Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins

Abraham Bernstein
Department of Informatics
University of Zurich
Switzerland
bernstein@ifi.unizh.ch

Christoph Kiefer
Department of Informatics
University of Zurich
Switzerland
kiefer@ifi.unizh.ch

ABSTRACT

Traditional semantic web query languages support a logic-based access to the semantic web. They offer a retrieval (or reasoning) of data based on facts. On the traditional web and in databases, however, exact querying often provides an incomplete answer as queries are overspecified or the mix of multiple ontologies/modelling differences requires "interpretational flexibility." Therefore, similarity measures or ranking approaches are frequently used to extend the reach of a query. This paper extends this idea to the semantic web. It introduces iRDQL—a semantic web query language with support for similarity joins. It is an extension of traditional RDQL (RDF Data Query Language) that enables the users to query for similar resources ranking the results using a similarity measure. We show how iRDQL allows to extend the reach of a query by finding additional results. We quantitatively evaluated four similarity measures for their usefulness in iRDQL in the context of an OWL-S semantic web service retrieval test collection and compared the results to a specialized OWL-S matchmaker. Initial results of iRDQL indicate that it is indeed useful for extending the reach of queries and that it is able to improve recall without overly sacrificing precision. We also found that our generic iRDQL approach was only slightly outperformed by the specialized algorithm.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval–*query formulation, search process*

General Terms

Similarity, Information Retrieval, RDQL, Matchmaking

Keywords

Similarity Measures, Semantic Web, Similarity Joins, Imprecise Queries, Service Matchmaking, Evaluation

1. INTRODUCTION

Imagine the following situation: you want to buy a used car—not just any car, but a car that has certain properties such as a minimum age, a favorite color, and a certain brand. All you have is a web interface that is connected to a large, semantically annotated database of cars, trucks, and other vehicles. When executing the query, however, you are buried in hundreds of results (or you may not get any answer as you overspecified the query). This situation is very typical. People querying the semantic web, databases, or also the web in general often find themselves either buried in results to their queries or with no results whatsoever. A common approach to handle these problems is to rank the results of a query, in the case of too many answers, or to return similar results, when no precise matches to the query exist [1, 5]. *Both solutions require a measure of similarity between queries and answers.*

One means for querying the semantic web or ontologies is RDQL (RDF Data Query Language) [19, 20] that is a query language for RDF [17] in Jena models [6]. RDQL allows the user to formulate queries which return precise results. *We extended traditional RDQL with similarity joins [7] to retrieve not only the precise results of a query but also similar ones.* In other words, our approach exploits the semantic annotation on the semantic web in conjunction with similarity measures to improve the performance of queries. It finds similar results when no precise results to a query exist. We called our approach iRDQL. The *i* stands for *imprecise* indicating that two or more resources do not have to be precisely equal, but should be considered as equal (or *similar*) with respect to their similarity value as computed by the similarity measure. The ranking of the results may be improved (or worsened) by specifying different similarity measures. Note that it is not necessarily clear which measure is best. On the contrary, the choice of the best performing similarity measure is highly context and data dependent [2, 8, 13]. We, therefore, implemented a set of such similarity measures in SimPack, our Java library of similarity measures for the use in ontologies [3].

The contribution of this paper is our proposed iRDQL framework that extends traditional RDQL. It implements a generic retrieval approach to allow the use of generic similarity elements hereby improving query performance.

The paper is organized as follows: Section 2 explains our extensions to RDQL that make use of similarity joins. In Section 3 we illustrate the usefulness of our approach with the preliminary results from an evaluation that uses a se-

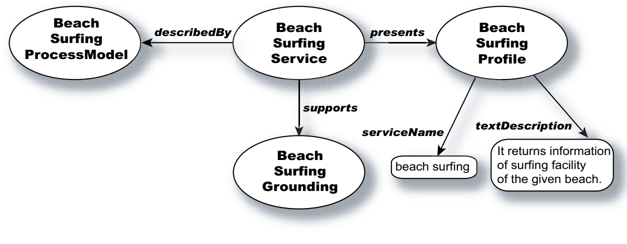


Figure 1: Partial semantic network representation of an OWL-S service instance called *Beach Surfing Service* showing its service *ProcessModel, Grounding, and Profile* (compare [14] Figure 1)

mantic web service retrieval test collection as dataset. We close the paper with some related and future work as well as conclusions.

2. iRDQL: RDQL WITH SIMILARITIES

RDQL (RDF Data Query Language) [19, 20] is a query language to formulate queries over RDF [17] in Jena models [6]. It looks at the information held in the models as RDF triples which consist of a subject, a property, and an object, where the object may again be a typed resource or a literal value. For illustration purposes refer to Figure 1 that shows a fragment of an RDF-graph of a particular OWL-S [14] service instance (*Beach Surfing Service*). Given a model that holds this instance, the following RDQL query (shortened) finds exactly the service instance shown in Figure 1.

```
SELECT ?S1,?P1
WHERE  ?S1 presents ?P1
       ?P1 serviceName ‘‘beach surfing’’
       ?P1 textDescription ‘‘It returns ...’’
```

Suppose now the main goal is NOT to find exactly this instance, but to find services which present profiles similar to the *Beach Surfing Profile*. To achieve this goal, we extended the traditional RDQL language with three additional language constructs that we call `IMPRECISE`, `SIMMEASURE`, and `OPTIONS`. The `IMPRECISE` clause defines the variables of the query whose bindings (found resources) will be matched imprecisely when executing the query. That is, they are added to the result set of the query together with their corresponding similarity value as computed by the similarity measure. The measure to compare two resources is specified by the `SIMMEASURE` clause. Here, any similarity measure implemented in SimPack [3] can be used. Additional parameters of the similarity measure are specified by the `OPTIONS` clause. The extended iRDQL syntax looks as follows (shortened):

```
SELECT      [selectClauseVariables]
FROM        [fromClause]
WHERE       [whereClause]
AND         [filterClause]
USING       [usingClause]

IMPRECISE  [impreciseClauseVariables]
SIMMEASURE [similarityMeasureClause]
OPTIONS    [similarityMeasureOptionsClause]
```

As an example, assume that the three services *Beach Surfing Service, Beach Broker Service,* and *Abstract Broker Service* together with their corresponding service profiles *Beach Surfing Profile, Beach Broker Profile,* and *Abstract Broker Profile* are stored in a Jena model. The query corresponding to our users desiderata would look like the following:

```
SELECT ?S1,?P1,?P2
WHERE  ?S1 presents ?P1
       ?P2 serviceName ‘‘beach surfing’’
       ?P2 textDescription ‘‘It returns ...’’
IMPRECISE ?P1,?P2
SIMMEASURE Levenshtein
OPTIONS IGNORECASE false THRESHOLD 0.7
```

The query looks for a service `?S1` with service profile `?P1`. It retrieves all profiles `?P2` which have the service name *“beach surfing”* and the textual description *“It returns information of surfing facility of the given beach.”*[1] The similarity between `?P1` and `?P2` is computed using the Levenshtein string edit distance (see Equation 1) and is returned with the possible combinations of `?S1`, `?P1`, and `?P2` as shown in Table 1. String comparison is case sensitive and a threshold of 0.7 is used expressing that two strings are equal if their similarity is at least 0.7 in this example.[2]

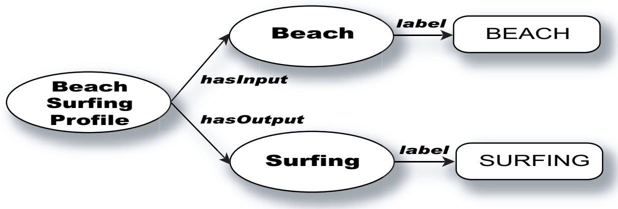


Figure 2: Partial semantic network representation of an OWL-S service profile instance called *Beach Surfing Profile*

The implementation of iRDQL consists of a pre- and postprocessor to the Jena-based RDQL engine. The preprocessor divides the iRDQL statement into independent RDQL statements which are executed by the Jena RDQL engine. The postprocessor combines the results of those subqueries, computes the necessary similarities using the specified similarity measure, and returns the results.

For this paper, we evaluated four such measures: the Levenshtein string edit distance measure, the cosine vector measure, TFIDF, and the Jensen-Shannon information divergence based similarity measure, that we all took from SimPack, our library of similarity measures [3].

The *Levenshtein* measure first takes an RDF resource (e.g. R_x) and maps it to a vector. Thus, the mapping of the RDF graph in our example shown in Figure 2 results in the vector $\mathbf{x}$ = [BeachSurfingProfile, hasInput, Beach, label, BEACH, hasOutput, Surfing, label, SURFING]^T. Then, it uses the Levenshtein string edit distance algorithm [11]

[1]In our current approach, `?P2` acts like a *target* or *template* for comparison, i.e., all retrieved service profiles `?P1` are compared against this target `?P2`.

[2]Although not applied at the moment, additional query expansion methods can be used as, for instance, described in [21] to further extend a query's search space.

S1	P1	P2	Sim
Beach Surfing Service	Beach Surfing Profile	Beach Surfing Profile	1.0
Beach Broker Service	Beach Broker Profile	Beach Surfing Profile	0.85
Abstract Broker Service	Abstract Broker Profile	Beach Surfing Profile	0.7

Table 1: Output of the iRDQL example query

that calculates the number of insert, remove, and replacement operations to transform vector $\mathbf{x}$ to vector $\mathbf{y}$ that is obtained from another resource R_y. This edit distance is defined as $xform(\mathbf{x}, \mathbf{y})$. But should each type of transformation have the same weight? Is the replacement transformation, for example, not comparable with a deleting procedure followed by an insertion procedure? Hence, we could argue that the cost function c should have the behavior $c(delete) + c(insert) \geq c(replace)$. Consequently, we can calculate the worst case transformation cost $xform_{wc}(\mathbf{x}, \mathbf{y})$ from $\mathbf{x}$ to $\mathbf{y}$ replacing all concept parts of $\mathbf{x}$ with parts of $\mathbf{y}$, then deleting the remaining parts of $\mathbf{x}$, and inserting additional parts of $\mathbf{y}$. The worst case cost is then used to normalize the edit distance, resulting in

$$sim_{levenshtein}(R_x, R_y) = \frac{xform(\mathbf{x}, \mathbf{y})}{xform_{wc}(\mathbf{x}, \mathbf{y})} \quad (1)$$

The *cosine* measure aligns two vectors $\mathbf{x}$ and $\mathbf{y}$ along their properties, like, for instance, in the following example where $\mathbb{R}_x$ and $\mathbb{R}_y$ denote sets of (RDF) properties declared on resources R_x and R_y.

$$\mathbb{R}_x = \{type, name\} \Rightarrow \mathbf{x}' = \begin{pmatrix} 0 \\ name \\ type \end{pmatrix} \Rightarrow \mathbf{x} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\mathbb{R}_y = \{type, age\} \Rightarrow \mathbf{y}' = \begin{pmatrix} age \\ 0 \\ type \end{pmatrix} \Rightarrow \mathbf{y} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The measure computes the similarity of $\mathbf{x}$ and $\mathbf{y}$ as the cosine of the angle between the two vectors. In equation 2, $||\mathbf{x}||_2$ is the L^2-norm, thus $||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$.

$$sim_{cosine}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||_2 \cdot ||\mathbf{y}||_2} \quad (2)$$

The *TFIDF* measure (term frequency–inverted document frequency) uses a corpus of text documents generated from all resources in the knowledge base. A traditional TFIDF-based information retrieval algorithm [1] is applied to compute the similarity between a query document and another document of the corpus.

The *Jensen-Shannon information divergence* based similarity measure (JSD) is based on the Kullback-Leibler distance d_{kl} [12] which measures the relative entropy of p with respect to q where p and q are probability distributions of some random variable X. The Kullback-Leibler distance is not symmetric because of $d_{kl}(p, q) \neq d_{kl}(q, p)$. To obtain a symmetric, non-negative, and additive measure, the Jensen-Shannon information divergence is given, as defined in [10], by

$$sim_{jsd}(p, q) = \frac{1}{2 \log 2} \sum_{x \in X} h(p(x)) + h(q(x)) - h(p(x) + q(x)) \quad (3)$$

where $h(x)$ is the entropy loss function $h(x) = -x \log x$. Like TFIDF, this measure makes use of the same corpus of documents and corresponding term-by-document probability matrix containing a term's probability of occurrence ($p(x)$ and $q(x)$ in Equation 3) in any document of the collection to compute the similarity of two documents. In the next section, we evaluate the results obtained with iRDQL in combination with all four similarity measures and a particular semantic web service test collection.

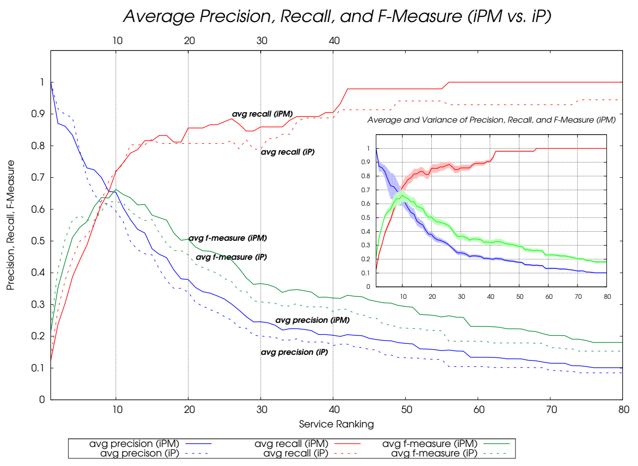


Figure 3: Precision, recall, and f-measure for all queries averaged for both query styles (*iP* vs. *iPM*)

3. EXPERIMENTAL EVALUATION

To evaluate the performance of iRDQL, we have chosen the `OWL-S-TC-v1` [9] semantic web service retrieval test collection as the knowledge domain for our experiments. This collection specifies a set of 406 OWL-S services of six different domains (i.e., *communication, economy, education, food, medical,* and *travel*). The collection is intended to support the evaluation of the performance of OWL-S semantic web service matchmaking algorithms. For each domain, the collection specifies a number of queries along with a set of relevant answers.

Inspecting the test collection, we found that both the service profiles (what the service does) and the service process models (how the service works) capture a suitable amount of information to run our evaluation with the chosen similarity measures. We, therefore, generated two iRDQL statements for each query of the test collection: one applying the similarity join to the *service **profile*** (called *iP*) and a second one applying it to the *service **profile** as well as the service's process **model*** (called *iPM*). This allowed us to compare the retrieval/matchmaking performance with varying numbers of similarity joins: *none* in the benchmark setup finding exactly one correct result, *one* in the case of *iP*, and *two* in the case of *iPM*.

3.1 Different Types of Similarity Joins

To compare the performance of different types of similarity joins, we used the similarity algorithms introduced in Section 2. Figure 3 shows average precision, recall, and f-measure for both query styles (*iP* vs. *iPM*) using the Levenshtein similarity measure. The numbers on the x-axis express the ranking of the services in the query's result set. The services within the result set are ranked in descending order of similarity to the query service (highest ranking indicating most similar). We graphed the retrieval performance for a maximum of 80 retrieved services, thus focusing on the most interesting range of the result set. The precision of the iRDQL query is 1.0 for a result set of size one indicating that, on average, all of the services inside this result set are correct answers. The overall trend of precision is decreasing since the result set is constantly growing until its size reaches the total number of services of the test collection. The recall of the iRDQL query constantly increases as additional (relevant) services are added to the result set.

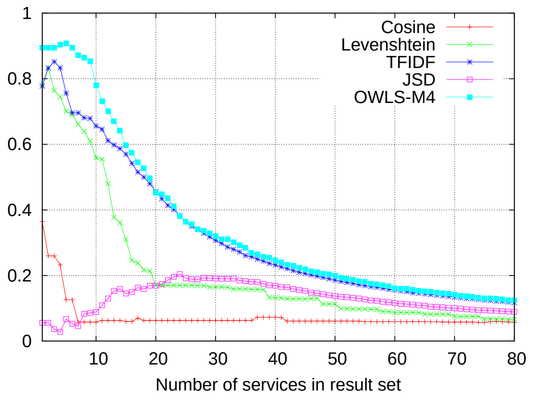
The behavior observed in Figure 3 illustrates the usefulness of our approach. In each of the domains, an exact query (i.e., an RDQL query with no similarity extension) yields exactly one result: the only perfect match. While this result has 100% precision it has a rather poor recall. The use of the imprecision extension for RDQL allowed us to simply extend the reach of the query and find additional correct matches without (at least initially) overly sacrificing precision. Actually, as a comparison of both query styles shows, an increased use of similarity operators leads to better retrieval performance: *iPM* that has two similarity joins significantly outperforms *iP* (only one similarity join) on all measures as shown by a t-test (precision: $1.4e^{-19}$, recall: $9.8e^{-21}$, f-measure: $5.7e^{-19}$). These results are consistent across all four similarity measures (*cosine:* precision: $0.2e^{-3}$, recall: $7.9e^{-23}$, f-measure: $1.1e^{-5}$; *TFIDF:* precision: $1.1e^{-44}$, recall: $1.1e^{-58}$, f-measure: $8.1e^{-60}$; *JSD:* precision: $3.6e^{-48}$, recall: $2.3.e^{-61}$, f-measure: $7.4e^{-62}$). We omitted the graphs for the other three measures due to space constraints.

3.2 iRDQL vs. OWLS-M4

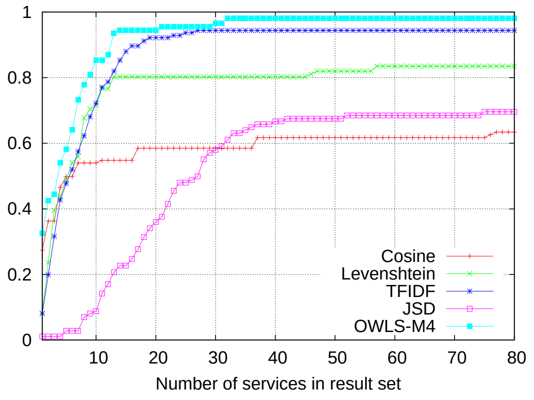
Figures 4(a) through 4(c) compare precision, recall, and f-measure of the *iPM* queries for the four similarity measures together with the best performing matchmaking algorithm OWLS-M4 of the OWLS-MX hybrid semantic web service matchmaker [10].[3] This matchmaker uses both logic-based as well as information-retrieval-based matching criteria to identify services which match with a given query service. The main focus of the matcher lies on a service profile's input and output parameters. OWLS-MX uses five different matching filters: *Exact match*, *Plug-in match*, *Subsumes match*, *Subsumed-by match*, and *Nearest-neighbor match*. We depicted the results of the best performing algorithm *Nearest-neighbor match* (also called OWLS-M4) as a benchmark for iRDQL in Figures 4(a) to 4(c). OWLS-M4 combines logic-based features with the Jenson-Shannon information divergence based similarity measure that is applied to terminologically unfolded service profile input and output concepts.

As Figure 4(a) shows, both Levenshtein and TFIDF start with highest precision (except for OWLS-M4) with few returned services but then TFIDF significantly outperforms Levenshtein and all other measures. When focusing on recall (Figure 4(b)), however, the cosine measure dominates both TFIDF and Levenshtein with a very small number of returned services. The combined f-measure (Figure 4(c)) reflects these observations allowing for a lead of TFIDF and the Levenshtein measure until about 8 returned services fol-

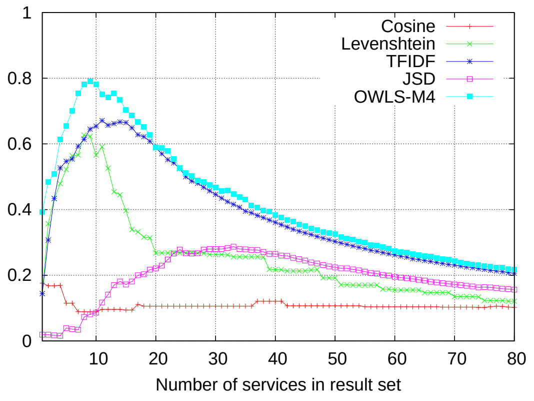
[3]OWLS-MX is available as open source at `http://www.dfki.de/~klusch/owls-mx/`



(a) Average precision of the cosine, Levenshtein, TFIDF, and JSD measure (iPM)



(b) Average recall of the cosine, Levenshtein, TFIDF, and JSD measure (iPM)



(c) Average f-measure of the cosine, Levenshtein, TFIDF, and JSD measure (iPM)

Figure 4: Comparison of the iPM-results and the specialized OWLS-MX algorithm

lowed by a clear dominance of TFIDF thereafter.

As shown by the Figures 4(a) to 4(c), OWLS-M4 outperforms all iRDQL queries in terms of precision, recall, and f-measure. However, the difference in performance compared to the TFIDF-based iRDQL statement is small. This is a success for iRDQL. *OWLS-M4 is an algorithm that was build with a lot of domain knowledge regarding OWL-S and the matchmaking task.* To that end it combines logic-based features and a focus on the services' input and output concepts boosting its precision. *In contrast, iRDQL implements a generic retrieval approach; for the matchmaking task, all we used were at most two similarity joins.* Hence, it is surprising that it performed so well compared to this specialized algorithm.

Obviously, this evaluation of iRDQL can only serve as an illustration. A thorough evaluation will have to (1) explore different combination approaches for multiple similarity measures, (2) investigate the computational consequences of using similarity joins over precise joins, (3) use additional test collections besides `OWL-S-TC-v1`, and (4) investigate the quality of queries constructed by developers in the light of these new constructs.

4. RELATED AND FUTURE WORK

Our iRDQL implementation was inspired by Cohen's approach [7] of using similarity joins to solve the problem of combining information from different databases. He uses a standard TFIDF scheme [1] to compute the similarity between columns from different tables. The main difference to our approach is that we are not dealing with flat tables (i.e., data in first normal form) but with complex (ontologized) objects (i.e., data stored in NF^2—non first normal form [18]).

The *literature on service matching* is too elaborate for a complete discussion in addition to the elaborations on OWLS-MX [10]. Most notably, Di Noia and colleagues [15] discuss a purely logic-based approach that matches service demands and supplies based on their explicit normal form, i.e., demands and supplies are terminologically unfolded into their names, number restrictions, and universal role quantifications. The Classic-based [4] matchmaking algorithm then distinguishes between *potential* and *partial* matches of demands and supplies, i.e., matches with no conflicts and matches with conflicting properties of demands and supplies. This approach differs from ours in its strong reliance on reasoning rather than similarity measures. In the future, we intend to investigate whether we can capture the algorithm's functionality with a preprocessing of iRDQL queries which includes logical reasoning elements. More generally, we plan to investigate different approaches of improving iRDQL's performance with both logical reasoning and domain knowledge. This might, among other things, allow iRDQL to meet OWLS-MX on equal footing.

We know of two projects that focus on *the implementation of similarity measures*: *SecondString*[4] that implements a set of approximate string-matching algorithms, and the *SimMetrics*[5] project that presents a large set of similarity/distance metrics for information integration tasks. We intend to integrate those libraries into SimPack, our own Java library of similarity measures for the use in ontologies [3], making them available in iRDQL. This is especially pressing as our findings were not as conclusive about which measure was best reflecting the findings of [2, 8] and calling for further research to find the best performing similarity measures in different task domains.

Last but not least, the current syntax of an iRDQL query is rather cryptic. Therefore, we plan to further simplify its syntax and also switch to SPARQL [16], the new proposed w3c standard, setting the stage for future *iSPARQL*.

[4]More information about SecondString can be found at `http://secondstring.sourceforge.net/`

[5]SimMetrics can be found at `http://www.dcs.shef.ac.uk/~sam/simmetrics.html`

5. CONCLUSIONS

In this paper we presented our approach of extending RDQL with similarity joins to find not only precise matches to a query but also similar ones. Inspired by the work of Cohen [7], we extended normal RDQL with similarity joins, thus, using traditional information-retrieval-based similarity measures in combination with RDQL to improve a query's precision and recall. We found that an increased usage of similarity operators leads to better, overall retrieval performance. We showed that our generic IR-based approach is outperformed only to a very small extend by OWLS-MX [10], a matchmaker for OWL-S services, that uses domain knowledge and extensive logical reasoning. Using iRDQL with a test collection of OWL-S services [9], we found that TFIDF and the Levenshtein string edit distance were best performing. In accordance to Cohen's work we claim that the approach presented in iRDQL provides the basis for combining the strengths of logic-based precise querying and similarity-based retrieval.

6. REFERENCES

[1] R. Baeza-Yates and B. d. A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999.

[2] A. Bernstein, E. Kaufmann, C. Bürki, and M. Klein. How Similar Is It? Towards Personalized Similarity Measures in Ontologies. In *7. Internationale Tagung Wirtschaftsinformatik*, February 2005.

[3] A. Bernstein, E. Kaufmann, and C. Kiefer. SimPack: A Generic Java Library for Similarity Measures in Ontologies. Technical report, University of Zurich, Department of Informatics. `http://www.ifi.unizh.ch/ddis/staff/goehring/btw/files/ddis-2005.01.pdf`, 2005.

[4] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A Structural Data Model for Objects. In *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 58–67, New York, NY, USA, 1989. ACM Press.

[5] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[6] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. Technical report, HP Labs, 2003.

[7] W. W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000.

[8] D. Gentner and J. Medina. Similarity and the Development of Rules. *Cognition*, 65:263–297, 1998.

[9] M. Klusch. OWLS-TC-v1: OWL-S Service Retrieval Test Collection. `http://projects.semwebcentral.org/projects/owls-tc/`, 2005.

[10] M. Klusch, B. Fries, M. Khalid, and K. Sycara. OWLS-MX: Hybrid Semantic Web Service Retrieval. Arlington, VA, USA, 2005.

[11] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[12] J. Lin. Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[13] P. W. Lord, R. D. Stevens, A. Brass, and C. A. Goble. Investigating semantic similarity measures across the gene ontology: the relationship between sequence and annotation. *Bioinformatics*, 19(10):1275–83, 2003.

[14] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. `http://www.w3.org/Submission/OWL-S/`, November 2004.

[15] T. D. Noia, E. D. Sciascio, F. M. Donini, and M. Mongiello. A System for Principled Matchmaking in an Electronic Marketplace. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 321–330, New York, NY, USA, 2003. ACM Press.

[16] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`, 2005.

[17] RDF Core Working Group. RDF Primer. `http://www.w3.org/TR/rdf-primer/`, 2004.

[18] H. J. Schek and M. H. Scholl. The Relational Model With Relation-Valued Attributes. *Information Systems*, 11(2):137–147, 1986.

[19] A. Seaborne. Jena Tutorial–A Programmer's Introduction to RDQL. `http://jena.sourceforge.net/tutorial/RDQL/`, 2004.

[20] A. Seaborne. RDQL–A Query Language for RDF. `http://www.w3.org/Submission/RDQL/`, 2004.

[21] E. M. Voorhees. Query Expansion Using Lexical-Semantic Relations. In *SIGIR '94: Proceedings of the 17th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 61–69, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

APPENDIX

A. COMPLETE iRDQL QUERY

```
SELECT ?s, ?p, ?p1, ?m, ?m1
WHERE (?s rdf:type service:Service)
      (?s service:supports ?g)
      (?g rdf:type grounding:WsdlGrounding)
      (?g service:supportedBy ?s)
      (?s service:presents ?p1)
      (?p1 rdf:type profile:Profile)
      (?p1 service:isPresentedBy ?s)
      (?s service:describedBy ?m1)
      (?m1 rdf:type processmodel:ProcessModel)
      (?m1 service:describes ?s)
      (?p rdf:type profile:Profile)
      (?p service:isPresentedBy ?s1)
      (?s1 rdf:type service:Service)
      (?p profile:serviceName ?sn)
      (?p profile:textDescription ?sd)
      (?p profile:hasInput ?in1)
      (?p profile:hasOutput ?out1)
      (?in1 processmodel:parameterType ?in1PT)
      (?in1 rdfs:label ?in1L)
      (?out1 processmodel:parameterType ?out1PT)
      (?out1 rdfs:label ?out1L)
      (?m rdf:type processmodel:ProcessModel)
      (?m service:describes ?s2)
      (?s2 rdf:type service:Service)
      (?m processmodel:hasProcess ?x)
      (?x rdf:type processmodel:AtomicProcess)
      (?x processmodel:hasInput ?in2)
      (?x processmodel:hasOutput ?out2)
      (?in2 processmodel:parameterType ?in2PT)
      (?in2 rdfs:label ?in2L)
      (?out2 processmodel:parameterType ?out2PT)
      (?out2 rdfs:label ?out2L)

AND ?sn =~ /beach surfing/i
AND ?sd =~ /It returns information.../i
AND ?in1 =~ /_BEACH/
AND ?out1 =~ /_SURFING/

USING
 service for <http://Service.owl#>
 profile for <http://Profile.owl#>
 processmodel for <http://Process.owl#>
 grounding for <http://Grounding.owl#>

IMPRECISE ?p, ?p1
IMPRECISE ?m, ?m1

SIMMEASURE Levenshtein
OPTIONS IGNORECASE false THRESHOLD 0.7;
```

Abraham Bernstein is Associate Professor and heads the Dynamic and Distributed Information Systems Group in the Department of Informatics at the University of Zurich, Switzerland. Before joining the University of Zurich he was Assistant Professor in the Department of Information, Operations and Management Sciences at New York University's Leonard N. Stern School of Business and received a Ph.D. from MIT's Sloan School of Management. His research interests include the various aspects of supporting dynamic (intra- and inter-) organizational processes with a special focus on machine learning, the semantic web, and pervasive computing.

Christoph Kiefer is a PhD. candidate at the Dynamic and Distributed Information Systems Group in the Department of Informatics at the University of Zurich, Switzerland. He holds a master's degree in computer science obtained at ETH Zurich, Switzerland. His research interests include the exploration and application of similarity measures to various application domains including the semantic web, query formulation, and software clone detection.