# SPARQLByE: Querying RDF data by example

Gonzalo Diaz
University of Oxford
gonzalo.diaz@cs.ox.ac.uk

Marcelo Arenas
PUC Chile
marenas@ing.puc.cl

Michael Benedikt
University of Oxford
michael.benedikt@cs.ox.ac.uk

## ABSTRACT

Semantic Web technologies such as RDF and its query language, SPARQL, offer the possibility of opening up the use of public datasets to a great variety of ordinary users. But a key obstacle to the use of open data is the unfamiliarity of users with the structure of data or with SPARQL. To deal with these issues, we introduce a system for querying RDF data by example. At its core is a technique for reverse-engineering SPARQL queries by example. We demonstrate how reverse engineering along with other techniques, such as query relaxation, enables our system, SPARQLByE, to guide users who are unfamiliar with both the dataset and with SPARQL to the desired query and result set.

## 1. INTRODUCTION

Semantic Web systems provide programmatic interfaces for endusers to access data with standard formats. An enormous range of data of broad interest is available over these interfaces, leading to the possibility of a huge increase in the number of active "data queriers". A chief bottleneck for exploiting the availability of this data is the query interface. The data model views data as collections of RDF triples, making heavy use of web identifiers (URIs). This is a fairly low-level representation that is not easy for users to deal with by navigating or browsing the data one item at a time. The Web APIs also expose a powerful declarative query language, SPARQL, which allows users to pose queries that combine and filter information. Making use of SPARQL still requires familiarity both with the way data is represented as triples and with the query language itself.

An alternative paradigm for querying is *query-by-example*, where users present examples of what they want, and the system generalises them [2, 5, 6]. Querying by example is particularly attractive in an open data setting, since it eliminates the need to understand the structure of data as well as the features of SPARQL needed to express a query. Even users familiar with SPARQL and with a given dataset may prefer to explore the data via example and have the system suggest generalisations.

EXAMPLE 1. *Consider* DBpedia*, a public repository of RDF triples which represent knowledge extracted from Wikipedia, and consider a user who would like to extract a list of all Spanish-speaking countries. Although DBpedia has a free public SPARQL endpoint available (see:* `http://dbpedia.org/sparql`*), this interface does not provide much in the way of help for a user who is not familiar with the syntax and semantics of SPARQL queries. Moreover, formulating the appropriate SPARQL query would be challenging even for an experienced SPARQL user, as intimate knowledge of the DBpedia ontology and specific URIs is necessary to express the correct required triples.*

In contrast, the query-by-example paradigm allows users to specify their information need using positive and negative examples.

EXAMPLE 2. *Continuing with the previous example, consider now that the user has access to a query-by-example system like SPARQLByE. In this case, the user might indicate Chile, Bolivia, Venezuela, and Spain as positive examples, while indicating Brazil and Angola as negative examples. The system would then be capable of guessing that the user is interested in the following query, and presenting both the query and its results to the user:*

```
SELECT * WHERE {
?s  <http://www.w3.org/1999/02/22-rdf-syntax
-ns#type>  <http://dbpedia.org/ontology/Country> .
?s  <http://purl.org/dc/terms/subject>
    <http://dbpedia.org/resource/Category:
Spanish-speaking_countries_and_territories> }
```

*Each URI in the query is of the form* `<http:// ... >`*. In particular,* `<http://dbpedia.org/ontology/Country>` *is the URI for the concept* `Country`*, so in the first triple of the query we are asking to store in the variable* `?s` *all the countries in DBpedia. Moreover, the second triple asks for all the Spanish-speaking countries and territories. Finally, these two triples are combined by means of the period symbol in SPARQL, which essentially represents a join operator — it is referred as the* `AND` *operator in this paper. Thus, this combination is used to extract the list of all Spanish-speaking countries in DBpedia. Notice that the URIs and triples used in the query are not trivial to remember, even for experienced SPARQL users.*

We present a system for querying Semantic Web data by example. Our system, SPARQLByE (SPARQL By Example), allows users to obtain information without any knowledge of SPARQL. At the same time, SPARQL plays a key role in SPARQLByE behind the scenes: a core component of SPARQLByE is a reverse-engineering algorithm that abstracts user examples into a SPARQL query, which is evaluated to present answers to the user. The system supplements reverse engineering with techniques for guiding the user to further positive and negative examples.


*Proceedings of the VLDB Endowment,* Vol. 9, No. 13

**Related work.** Query-by-example dates back to the early days of relational databases [8], with the initial focus being on a specification using only positive examples. Reverse engineering of queries from positive examples has been studied for a number of query languages [6, 7]. The use of positive and negative examples within query learning has been explored in the context of XML queries [3], and later for relational data in the JIM system of [2]. Like SPARQLByE, JIM does not simply reverse engineer a query, but allows interaction with a user. But the underlying reverse-engineering algorithms for SPARQL and relational data are quite different, due to the presence in SPARQL of a command `OPTIONAL` for extracting optional information. The additional features in SPARQLByE for query-by-example, beyond reverse engineering, are tailored to issues specific to the open data setting, such as the difficulty of mapping entities to URIs. Furthermore, SPARQLByE is designed to connect to a predefined (and customisable) SPARQL endpoint, thus achieving a modular design, allowing it to be added onto existing Semantic Web infrastructure.

## 2. SPARQLByE SYSTEM OVERVIEW

SPARQLByE can be attached to any RDF dataset $D$. The main input is a set of *annotated mappings*, which are specified by the user. A mapping is analogous to a tuple in a relational setting: it consists of a set of associations of an attribute or variable name with a value. An annotated mapping is a mapping labelled as either a positive example or a negative example. A SPARQL query $Q$, when evaluated on a RDF dataset $D$, returns a set of mappings $Q(D)$ — the *result set* of $Q$ on $D$. A result set implicitly defines an annotated mapping set where the mappings in $Q(D)$ are positive examples and the mappings outside of $Q(D)$ are the negative examples. A set of annotated mappings $\Omega$ is *consistent* with a query $Q$ on $D$ if the positive mappings in $\Omega$ are in $Q(D)$ and the negative mappings in $\Omega$ are all outside of $Q(D)$. The system proceeds by discovering a query $Q$ which is consistent with the annotated examples presented by the user, and to present the full result set $Q(D)$ (the query itself need not be shown). The user may then refine the set of annotated mappings in order to improve the result set; this refinement step will be explained in detail below.

At the core of SPARQLByE is a set of *reverse-engineering algorithms*, which take a set of annotated mappings and return a SPARQL query that is consistent with the annotated examples. SPARQLByE focuses on SPARQL queries which make use of only the `AND` and `OPTIONAL` operators [1,4]. Given a set of mappings, a basic step in the system is to reverse engineer a query, evaluate it, and return the result set (or a pointer into a page of it, in case it is too large) to the user. This is the *reverse-engineering step*.

Reverse-engineering steps are interleaved with *example refinement steps*, in which a user responds to the displayed result set by refining the set of annotated mappings. SPARQLByE provides several forms of assistance to a user in the refinement step, including support for translating text descriptions to URIs (needed in constructing positive examples) and for creating pools of *candidate examples*, which the user may add to the labelled set of mappings.

Figure 1 (right) shows the architecture of the SPARQLByE system. A HTML-JavaScript front-end is backed by the Java server which is responsible for running the reverse engineering algorithms in the *reverse-engineering* module and obtaining candidate mappings in the *refinement module*. An auxiliary *entity search module* translates keyword searches to URIs.

The two main modules depend on several lower-level ones. In particular, the reverse-engineering algorithms require the execution of auxiliary SPARQL queries, which are delegated to a SPARQL endpoint; this may be either a local or a public endpoint, allowing for flexible setup.

**Reverse Engineering module.** Our reverse-engineering approach is based on [1]. We look for well-designed SPARQL queries with `AND` and `OPTIONAL` [4] that are consistent with a set of positive and negative examples, and which satisfy a natural restriction: they are *tree-like*, which means that the domains of different mappings (that is, sets of variables) should form a tree-like order. Roughly speaking, the algorithm proceeds by induction on domains starting with the the smallest ones. At each inductive step it looks to put together a set of *safe patterns* through the `AND` operator, that is, patterns which are satisfied by the positive examples, are not fit by the negative examples, and which are consistent with the absent variables (that is, NULL values) present in positive examples. We make use of a "greedy" version of the algorithm that iteratively adds patterns until it obtains a safe set. This greedy approach has the desirable property of generating relatively small reverse-engineered queries. The algorithm needs to query the data to generate potential patterns, and also to check that a candidate query does correctly fit the data. This is done via issuing SPARQL queries to the endpoint.

Although arbitrary tree-like example sets are supported in [1], the meaning of a "negative partial example" (i.e. a negative example with NULLs) is unintuitive for users – it could mean that there is no binding that matches the example exactly, or no binding that subsumes the example. Thus in SPARQLByE we allow only "full negative examples": those which mention all variables included in any positive example.

SPARQLByE allows user-driven customization of the reverse-engineering algorithm to avoid both over-fitting and over-generalization. For instance, a common scenario is that all the positive examples are associated with a concept such as "Thing" (`http://www.w3.org/2002/07/owl#Thing`) and hence the resulting reverse-engineered query hardcodes that URI within it. Users can select a set of "forbidden URIs" (e.g. from the current result set) and the reverse-engineering algorithm will then avoid generating a query containing these.

**Example Refinement module.** SPARQLByE provides several features to assist users in refining annotated examples.

The right-hand side of the user interface houses the results panel. Here, the generated query is evaluated and a selection of results is presented. A user may use this panel to judge whether the answers to the generated query are satisfactory; in particular, an answer from this panel may be marked as a negative example, forcing SPARQLByE to generate a new, more restrictive, query. Thus, the results panel also serves as a *pool for potential negative examples*.

For suggesting new positive examples, the first feature SPARQLByE provides is for entity and URI searching. Finding the correct entity identifiers (i.e. URIs) presents one of the main barriers to the formulation of meaningful SPARQL queries. Entity URIs are notoriously hard to remember; even the use of standard URI prefixes do not solve this problem, as several such prefixes exist and must be chosen correctly. The entity and URI search module of SPARQLByE provides a simple keyword search for users to find appropriate URIs.

EXAMPLE 3. *Figure 1 (left) shows the result of inputting* `tennis` *into the entity search module (left-most panel of the interface). The result is a list of entities, where, for each, their* `label` *is presented, along with a* type*. The buttons allow for assigning the URI as a positive or negative example (note that this will only apply if the examples have arity 1).*

The URI and entity search allows users to find candidate values

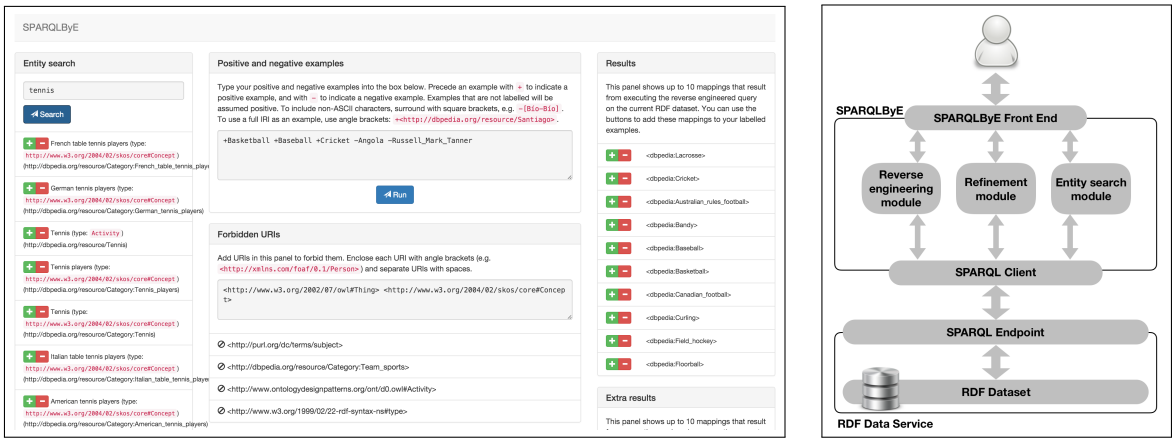


**Figure 1: SPARQLByE user interface (left) showing the result of adding labelled examples, and SPARQLByE architecture (right).**

to use in mappings. A second feature allows the user to find appropriate mappings "in bulk". Below the answers panel is the extra results panel. Here SPARQLByE generates relaxed versions of the current reverse-engineered query, by choosing a triple and removing it from the query. More precisely, given the generated query $Q$ (considered as a set of triple patterns), for each triple pattern $t$ in $Q$ the relaxed query $Q_t$ not including $t$ is evaluated. A selection of the answers of all $Q_t$ queries is shown in the extra results panel.

The purpose of the extra results panel is to serve as a *pool for potential positive examples*. This is due to the fact that the relaxed queries may include answers which are not captured by the main query $Q$, and which may be of interest to the user. Labelling one of these as a positive example would force the system to generate a new–and possibly more relaxed–query.

**SPARQL endpoint.** SPARQLByE relies on a SPARQL endpoint for its operation. The reverse engineering of SPARQL queries from positive and negative examples is achieved by generating intermediate SPARQL queries which allow the reverse-engineering module to first *produce* a candidate query, and then to *check* said query. The fact that SPARQLByE can operate on a SPARQL endpoint allows for flexible setup including: 1. using a local RDF backend for quicker access and custom indices tailored for the query loads that the reverse-engineering module will generate, 2. plugging in the SPARQLByE system into an existing SPARQL endpoint, and 3. installing SPARQLByE as a third-party client to a public SPARQL endpoint.

**Entity search.** The entity search module of SPARQLByE provides a simple way to search for URIs. Keyword searches in the entity search box are ultimately used to supply the user with a list of entities (URIs). This module allows for different techniques to be used, ranging from simple text search to using third-party semantic annotator APIs. Currently, the entity search module translates the keyword search to a SPARQL query which asks for a `uri` such that the triple (`uri`, `rdfs:label`, `X`) is in the dataset with label `X` containing each keyword as a substring.

**Demo Scenario.** The demonstration will focus on showcasing the capabilities of SPARQLByE to ease navigation and querying of a SPARQL endpoint. In contrast to a standard SPARQL query interface, SPARQLByE allows for a more interactive experience.

An obvious scenario showcasing SPARQLByE is obtaining a query capturing a class of entities.

EXAMPLE 4. *Continuing with our running example, consider the case of obtaining a list of Spanish-speaking countries. A possible first approach for a user is to input the countries* `Chile`, `Bolivia`, *and* `Venezuela` *as positive examples (see Figure 3, top frame). In this case, the corresponding reverse-engineered query asks for all* `Country` *entities, which is too general. The right hand panel has the country* `Angola`, *which can be added next as a negative example. The result is shown in the middle frame of Figure 3. The reverse-engineered query is asking for countries in South America. The problem is its exclusion of the Spanish-speaking country* `Spain`, *which may then be added as a positive example. The bottom frame of Figure 3 shows the final result set and the corresponding reverse engineered query, asking for* `Country` *entities which have the subject* `Spanish-speaking_countries_and_territories`. *Notice that although the query is intuitive, it would have been difficult for a user to arrive at the URIs and triples mentioned in the query without extensive knowledge of the DBpedia data and types.*

In the previous example, the final positive example (`Spain`) was crucial to obtaining the desired result set. However, it is not always clear what positive example to add. This problem is addressed by the Extra Results panel, which suggests positive examples.

EXAMPLE 5. *Consider a user who wishes to obtain a list of English-language movies. The user begins with the positive example* `+Dances_with_Wolves`, *and subsequently adds the negative example* `-Thirupathi`. *The resulting query, shown in Figure 2, asks for films which have the label "*`Dances with Wolves`*", which only returns one result. Clearly this is overfitting the example data. The Extra Results panel, however, has executed a relaxed query which asks for all films, and thus is able to suggest the film* `Newcastle_(film)`. *Adding Newcastle as a positive example gives the desired result set, corresponding to the query;*

```
SELECT * WHERE {
?x  <http://purl.org/dc/terms/subject>
    <http://dbpedia.org/resource/
    Category:English-language_films> .
?x  rdf:type  <http://dbpedia.org/ontology/Film> }
```

Finally, a key feature of SPARQLByE is that it is able to reverse engineer SPARQL queries which make use of the `OPTIONAL` operator. In other words, labelled examples may contain unmapped (*null*) values. The following example illustrates this scenario.

EXAMPLE 6. *Consider a user who wishes to obtain a list of authors along with their places of death, the latter being relevant only if they have passed away. If the user*

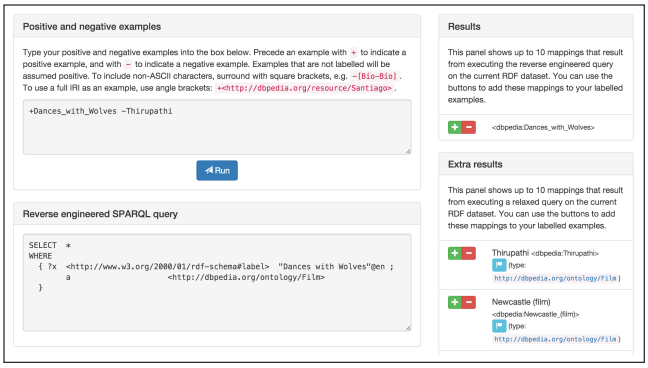


**Figure 2: SPARQLByE interface after inputting two movies as labelled examples. The Extra Results panel suggests further movies.**

*uses the positive examples* {Lewis, Oxford}, {Tolkien, Dorset}, *and* {Rowling, NULL}, *and the negative examples* {11th_Dalai_Lama, Tibet} *and* {Abbey_Lincoln, Manhattan}, *then the reverse-engineered SPARQL is:*

```
SELECT * WHERE {
?x rdf:type <http://dbpedia.org/ontology/Writer> .
?x rdf:type <http://xmlns.com/foaf/0.1/Person> .
?x rdf:type <http://dbpedia.org/ontology/Artist>
OPTIONAL {
  ?x <http://dbpedia.org/ontology/deathPlace> ?y } }
```

*The previous query thus allows SPARQLByE to present to the user the desired result set.*

## 3. CONCLUSIONS

We demonstrate here a system for querying RDF data "by example". On the one hand, a user need not write SPARQL queries; indeed, the user does not even have to understand SPARQL. On the other hand, the system exploits the power of the SPARQL language for expressing natural user queries, by inducting a SPARQL query from user examples "under the hood".

Arenas was funded by Millennium Nucleus Center for Semantic Web Research under Grant NC120004; Diaz by Becas Chile of CONICYT Chile; Benedikt's work was sponsored by the Engineering and Physical Sciences Research Council of the United Kingdom, grant EP/M005852/1.

## 4. REFERENCES

[1] M. Arenas, G. I. Diaz, and E. Kostylev. Reverse engineering SPARQL queries. In *WWW*, 2016.

[2] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive join query inference with JIM. *PVLDB*, 7(13):1541–1544, 2014.

[3] S. Cohen and Y. Y. Weiss. Learning tree patterns from example graphs. In *ICDT*, 2015.

[4] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM TODS*, 34(3), 2009.

[5] Q. T. Tran, C. Chan, and S. Parthasarathy. Query by output. In *SIGMOD*, 2009.

[6] Q. T. Tran, C. Y. Chan, and S. Parthasarathy. Query reverse engineering. *VLDB J.*, 23(5):721–746, 2014.

[7] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava. Reverse engineering complex join queries. In *SIGMOD*, 2013.

[8] M. M. Zloof. Query-by-example: The invocation and definition of tables and forms. In *VLDB*, 1975.

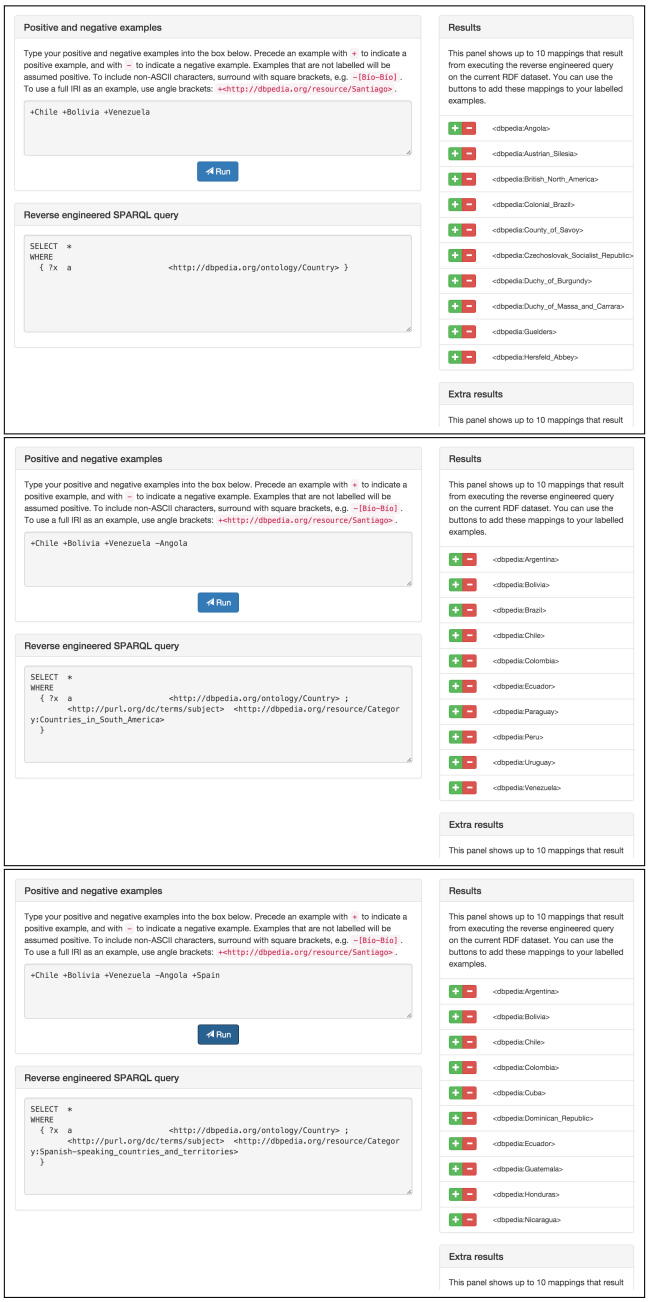


**Figure 3: SPARQLByE interface converging towards a query that retrieves Spanish-speaking countries.**