SWISH: An Integrated Semantic Web Notebook

Wouter Beek and Jan Wielemaker
`{w.g.j.beek,j.wielemaker}@vu.nl`

Dept. of Computer Science, VU University Amsterdam, NL

1 Introduction

SPARQL editors like Yasgui [6] make it easier to write and inspect their results. Notebooks like Jupyter/IPython [5] already support computer- and data scientists in domains like statistics and machine learning. There is currently not an integrated notebook solution for Semantic Web programming that combines the strengths of SPARQL editors with the benefits of notebooks. The challenge is that Semantic Web formalisms are mostly logic-based and declarative, which does not always align naturally with imperative programming paradigm. SWISH takes a different approach by presenting an integrated notebook experience to the Semantic Web programmer that uses a declarative programming paradigm (SWI) as an integration layer.

2 Requirements

An integrated Semantic Web notebook must implement the following requirements:

1. Be able to write queries in a modular way.
2. Be able to share these modules with others.
3. Be able to online collaborate with others on building, altering and combining query modules.
4. Be able to interleave SPARQL patterns and filters with functions from other programming paradigms (e.g., NLP, statistics, ML).
5. Be able to calculate query results under standardized and user-defined entailment regimes.

One of the main problems with existing SPARQL editors is that they do not allow queries to be written in a modular way. This issue is only partially solved by recent innovations like grlc [4] that allow full queries to be shared with others. The problem is that SPARQL queries cannot be easily reused as self-contained building blocks, which is possible in programming languages that allow self-contained functions to be reused by other functions. The main challenge is that SPARQL, like most other Semantic Web formalisms, follows a declarative paradigm. What is needed is a programming paradigm that allows subqueries to be naturally encapsulated in functions/predicates and modules (Requirement 1).

Once queries can be written as modular code snippets, the online notebook environment must allow these code snippets to be shared with other users (Requirement 2). Existing technologies like ShareJS ([1]) make it easy for users to collaboratively work on the same code. This functionality must be integrated into a Semantic Web notebook as well (Requirement 3).

In existing notebook systems one is not restricted to using only one standardized syntax for querying. In fact, it is very important for data scientists to be able to mix code from different programming and query languages (Requirement 4). Use cases for these are evident in many areas, for instance the ability to use Natural Language Processing (NLP) tools for fuzzy string matching (not included in the SPARQL query language) or the ability to perform a statistical test in R ([2]).

The user must be able to perform entailment under arbitrary regimes. SPARQL editors are tied to the restrictions of the entailment functionality that is exposed by contemporary triples stores. Support for standardized entailment regimes (RDF(S), OWL) is often partial and it is not always possible to specify alternative entailment regimes or domain-specific custom rules. A Semantic Web notebook should allow a user to specify her own deduction rules in addition to standardized entailment regimes (Requirement 5).

3 Implementation

SWISH is implemented as a JavaScript (browser) client that runs in combination with the Prolog-based ClioPatria triple store [7]. The client/server communication is implemented by using *Pengines* [3]. A Pengine is a Prolog engine that can be controlled through (remote) HTTP requests. It allows Prolog queries to be performed from within JavaScript. Since arbitrary programs can be executed, **SW**ISH is not limited to functionality that is provided by standardized Semantic Web query languages like SPARQL. For instance, the user can choose to perform SQL and Datalog queries in addition to SPARQL queries. She can perform entailment under a domain-specific or otherwise non-standard regime in addition to RDF(S) and OWL.

On the server-side code is executed within a sandboxed environment for security and sustainability reasons. If full/unrestricted functionality is needed at the server-side a user can deploy a remote or local **SW**ISH instance herself by cloning the **SW**ISH repository[3].

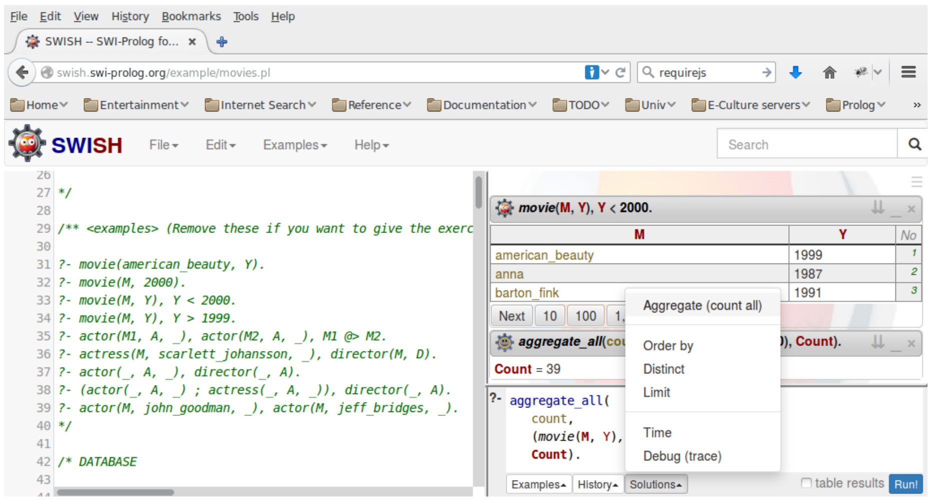
4 Illustration

As an example we take the following SPARQL query that enumerates labor strikes that took place in Amsterdam in 1903:

[1] See `https://github.com/share/ShareJS`

[2] See `https://www.r-project.org/`

[3] See `https://github.com/SWI-Prolog/swish`



File Edit View History Bookmarks Tools Help
SWISH -- SWI-Prolog fo...
swish.swi-prolog.org/example/movies.pl
requirejs
Home Entertainment Internet Search Reference Documentation TODO Univ E-Culture servers Prolog
SWISH File Edit Examples Help
Search
Next 10 100 1,
Aggregate (count all)
Order by
Distinct
Limit
Time
Debug (trace)
Examples History Solutions
table results
Run!

```
*/

/** <examples> (Remove these if you want to give the exerc

?- movie(american_beauty, Y).
?- movie(M, 2000).
?- movie(M, Y), Y < 2000.
?- movie(M, Y), Y > 1999.
?- actor(M1, A, _), actor(M2, A, _), M1 @> M2.
?- actress(M, scarlett_johansson, _), director(M, D).
?- actor(_, A, _), director(_, A).
?- (actor(_, A, _) ; actress(_, A, _)), director(_, A).
?- actor(M, john_goodman, _), actor(M, jeff_bridges, _).
*/

/* DATABASE
```

movie(M, Y), Y < 2000.

M	Y	No
american_beauty	1999	1
anna	1987	2
barton_fink	1991	3

aggregate_all(co ...), Count).

Count = 39

```
?- aggregate_all(
     count,
     (movie(M, Y),
     Count).
```

Figure 1. Screenshot of the **SWISH** interface.

```
SELECT ?strike ?days ?workers ?place ?date ?place
WHERE {
  ?strike ex:days ?days .
  ?strike ex:workers ?workers .
  ?strike ex:place ?place
  FILTER (langMatches(lang(?place), "nl"))
  FILTER (lcase(str(?place)) = "Haarlem")
  ?strike ex:date ?date .
  FILTER (year(?date) == 1903)
}
LIMIT 10
```

In **SWISH** we can write any SPARQL query by using the `rdf/3` predicate that implements Simple Graph Pattern queries. SPARQL FILTER expressions are implemented using a Domain-Specific Language extension (DSL): `lang_matches/2` shows how this works (notation between curly braces). `sounds/2` performs 'sounds like' string matching as implemented by the NLP metaphone algorithm. This illustrates how custom functions can be applied as filters within the query.[4]

```
strike_by_place_and_year(Strike, PlaceMatch, Year) :-
  rdf(Strike, ex:numberOfDays, NumDays),
  rdf(Strike, ex:numberOfWorkers, NumWorkers),
  rdf(Strike, ex:place, Place),
  {lang_matches(Place, nl)},
```

[4] Using `lcase/2` would have replicated the SPARQL query.

```
  {sounds(Place, PlaceMatch)},
  rdf(Strike, ex:date, Date),
  {Date = date(1903,_,_)}.

?- strike_by_place_and_year(Strike, "Haarlem", 1903).
```

The predicate `strike_by_place_and_year/3` has advantages over the SPARQL version. The Prolog predicate can be used to enumerate the labor strikes in any city and in any year. It can also be reused in other queries. Since **SW**ISH programs can be shared online, the Prolog predicate can also be reused in someone else's query. This functionality allows developers to incrementally build more sophisticated queries on top of existing, proven and tested building blocks. This is an effective way to avoid the large and complex SPARQL queries often found in existing Semantic Web applications.

5 Use cases & Conclusion

SWISH is able to support a variety of use cases. Recently TRILL-on-SWISH [2] was released: a fuzzy OWL reasoner built on top of **SW**ISH. It illustrates that **SW**ISH can be used to provide functionality that no existing SPARQL editor or Semantic Web-compatible notebook can provide: reasoning over a non-standard entailment regime[5]. **SW**ISH development is still ongoing. The LOD Laundromat team is currently using **SW**ISH in order to expose the next version of LOD Laundromat [1] for others to query online.

References

1. Beek, W., Rietveld, L., Bazoobandi, H.R., Wielemaker, J., Schlobach, S.: LOD Laundromat: A uniform way of publishing other people's dirty data. In: ISWC 2014, pp. 213–228. Springer (2014)
2. Bellodi, E., Lamma, E., Riguzzi, F., Zese, R., Cota, G.: A web system for reasoning with probabilistic OWL. Software: Practice and Experience (2016)
3. Lager, T., Wielemaker, J.: Pengines: Web logic programming made easy. Theory and Practice of Logic Programming 14(4-5), 539–552 (2014)
4. Meroño-Peñuela, A., Hoekstra, R.: grlc makes GitHub taste like Linked Data APIs. In: Proceedings of the Services and Applications over Linked APIs and Data workshop, ESWC (2016)
5. Ragan-Kelley, M., Perez, F., Granger, B., Kluyver, T., Ivanov, P., Frederic, J., Bussonier, M.: The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication. In: AGU Fall Meeting Abstracts. vol. 1, p. 07 (2014)
6. Rietveld, L., Hoekstra, R.: Yasgui: Not just another SPARQL client. In: Extended Semantic Web Conference. pp. 78–86. Springer (2013)
7. Wielemaker, J., Beek, W., Hildebrand, M., van Ossenbruggen, J.: ClioPatria: A SWI-Prolog infrastructure for the Semantic Web. Semantic Web Journal 7(5), 529–541 (2016)

[5] See `http://trill.lamping.unife.it/`