván Herman 2 more
2005

JSViz Timeline RSS

previous next Results 1 - 5 of 749

Fig. 1. User interface displaying "objects `foaf:made` by objects that `timbl:i foaf:knows`, sorted by `dc:date`" (Query 7). The interface consists of three main sections: i) the current result set in the main content area, ii) the current query in the top part and iii) the prospective choices on the left, divided into datatype properties and object properties to reflect the different operations possible on each.

3 Search and Navigation Operations

In the following we introduce our query operations and a grammar describing how to compose complex queries from atomic operations, and present a formalisation of query results using trees.

3.1 Query Operations

– **Keyword Search** A search session can start with the user specifying 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, ie. the fact that the same resource might be annotated using different spellings or different languages.

- **Object Navigation** The object navigation operation is similar to following a hypertext link in a web browser. The user either starts with a set of results or a single result, and clicks on a node to bring it into focus. The operation yields always a result set with a single element.
- **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 result sets.
- **Path Traversal** Rather than arriving at a single result by performing a object navigation operation, users are also able to navigate 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, ie. they follow a certain link, either from a single result or a set of results.
- **Projection** For views which display individual values of datatype properties (such as the table view), our framework includes a projection operation to select only a number of datatype properties for display.
- **Sorting** Users are often required to sort the result set according to specific datatype property values. In our model, users can select one or more sorting criteria which can be applied to the current result set.

Users start a query building process via specifying a keyword or a URI of the object to bring into focus. The Extended Backus-Naur Form grammar in Figure 2 describes how the individual operations can be combined (via interactions with the user interface) to form complex queries.

```
<query>           ::= <init> { <refine> | <modify> } ;
<init>            ::= keyword search | object navigation ;
<refine>          ::= <facet> | path traversal ;
<facet>           ::= datatype facet | object facet;
<modify>          ::= project | sort ;
keyword search    ::= specify keyword ;
object navigation ::= specify object focus ;
datatype facet    ::= restrict result by P_D, L facet ;
object facet      ::= restrict result by P_O, U facet ;
path traversal    ::= traverse path P_O ;
project           ::= add P_D to projection criteria ;
sort              ::= sort results according to P_D ;
```

Fig. 2. EBNF grammar describing queries. Terminals describe end user actions.

3.2 Result Trees

Iterative application of the restriction and navigation operations leads to a set of focus nodes R. Using one result set is sufficient for keyword searches, object navigation (specifying a single-element result set), and faceted browsing (incrementally reducing the size of the result set). The path traversal (or set-based navigation) operation is different: often, users are interested in the objects on the navigation path that led them to the current result set. Thus, the system adds a new result set R_i whenever the user performes a path traversal operation. The result of multiple path traversal operations are result sets $R_0 \ldots R_n$ where n is the number of path traversal operations in a query. Users are able to select result sets $R_0 \ldots R_n$ for display; R_n is the result set displayed as default.

Consider, for example, the query "objects `foaf:made` by objects that `timbl:i` `foaf:knows`" (Query 6). That query yields three result sets R_0, R_1, R_2. We assume that the query was constructed in the following way: the user teleports to Tim Berners-Lee $R_0 =$ `timbl:i`, from there performs a path traversal along the `foaf:knows` property $R_1 =$ people that Tim knows, and from there again perform a path traversal along the `foaf:made` property yielding $R_2 =$ things made by people Tim knows. When inspecting the results, users might be interested not only in the things made by Tim's acquaintances, but also in retaining the connection between the things and the person who made them. To this end, we incorporate the notion of result trees, ie. objects connected to each other based on the path traversal steps performed. An example result tree is shown in Figure 3 below.

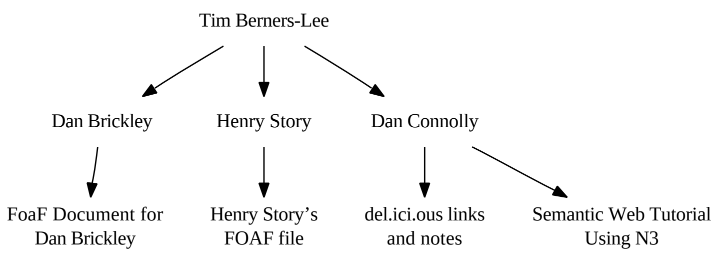

Tim Berners-Lee
Dan Brickley
Henry Story
Dan Connolly
FoaF Document for Dan Brickley
Henry Story's FOAF file
del.ici.ous links and notes
Semantic Web Tutorial Using N3

Fig. 3. Partial result tree for query "objects `foaf:made` by objects that `timbl:i` `foaf:knows`" (Query 6). Labels displayed instead of URIs for clarity.

4 Architecture and Implementation

To verify our ideas, we implemented a prototype system as a Java web application. We present first the architecture, describe our indexing and query processing component, explain how we generate a set of prospective choices for the current result set, and finally describe the rendering pipeline.

4.1 System Architecture

The architecture is based on the Model-View-Controller (MVC) paradigm. The Controller, implemented as servlet, receives queries from the users and retrieves the topical subgraph (Section 4.2) from the database – we use top-k processing over specialised index structures described in Section 4.3. The Model classes parse the statements comprising the topical subgraph for the query into Java objects and generate the set of prospective choices (Section 4.4). Finally, in the View, the results are rendered and returned to the web browser (Section 4.5). Optionally the user can request the results in a format suitable for import in external services or applications.

4.2 Topical Subgraphs

We use the notion of a "topical subgraph", which contains all information required to firstly display the results tree and secondly calculate possible next steps for navigation. Figure 4 depicts a topical subgraph. To retrieve the topical subgraph we first calculate the sets of focus nodes: the nodes directly matching the query criteria. Then, we expand the sets by following outgoing links up to a specified limit ϵ. The parameter ϵ denotes how much information in the neighbourhood of the focus nodes should be returned as input to the subsequent processing steps. In our current implementation we use $\epsilon = 2$, however, applications might require larger portions of the graph to operate. To be able to track the sources of a given piece of data, the topical subgraph refers to a directed labeled graph of data with context [4] derived for a particular query.

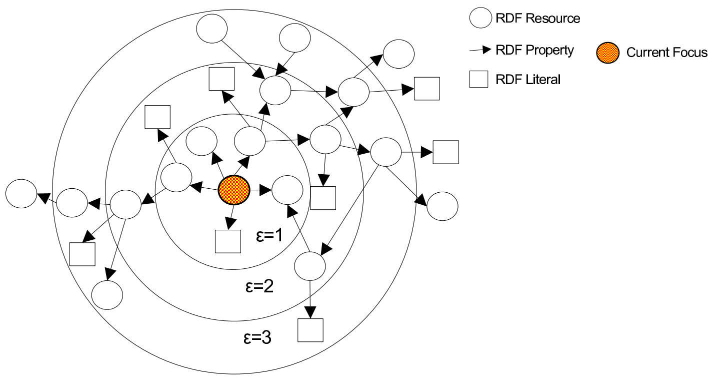


Fig. 4. Topical subgraph for one focus node with $\epsilon = 1, 2, 3$.

Please observe that our notion of topical subgraph does not contain in-links 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 out-links 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 [7], a link is inferred in both directions.

4.3 Indexing and Query Processing

The query processing component performs top-k processing of the queries, based on a set of rankings for all identifiers in the system. Top-k processing – which is not sufficiently supported in current RDF query processors – is a crucial feature for the application since the intermediate result sizes become large, leading to performance degradation, and a simple cut-off of unranked identifiers leads to suboptimal results. Ranking is out of scope for this paper, however, one can assume a simple frequency-based ranking where the rank of an identifier depends on how often the identifier occurs.

We devise a set of index structures to match the navigation primitives offered to the user. Conceptually, our index structures match the <key, posting list> structure known from Information Retrieval systems. The current prototype utilises the following indices:

– Statement Index (<subject, poc list>): store a list of predicate/object/context tuples (poc) per subject. This index is used for topical subgraph lookups where $\epsilon = 1$.
– Path Index (<subject, path list>): store the topical subgraph with $\epsilon = 2$ per subject. This index is used for topical subgraph lookups where $\epsilon = 2$.
– Text Index (<term, subject list>): store a list of subjects per term. This inverted index is used for keyword lookups, intersecting the postings list in case of multiple search terms.
– Facet Index (<po, subject list>): store a list of subjects per facet (predicate/object pair). The index is used for facet restrictions.
– Out-link Index (<sp, object list>): store a list of objects per subject/predicate (sp) pair. The index is used for the path traversal operation.

The Statement Index in combination with the Path Index is used for topical subgraph lookup. In case there is no Path Index available, or subgraphs with $\epsilon > 2$ are requested, the query processor computes the joins between Statement Index and Path Index in a breadth-first manner.

The query processing is carried out as follows: execute each navigation operation using the respective index, sort the posting lists according to the global ranks, intersect the posting lists (starting with the smaller one), and look up the topical subgraph for the resulting focus nodes. During query processing, each navigation primitive is applied to an index, which returns a set of focus nodes, which are in turn again used as input for the next operation. Lastly, the topical subgraph for the final result set is retrieved, and returned as set of statements. In case of path traversals, the topical subgraphs for the multiple result sets are retrieved, and information to link together the objects on the results path is added. We use the sets of statements abstractions rather than storing objects directly to be able to optionally plug in an RDF store as the back-end, or perform lookups on live RDF sources.

The Model component converts the information in the topical subgraphs to Java objects that other components can conveniently process the data.

4.4 Computing Prospective Choices

The users should be able to refine their queries relative to the current result set (now encoded in Java objects). The system computes prospective choices (possible facets and path traversals) from the current result set. Similarly to result sets, we rank the properties and also rank the values and objects that are part of a facet based on their global rank.

4.5 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 geographic coordinates) or a timeline view (for objects with associated date) can be selected. In case users performed path traversal operations, they can optionally select a graph view which renders the result tree in a node-link diagram.

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 finally render the 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 offers to generate certain files in matching data formats for subsequent processing by the user via software programs. For example, geographic coordinates can be exported to KML[10] or objects with associated dates to iCalendar format (RFC 2445).

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 plugins are becoming important. The system currently allows to export objects containing RDF literals

[10] Keyhole Markup Language, `http://www.opengeospatial.org/standards/kml/`

of type `xsd:date` in RSS and Timeline[11] formats, `geo:lat` and `geo:long` in KML, and result trees in JSViz[12]. We provide rendering views of these formats in Timeline, Google Maps, and JSViz widgets directly in the user interface.

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. We tested the system on a number of datasets: the Mondial database[13] consisting of information about countries, an RDF version of CrunchBase[14] containing information about technology startups, and an RDF web crawl, seven hops from a seed URI[15], containing information mainly about people. We first provide anecdotal evidence of the utility of our system, and then present measurements evaluating the performance of query processing and view rendering.

5.1 Iterative Design and Continuous Feedback

We initiated the design process using the Mondial dataset and several queries (e.g. "islands in calabria", "gdp of countries bordering italy"). We asked in total ten participants to evaluate early versions of the interface design based on several user tasks and a questionnaire. One session took around 20 minutes; we asked users to interact with the system immediately without a training or introduction phase since visitors to the web site would not receive training either.

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 to gain insight into what the users would expect from the system. The results were mixed: some users picked up quickly the conceptual model behind the user interface and were able to complete all queries, while the majority were able to retrieve the right answers only for about half of the queries.

The suggestions and comments of the first round of evaluations were taken into account for subsequent versions of the interface, 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 the changes made were actually benefitting the users (which the small study confirmed).

Finally, we performed a series of user tests with the current version of the user interface on the web dataset with five participants. All five participants were able to find the correct answers to queries over the web dataset ("find

[11] `http://simile.mit.edu/timeline/`
[12] JavaScript graph visualisation, `http://www.jsviz.org/`
[13] `http://www.dbis.informatik.uni-goettingen.de/Mondial/`
[14] `http://cb.semsol.org/`
[15] `http://www.w3.org/People/Berners-Lee/card`

`foaf:Person` X", "find `foaf:Person`s that X `foaf:knows`", "find objects that X `foaf:made`"). While the initial user studies have proved very valuable during the design phase, we plan to conduct larger, more rigorous user studies.

5.2 Performance Evaluation

For the performance evaluation we use the queries from Table 1 as templates; we inserted into the query templates the names and URIs of six selected people for which a sizable amount of information is available. We measure separately the time elapsed in query processing and rendering the view on the server; data transmission time and rendering performance on the client are independent of our method and thus not covered in the measurements. The measurements were carried out on a machine with a 2.2GHz AMD Opteron CPU, 4GB of main memory and a 160GB hard disk. The servlet container was Tomcat 5.5 in combination with a Sun Java 1.6.0. Figure 5 shows the average performance of query processing and view rendering.

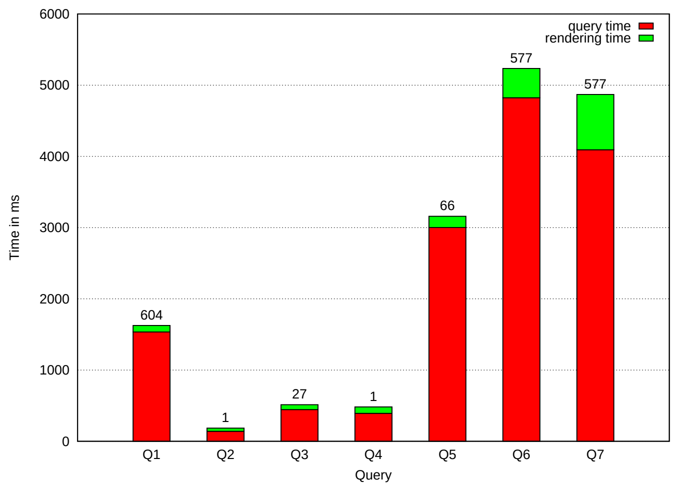


Fig. 5. Average query processing and view rendering performance for queries from Table 1. The numbers above the bars denote the average result sizes for the final layer in the result tree.

The results indicate that the majority of time spent is on query processing; we indeed removed a previous bottleneck due to the use of a method which renders the XML in-memory rather than stream-based.

Queries with many path traversal steps (Q5 - Q7) are the most expensive ones due to the join method used. We expect major performance improvements by replacing the current hash join implementation with an index nested loops join algorithm.

6 Related Work

NLMenu [16] is an early system advocating the use of multi-step query construction based on menus. Faceted browsing [18], 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 [15] provides complex query and aggregation operations, however, operates over relational data and thus requires a priori knowledge about the schema used.

A number of systems exist that operate over graph-structured data, which range from quite basic browsing facilities (e.g. Disco[16] allows only object navigation) to systems allowing complex constructs such as negation [13] or nested facets [17]. Table 2 provides a feature-set comparison of related systems.

System	Keywords	Facets	Navigation	Results	Ranking	Configuration	Data sources
Flamenco [18]	x	x	-	set	-	manual	one
Magnet [14]	x	x	-	set	-	schema	one
MuseumFinland [9]	x	x	-	set	-	rules	several
GRQL [1]	-	o	x	set	-	schema	one
/facet [6]	x	x	-	set	group-by	auto	one
BrowseRDF [13]	x	x	-	set	facets	auto	one
ESTER [2]	x	x	-	set	top-k	auto	one
TcruziKB [12]	x	-	x	set	-	schema	several
Humboldt [11]	x	x	x	sets	-	auto	one
Parallax [8]	x	x	x	sets	-	auto	several
VisiNav	x	x	x	trees	facets/top-k	auto	web

Table 2. Feature comparison of related systems.

In general, system designs have to balance a trade-off between ease of use and query expressiveness. Our system uses the combined set of query primitives offered by a range of established browsing and navigation systems for graph-structured data, providing evidence that the selection of features in our system represent a consensus in the community. This suggests that a sizeable user community is able to understand the operations.

The systems most closely related to our system in terms of features are GRQL [1], Humboldt [11] and Parallax [8]. 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,

[16] `http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/`

since keywords are independent of any schema. Rather than allowing arbitrary facets, GRQL allows to restrict based on the `rdf:type` predicate. GRQL is, to our knowledge, the earliest system that provides functionality to perform set-based navigation. Parallax [8] 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 [3]. We describe our abstract interface – the information exchange between users and system – in terms of user operations, formalised in EBNF, and result trees. Our concrete interface – the look and feel – is implemented using a multi-layered rendering pipeline spanning server (RDF, queries and XML) and client (XSLT and CSS).

7 Conclusion

Established efforts such as the Linked Open Data[17] already provide large corpora of structured data in various domains, and more efforts are underway[18]. Projects such as FOAF and SIOC provide vocabularies and best practices, enabling both individuals and organisations to publish high-quality data on the web. More structured and interlinked data, in combination with a search and navigation system as presented in this paper, represents an opportunity to bring novel and powerful ways for interacting with data to the web. To this end, we have demonstrated VisiNav, a system based on a formal interaction model that empowers users to search, browse, and navigate a large, domain-independent dataset collectively created by a global user community. Future work includes further improvements of the usability of the system; in particular we would like to enhance the query response times and streamline the user experience based on insights obtained through additional user testing.

Acknowledgements

I gratefully acknowledge the comments of the participants of the user studies, and discussions with Brian Davis, Renaud Delbru, Armin Haller, Aidan Hogan, Sheila Kinsella, Axel Polleres, Rene Schubotz, and Jürgen Umbrich. This work has been supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

[17] `http://linkeddata.org`
[18] e.g. `http://openflydata.org/` and `http://www.w3.org/2001/sw/hcls/`

References

1. N. Athanasis, V. Christophides, and D. 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, Nov 2004.
2. H. Bast, A. Chitea, F. Suchanek, and I. 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.
3. S. S. de Moura and D. 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.
4. A. Harth and S. Decker. Optimized index structures for querying rdf from the web. In *3rd Latin American Web Congress*, pages 71–80, 2005.
5. M. Henzinger. Search Technologies for the Internet. *Science*, 317(5837):468–471, 2007.
6. M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. In *5th International Semantic Web Conference*, pages 272–285, Nov 2006.
7. A. Hogan, A. Harth, and A. Polleres. Saor: Authoritative reasoning for the web. In *3rd Asian Semantic Web Conference*, pages 76–90, 2008.
8. D. F. Huynh and D. 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.
9. E. Hyvnen, E. Mkel, M. Salminen, A. Valo, K. Viljanen, S. Saarela, M. Junnila, and S. Kettula. Museumfinland – finnish museums on the semantic web. *Journal of Web Semantics*, 3(2):25, 2005.
10. E. Kaufmann and A. Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? In *6th International Semantic Web Conference*, pages 281–294, Nov 2007.
11. G. Kobilarov and I. Dickinson. Humboldt: Exploring linked data. In *Linked Data on the Web Workshop*, 2008.
12. P. Mendes, B. McKnight, A. Sheth, and J. Kissinger. Tcruzikb: Enabling complex queries for genomic data exploration. *IEEE International Conference on Semantic Computing*, pages 432–439, Aug. 2008.
13. E. Oren, R. Delbru, and S. Decker. Extending faceted navigation for rdf data. In *5th International Semantic Web Conference*, Nov 2006.
14. V. Sinha and D. R. Karger. Magnet: supporting navigation in semistructured data environments. In *ACM SIGMOD International Conference on Management of Data*, pages 97–106, 2005.
15. C. Stolte, D. Tang, and P. 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.
16. C. W. Thompson, K. M. Ross, H. R. Tennant, and R. M. Saenz. Building usable menu-based natural language interfaces to databases. In *9th International Conference on Very Large Data Bases*, pages 43–55, 1983.
17. M. Tvarozek and M. Bielikova. Adaptive faceted browser for navigation in open information spaces. In *16th International Conference on World Wide Web*, pages 1311–1312, 2007.
18. K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *SIGCHI Conference on Human factors in Computing Systems*, pages 401–408, 2003.