INCREMENTAL END-USER QUERY CONSTRUCTION FOR THE SEMANTIC DESKTOP

Ricardo Kawase, Enrico Minack, Wolfgang Nejdl
L3S Research Center, Leibniz Universität Hannover, Appelstrasse 9a, 30167 Hannover, Germany

Samur Araújo, Daniel Schwabe
Departamento de Informática, Catholic University of Rio de Janeiro
Rua M. de S. Vicente, 222 Rio de Janeiro RJ 224530-900, Brazil

Keywords: Semantic Desktop, User interface, Visual query system, Graphical query language, Ontology, RDF, SPARQL, Personal information management.

Abstract: This paper describes the design and implementation of a user interface that allows end-users to incrementally construct a query over the information in the Personal Information Management (PIM) domain. It allows semantically enriched keyword queries, implemented in the Semantic Desktop of the NEPOMUK Project. The Semantic Desktop user is able to explicitly articulate machine-processable knowledge, as described by its metadata. Therefore, searching this semantic information space can also benefit from the knowledge articulation within the query. Contrary to keyword queries, where it is not possible to provide semantic information, structured query languages as SPARQL enable exploiting this knowledge explicitly.

1 INTRODUCTION

Whereas traditional folder-based storage and navigation-based retrieval used to be sufficient for our personal information management needs, the sheer number of items on our Desktop calls for the integration of Desktop search in its interfaces. The paradigm of Desktop search is similar to Web search, but shows some interesting differences:

- We typically search for items that we have created, received or stored earlier – which means that we try to re-locate (re-find) rather than to find (discover) something new.
- We typically have a (most likely incomplete) picture of how the data on our Desktop is organized (which means that navigation might be more apt than keyword search).

From Teevan et al. (Teevan, 2004) we know that users use a combination of search and navigation to relocate information on the Web. For personal information management this is likely to be the same, provided that the appropriate tools are available.

To allow the personal information management in the Desktop we first need to introduce an ontology language that can be used to express personal mental models – the personal information model ontology (PIMO)[1]. The use of such an ontology gives the user a better understanding of her Desktop and her tasks, and a way to express her knowledge about the information items. By adding explicit, processable meta-data to the information items in the Desktop, it becomes a "Semantic Desktop", a concept we will elaborate later.

As it has been identified in previous works (Teevan, 2004) (Bates, 1989) (Belkin, 1993) a progressive modelling search activity can provide better results than a single, static search. Yet, strong methods and interaction mechanisms are still missing for searching the Desktop.

In this paper we fill this gap by presenting a methodologically designed interface for the

[1] http://www.semanticdesktop.org/ontologies/2007/11/01/pimo/

Kawase R., Minack E., Nejdl W., AraÃžjo S. and Schwabe D.
INCREMENTAL END-USER QUERY CONSTRUCTION FOR THE SEMANTIC DESKTOP.
DOI: 10.5220/0001825902700275
In *Proceedings of the Fifth International Conference on Web Information Systems and Technologies (WEBIST 2009)*, page
ISBN: 978-989-8111-81-4

(Semantic) Desktop search that combines and exploits current interaction mechanisms. It benefits from what the user knows (the vocabulary and the mental model), respects what the user does not know (the data structure and query languages), to finally give her what she wants.

In the remaining of this paper we discuss in more details searching activities in the Semantic Desktop and our proposed work in Section 2. In Section 3 we describe our implementation and the architecture. Finally, in Section 4 we draw our conclusions and the sketch future work.

2 SEARCHING THE SEMANTIC DESKTOP

In (Sauermann, 2005), the authors argued, that the typical user uses the (information on the) Desktop to complete a certain task. For this, documents that are relevant to the user's current task are retrieved, processed and stored. Such documents contain relevant information that is processed by the user, allowing the user to generate knowledge. This knowledge is implicitly stored in the documents. Making this implicit knowledge explicitly expressible and machine-processable, is one of the goals of the Semantic Desktop. Allowing the user to exploit knowledge for retrieval at the Semantic Desktop, as implemented in the NEPOMUK[2] project, is the goal of our proposed user interface.

The definition given by Sauermann et al. (Sauermann, 2005) depicts, that the Semantic Desktop paradigm brings the ideas of the Semantic Web paradigm to the user's personal Desktop where the conceptualization of the personal mental model is described in formal ontologies. The standard data format for a common representation is RDF (Resource Description Format). Finally, the different Desktop applications are integrated using the same concept of the Semantic Web, for exchanging data and accessing resources.

The NEPOMUK project integrates research, industrial and open source community efforts to develop a new technical and methodological platform: the Social Semantic Desktop. This is an extension of the personal Desktop that aims at collaboration and personal information management.

The NEPOMUK framework PSEW[3] (P2P Semantic Eclipse Workbench) is an integrated environment that is based on the NEPOMUK architecture. Since NEPOMUK still requires some semantic knowledge from the user, user-friendly interfaces are a crucial milestone to achieve the goal of bringing the Semantic Desktop to the common user. Walking in a two way path, first we aim at designing interfaces that solve the user's needs on the Semantic Desktop, and on the other direction, we design interfaces to show the user the potential of the Semantic Desktop and still hide its complexity. In both cases, first we take a look at the way people think and express their mental models, so that we understand how the Semantic Desktop can support this (Sauermann, 2005).

As we mentioned before, we use the Personal Information Model Ontology (PIMO) to describe and work within the PIM in the Semantic Desktop. The PIMO forms the basis for all custom, user-created types and relations. It defines basic types such as *Document*, a *Person*, a *Location*, etc. and relations such as *creator*, *hasLocation*, etc. and is intended to be extended by the user in any way he or she likes. Note that we use the terms "type" and "relation" throughout the paper as user-friendly terms for RDF Class and RDF Property. We also use these less technology-based terms consistently in our user interface and the implementation.

The user can extend the PIMO ontology and use it to articulate arbitrary knowledge in an explicit way. For the task of re-finding information on the Semantic Desktop, NEPOMUK provides two different mechanisms. First, a type and instance browser that is analogous to most operating system file browsers where users type hierarchy and the instances of each type. Alternatively, there's a full-text keyword search that analyses the extracted metadata from the instances returning a relevance sorted list classified using complex ranking algorithms.

It happens that in both cases the potential combination of user knowledge and system functionality is not fully merged. The browser does not allow the user to input her knowledge about the instances and its relations. Conversely, the keyword search does not permit the user to use her knowledge of her PIMO.

Consider the case when the user is looking for an email (or a presentation document) that was sent (or created) by a certain person, containing a certain information (e.g., a telephone number or a quote). In a pure keyword-based interface, the user should input a query such as "email sent by person telephone number". However, this is very unlikely, since most purely keyword-based interfaces assume

[2] http://nepomuk.semanticdesktop.org/

[3] http://nepomuk-eclipse.semanticdesktop.org/xwiki/bin/view/Main/PSEW

the search will be made exclusively in the *contents*, and it would not be expected to understand the semantics of "sent by", or the type of item (email). Being able to leverage such semantic information greatly reduces the search space, since, for instance, only a small fraction of items in the Desktop would have a relation "sent by".

A similar argument can be made when the query must span several items, e.g., "email sent by the author of a given presentation". In such cases, keyword-based searches would, at best, retrieve partial answers, that must be combined by the user by filling in the semantic information that allows the binding between the partial answers returned.

The user would now need to know the exact semantic terms that are used in her semantic repository. Recommendations help to reduce the burden to find them, by presenting only sensible choices of types and relations for selection. Keyword filters further provide quick access in case parts of the semantic terms are known.

From these observations, we developed a modular incremental end-user query construction interface that enables the users to build coherent semantic queries. Our interface does not cover all possible queries expressible in a query language such as SPARQL (Prud'hommeaux, 2008). However, we are confident that our user interface allows for the construction of most of the usual structured user queries. This can be justified by looking at the features of the SPARQL query language, see Section 12 of (Prud'hommeaux, 2008).

Hence, while it is not the ultimate user interface for RDF querying, it is still a clear step ahead from the traditional textual queries demonstrating the benefits of knowledge-enriched queries.

2.1 Incremental Query

Our task in the development of this tool was to provide to NEPOMUK users a mechanism for re-finding information where the user could also reuse her knowledge about the instances, their types and relations. To achieve this goal we developed a modular incremental end-user query construction interface based on the query-by-example paradigm (Zloof, 1977).

The crux of this paradigm is to create a query mechanism analogous to the mental model that the user has about the representation of data. For the relational database model, Zloof (Zloof, 1977) used a two-dimensional table. The query is formulated filling the blank spaces of the table with examples of the solution. The results are given based on the records that match the pattern of this table.

Considering the graph properties of the RDF model (Pérez, 2006), our solution was based on allowing the user to create a graph visually and incrementally, which then is translated to a graph-matching query language. Analogous to what was done by Zloof, we are giving to the user a visual model where she can provides examples of the reality it wants to get. The incremental feature of our solution is that each new node added to the graph, the user is able to view the intermediate result of her query.

Below is an example of how the user can make a query using a graph structure. Suppose you want to get all emails sent by John. This query can be made in 3 steps:

Figure 1: Retrieve all instances of type Email.

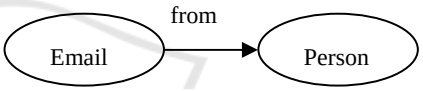

Figure 2: Retrieve all instances of type Email that have been sent by instances of type Person.

Figure 3: Retrieve all instances of type Email that have seen sent by instances of type Person whose name is John.

This graph would be translated to SPARQL as:

```
SELECT ?s WHERE {
  ?s <http://ontologies.opendfki.de/repos/
ontologies/pim/pimo#from> ?o.
  ?s <http://www.w3.org/1999/02/22-rdf-synta
x-ns#type> <http://ontologies.opendfki.de/r
epos/ontologies/pim/pimo#Email> .
  ?o <http://www.w3.org/1999/02/22-rdf-synta
x-ns#type> <http://ontologies.opendfki.de/r
epos/ontologies/pim/pimo#Person> .
  ?o <http://ontologies.opendfki.de/repos/on
tologies/pim/pimo#name> "John" .
}
```

As we can see in this translation, creating such a query in a traditional textual query interface demands from the users a full comprehension of the data structure and the query language syntax, therefore leading to a high cognitive load for less experienced users.

Our tool is able to query all instances of a given type, possibly restricting instances through their relations with other instances, which in turn may also be related to other instances, and so on, recursively. Since NEPOMUK users only instantiate PIMO types, this kind of query is enough to allow them to find them, for example, to find the author of a presentation mentioned in a given email .

While not being able to build completely arbitrary queries, the allowed ones enable the user to leverage her knowledge about the ontology in the incremental query construction interface. The system recommends only valid types and relations for the user, thus constraining her to formulate valid queries within the domain of the instances. In short, this system allows the user to:

- apply the knowledge about her PIMO,
- formulate a query visually without recurring to a textual language,
- formulate a query step-by-step

2.2 Proposed Interface

As discussed before, our system is based on a complex semantic architecture that establishes an integrated Semantic Desktop. However, the functionality of its architecture would be blurred by the conventional user interface paradigm or by ordinary developers' semantic interfaces present on most of semantic systems so far.

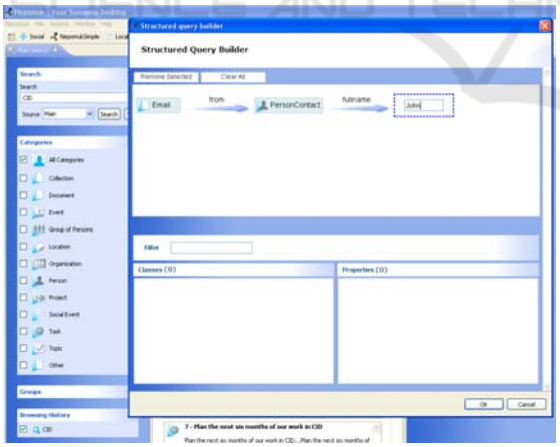

Figure 4: The structured query builder interface in NEPOMUK.

Our interface is divided into three main canvases: the construction area, the instances container and the properties/relations container (Figure 4).The graphic representation of the instances and relations is straightforward and similar to the modelling language. We have a box representation for the instances containing its name and an icon (e.g. the "Email" box at the left of Figure 5). The incoming and outgoing relations are represented by oriented arrows (e.g. the "from" arrow in Figure 5). Note that both instances and properties may be an unlabeled item representing "any type" and "any relation".

For the literal values required we use the HTML text input box. And to structure the query visualization we incrementally build a horizontal tree where the branches are visually grouped by a curly bracket. This arrangement allows indefinitely number of relations on the query, moreover, it's visually clean and of easy recognition.

Figure 5: The graphic representation of a constructed query.

For the query construction interactions steps, the user begins with a node of any type. This node is select in the construction canvas and the possible recommendations are listed in their respective canvas. Optionally, the user can filter out some items using a dynamic text filter. Once the user clicks on a type, the selected node is set to be of the clicked type. Clicking on a relation adds both, the relation and a new target node or a literal input text box.

Our first designs focused on the core functionalities of the incremental query construction and the recommendations algorithms having the features and usability fulfilling the user needs within the NEPOMUK project context. In this context we assume that the user is trying to apply her knowledge to reduce the number of results and to raise the precision of her query. The neatness of the interface, the usability and the clear-cut interactions are possible due to the underlying recommendation system discussed in the next session. Moreover, the interface implementation uses HTML, Javascript and Cascading Style Sheets (CSS) to style all components. This simple implementation brings the benefit of modularity of the interface and easy design tailoring possibilities.

2.3 Type and Relation Recommendations

We argued for the usefulness of type and relation recommendations in the user interface. In order to control the amount of information and options that have to be processed by the user, good

recommendations are essential. In case of sufficiently large ontologies, a type can have hundreds of possible (sub- or super-) types. This occasionally large list can be reduced to:

1. direct super- / subtypes
2. only the types that actually have instances
3. closest super- / subtypes that have instances
4. only meaningful types
5. etc...

However, reducing the number of options per step to a handy list, even if then the number of steps for the complete query construction increases is a desirable goal. The proposed keyword filter is a first helpful feature to handle such large collections of options.

On top of that, ordering can push more relevant types to the top of the list and thus push them faster into the awareness of the user. Such an ordering could be based on:

i. number of according instances
ii. utility estimation metrics
iii. usage frequency during previous queries
iv. etc...

The fact that the subtypes and subrelations form a hierarchy is exploited to reduce the number of choices per step. While (1) decomposes any level of super- or subtypes into a single step, (2) and (3) reduce this number to useful target types. However, if all intermediate types have instances, this leads to the same number of steps.

3 IMPLEMENTATION

We decided for a two layer architecture which separates the upper presentation layer from the lower query construction model layer. The lower layer wraps the data and its schema, and implements the iterative query construction based on the given data and schema. The upper presentation layer uses the functions of the lower layer to provide query construction options to the user. It collects and forwards feedback to the query construction layer in order to proceed with to the next construction step. This separation allows for easily plugging different presentation layer implementations (i.e. Graphical User Interfaces) on top of the same query construction layer. Further, the GUI does not need any knowledge about the data itself, it only needs to know the idea of the graphical query and its construction process. The query construction in return does not need to care about presentation of recommendations and graphical queries.

For the implementation of the proposed interface we adopted the approach of add semantic annotations in the HTML code to define the behaviour of the interface widgets. To that end, we used the Prototype[4] library, which allows us to select elements in the DOM tree by their class attribute values - by its CSS- and link operations to interface events like *onclick*, *onmouseover*, *onkeyup*, etc., of each selected HTML elements. This technique enables us to create dynamical interfaces for direct manipulation.

The query construction model layer provides two services to the upper layer. Firstly, this is the recommendation of types and relations. Secondly, this is the actual modification of the current structured query based on the user's action, as it was described in Section 2.1. These actions are:

- set the type of a resource
- set the type of a relation
- add a relation to a resource

After each of these actions, the current structured query object is updated and provided to the presentation layer, which in return updates the visualization of the query and the recommendations. This cleanly separates visualisation and user actions from the query construction model and the underlying ontology.

The structure finally provides a method that generates an equivalent SPARQL which is syntactically correct and uses the proper ontology elements. This query is then evaluated against the stored data.

4 RELATED WORK

Several visual query systems (VQS) were proposed since the QBE (Zloof, 1977) developed in 1970. Below, we will discuss some significant works in this area..

Harth et al. (Harth, 2006) sketch the piece-by-piece construction of a SPARQL query, and the possible visualization of these pieces. This is more a graphical notation than a query construction system. However, it is a remarkable early visualization approach of RDF queries. Due to the piece-wise graphical translation, this notation contains the same technical complexity and terminology as SPARQL, which we want to hide from the user in our system.

[4] http://www.prototypejs.org/

The iSPARQL[5] is powerful tool for specifying all kinds of SPARQL queries. However, its visual concepts are far away from the user's mental model. To formulate even a simple query, the user must know technical concepts such as variable and properties. Another tool, the NITELIGHT (Russell, 2008), is quite similar to the iSPARQL. The main difference between them lies in the visual notations adopted by each one. In spite of all the expressiveness of these tools, its visual notations are not suitable for the profile of the NEPOMUK's users.

SEWASIE (Catarci, 2004) uses an interface closer to the mental model of users, allowing them to make a limited set of queries; furthermore, the user is guided during the query formulation by a recommendation system based on the ontology behind the data domain. It is a work conceptually very similar to what we have done but little conclusive due to limited information contained in the paper.

To sum up, none of these tools were focused on the scenario of a Semantic Desktop. We designed an approach that takes into account the user's knowledge about her virtual belongings, and it is simple enough to allow naïve users to perform a query.

5 CONCLUSIONS AND OUTLOOK

In this paper we presented a simple nevertheless powerful interface to visually construct structured SELECT SPARQL queries through direct manipulation. We pondered functionality and simplicity in real user cases and scenarios in a way to provide the naïve user means to interact with the ontology underneath without the usual cognition load of a semantic querying tool. The first version of the UI has been completed and it is available within NEPOMUK. A usability test for evaluating the UI's efficiency is underway together with a full evaluation of the project and the results will be soon posted.

ACKNOWLEDGEMENTS

We wish to express our thanks to DLR and CNPq for the financial support, as well as the European Union IST fund (Grant FP6-027750, Project NEPOMUK).

[5] http://demo.openlinksw.com/isparql/

REFERENCES

Bates, M., 1989. The design of browsing and berrypicking techniques for the online search interface. Online Review, 13: 407-424.

Belkin, N. J., 1993. Interaction with texts: Information retrieval as information-seeking.

Catarci, T., Dongilli, P., Mascio, T. D., Franconi, E., Santucci, G., and Tessaris, S., 2004. An ontology based visual tool for query formulation support, 16th European Conference on Artificial Intelligence 2004.

Harth, A., Kruk, S. R., Decker, S., 2006. Graphical representation of RDF queries. Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006.

Pérez P., Arenas M., and Gutierrez C., 2006. Semantics and Complexity of SPARQL , 5th International Semantic Web Conference, ISWC 2006, Athens, Georgia, USA, November 2006.

Prud'hommeaux, E., Seaborne, A., 2008. SPARQL Query Language for RDF. W3C Recommendation, 15 Jan., 2008, http://www.w3.org/TR/rdf-sparql-query/

Russell, A., Smart, P. R., Braines, D. and Shadbolt, N. R. (2008) NITELIGHT: A Graphical Tool for Semantic Query Construction. In: Semantic Web User Interaction Workshop (SWUI 2008), 5th April 2008, Florence, Italy.

Sauermann, L., Bernardi, A., Dengel, A., 2005. Overview and Outlook on the Semantic Desktop, in: Stefan Decker, Jack Park, D.Q., Sauermann, L., eds.: Proceedings of the 1st Workshop on The Semantic Desktop at the ISWC 2005 Conference.

Teevan, J., Alvarado, C., Ackerman, M. and Karger, D., 2004. The Perfect Search Engine Is Not Enough: A Study of Orienteering Behavior in Directed Search. Proceedings of CHI 2004, pp. 415–422. ACM Press

Zloof, M. M., 1977. Query-by-example: a database language. IBM System Journal 16, 324-343, 1977.