Extending faceted navigation for RDF data

Eyal Oren, Renaud Delbru, and Stefan Decker

DERI Galway, Ireland
firstname.lastname@deri.org

Abstract. Data on the Semantic Web is semi-structured and does not follow one fixed schema. Faceted browsing [23] is a natural technique for navigating such data, partitioning the information space into orthogonal conceptual dimensions. Current faceted interfaces are manually constructed and have limited query expressiveness. We develop an expressive faceted interface for semi-structured data and formally show the improvement over existing interfaces. Secondly, we develop metrics for automatic ranking of facet quality, bypassing the need for manual construction of the interface. We develop a prototype for faceted navigation of arbitrary RDF data. Experimental evaluation shows improved usability over current interfaces.

1 Introduction

As Semantic Web data emerges, techniques for browsing and navigating this data are necessary. Semantic Web data, expressed in RDF[1], is typically very large, highly interconnected, and heterogeneous without following one fixed schema [1]. Any technique for navigating such datasets should therefore be scalable; should support graph-based navigation; and should be generic, not depend on a fixed schema, and allow exploration of the dataset without a-priori knowledge of its structure.

We identified four existing interface types for navigating RDF data: (1) keyword search, e.g. Swoogle[2], (2) explicit queries, e.g. Sesame[3], (3) graph visualisation, e.g. IsaViz[4] , and (4) faceted browsing [12, 19, 23]. None of these fulfill the above requirements: *keyword search* suffices for simple information lookup, but not for higher search activities such as learning and investigating [13]; writing *explicit queries* is difficult and requires schema knowledge; *graph visualisation* does not scale to large datasets [7]; and existing *faceted interfaces* are manually constructed and domain-dependent, and do not fully support graph-based navigation.

In this paper we 1. improve faceted browsing techniques for RDF data, 2. develop a technique for automatic facet ranking, 3. develop a formal model of faceted browsing, allowing for precise comparison of interfaces, and 4. support our conclusions with a formal analysis and an experimental evaluation.

[1] http://www.w3.org/RDF/
[2] http://swoogle.umbc.edu/
[3] http://www.openrdf.org/
[4] http://www.w3.org/2001/11/IsaViz/

2 Faceted browsing

An exploratory interface allows users to find information without a-priori knowledge of its schema. Especially when the structure or schema of the data is unknown, an exploration technique is necessary [21]. Faceted browsing [23] is an exploration technique for structured datasets based on the facet theory [17].

In faceted browsing the information space is partitioned using orthogonal conceptual dimensions of the data. These dimensions are called facets and represent important characteristics of the information elements. Each facet has multiple restriction values and the user selects a restriction value to constrain relevant items in the information space. The facet theory can be directly mapped to navigation in semi-structured RDF data: information elements are RDF subjects, facets are RDF predicates and restriction-values are RDF objects.

A collection of art works can for example have facets such as type of work, time periods, artist names and geographical locations. Users are able to constrain each facet to a restriction value, such as "created in the 20th century", to limit the visible collection to a subset. Step by step other restrictions can be applied to further constrain the information space.

A faceted interface has several advantages over keyword search or explicit queries: it allows exploration of an unknown dataset since the system suggests restriction values at each step; it is a visual interface, removing the need to write explicit queries; and it prevents dead-end queries, by only offering restriction values that do not lead to empty results.

3 A faceted interface for RDF data

In this section we introduce our faceted interface for arbitrary RDF data, explain its functionality, and formally describe its expressive power. The formal treatment allows us to clearly show the improvement in expressive power over existing interfaces, which we will do in Sec. 5.1.

3.1 Overview

A screenshot of our BrowseRDF prototype[5], automatically generated for arbitrary data, is shown in Fig. 1. This particular screenshot shows the FBI's most wanted fugitives[6]. These people are described by various properties, such as their weight, their eye-color, and the crime that they are wanted for. These properties form the facets of the dataset, and are shown on the left-hand side of the screenshot.

Users can browse the dataset by constraining one or several of these facets. At the top-center of the screenshot we see that the user constrained the dataset to all fugitives that weigh 150 pounds, and in the middle of the interface we see

[5] available at `http://browserdf.org`.

[6] `http://sp11.stanford.edu/kbs/fbi.zip`

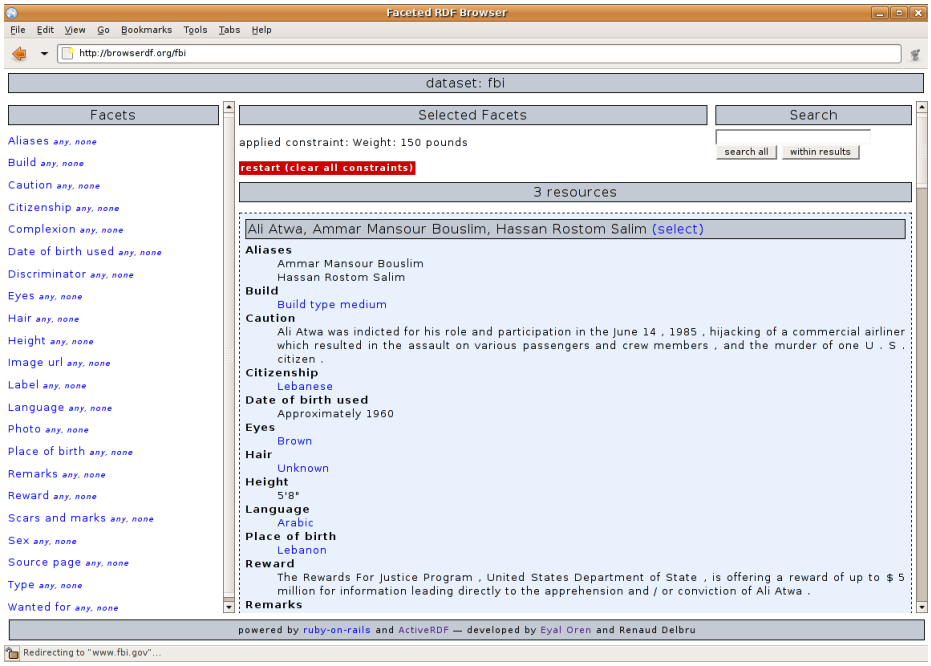


Fig. 1: Faceted browsing prototype

that three people have been found conforming to that constraint. These people are shown (we see only the first), with all information known about them (their alias, their nationality, their eye-color, and so forth). The user could now apply additional constraints, by selecting another facet (such as citizenship) to see only the fugitives that weigh 150 pounds and speak French.

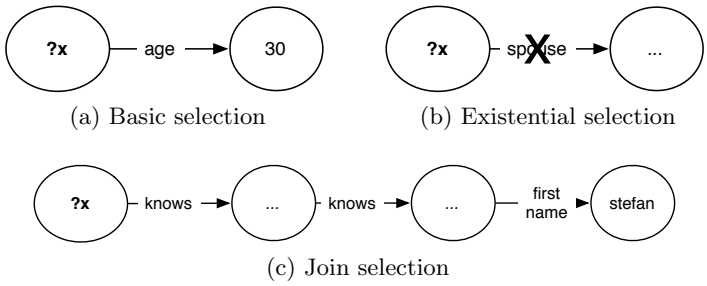
3.2 Functionality

The goal of faceted browsing is to restrict the search space to a set of relevant resources (in the above example, a set of fugitives). Faceted browsing is a visual query paradigm [15, 9]: the user constructs a selection query by browsing and adding constraints; each step in the interface constitutes a step in the query construction, and the user sees intermediate results and possible future steps while constructing the query.

We now describe the functionality of our interface more systematically, by describing the various operators that users can use. Each operator results in a constraint on the dataset; operators can be combined to further restrict the results to the set of interest. Each operator returns a subset of the information space; an exact definition is given in Sec. 3.3.

Basic selection The basic selection is the most simple operator. It selects nodes that have a direct restriction value. The basic selection allows for example to

"find all resources of thirty-year-olds", as shown in Fig. 2a. It selects all nodes that have an outgoing edge, labelled "age", that leads to the node "30". In the interface, the user first selects a facet (on the left-hand side) and then chooses a constraining restriction value.



(a) Basic selection (b) Existential selection

(c) Join selection

Fig. 2: Selection operators

Existential selection There might be cases when one is interested in the existence of a property, but not in its exact value, or one may be interested simply in the non-existence of some property. For example, we can ask for "all resources without a spouse" (all unmarried people), as shown in Fig. 2b. In the interface, instead of selecting a restriction value for the facet, the user clicks on "any" or "none" (on the left-hand side, after the facet name).

Join selection Given that RDF data forms a graph, we often want to select some resources based on the properties of the nodes that they are connected to. For example, we are looking for "all resources who know somebody, who in turn knows somebody named Stefan", as shown in Fig. 2c. Using the join-operator recursively, we can create a path of arbitrary length[7], where joins can occur on arbitrary predicates. In the interface, the user first selects a facet (on the left-hand side), and then in turn restricts the facet of that resource. In the given example, the user would first click on "knows", click again on "knows" and then click on "first-name", and only then select the value "Stefan".

Intersection When we define two or more selections, these are evaluated in conjunction. For example, we can use the three previous examples to restrict the resources to "all unmarried thirty-years old who know some other resource that knows a resource named Stefan Decker", as shown in Fig. 3. In the interface, all constraints are automatically intersected.

Inverse selection All operators have an inverse version that selects resources by their inverse properties. For example, imagine a dataset that specifies companies and their employees (through the "employs" predicate). When we select

[7] The path can have arbitrary length, but the length must be specified; we, or any RDF store [1], do not support regular expression queries, as in e.g. GraphLog [3].

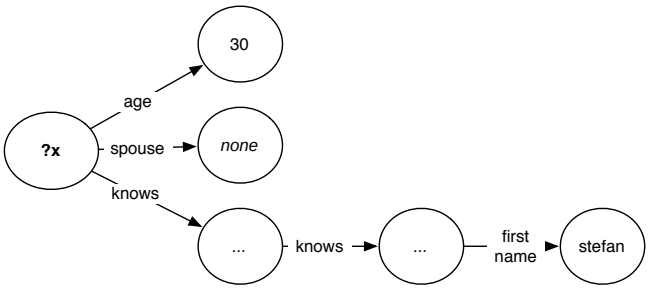


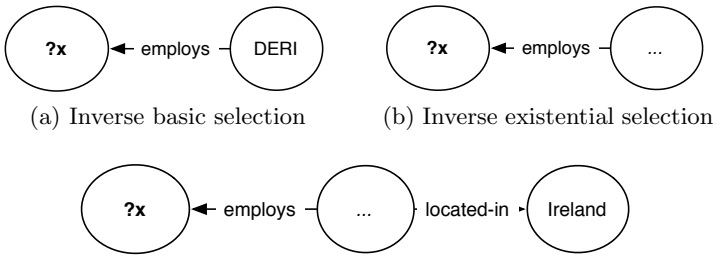
Fig. 3: Intersection operator

a person, we might be interested in his employer, but this data is not directly available. Instead, we have to follow the inverse property: we have to look for those companies who employ this person. In the user interface, after all regular facets, the user sees all inverse facets. The inverse versions of the operators are:

Inverse basic selection For example, when the graph only contains statements such as "DERI employs ?x", we can ask for "all resources employed by DERI", as shown in Fig. 4a.

Inverse existential selection We could also find all employed people, regardless of their employer, as shown in Fig. 4b.

Inverse join selection The inverse join selection allows us to find "all resources employed by a resource located in Ireland", as shown in Fig. 4c.



(a) Inverse basic selection

(b) Inverse existential selection

(c) Inverse join selection

Fig. 4: Inverse operators

We can merge the last example with the intersection example to find "all unmarried thirty-year-olds who know somebody –working in Ireland– who knows Stefan", as shown in Fig. 5.

3.3 Expressiveness

In this section we formalise our operators as functions on an RDF graph. The formalisation precisely defines the possibilities of our faceted interface, and allows us to compare our approach to existing approaches (which we will do in Sect. 5.1).

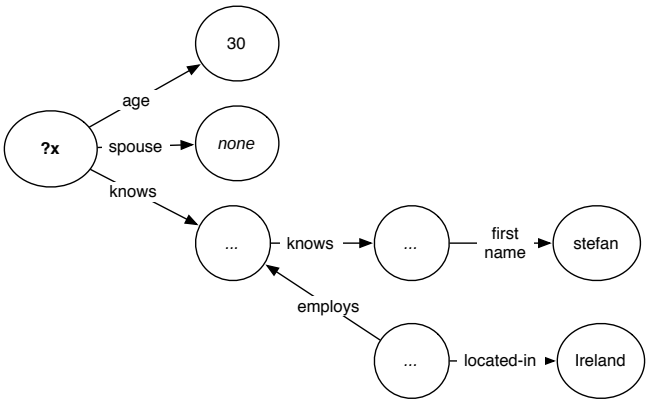


Fig. 5: Full selection

First, we define the graph on which the operations are applied. Our notion of an RDF graph differs from the standard one[8]: we only consider the explicit statements in an RDF document and do not infer additional information as mandated by the RDF semantics. The latter is not a "violation" of the semantics, because we assume the RDF store to perform the necessary inferences already; we regard a given RDF graph simply as the graph itself.

Definition 1 (RDF Graph). *An RDF graph G is defined as $G = (V, E, L, l)$ where V is the set of vertices (subjects and objects), E is the set of edges (predicates), L is the set of labels, $l : E \rightarrow label$ is the labelling function for predicates and with V and E disjoint*[9]*. The projections, $source : E \rightarrow V$ and $target : E \rightarrow V$, return the source and target nodes of edges.*

Table 1 gives a formal definition for each of the earlier operators. The operators describe faceted browsing in terms of set manipulations: each operator is a function, taking some constraint as input and returning a subset of the resources that conform to that constraint. The definition is not intended as a new query language, but to demonstrate the relation between the interface actions in the faceted browser and the selection queries on the RDF graph. In our prototype, each user interface action is translated into the corresponding SPARQL[10] query and executed on the RDF store.

The primitive operators are the basic and existential selection, and their inverse forms. The basic selection returns resources with a certain property value. The existential selection returns resources that have a certain property, irrespective of its value. These primitives can be combined using the join and the intersection operator. The join returns resources with a property, whose value is part of the joint set. The intersection combines constraints conjunctively. The

[8] `http://www.w3.org/TR/rdf-mt/`

[9] In RDF E and V are not necessarily disjoint but we restrict ourselves to graphs in which they actually are.

[10] `http://www.w3.org/TR/rdf-sparql-query/`

join and intersection operators have closure: they have sets as input and output and can thus be recursively composed. As an example, all thirty-year-olds without a spouse would be selected by: $intersect(select(age, 30), not(spouse))$.

operator	definition
basic selection	$select(l, v') = \{v \in V \mid \forall e \in E : label(e) = l, source(e) = v \; target(e) = v'\}$
inv. basic selection	$select^{-}(l, v') = \{v \in V \mid \forall e \in E : label(e) = l, source(e) = v' \; target(e) = v\}$
existential	$exists(l) = \{v \in V \mid \forall e \in E : label(e) = l, source(e) = v\}$
inv. existential	$exists^{-}(l) = \{v \in V \mid \forall e \in E : label(e) = l, target(e) = v\}$
not-existential	$not(l) = V - exists(l)$
inv. not-existential	$not^{-}(l) = V - exists^{-}(l)$
join	$join(l, V') = \{v \in V \mid \forall e \in E : label(e) = l, source(e) = v \; target(e) \in V'\}$
inv. join	$join^{-}(l, V') = \{v \in V \mid \forall e \in E : label(e) = l, source(e) \in V' \; target(e) = v\}$
intersection	$intersect(V', V'') = V' \cap V''$

Table 1: Operator definitions

4 Automatic facet ranking

By applying the previous definitions a faceted browser for arbitrary data can be built. But if the dataset is very large, the number of facets will typically also be large (especially with heterogeneous data) and users will not be able to navigate through the data efficiently. Therefore, we need an automated technique to determine which facets are more useful and more important than others. In this section, we develop such a technique.

To automatically construct facets, we need to understand what characteristics constitute a suitable facet. A facet should only represent one important characteristic of the classified entity [17], which in our context is given by its predicates. We need to find therefore, among all predicates, those that best represent the dataset (the best descriptors), and those that most efficiently navigate the dataset (the best navigators).

In this section, we introduce facet ranking metrics. We first analyse what constitutes suitable descriptors and suitable navigators, and then derive metrics to compute the suitability of a facet in an dataset. We demonstrate these metrics on a sample dataset.

4.1 Descriptors

What are suitable descriptors of a data set? For example, for most people the "page number" of articles is not very useful: we do not remember papers by their page-number. According to Ranganathan [17], intuitive facets describe a property that is either temporal (e.g. year-of-publication, date-of-birth), spatial

(conference-location, place-of-birth), personal (author, friend), material (topic, color) or energetic (activity, action).

Ranganathan's theory could help us to automatically determine intuitive facets: we could say that facets belonging to either of these categories are likely to be intuitive for most people, while facets that do not are likely to be unintuitive. However, we usually lack background knowledge about the kind of facet we are dealing with since this metadata is usually not specified in datasets. Ontologies, containing such background knowledge, might be used, but that is outside the scope of this paper.

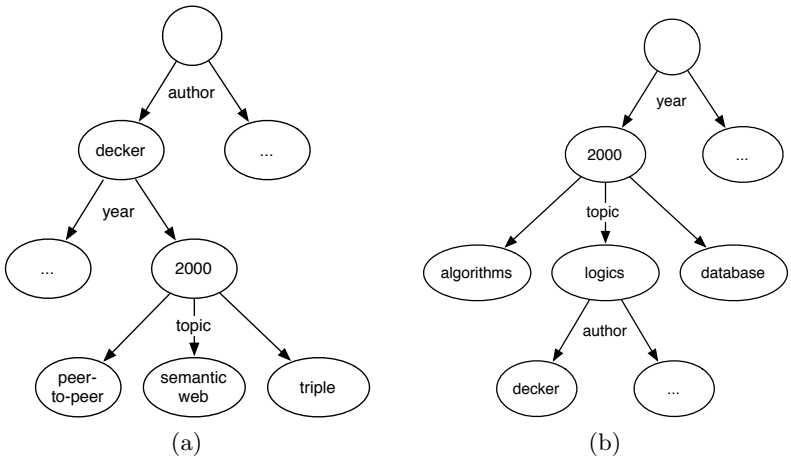


Fig. 6: Faceted browsing as decision tree traversal

4.2 Navigators

A suitable facet allows efficient navigation through the dataset. Faceted browsing can be considered as simultaneously constructing and traversing a decision tree whose branches represent predicates and whose nodes represent restriction values. For example, Fig. 6a shows a tree for browsing a collection of publications by first constraining the author, then the year and finally the topic. Since the facets are orthogonal they can be applied in any order: one can also first constrain the year and topic of publication, and only then select some author, as shown in Fig. 6b.

A path in the tree represents a set of constraints that select the resources of interest. The tree is constructed dynamically, e.g. the available restriction values for "topic" are different in both trees: Fig. 6b shows all topics from publications in 2000, but Fig. 6a shows only Stefan Decker's topics.

4.3 Facet metrics

Regarding faceted browsing as constructing and traversing a decision tree helps to select and use those facets that allow the most efficient navigation in the tree.

In this section we define this "navigation quality" of a facet in terms of three measurable properties (metrics) of the dataset. All metrics range from [0..1]; we combine them into a final score through (weighted) multiplication. We scale the fontsize of facets by their rank, allowing highlighting without disturbing the alphabetical order[11].

The metrics need to be recomputed at each step of the decision tree, since the information space changes (shrinks) at each decision step. We give examples for each metric, using a sample[12] of the Citeseer[13] dataset for scientific publications and citations, but these example metrics only apply on the top-level (at the root of the decision-tree).

We would like to rank facets not only on their navigational value, but also on their descriptive value, but we have not yet found a way to do so. As a result, the metrics are only an indication of usefulness; badly ranked facets should not disappear completely, since even when inefficient they could still be intuitive.

Predicate balance Tree navigation is most efficient when the tree is well-balanced because each branching decision optimises the decision power [20, p. 543]. We therefore use the balance of a predicate to indicate its navigation efficiency.

For example, we see in Table 2a that *institution* and *label* are well balanced, but publication *type* is not, with a (rounded) normalised balance of 0. Table 2b shows in more detail why the type of publications is unbalanced: among the 13 different types of publications, only three occur frequently (proceeding papers, miscellaneous and journal articles); the rest of the publication types occur only rarely. Being a relatively unbalanced predicate, constraining the publication type would not be the most economic decision.

We compute the predicate balance $balance(p)$ as the normalised[14] variance of the number of subjects $n_s(p, o_i) = |select(p, o_i)|$ for each object value o_i of p:

$$balance(p) = 1 - \frac{var\big[n_s(p, o_i), \cdots, n_s(p, o_n)\big]}{1 + var\big[n_s(p, o_i), \cdots, n_s(p, o_n)\big]}$$

Object cardinality A suitable predicate has a limited (but higher than one) amount of object values to choose from. Otherwise, when there are too many choices, the options are difficult to display and the choice might confuse the user.

For example, as shown in Table 2c, the predicate *type* is very usable since it has only 13 object values to choose from, but the predicate *author* or *title* would not be directly usable, since they have around 4000 different values. One solution for reducing the object cardinality is object clustering [11, 22], but that is outside the scope of this paper.

[11] font scaling has not yet been implemented.

[12] `http://www.csd.abdn.ac.uk/~ggrimnes/swdataset.php`

[13] `http://citeseer.ist.psu.edu/`

[14] the given normalisation is not linear: the balance becomes decreases too quickly compared to the variance; we have not yet found a better normalisation.

We compute the object cardinality metric $card(p)$ as the number of different objects (restriction values) $n_o(p)$ for the predicate p and normalise it using the a function based on the Gaussian density. For displaying and usability purposes the number of different options should be approximately between two and twenty, which can be regulated through the μ and σ parameters.

$$card(p) = \begin{cases} 0 & \text{if } n_o(p) \leq 1 \\ \exp^{-\frac{(n_o(p)-\mu)^2}{2\sigma^2}} & \text{otherwise} \end{cases}$$

Predicate frequency A suitable predicate occurs frequently inside the collection: the more distinct resources covered by the predicate, the more useful it is in dividing the information space [4]. If a predicate occurs infrequently, selecting a restriction value for that predicate would only affect a small subset of the resources.

For example, in Table 2d we see that all publications have a type, author, title, and URL, but that most do not have a volume, number, or journal.

We compute the predicate frequency $freq(p)$ as the number of subjects $n_s(p) = |exists(p)|$ in the dataset for which the predicate p has been defined, and normalise it as a fraction of the total number of resources n_s: $freq(p) = \frac{n_s(p)}{n_s}$.

predicate	balance
institute	1.0
label	1.0
url	1.0
title	0.99
text	0.99
author	0.92
pages	0.87
organization	0.8
⋮	⋮
type	0.00

(a) balance

type	perc.
inproc.	40.78%
misc	28.52%
article	19.44%
techrep.	7.59%
incoll.	2.66%
phd	0.47%
book	0.21%
unpub.	0.19%
msc	0.07%
inbook	0.05%
proc.	0.02%

(b) objects in *type*

predicate	objects
title	4215
url	4211
author	4037
pages	2168
text	1069
booktitle	1010
number	349
address	341
journal	312
editor	284
⋮	⋮
type	13

(c) cardinality

predicate	freq.
type	100%
author	99%
title	99%
url	99%
year	91%
pages	55%
booktitle	37%
text	25%
number	23%
volume	22%
journal	20%
⋮	⋮

(d) frequency

Table 2: Sample metrics in Citeseer dataset

5 Evaluation

We first evaluate our approach formally, by comparing the expressiveness of our interface to existing faceted browsers. We then report on an experimental evaluation.

5.1 Formal evaluation

Several approaches exist for faceted navigation of (semi-)structured data, such as Flamenco [23], mSpace [19], Ontogator [12], Aduna Spectacle[15], Siderean Seamark Navigator[16] and Longwell[17]. Our formal model provides a way to compare their functionality explicitly.

Existing approaches cannot navigate arbitrary datasets: the facets are manually constructed and work only on fixed data structures. Furthermore, they assume data homogeneity, focus on a single type of resource, and represent other resources with one fixed label. One can for example search for publications written by an author with a certain name, but not by an author of a certain age, since authors are always represented by their name.

Table 3 explicitly shows the difference in expressive power, indicating the level of support for each operator. The existing faceted browsers support the basic selection and intersection operators; they also support joins but only with a predefined and fixed join-path, and only on predefined join-predicates. The commercial tools are more polished but have in essence the same functionality. Our interface adds the existential operator, the more flexible join operator and the inverse operators. Together these significantly improve the query expressiveness.

operator	BrowseRDF	Flamenco	mSpace	Ontogator	Spectacle	Seamark
selection	+	+	+	+	+	+
inv. selection	+	−	−	−	−	−
existential	+	−	−	−	−	−
inv. exist.	+	−	−	−	−	−
not-exist.	+	−	−	−	−	−
inv. not-exist.	+	−	−	−	−	−
join	+	±	±	±	±	±
inv. join	+	−	−	−	−	−
intersection	+	+	+	+	+	+

Table 3: Expressiveness of faceted browsing interfaces

Other related work Some non-faceted, domain-independent, browsers for RDF data exist, most notably Noadster [18] and Haystack [16]. Noadster (and its predecessor Topia) focuses on resource presentation and clustering, as opposed to navigation and search, and relies on manual specification of property weights, whereas we automatically compute facet quality. Haystack does not offer faceted browsing, but focuses on data visualisation and resource presentation.

Several approaches exist for generic visual exploration of RDF graphs [6, 5] but none scale for large graphs: OntoViz[18] cannot generate good layouts for more than 10 nodes and IsaViz[19] is ineffective for more than 100 nodes [7].

[15] `http://www.aduna-software.com/products/spectacle/`
[16] `http://www.siderean.com/`
[17] `http://simile.mit.edu/longwell`
[18] `http://protege.stanford.edu/plugins/ontoviz/`
[19] `http://www.w3.org/2001/11/IsaViz/`

Related to our facet ranking approach, a technique for automatic classification of new data under existing facets has been developed [4], but requires a predefined training set of data and facets and only works for textual data; another technique [2], based on lexical dispersion, does not require training but it is also limited to textual data.

5.2 Experimental evaluation

We have performed an experimental evaluation to compare our interface to alternative generic interfaces, namely keyword-search and manual queries.

Prototype The evaluation was performed on our prototype, shown earlier in Fig. 1. The prototype is a web application, accessible with any browser. We use the Ruby on Rails[20] web application framework to construct the web interface. The prototype uses ActiveRDF[21] [14], an object-oriented API for arbitrary RDF data, to abstract the RDF store and translate the interface operators into RDF queries. The abstraction layer of ActiveRDF uses the appropriate query language transparently depending on the RDF datastore. We used the YARS [10] RDF store because its index structure allows it to answer our typical queries quickly.

Methodology Mimicking the setup of Yee *et al.* [23], we evaluated[22] 15 test subjects, ranging in RDF expertise from beginner (8), good (3) to expert (4). None were familiar with the dataset used in the evaluation.

We offered them three interfaces, keyword search (through literals), manual (N3) query construction, and our faceted browser. All interfaces contained the same FBI fugitives data mentioned earlier. To be able to write queries, the test subjects also received the data-schema.

In each interface, they were asked to perform a set of small tasks, such as "find the number of people with brown eyes", or "find the people with Kenyan nationality". In each interface the tasks were similar (so that we could compare in which interface the task would be solved fastest and most correctly) but not exactly the same (to prevent reuse of earlier answers). The questions did not involve the inverse operator as it was not yet implemented at the time. We filmed all subjects and noted the time required for each answer; we set a two minute time-limit per task.

Results Overall, our results confirm earlier results [23]: people overwhelmingly (87%) prefer the faceted interface, finding it useful (93%) and easy-to-use (87%).

As shown in Table 4, on the keyword search, only 16% of the questions were answered correctly, probably because the RDF datastore allows keyword search only for literals. Using the N3 query language, again only 16% of the questions

[20] `http://rubyonrails.org`
[21] `http://activerdf.org`
[22] evaluation details available on `http://m3pe.org/browserdf/evaluation`.

were answered correctly, probably due to unfamiliarity with N3 and the unforgiving nature of queries. In the faceted interface 74% of the questions were answered correctly.

Where correct answers were given, the faceted interface was on average 30% faster than the keyword search in performing similar tasks, and 356% faster than the query interface. Please note that only 10 comparisons could be made due to the low number of correct answers in the keyword and query interfaces.

Questions involving the existential operator took the longest to answer, indicating difficulty understanding that operator, while questions involving the basic selection proved easiest to answer — suggesting that arbitrarily adding query expressiveness might have limited benefit, if users cannot use the added functionality.

	solved	unsolved
keyword	15.55%	84.45%
query	15.55%	84.45%
faceted	74.29%	25.71%

(a) Task solution rate

	keyword	query	faceted
easiest to use	13.33%	0%	86.66%
most flexible	13.33%	26.66%	60%
most dead-ends	53.33%	33.33%	13.33%
most helpful	6.66%	0%	93.33%
preference	6.66%	6.66%	86.66%

(b) Post-test preferences

Table 4: Evaluation results

6 Conclusion

Faceted browsing [23] is a data exploration technique for large datasets. We have shown how this technique can be employed for arbitrary semi-structured content. We have extended the expressiveness of existing faceted browsing techniques and have developed metrics for automatic facet ranking, resulting in an automatically constructed faceted interface for arbitrary semi-structured data. Our faceted navigation has improved query expressiveness over existing approaches and experimental evaluation shows better usability than current interfaces.

Future work Our additional expressiveness does not necessarily result in higher usability; future research is needed to evaluate the practical benefits of our approach against existing work. Concerning the ranking metrics, we performed an initial unpublished evaluation showing that although the search space is divided optimally, the ranking does not always correspond to the intuitive importance people assign to some facets; again, further research is needed.

Acknowledgements This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694. We thank Jos de Bruijn and the anonymous reviewers for valuable comments on a previous version.

References

[1] R. Angles and C. Gutierrez. Querying RDF data from a graph database perspective. In *ESWC*, pp. 346–360. 2005.

[2] P. Anick and S. Tipirneni. Interactive document retrieval using faceted terminological feedback. In *HICSS*. 1999.

[3] M. P. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *PODS*, pp. 404–416. 1990.

[4] W. Dakka, P. Ipeirotis, and K. Wood. Automatic construction of multifaceted browsing interfaces. In *CIKM*. 2005.

[5] C. Fluit, M. Sabou, and F. van Harmelen. Ontology-based information visualization. In [8], pp. 45–58.

[6] C. Fluit, M. Sabou, and F. van Harmelen. Supporting user tasks through visualisation of light-weight ontologies. In S. Staab and R. Studer, (eds.) *Handbook on Ontologies*, pp. 415–434. Springer-Verlag, Berlin, 2004.

[7] F. Frasincar, A. Telea, and G.-J. Houben. Adapting graph visualization techniques for the visualization of RDF data. In [8], pp. 154–171.

[8] V. Geroimenko and C. Chen, (eds.) *Visualizing the Semantic Web*. Springer-Verlag, Berlin, second edn., 2006.

[9] N. Gibbins, S. Harris, A. Dix, and mc schraefel. Applying mspace interfaces to the semantic web. Tech. Rep. 8639, ECS, Southampton, 2004.

[10] A. Harth and S. Decker. Optimized index structures for querying RDF from the web. In *LA-WEB*. 2005.

[11] M. A. Hearst. Clustering versus faceted categories for information exploration. *Comm. of the ACM*, 46(4), 2006.

[12] E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: Combining view- and ontology-based search with semantic browsing. In *Proc. of XML Finland*. 2003.

[13] G. Marchionini. Exploratory search: From finding to understanding. *Comm. of the ACM*, 49(4), 2006.

[14] E. Oren and R. Delbru. ActiveRDF: Object-oriented RDF in Ruby. In *Scripting for Semantic Web (ESWC)*. 2006.

[15] C. Plaisant, B. Shneiderman, K. Doan, and T. Bruns. Interface and data architecture for query preview in networked information systems. *ACM Trans. Inf. Syst.*, 17(3):320–341, 1999.

[16] D. Quan and D. R. Karger. How to make a semantic web browser. In *WWW*. 2004.

[17] S. R. Ranganathan. *Elements of library classification*. Bombay: Asia Publishing House, 1962.

[18] L. Rutledge, J. van Ossenbruggen, and L. Hardman. Making RDF presentable: integrated global and local semantic Web browsing. In *WWW*. 2005.

[19] m. schraefel, M. Wilson, A. Russell, and D. A. Smith. mSpace: Improving information access to multimedia domains with multimodal exploratory search. *Comm. of the ACM*, 49(4), 2006.

[20] R. Sedgewick. *Algorithms in C++*. Addison-Wesley, 1998.

[21] R. W. White, B. Kules, S. M. Drucker, and mc schraefel. Supporting exploratory search. *Comm. of the ACM*, 49(4), 2006.

[22] R. Xu and D. W. II. Survey of clustering algorithms. *IEEE Trans. on Neural Networks*, 16(3):645–678, 2005.

[23] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI*. 2003.