A query refinement model for exploratory semantic search

Dave Reynolds, Ian Dickinson, Dave Grosvenor

HP Laboratories
HPL-2009-167

Keyword(s):
semantic search; RDF; SPARQL; query refinement

Abstract:
We describe a novel approach to exploratory semantic search applied to the problem domain of e-discovery. In this domain, as in many others, we need to support non-expert users in composing queries over a complex and heterogeneous knowledge base. We achieve this by enabling users to initiate a query through a few seed concepts and terms, and then guide them in incrementally refining the query and exploring the dataset without requiring prior knowledge of the ontology. This is achieved through a query refinement service which suggests and ranks queries which connect the initial seed concepts (and an optional result type), and then suggests query refinements based on the instance data and associated ontology. This service is enabled by an abstract query representation which reflects the user exploration model and simplifies the interface between the refinement service and the user-facing components.

External Posting Date: July 21, 2009 [Fulltext]
Internal Posting Date: July 21, 2009 [Fulltext]

Approved for External Publication

© Copyright 2009 Hewlett-Packard Development Company, L.P.

A query refinement model for exploratory semantic search

Dave Reynolds, Ian Dickinson, and Dave Grosvenor

Hewlett Packard Laboratories, Bristol
{dave.reynolds,ian.dickinson,dave.grosvenor}@hp.com

Abstract. We describe a novel approach to exploratory semantic search applied to the problem domain of e-discovery. In this domain, as in many others, we need to support non-expert users in composing queries over a complex and heterogeneous knowledge base. We achieve this by enabling users to initiate a query through a few seed concepts and terms, and then guide them in incrementally refining the query and exploring the dataset without requiring prior knowledge of the ontology. This is achieved through a query refinement service which suggests and ranks queries which connect the initial seed concepts (and an optional result type), and then suggests query refinements based on the instance data and associated ontology. This service is enabled by an abstract query representation which reflects the user exploration model and simplifies the interface between the refinement service and the user-facing components.

Key words: query refinement; semantic search

1 Introduction

In this paper we describe an approach to exploratory semantic search, motivated by specific user requirements in the domain of *e-discovery*. While we believe the techniques we report are broadly applicable, the e-discovery domain has some interesting characteristics which motivate our extensions to other exploratory search approaches.

E-discovery [1] is the process of collecting, preparing, reviewing, and producing electronically stored information (ESI) in the context of a legal process. For example, a company named in a patent dispute may be required to produce a wide range of design documents, letters, emails, product and part descriptions and meeting records, possibly dating back many years. E-discovery is typically characterised through a phased model[1], of which the first phase, *identification*, is our concern in this paper. Although it clearly overlaps with enterprise search, e-discovery is not just a search problem. The priority during the identification phase is completeness: all potentially responsive ESI must be located [2]. Failure to locate relevant ESI may directly (and adversely) affect the outcome of the legal action. The initial identification phase is concerned with finding not only

[1] The Electronic Discovery Reference Model, http://www.edrm.net/

specific documents, but also relevant people, projects, organizations and data repositories. In contrast, enterprise search typically focuses on finding the few documents most likely to answer a user's question.

We have been developing a system to support this e-discovery identification phase by integrating data from many diverse sources across an enterprise. Such information includes contact and organizational information (as is typically found in an enterprise directory), product descriptions, and role information indicating the roles that individuals and workgroups play in relation to products and business activities. The sources from which information is extracted include formal structured published sources (e.g. databases, LDAP directory services or centralised XML product catalogues), and information gleaned from intranet web sites and content management systems by text mining algorithms. The heterogeneity of these sources leads to a complex knowledge representation requiring an open, extensible data model. In our work, we use RDF[2] as the common information model, and a collection of OWL[3] ontologies to enable semantic integration. We use the term *knowledge base* to describe the collection of RDF information and OWL ontologies available to our system.

In this paper we describe the exploratory search and query interface which enables users to exploit this complex, interconnected knowledge base to assist in discovering people, products, organizations and repositories for a given e-discovery case. A goal of our research is not only to improve the underlying mechanisms by which users can navigate a rich knowledge base, but also to ensure that those enhanced mechanisms really do help people to achieve their information-seeking goals.

The nature of the domain problem highlights a number of requirements we need to address:

- users are searching over both documents and conceptual entities (people, workgroups, products, locations);
- most searches involve locating a complete set of resources meeting some criteria, not simply retrieving a few most relevant documents;
- there are many different ways in which concepts can be related, with multiple paths of evidence to follow;
- our users are not conversant with the schemas used to store structured data in the enterprise, nor are they necessarily aware of the concepts and properties modelling this data in our knowledge base;
- the user does not normally have a specific well formulated query but has to explore the knowledge base to find relevant entities and connections;
- the knowledge base may change dynamically as new sources become integrated, opening new paths to investigate;
- our users must be able to defend the claim that the search results are complete, so it is important to record the process of exploration as well as the results.

[2] Resource Description Framework (RDF), http://www.w3.org/RDF/
[3] OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/

We summarise our approach as supporting iterative exploration of the knowledge base allowing a user to initiate an approximate query, and then refine it to home in on the set of concepts and connections relevant to the case. This raises two primary challenges:

1. How can a user initiate a query when the data model is not known?
2. How can we effectively guide a user in refining the query?

1.1 Motivating example

The following brief example illustrates the kinds of queries we wish to assist our users with. In a patent violation dispute involving Company B, Company C is called to give documentary evidence germane to the case. Company C was a large customer of Company B, and therefore may be able to throw light on the intellectual property claims. The presiding judge has mandated Company C to produce all relevant documents relating to Company C's use of Company B's products, possibly including: emails, letters, meeting records, marketing material, technical specifications and customer support case records. Company C last worked with Company B five years ago, and members of the project team have variously moved to new roles in the company, or have left Company C altogether. C's legal team need to identify all relevant internal sources of data and make sure that nothing is deleted. They must identify authors of documents from the time of the project, and locate those that are still within the organization. They must identify current possessors of relevant documents, who may not have been the original authors. For project team members who have now left Company C, the legal team must identify from the documents of the period who the likely collaborators and managers of those staff would have been, to check if those persons retain any relevant documentation.

2 User query model and exploratory user interface

The basic user interaction paradigm we are basing our work on has been characterised elsewhere as *exploratory search* [3]. In exploratory search, information seeking is regarded as a long-lived activity, in which search actions are interleaved with responses from the system, including partial results or result summaries. These responses in turn inform and guide the user's future search actions. Exploratory search is well-suited to situations in which users do not start with clearly articulated search goals and strategies, but expect to have these goals and strategies instantiated and refined as a result of the exploration process itself.

The eDiscovery domain presents some differences to exploratory search as typically characterised in the literature. In particular, we note:

– our corpus contains both structured data and unstructured documents, so during an exploration session users will effectively be issuing both structured queries and unstructured searches by keyword;

– the primary goal is to identify a complete set of responsive items, not to retrieve them for the consumption of the user

We have designed and partially implemented a user interface that attempts to address these issues and provide an exploration interface suited to our domain. The remainder of this section describes the overall design of the user experience, illustrated with notes from the current prototype.

2.1 Abstract user interaction model

We abstract the overall user interaction as a series of moves through a connected space of *interaction states*. Each node in the space corresponds to an interaction state, represented by a 5-tuple:

$$S_i = \langle \psi_i, \rho_i, \sigma_i, c_i, h_i \rangle$$

where S_i is the i'th state, ψ is the current query (encoded in our abstract query notation AQL, described below), ρ is the set of results obtained from interpreting and running ψ, σ is a ranked list of candidate adjacent states, c is the current context, and h is the history of commands that the user has issued. Each state denotes a complete set of results, and is annotated with the set of moves (the history) that produced it. This supports both the exploration capabilities we want to enable, but also the case-recording requirements in the e-discovery domain.

The *context* has two roles in our system. First, it records relevant information that helps present a manageable presentation of possibly complex results to the user. Second, it provides (possibly heuristic) guidance to the query refinement service in order that it can better rank the candidate suggestions to present to the user. These roles overlap to some extent. In our case, we use an RDF model to encode the context, thus providing an extensible container which is well suited to holding context terms which may be extended or refined in future. A simple example of the context would be to record the current resultset window: if ρ_i contains 10^5 items, it is neither efficient nor helpful to the user to present them all at once. Instead, we record the start and length of a window onto the current resultset in the context.

There are two distinct phases in the user-experience model, corresponding to the initial state S_0 and subsequent exploration states $S_{i\geq 1}$. In particular, for S_0 we need to generate an initial query ψ_0 based on some initial user inputs or *seeds*. In principle there can be many kinds of user input seeds; currently we support four: unstructured text keywords, formal concepts (representing individuals or classes from the domain ontologies), relations, and desired result types. An initial dialogue in the user interface allows users to quickly and iteratively build a set of seed inputs.

Following the initial seed selection, given an interaction state S_i, we present an interface to the user that contains the following categories of affordance:

- a presentation of (some subset of) the resultset ρ_i. By default this uses a table presentation, but other presentation and visualization forms are supported by the architecture. Presentations may be selected dynamically based on result types and information from the current context;
- UI actions to move directly to an adjacent state S_j, by selecting one of the suggested query refinements – e.g. to filter the resultset, or to pivot along some relationship in the data;
- UI actions to move to an adjacent state by directly editing the current state description – e.g. by adding or changing search terms, changing the desired output type, etc.
- UI actions to alter the context
- UI actions to move directly to a previously explored state via the history

Each action taken by the user is added to the state history, and takes the user to a new interaction state. In addition to providing an abstract understanding of the overall process of interacting with this system, this abstract model is the basis for the user model, which is recorded in an RDF model on the server. This model then provides a resource that can be used by the query refinement, query suggestion and presentation tuning services as inputs to the choices made by those algorithms[4].

2.2 User-interface implementation

We use a rich-client architecture for our experimental system, in which a front-end written in Flex[5] manages presentation and user-gesture translation. Gestures that cannot be directly performed on the client are encoded as command objects which act to change the current state description. These commands are serialised and sent to the server. The Java back-end responds reactively to the state changing commands by dynamically invoking services, such as the query refiner and query suggester, below. To keep responsiveness high while allowing for slower-running services, responses are streamed back to the client asynchronously as they are generated by the query processing services.

Concept resolution Concept resolution is primarily used during the initial query formulation dialogue, though it is also available to users during the later exploration phase. We make a distinction between query terms that are recognised members of a controlled vocabulary – terms in the one of the underlying ontologies – and keyword-matching terms that are not controlled. The UI provides easy mapping between these two disjoint term spaces by using auto-completion: as the user enters characters in a query term, for example *"enterprise informatics"*, the look-ahead sub-system will match those characters to the `skos:label` or `rdfs:label` of knowledge base terms, such as the individual representing the

[4] Use of the context user model in the query refining algorithms is not yet implemented in the current prototype

[5] Adobe Flex 3, http://www.adobe.com/products/flex/

Enterprise Informatics Lab. If the user accepts the match, the term is replaced with the concept's URI. However, the original user input is retained, making it easy for the user to flip the interpretation of the term between a concept and an unstructured text keyword match, or to select an alternative mapping from the input text to a concept term in the KB.

Query refinement The algorithms underlying the query refinement services are discussed in more detail below. Here we outline how the possible exploration moves are presented to the user. Figure 1 shows a screenshot of the current prototype UI (which is still in early development). The query shown lists co-authors of members of a given laboratory who have authored documents containing the keyword 'rdf'. This state was achieved with two moves: an initial query formulation and a navigation via the 'co-authors' suggested link. Further suggested related queries are listed to the left of the resultset display area. The current state is summarised just above the result and navigation displays.

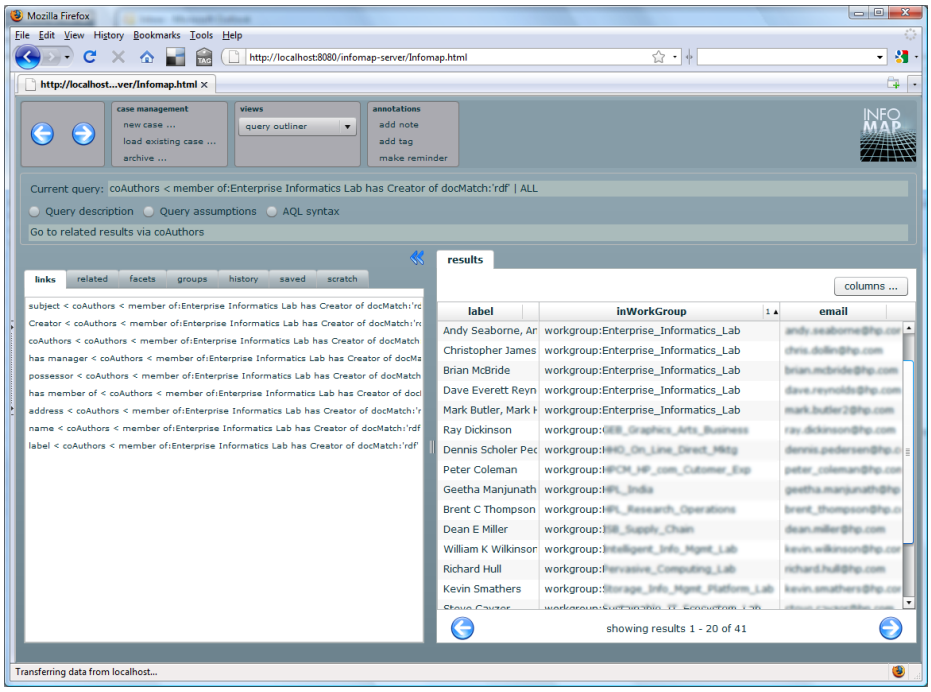

Fig. 1. Screenshot of prototype user interface

The labels on the suggested queries are auto-generated from the abstract query language expressions generated by the query refinement service. A significant challenge in pursuing this form of query refinement is to synthesise labels that are both compact and meaningful to our users. We hope to address this issue in future research.

A further challenge is that there may be many possible refinements from any given interaction state, but presenting too many such choices will overwhelm the user and reduce usability. One tactic we have used is to group candidate refinements into thematic groups: for example, related queries, link traversals (e.g. from people to their managers), narrowing or widening via the concept hierarchy, or facet-based filtering. Such groups are presented in different UI controls, and allow the user to make tactical choices at a coarser granularity than individual refinements. A future extension is to allow user preference for a particular class of refinement, which becomes a parameter to the query refinement service.

3 Architecture

A high level outline of the architecture which supports this user interface is illustrated in figure 2.

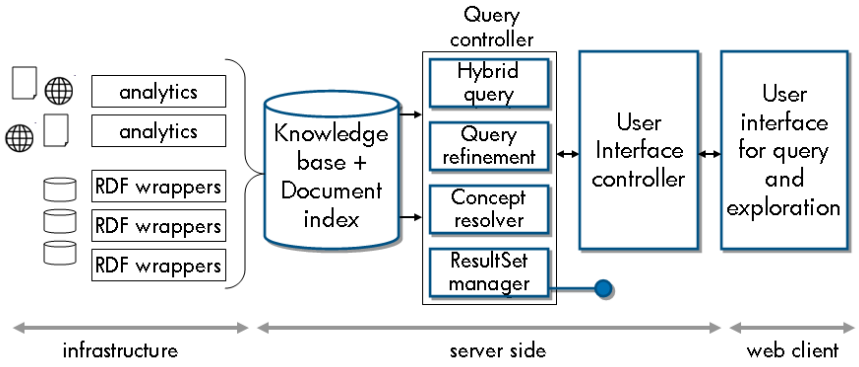

Fig. 2. High level architecture

We omit the details of the infrastructure for wrapping existing information sources, mapping between conceptual models and performing off-line text analytics (but see Butler et al [4] for more details). We take as given that the query refining capability has access to an RDF knowledge base, a document index over documents associated with entities in the knowledge base and to the necessary ontologies including the common upper ontology.

The core component is the *query controller*, which provides four services: hybrid query service, concept resolver, resultset manager and query refiner. These services are located in a Java servlet container, so expose a Java API to co-resident components such as the user interface controller in the normal way. However, they each also expose a RESTful[5] API over HTTP, allowing the services to also be utilised in a larger, decentralised service-oriented architecture. RDF, OWL and SPARQL processing are based on the Jena platform [6].

The *hybrid query* service provides standard SPARQL query over the knowledge base. It also handles embedded queries for documents by delegating some query terms to the document index, via the use of property functions[6].

The *concept resolver* service maps lexical terms to knowledge base concepts such as names of people or workgroups, or ontology concepts such as user-relevant classes and properties. Our initial implementation is simply a text index over labels and descriptive terms associated with the knowledge base resources. However, these descriptive terms may be derived from text analytic processes which can identify aliases, and expand acronyms and abbreviations.

The *ResultSet manager* provides an abstraction above the raw SPARQL query layer to enable the query model described earlier. In our system, a *ResultSet* is an ordered list of RDF resources which match a query specification, together with a set of RDF statements that describes each resource. By query specification, we mean a SPARQL SELECT query, with optional ordering and text index annotations. The RDF description to be associated with each matched resource can be specified separately from the main query and can range from a generic concise bounded description[7] to an application-specific template. The ResultSet also includes metadata for the query and the whole set of results (also encoded in RDF); this enables abstracted views of the set such as clustering or summarization. The interface is designed to support lazy evaluation and a query can be transparently suspended, cached or restarted without affecting any client services. ResultSets are first class objects in the system. They appear as a web resources, identified by URIs, and so references to them (and their associated content and metadata) can be easily passed around the distributed architecture.

The final service, the *query refinement* is the primary focus of this paper. It provides two core functions. The first, *instantiate*, takes a set of keywords and seed concepts and generates a ranked set of possible queries relevant to those seed terms. The user interface is thus able to combine the concept resolution service (to map lexical terms to concepts) with the instantiate service to initiate a search. The second function, *refine*, takes a previously generated query and returns possible refinements of it. In essence, it suggests possible moves in the exploratory search. The results of each service operation are encoded as RDF models allow us to include rich and extensible metadata to describe the queries and their interpretation.

4 Query refinement model

We divide the space of possible query refinements into a number of categories of refinement types. This enables the user interface to group and organize refinement suggestions. We capture this conceptual model of the space of refinements as a simple OWL ontology, thereby enabling us to encode sets of related refinement suggestions as RDF models.

When the user interface controller requests query refinement suggestions they can specify a set of categories to suggest. The categories we define are:

[6] http://jena.sourceforge.net/ARQ/extension.html#propertyFunctions

Instantiate Generate an initial query which connects a set of seed concepts and/or document keywords and an optional target class. For example given a document keyword, an organization and a target class of `Person`, the instantiation might look for people within the given organization who authored (or possess) documents with that keyword. For any seed query, many possible instantiations are possible.

Narrow Reduce the ResultSet in size by focussing on some aspect. Includes:

- **Constrain** Add a constraint to an existing query. For example, limit the returned people to those who are based in a particular geography.
- **Specialize** Narrow the scope of a constraint in an existing query by descending a hierarchy. Such hierarchies can include `part-of` relations (e.g. narrow the query to just those people within a sub-organization of the original seed organization) as well as taxonomic hierarchies.

Widen Increase the ResultSet in size by generalizing some aspect. Includes:

- **WeakenConstraint** Weaken or remove an existing constraint.
- **Generalize** Widen the scope of a constraint by ascending a hierarchy. Like `Specialize`, this applies to `part-of` relations as well as taxonomic hierarchies.
- **AddRelated** Broaden a query by adding in additional results which are linked to those in the base query. For example, expand the set of people in our original instantiate example by also returning all people who co-authored matching documents.

Link Find all resources connected to those in the base query by some relation. That relation might be an explicit RDF property or might be an induced relation computable over the knowledge base. Examples include the *co-authors* relationship pattern, and various textual similarity measures. In the current implementation such induced relations are defined by pre-defined query templates. In future, we hope to add user generation of link refinements through some form of "macro" mechanism.

Related Find queries which are alternates to some base query. Typically this case arises after an instantiate step where alternative property paths could have been used (e.g. selecting `possessor` instead of `creator` as the link from a document to a person.

In this paper we concentrate on *Instantiate* and *Link* refinements but our overall framework is designed to support the full range described above.

5 Abstract query representation

As noted earlier, we use SPARQL, extended with document indexing, as our underlying query mechanism. However, we found it cumbersome to work directly with raw SPARQL in the query refinement service. Instead, we developed an *abstract query representation* (AQL) as an intermediate layer used both within the query refinement algorithms and for external access to the refinement service. This abstraction meets several design goals:

- compositional, it is easy to take an existing abstract query and add constraint filtering, link following or description templates without needing to decompose the base query
- support for our query model – queries denote a ResultSet which combines an ordered set of matching resources with a set of RDF statements describing those resources. We thus combine SPARQL `select`, `describe` and `construct` in one abstraction, combining control of resource selection and ordering, and resource description
- easy to manipulate programmatically
- serializable to RDF so that we store set of suggested query refinements along with their inter-relationships and metadata.

Note that this representation is not intended as a new RDF query language. It is a convenient abstraction which simplifies the query refinement service interface and its internal design.

The abstract query representation is constructed as an algebra of operators, encoded as S-expressions. The key operators in our representation are:

`(ALL)` corresponds to all resources across the entire knowledge base (and associated document index).

`(ENUM` *`value*`*`)`* a set of explicit enumerated values.

`(SOURCE` *`uri`*`)` all resources corresponding to a particular named graph within the overall knowledge base.

`(SPARQL` *`query`*`)` all resources which match a SPARQL SELECT query with a single distinguished variable.

`(FILTER` *`constraint*`* *`Q1`*`)` filter the results from query `Q1` retaining only those that match all the constraints. A constraint is a pair *`(pexpr cvalue)`*. A *`pexpr`* is one of: *`predicate`* for an RDF property or virtual property, `(INV` *`predicate`*`)` for the inverse of a predicate, `(CHAIN` *`predicate*`*`)` for a linked sequence of predicates, `(TEMPORAL` *`predicate range`*`)` for reference to a temporally qualified predicates[7] or `ANY` for a wildcard predicate. A *`cvalue`* can be an RDF resource, a range expression, or a nested abstract query expression.

`(FOLLOW` *`pexpr Q1`*`)` take all results from query `Q1` and find the set of values related to those resources via the given predicate expression (i.e. implements the *link* refinement).

`(GROW` *`pexpr Q1`*`)` perform a `FOLLOW` from the base query `Q1` but return the union of `Q1` and the linked results.

`(DESCRIBE` *`Q1`*`)` includes in the ResultSet a concise bounded description [7] of each resource that matches *Q1*.

`(DESCRIBEBY` *`BGP Q1`*`)` includes in the ResultSet a description of each resource that matches *Q1* as defined by a SPARQL basic graph pattern.

[7] Support for temporal qualification is important in this application domain but is outside the scope of this paper.

`(ORDERBY predicate Q1)` order the resource matches in the result set according the value of the given predicate.

`(UNION Q1 Q2)` the union of the results from *Q1* and *Q2*.

`(INTERSECTION Q1 Q2)` the intersection of the resources matching *Q1* and *Q2*.

`(DIFFERENCE Q1 Q2)` all resources matching *Q1* which do not match *Q2*.

We encode these abstract query expressions in RDF using the SPARQL s-expression (SSE) support from Jena's ARQ processor[8]. The query arguments can be expressed inline with nested s-expressions or by a URI reference which identifies the ResultSet arising from a previously explored query.

We can encode in a single RDF model an entire set of query refinements, the abstract query expressions they correspond to, and their associated descriptive metadata. This allows us to experiment with additional services which analyze and describe the query options by enriching the metadata in the RDF model.

When a query is to be executed, the *hybrid query* service translates the abstract expressions to SPARQL, dereferencing any embedded query URI references. In the current implementation this is done directly at the SPARQL algebra level, avoiding the need to generate strings in SPARQL surface syntax.

6 Refinement algorithms

In this section, we discuss how the query suggestions are generated using a type graph, and ranked using both query complexity and frequency-of-use statistics. The type graph and statistics of use are derived offline from the knowledge base, rather at query time.

Type-graph The type graph is used to determine the possible connections between a seed and result types. The edges in the type graph are predicates from the knowledge base, while the nodes contain type information about the entities related by the predicates. For example, the `possessor` predicate in the type graph links a `Document` type node to a `Person` type node. The type graph can be derived from the ontology by using either using the domain and range information for predicates provided by the ontology, or the type information about instances related by a predicate.

We introduce inverse predicates into the type graph for every predicate added into the type graph, to make it easy to generate exploration links that traverse the graph in any order.

Figure 3 shows a simple example of a type graph (omitting inverse predicates for clarity). This type graph can be used to determine multiple connections from `Person` to `Organization`, using the direct connection using the `member-of` predicate, or an indirect connection going via the `BusinessGroup`, `BusinessUnit` or `WorkGroup` type nodes.

[8] http://jena.hpl.hp.com/wiki/SSE

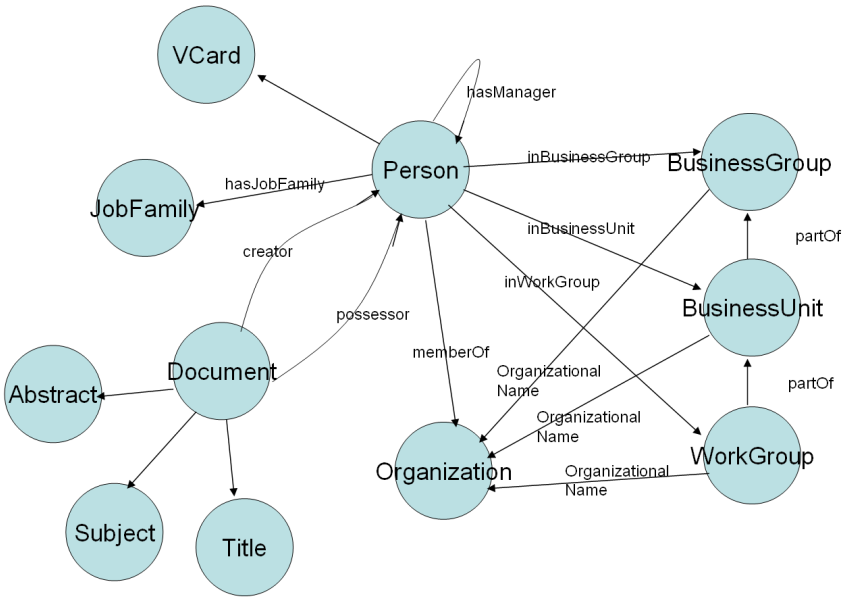

Fig. 3. Simplified type graph omitting some predicate names and inverses

A guided-heuristic search of the graph is used to find the potential connections between any two types. Each connection determines a sequence of predicates, which are composed in a `follow` AQL expression. For example, assume we start with the person 'Dave Grosvenor', and want to find connections to `Organization`. The query refiner computes that the seed type is `Person`, and generates the following AQL expressions:

```
(FOLLOW :member-of (ENUM hp:dave_grosvenor))
(FOLLOW :organizationName (FOLLOW :inBusinessGroup (ENUM hp:dave_grosvenor)))
```
... omitted for brevity ...
```
(FOLLOW :organizationName (FOLLOW :part-of (FOLLOW :inWorkGroup
        (ENUM hp:dave_grosvenor))))
```

Clearly the number of possible connections will grow very rapidly, which we control though ranking query suggestions by the complexity of the AQL expression, as shown below.

Instantiate The instantiate refinement takes a set of seed concepts (including keywords, relations and result types) and generates an initial query which is likely to represent the user's initial information-seeking goal.

To do this, we first we find the potential connections from each of the given seeds to the result-type. This generates a set of query suggestions for each seed. We then compose a set of combined query suggestions from the individual seed suggestions, by forming an `intersect` query expression. The arguments to the

intersection are taken from the contributing individual seed suggestions. This composition creates a combinatorial explosion in the number of possible query suggestions. Thus the use of mutliple seeds provides a further motivation for the use of ranking over query suggestions.

If there is no connection from any given seed to the result type, we construct an intersect query from those seeds for which there is a non-empty suggestions set. This is motivated by search-like behaviour, which would attempt to provide the best results to the set of criteria even when some combination of criteria cannot be met. This would require some explanation and feedback to the user that one or more of their seeds are not affecting the suggestions.

Link refinements A link refinement identifies resources connected to the resources in a base query, using some linking predicate. We take a sample of the results from the base query and determine which outgoing or incoming predicates are present in the instances in the knowledge base. These instance samples are then augmented by examining the type graph for arcs which connect to the type node corresponding to the base query. The instance sampling allows us to detect links which are not known to the ontology but are nevertheless present in the actual data, whereas the ontology-derived links enable us to detect links which may not be present in our particular sampling. We can then use a weighting function, as shown below, to order the sets of possible links, though we bias the ranking to favour instance-derived links since we prefer queries which we know will return at least some results.

Ranking As described above, we use ranking of query suggestions in a variety of ways to control the combinatorics of the suggestion generation process. There is a need to rank suggestions from all refinement operations, not just instantiate.

We can rank suggestions based on a number of criteria. One criterion is the complexity of the AQL expression:

$$\begin{aligned} rank(op_0) &= 1 \\ rank((op_1\ x)) &= 1 + rank(x) \\ rank((op_2\ x\ y)) &= 1 + rank(x) + rank(y) \end{aligned}$$

where op_i is any AQL operation with arity i. Secondly we can weight the connections used in the query, and prefer queries which make use of more highly weighted connections. We weight connections by $w_i * ln(f_i)$ where f_i is the frequency with which the associated predicate is used in the knowledge base and w_i is an application-specific weighting factor which allows us to adapt the recommendations to the user's task. In the current implementation the weighting factors are manually entered as part of system configuration; however, in future work we would like to explore weighting based on learned preferences for individual users and case types.

We can combine the search of the type graph with the weighting. We set the length of a connection arc to be the inverse of the weight (so important, high

frequency connections are regarded as short) and then search for the N-shortest paths in the graph. Such shortest paths will also typically correspond to low complexity queries.

7 Status

We have developed a prototype implementation of the architecture, query services and user interface described here. This work has been carried out in the context of a larger project looking at the application of semantic web technology to e-discovery. We have developed and tested the exploratory user interface over an experimental knowledge base drawn from HP's internal information sources. The current implementation supports only the instantiate, related and link refinements but we do not anticipate difficulties extending the approach to the other refinement types discussed.

At the time of writing no formal user evaluations have been carried out. However, initial reactions from our internal sponsors has been very positive indicating that the style of interaction this approach facilitates is a good match to the requirements of this application domain.

8 Related work

Exploratory search [8] is an active research area in the information retrieval community, and a variety of interface approaches can be found in the literature. One commonly used technique is *faceted browsing* [9], in which attributes of the set of objects currently displayed as results are used to dynamically generate UI affordances that can filter the resultset down to a narrower collection[9]. Basic facet browsing is widely used, especially on eCommerce sites, to filter a set of products to those meeting a customer's needs. An extension to basic facet filtering allows the user to select related resultsets by *pivoting* on a relational attribute, for example from US presidents to their children, as illustrated by Huynh's Parallax system [10]. Our work uses both facet filtering and pivoting to navigate the space of interaction states, but our query refiner is able to generate navigational links that might take several moves to accomplish with traditional facet browsing and pivoting and can include query generalization as well as narrowing. Moreover, our synthesized suggested query moves can, at least in principle, abstract away from the underlying data model and present navigational links in conceptual terms familiar to the user. However our current prototype has only limited capability in this regard.

The early work by the TAPSOFT project [11] gives a good overview of how search can exploit the semantic web. TAPSOFT used semantics to both improve the presentation of search results, and to improve search performance. More recently, work on semantic search by Cimiano et al [12] and Zhou et al [13] investigates the interpretation of keywords in semantic search. These approaches,

[9] We note that disjunctive facets can increase the size of the resultset.

and ours, aim to suggest structured queries which interpret keywords as concepts. We also suggest other refinement operations. Both approaches generate ranked lists of structured queries against the knowledgebase, and derive a subgraph of the knowledge base to analyze the connections between the entities denoted by the keywords in the query. This is similar to the approach we use with a type-graph, however we aim to avoid run-time retrieval of instances from the KB during the query suggestion process.

9 Conclusions and future work

We have described an implemented architecture to support exploratory query interfaces over a complex knowledge base. The current prototype illustrates some key features of a task interface to support users with challenging information seeking problem. Building on this initial prototype, future work will investigate ways to use context information to bias query refinement to the user's interests, better summariziations of the current interaction state, new query refinement operators and better ranking of query refinement candidates based on complexity measures and other metrics on the candidate query expressions.

Our approach illustrates a number distinctive features:

- It conceptualizes the user task as a series of moves in a space of possible queries, separating the generation of the query space from the user interface affordances for navigating it.
- It introduces the notion of *instantiating* a query from an initial set of seed concepts and then *refining* it based on a number of categories of refinement types. These refinements go beyond the filtering seen commonly-used faceted browsing interfaces, by adding link following and query generalization.
- We provide an abstract query representation which simplifies both the interface to the query refinement service and its internal implementation. We are able to encode a set of related and categorized query refinements as an RDF model which both simplifies the service interface and allows us to apply our RDF tools to the manipulation of this materialized set of search options.
- We have defined initial algorithms for the instantiation and refinement service, based on the generation of a type graph from both instance data and available ontologies.

We believe that our current design and initial implementation provide a promising start in developing a fresh approach to complex information discovery and management problems typified by the e-discovery domain. The system offers a framework in which we can explore new algorithms for suggesting and ranking moves in query space (as opposed to ranking of query results). This will enable us to explore mixed initiative interfaces in which the system can proactively offer navigation and query options tailored to the user task.

Acknowledgments. Brian McBride and Mark Butler developed the framework for ingesting time-varying data into knowledge base from multiple sources, and together with Andy Nelson contributed to the overall application goals and design. Our architecture depends on ARQ, developed by Andy Seaborne, and the overall Jena framework designed and built by the Jena team at HP Labs.

References

1. Crowley, C.R., Harris, S.B.: The sedona conference glossary: E-discovery & digital information management (second edition). Technical report, The Sedonda Conference Institute (2007)
2. Baron, J.R., Thompson, P.: The search problem posed by large heterogeneous data sets in litigation: possible future approaches to research. In: ICAIL '07: Proc. 11th Int. Conf. on Artificial Intelligence and Law, ACM (2007) 141–147
3. Marchionini, G.: Exploratory search: from finding to understanding. Communications of the ACM **49**(4) (2006) 41–46
4. Butler, M., Reynolds, D., Dickinson, I., Grosvenor, D., McBride, B., Seaborne, A.: Semantic middleware for e-discovery. Technical Report HPL-2009-76, HP Laboratories (2009)
5. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Technol. **2**(2) (2002) 115–150
6. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the Semantic Web recommendations. In: Proceedings of the 13th International World Wide Web Conference, ACM (2004) 74–83
7. Stickler, P.: Cbd - concise bounded description. Technical report, W3C Member Submission (2005)
8. White, R., Kules, B., Drucker, S., schraefel, m.c.: Supporting exploratory search: Introduction to the special issue. Communications of the ACM **49**(4) (2006) 36–39
9. Tunkelang, D.: Faceted Search (Synthesis Lectures on Information Concepts, Retrieval, and Services). Morgan & Claypool Publishers (2009)
10. Huynh, D.F., Karger, D.R.: Parallax and companion: Set-based browsing for the data web (2009)
11. Guha, R., McCool, R., Miller, E.: Semantic search. In: Proc. World Wide Web Conference (WWW'03), ACM (2003) 700–709
12. Tran, T., Cimiano, P., Rudolph, S., Studer, R.: Ontology-based interpretation of keywords for semantic search. In: ISWC/ASWC. Volume 4825 of Lecture Notes in Computer Science., Springer (2007) 523–536
13. Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: Spark: Adapting keyword query to semantic search. In: Proc. 6th International Semantic Web Conference (ISWC'07). Volume 4825 of LNCS., Springer Verlag (2007) 687–700