# iRDQL - Imprecise Queries Using Similarity Joins for Retrieval in Ontologies

**Abraham Bernstein** and **Christoph Kiefer**
Department of Informatics
University of Zurich
Winterthurerstr. 190, 8057 Zurich
{bernstein,kiefer}@ifi.unizh.ch

## Abstract

Traditional semantic web query languages support a logic-based access to the semantic web offering a retrieval of data based on facts. On the traditional web and in databases, however, exact querying often provides incomplete answers as queries are over-specified or the mix of multiple ontologies/modelling differences requires "interpretational flexibility." This paper introduces iRDQL, a semantic web query language with support for similarity joins. It is an extension to RDQL that enables the user to query for similar resources in an ontology. In the context of an OWL-S matchmaking test collection we show that iRDQL is indeed useful for extending the reach of the query improving recall without sacrificing too much precision.

## 1 Introduction

Imagine the following situation: you want to buy a used car, which has certain properties such as a minimum age, a favorite color, etc. When executing the query on a semantically annotated database of cars, however, you are buried in hundreds of results (or you may not get any answers as you over-specified the query). This situation is very typical. People querying the semantic web, databases, or also the web in general frequently 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 [Baeza-Yates and Ribeiro-Neto, 1999]. To achieve the same goal for the semantic web *we extended RDQL*, a query language for RDF [RDF Core Working Group, 2004] in Jena models [Seaborne, 2004], *with similarity joins* [Cohen, 2000] *to retrieve not only the precise results of a query but also similar ones.* Thus, our approach, called iRDQL for *imprecise* RDQL, exploits the semantic annotation on the semantic web in conjunction with a similarity measure to improve the ranking of the results of queries for such resources. Hence, similar results may be found in the case where no precise results to a query exist. Additionally, if too many results are found, iRDQL uses similarity measures to improve the ranking of the results.

## 2 iRDQL: RDQL with Similarity Joins

RDQL (RDF Data Query Language) is a query language to formulate queries over RDF [RDF Core Working Group, 2004] in Jena models [Seaborne, 2004]. For example, it allows to formulate a query that will retrieve all OWL-S [Martin *et al.*, 2004] service resources that have profiles which exactly match a profile called *Beach Surfing Profile* (the result being the *Beach Surfing Service*). But what if a user would like to find all services that have a profile similar to the *Beach Surfing Profile* to get a larger variety of services? To that goal we extended the RDQL language with two additional language constructs `IMPRECISE` and `SIMMEASURE`. The `IMPRECISE` clause defines the variables of the query whose bindings (found resources) should 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, our Java library of similarity measures can be used [Bernstein *et al.*, 2005]. The query corresponding to our users desiderata would look as follows (shortened):

```
SELECT     ?S1,?P1,?P2
WHERE      ?S1 presents ?P1
           ?P2 serviceName "beach surfing"
IMPRECISE  ?P1,?P2
SIMMEASURE Levenshtein
```

The query looks for a service `?S1` with profile `?P1` and retrieves all profiles `?P2` that have the service name *"beach surfing"*. It then computes the similarity between `?P1` and `?P2` returning the following result table:

| S1 | P1 | P2 | Sim |
|---|---|---|---|
| Beach Surfing Service | Beach Surfing Profile | Beach Surfing Profile | 1.0 |
| Beach Broker Service | Beach Broker Profile | Beach Surfing Profile | 0.5 |
| ... | ... | ... | ... |

## 3 Experimental Evaluation

For our evaluation we chose the `OWL-S-TC-v1`[1] service retrieval test collection, which specifies a set of 406 OWL-S [Martin *et al.*, 2004] services and 9 queries with their "correct" answers to evaluate service matchmaking algorithms. For each query, we generated an iRDQL statement with one

[1] The test collection is freely available at `http://projects.semwebcentral.org/projects/owls-tc/`

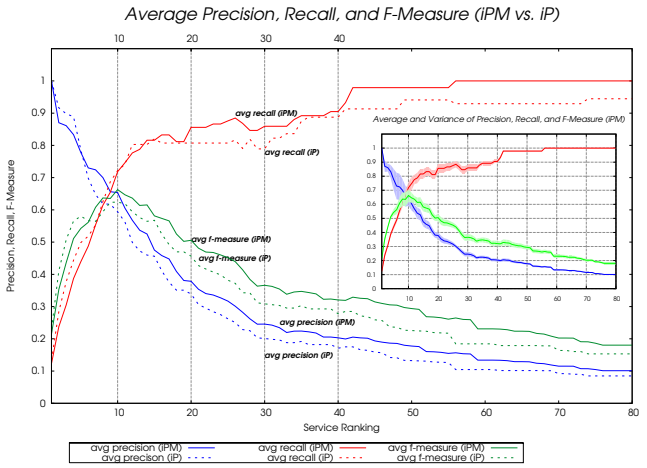


Figure 1: The figure shows precision, recall, and f-measure for all 9 Levenshtein-based [Levenshtein, 1966] queries averaged for both query styles (*iP* vs. *iPM*). The smaller subfigure illustrates the variance of the averaged *iPM*-queries.

or two similarity joins: one applying the similarity join only to the *service profile* (called *iP*-query style) and a second one applying it to the *service profile and the service's process model* (*iPM*-query style). We executed each resulting iRDQL query against all services belonging to the same domain as the query service and ranked the results according to their similarity (in the case of *iP*, or average similarity for *iPM*).

Figure 1 shows average precision, recall, and f-measure for both query styles (*iP* vs. *iPM*). The numbers on the x-axis express the ranking of the services in the query's 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 that 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 domain. The recall of the iRDQL query increases as additional (relevant) services are added to the result set of the query.

The behavior in Figure 1 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 rather poor recall. The use of the imprecision extension for RDQL allowed us to extend the reach of the query to find additional correct matches without (at least initially) overly decreasing recall. As a comparison of both query styles shows, an increased use of similarity operators leads to better retrieval performance: *iPM* significantly outperforms *iP* on all measures as shown by a t-test (precision: $1.4e^{-19}$, recall: $9.8e^{-21}$, f-measure: $5.7e^{-19}$). Obviously, this evaluation of iRDQL can only serve as an illustration. A thorough evaluation will have to consider (1) an evaluation of the approach's robustness towards the use of different similarity measures [Bernstein *et al.*, 2005], (2) explore different combination approaches for multiple similarity measures, and (3) investigate the computational consequences of using similarity joins over precise joins.

## 4 Conclusions

In this paper we introduced our approach of extending RDQL with similarity joins to find not only precise matches to a query but also a set of similar matches. Our implementation was inspired by Cohen's approach [Cohen, 2000] of using similarity joins to solve the problem of combining information from different databases. He uses a standard *tf-idf* scheme [Baeza-Yates and Ribeiro-Neto, 1999] 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 [Schek and Scholl, 1986]). This calls for a deeper investigation of similarity measures that rely on the structure of an ontology. Thus, further research is necessary to find the best performing similarity measure for the combination of iRDQL and OWL-S semantic web service descriptions. 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.

## References

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

[Bernstein *et al.*, 2005] Abraham Bernstein, Esther Kaufmann, and Christoph 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.

[Cohen, 2000] William W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. *ACM Trans. Inf. Syst.*, 18(3):288–321, 2000.

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

[Martin *et al.*, 2004] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. `http://www.w3.org/Submission/OWL-S/`, November 2004.

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

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

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

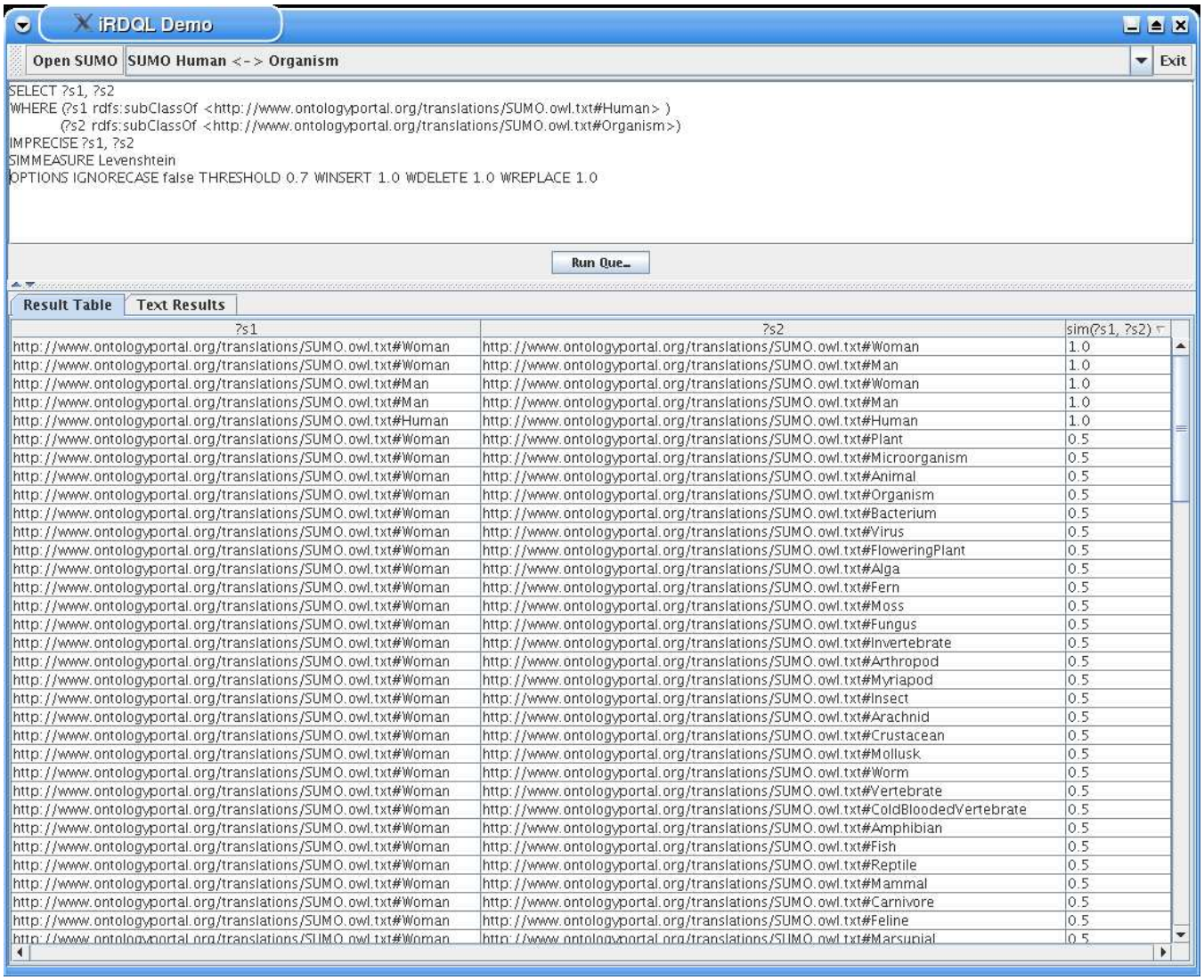


Figure 1: The figure shows the graphical user interface of our iRDQL query tool. The user can enter queries in the text field. In this particular example, every concept of type *human* is compared to every object of type *organism*. The results are ranked by their degree of similarity and displayed to the user in the table below.

## Brief Explanation of Demonstration

In the demonstration we will show our current implementation of iRDQL. It includes a graphical user interface that allows the user to load several ontologies in which similar objects are to be found. The user can enter iRDQL queries in a text field which subsequently get processed by the query engine. The results can then either be presented as the textual query evaluation engine output or in the form of a table (as shown in Figure 1), where each of the query-variables and the similarity measure's output get entered into their own columns. The table can be sorted by each column by clicking into its header.

All similarity measures implemented in SimPack, our generic Java library of similarity measures for the use in ontologies can be used in the iRDQL queries. Thus, the demonstration will also provide the user of the tool with a more detailed view and explanation of SimPack and its implementation. The tool is still under development will, therefore, still change in appearance and have additional functionality.