SPARKLIS: Expressive Semantic Search on Top of SPARQL Endpoints for (Almost) Everybody

Sébastien Ferré
Team LIS, Data and Knowledge Management, Irisa, Univ. Rennes 1

SemWeb.Pro, 5 November 2015, Paris

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

UMR IRISA LIS

Semantic Search with SPARQL

SPARQL is the W3C standard query language for semantic search

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT ?film WHERE {
  ?film a dbo:Film ;
        dbo:director res:Tim_Burton ;
        dbo:starring ?actor .
  ?actor dbo:birthDate ?date
        FILTER (?date >= "1980") }
ORDER by ?date
```

- PRO: very expressive (like SQL), standard, efficient engines
- CONS: limited to specialists, tedious (trial-and-error mode)

More Usable Approaches

- Navigation: surfing from entity to entity through relations
  - ex: France's capital's mayor's child...
- Faceted Search: guided iterative filtering of a set of entities
  - ex: Film / director: Tim Burton / release date: after 2000 / ...
- Question Answering: direct query formulation in natural language (NL)
  - ex: Give me all films whose director was born in France.

usability comes at the cost of expressivity

- one entity or a list of entity, no tables
- limited graph patterns: chains or shallow trees
- britleness of NL: ambiguity, synonymy

More Usable Approaches

- Navigation: surfing from entity to entity through relations
  - ex: France's capital's mayor's child...
- Faceted Search: guided iterative filtering of a set of entities
  - ex: Film / director: Tim Burton / release date: after 2000 / ...
- Question Answering: direct query formulation in natural language (NL)
  - ex: Give me all films whose director was born in France.

usability comes at the cost of expressivity

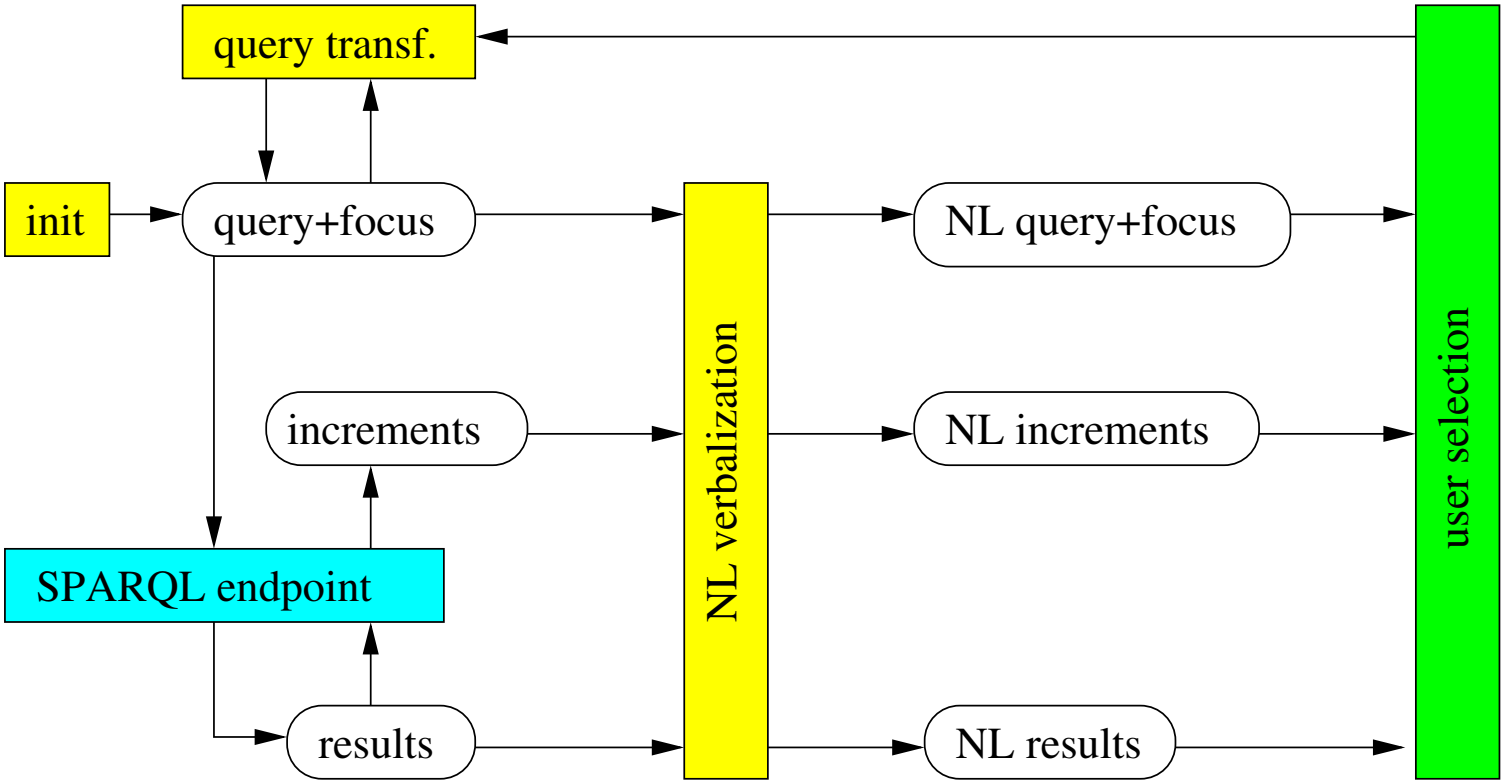
- one entity or a list of entity, no tables
- limited graph patterns: chains or shallow trees
- britleness of NL: ambiguity, synonymy

SPARKLIS: Reconciling Expressivity and Usability

Key ingredients:

1. query language for expressivity
   - *most of SPARQL features are covered*
2. NL verbalization of queries for readability
   - *no need to show any SPARQL to users*
3. faceted search for usability and guidance
   - *no need to write anything for users*
4. SPARQL endpoints for scalability and interoperability
   - *no need to configure to each dataset*

SPARKLIS: Dataflow and Interaction



Example and Screenshot (step 11)

Results of your query

results 1 - 10 of 15 Show 10 results

	the writer	the writer's nationality	the writer's birth date	the number of book
1	Sergei Lukyanenko	Russia	1968-04-11	18
2	Leo Tolstoy	Russians	1828-09-09	13
3	Fyodor Dostoyevsky	Russians	1821-11-11	11

Example and Screenshot (step 12)

Your query and its current focus permalink

Give me a writer
whose nationality is
Russians
or Russia
and whose birth date is after 1800
and that is the author of the highest-to-lowest number of book

Results of your query

results 1 - 10 of 13 Show 10 results

	the writer	the writer's birth date	the number of book
1	Sergei Lukyanenko	1968-04-11	18
2	Leo Tolstoy	1828-09-09	13
3	Fyodor Dostoyevsky	1821-11-11	11
4	Ivan Bunin	1870-10-22	7
5	Vladimir Sorokin	1955-08-07	5
6	Aleksey Konstantinovich Tolstoy	1817-09-05	4
7	Alexander Belyayev	1884-03-16	4
8	Nikolai Ostrovsky	1904-09-29	2
9	Joseph Brodsky	1940-05-24	1
10	Kir Bulychev	1934-10-18	1

results 1 - 10 of 13 Show 10 results

Your query in SPARQL

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?Writer_1 ?birthDate_13 (COUNT(DISTINCT ?Book_41) AS ?number_of_Book_51)
WHERE { ?Writer_1 a dbo:Writer .
        { ?Writer_1 dbo:nationality dbr:Russians . }
        UNION { ?Writer_1 dbo:nationality dbr:Russia . }
        ?Writer_1 dbo:birthDate ?birthDate_13 .
        FILTER ( str(?birthDate_13) >= "1800" )
        ?Book_41 a dbo:Book .
        ?Book_41 dbo:author ?Writer_1 . }
GROUP BY ?Writer_1 ?birthDate_13
ORDER BY DESC(?number_of_Book_51)
```

Demo

SPARKLIS is a Web application (no setup, no login)

http://www.irisa.fr/LIS/ferre/sparklis/

QALD-challenge questions over DBpedia

- QALD = Question Answering over Linked Data
- DBpedia = Semantic version of Wikipedia
- Examples:
  - Give me all films directed by Tim Burton, and starring some actor born since 1980. [YouTube]
  - Which rivers flow into a German lake? [YouTube]
  - How many languages are spoken in Colombia? [YouTube]
  - Give me all films produced by Steven Spielberg with a budget of at least $80 million. [YouTube]

Usage

Online since April 2014

- about 10 sessions per day
- by about 1000 unique users
- on more than 170 different endpoints (public and local)
- in various domains: encyclopedic, bioinformatics, ...
- with queries having sizes up to 29

Work in Progress and Future Work

- expressivity: full SPARQL coverage
  1. nested aggregations
     the [**average** [**number of** actors per film] per director]
  2. computations (like spreadsheet formulas)
     age = now - the birth date
  3. RDF graphs as results ⇒ updates
     every person whose age is higher than 18 is an adult
- usability
  1. visualization: generation of charts, maps, timelines
  2. use of lexicons (Wordnet?):
     better verbalization and filtering (synonyms)

Thanks! Questions?

Useful links:

- SPARKLIS at http://www.irisa.fr/LIS/ferre/sparklis/
- Demo/Tutorial at http://youtu.be/O999FVXn8Qc
- Usability Survey at http://tinyurl.com/kxozx9r
- Papers at ISWC'14, SWJ (under revision)

SPARKLIS: Expressivity compared to SPARQL (1/2)

Covered features:

- triple patterns: entities/values, classes/properties
- graph patterns (join): `and`, `that`, `is`
- cyclic graph patterns: anaphors (ex: the film's director)
- simple filters: `matching`, `higher than`, `after`, ...
- logic/algebra: `or`, `not`, `optionally`
- solution modifiers (projection, ordering): `any`, `the highest-to-lowest`, `the lowest-to-highest`
- aggregation: `number of`, `average`, `total`

SPARKLIS: Expressivity compared to SPARQL (2/2)

Missing features:

- expressions: `+`, `sqrt()`, `concat()`, `now()`
- nested aggregations:
  the [**average** [**number of** actors per film] per director]
- named graphs = querying on the source of knowledge
- transitive closures of property paths:
  ancestor = (father | mother)+
- federated search = querying over several SPARQL endpoints at once
- DESCRIBE and CONSTRUCT queries = returning RDF graphs as results (instead of tables)