SQUALL: a Controlled Natural Language for Querying and Updating RDF Graphs

Sébastien Ferré
Team LIS, Data and Knowledge Management, Irisa

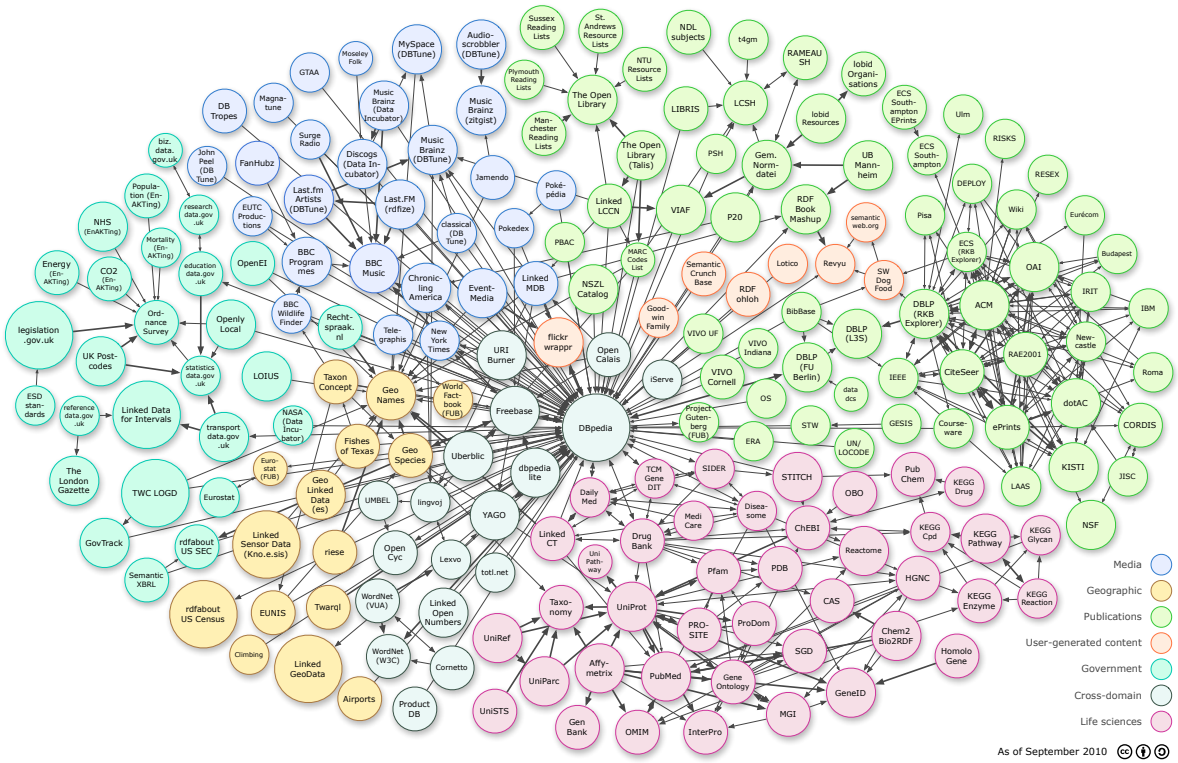
Controlled Natural Language, 30 August 2012, Zurich

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTEMES ALÉATOIRES

UMR IRISA LIS

The Web of Data

- How to search and explore the Web of data (RDF graphs) ?
- How to fill the gap between end users and formal languages (RDF, OWL, SPARQL) ?



Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

Formal vs Natural Languages

- SPARQL: a formal language *à la* SQL
  - very expressive and precise for querying and updating RDF graphs
  - requires understanding of low-level notions: relational algebra and logic
- natural language interfaces (ex., Aqualog, FREyA)
  - good usability through NL
  - difficult problems: ambiguity and adequacy w.r.t. the underlying system
  - in practice, generally limited to simple questions (much less expressive than SPARQL)
    - ex., Aqualog queries are limited to 2-triples queries

Controlled Natural Languages (CNL)

- on the natural/formal continuum [Kaufmann&Bernstein 2010]
- combine natural syntax and formal semantics
- *"There is no important theoretical difference between natural languages and the artificial languages of logicians."* (Montague)
- a few CNLs:
  - ACE [Fuchs *et al*]: a general purpose CNL
  - SOS, Rabbit: CNLs for verbalizing OWL axioms
  - *SQUALL*: the first CNL for SPARQL queries *and* updates

What SQUALL is *not*...

1. a pure (grammatically correct) subset of English
   - natural languages are a source of inspiration for flexibility, expressiveness, concision, high-level forms
   - I think that CNLs should be more regular than NLs because they have to be learnt anyway
2. concerned with morphology (lexicon, agreements, etc.)
   - should have the same requirements as SPARQL w.r.t. data
   - should be able to refer to every resource without preprocess
   - shares non-ambiguous notations with SPARQL
     - author → `hackcraft:authoredBy` or `purl:author` ... ?
3. a user interface
   - querying is difficult, whatever the language
     - syntax errors, empty results, preferences
   - syntactic guided input (e.g., Ginseng) is not enough
   - the objective is *semantic* guided input (done to a limited extent in previous work with Sewelis)

What SQUALL is...?

- an alternative CNL syntax for SPARQL
  - hence, same expressiveness as SPARQL
  - hence, full adequacy to RDF data
  - with natural high-level syntax
- its implementation is a compiler (3 phases)
  1. parsing of the source sentence (SQUALL)
  2. generation of an intermediate representation (Montague λ-terms)
  3. production of the target code (SPARQL)

RDF Graphs

- a RDF graph is a set of triples (labeled edges)
- a triple (of resources) has a subject, a predicate, and an object
  - a triple is a basic sentence
  - ex., `ex:John ex:loves ex:Mary .`
- a resource denotes an entity or a concept or a literal value (e.g., numbers, dates, strings)
- a property denotes a binary relation between resources
  - a property can be a transitive verb (ex., `ex:loves`) or a relational noun (ex., `ex:author`)
- a class denotes a set of resources
  - a class can be a noun (ex., `ex:woman`) or an intransitive verb (ex., `ex:works`)
- properties and classes are resources themselves

SPARQL 1.1: querying and updating RDF graphs

- SPARQL forms
  - closed question: `ASK` *`graph pattern`*
  - open question: `SELECT` *`vars`* `WHERE` *`graph pattern`*
  - update: `DELETE` *`graph`* `INSERT` *`graph`* `WHERE` *`graph pattern`*
- graph patterns
  - relational algebra: joins, unions, complements, selections, projections
  - constraints: logic, arithmetic, built-ins
  - named graphs
  - subqueries
  - aggregations

SPARQL query example

```
SELECT ?r
WHERE {
    ?r rdf:type :researcher .
    BIND (?r AS ?X)
    GRAPH :DBLP {
        FILTER NOT EXISTS {
            ?p rdf:type :publication .
            ?p :author ?X .
            ?p :year ?y .
            FILTER (?y >= 2000)
            FILTER NOT EXISTS {
                { SELECT COUNT(?a) AS ?n
                  WHERE { ?p :author ?a . } }
                FILTER (?n >= 2) } } } }
```

A Montague grammar of SQUALL

- Syntax and semantics
- Modules:
  1. lexical conventions
  2. triples as sentences
  3. relational algebra as coordinations
  4. natural constructs (headed NPs, relatives, ...)
  5. queries with *wh*-words
  6. quantifiers as determiners
  7. subordination and n-ary predicates with reification
  8. built-in predicates and aggregations
  9. resolving syntactic ambiguities

Lexical conventions

The same as in well-known notations (Turtle, SPARQL, N3)

- proper nouns, nouns, and verbs (URIs)
  - `<http://dbpedia.org/resource/Berlin>`: a full URI for the Berlin city
  - `dbpedia:Berlin`: an abbreviated URI with DBpedia namespace
  - `:Berlin`: an abbreviated URI with default namespace
  - `Berlin`: a bare URI (default namespace)
- literals
  - `"Hello world!"`: a plain literal
  - `"42"^^xsd:integer`: a typed literal
  - `42`: a bare integer
- variables: `?X`
- grammatical words (SQUALL reserved keywords)
  - `is`, `a`, `which`, `every`, ...

Triples as sentences

S	$\rightarrow$	$NP\ VP\ \{\ np\ vp\ \}$	"[$_{NP}$A] [$_{VP}$know-s B]"
NP	$\rightarrow$	$Term\ \{\ \lambda d.(d\ term)\ \}$	"A"
VP	$\rightarrow$	$P1\ \{\ \lambda x.(p1\ x)\ \}$	"[$_{P1}$work-s]"
	$\mid$	$P2\ NP\ \{\ \lambda x.(np\ \lambda y.(p2\ x\ y))\ \}$	"[$_{P2}$know-s] [$_{NP}$B]"
$P1$	$\rightarrow$	$ClassURI\ \{\ \lambda x.(\mathbf{type}\ x\ uri)\ \}$	"work"
$P2$	$\rightarrow$	$PropertyURI\ \{\ \lambda x.\lambda y.(\mathbf{stat}\ x\ uri\ y)\ \}$	"know"

Relational algebra as coordinations

They apply to most syntagms Δ: *S*, *NP*, *VP*, *P1*, *P2*, (next: *Rel*, *AP*, *PP*).

Δ	$\rightarrow$	**not** Δ_1 $\{$ **not** δ_1 $\}$	"not [$_{VP}$know-s B]"
	$\mid$	Δ_1 **and** Δ_2 $\{$ **and** δ_1 δ_2 $\}$	"[$_{VP}$work-s] and [$_{VP}$cite-s X]"
	$\mid$	Δ_1 **or** Δ_2 $\{$ **or** δ_1 δ_2 $\}$	"[$_{NP}$A] or [$_{NP}$B]"
	$\mid$	**maybe** Δ_1 $\{$ **option** δ_1 $\}$	"maybe [$_{VP}$know-s B]"

Headed NPs

NP	→	*Det NG1* { $\lambda d.(det\ (\textbf{init}\ ng1)\ d)$ }	"[$_{Det}$a] [$_{NG1}$woman]"
	|	*Det NG2* **of** *NP* { $\lambda d.(np\ \lambda x.(det\ (\textbf{init}\ (ng2\ x))\ d))$ }	
			"[$_{Det}$the] [$_{NG2}$author-s] of [$_{NP}$X]"
Det	→	**a**(**n**) { $\lambda d_1.\lambda d_2.(\textbf{exists}\ (\textbf{and}\ d_1\ d_2))$ }	
	|	**the** { $\lambda d_1.\lambda d_2.(\textbf{the}\ d_1\ d_2)$ }	
NG1	→	**thing** *AR* { **and thing** *ar* }	"thing [$_{AR}$that cite-s A]"
	|	*P1 AR* { **and** *p1 ar* }	"[$_{P1}$woman] [$_{AR}$?A]"
NG2	→	*P2 AR* { $\lambda x.\lambda y.(\textbf{and}\ (p2\ x\ y)\ (ar\ y))$ }	
			"[$_{P2}$author] [$_{AR}$?A]"
AR	→	*App Rel* { **and** *app rel* }	"[$_{App}$?A] [$_{Rel}$that X cite-s]"
	|	*App* { *app* }	
App	→	*URI* { $\lambda x.(\textbf{eq}\ x\ uri)$ }	"A"
	|	*Var* { $\lambda x.(\textbf{bind}\ x\ var)$ }	"?X"
	|	ϵ { $\lambda x.\textbf{true}$ }	

Relatives

$$
\begin{array}{lcl}
\textit{Rel} & \rightarrow & \textbf{that } \textit{VP} \ \{ \textbf{ init } \textit{vp} \} \qquad \text{“that } [_{VP}\text{know-s B]”} \\
& | & \textbf{that } \textit{NP P2} \ \{ \textbf{ init } \lambda x.(np\ \lambda y.(p2\ y\ x)) \} \\
& & \qquad \text{“that } [_{NP}\text{X}]\ [_{P2}\text{cite-s]”} \\
& | & \textbf{such that } S \ \{ \textbf{ init } \lambda x.s \} \qquad \text{“such that } [_{S}\text{?A work-s]”} \\
& | & \textit{Det NG2} \textbf{ of which } \textit{VP} \ \{ \textbf{ init } \lambda x.(det\ (ng2\ x)\ vp) \} \\
& & \qquad \text{“}[_{Det}\text{an}]\ [_{NG2}\text{author}] \text{ of which } [_{VP}\text{know-s B]”} \\
& | & \textbf{whose } \textit{NG2 VP} \equiv \textbf{the } \textit{NG2} \textbf{ of which } \textit{VP} \\
& & \quad \text{“whose } [_{NG2}\text{author ?A}]\ [_{VP}\text{cites-s a colleague of ?A]”} \\
& | & \textbf{whose } \textit{P2} \textbf{ is/are } \textit{NP} \ \{ \lambda x.(np\ \lambda y.(p2\ x\ y)) \} \\
& & \qquad \text{“whose } [_{P2}\text{author}]\ [_{VP}\text{is a woman]”}
\end{array}
$$

Auxiliary verbs

$$
\begin{array}{lcll}
VP & \rightarrow & \textbf{is}/\textbf{are}\ AP\ \{\ ap\ \} & \text{“is [}_{AP}\text{a woman]”} \\
 & | & \textbf{is}/\textbf{are}\ Rel\ \{\ rel\ \} & \text{“is [}_{Rel}\text{such that ?A work-s]”} \\
 & | & \textbf{has}/\textbf{have}\ Det\ P2\ AR\ \{\ \lambda x.(det\ (p2\ x)\ ar)\ \} & \\
 & & & \text{“have [}_{Det}\text{an] [}_{P2}\text{author] [}_{AR}\text{that X cite-s]”} \\
AP & \rightarrow & Term\ \{\ \lambda x.(\textbf{eq}\ x\ term)\ \} & \text{“A”} \\
 & | & \textbf{a}(\textbf{n})/\textbf{the}\ NG1\ \{\ ng1\ \} & \text{“a [}_{NG1}\text{woman]”} \\
 & | & \textbf{a}(\textbf{n})/\textbf{the}\ NG2\ \textbf{of}\ NP\ \{\ \lambda x.(np\ \lambda y.(ng2\ y\ x))\ \} & \\
 & & & \text{“the [}_{NG2}\text{author] of [}_{NP}\text{X]”}
\end{array}
$$

Queries with *wh*-words

$S \rightarrow$ **whether** S_1 { **ask** s_1 } "whether [$_S$A know-s a woman]"

$NP \rightarrow$ **what** $\equiv$ **which thing**

$\mid$ **whose** *NG2* $\equiv$ **the** *NG2* **of what**

"whose [$_{NG2}$author ?A]"

$Det \rightarrow$ **which** { $\lambda d_1.\lambda d_2.$(**select** (**and** d_1 d_2)) }

$\mid$ **how many** { $\lambda d_1.\lambda d_2.$(**count** (**and** d_1 d_2)) }

Quantifiers as determiners

Not present as such in SPARQL.

$$
\begin{array}{lcl}
\mathit{Det} & \rightarrow & \textbf{some}\ \{\ \lambda d_1.\lambda d_2.(\textbf{exists}\ (\textbf{and}\ d_1\ d_2))\ \} \\
 & | & \textbf{every}\ \{\ \lambda d_1.\lambda d_2.(\textbf{forall}\ d_1\ d_2)\ \} \\
 & | & \textbf{no}\ \{\ \lambda d_1.\lambda d_2.(\textbf{not}\ (\textbf{exists}\ (\textbf{and}\ d_1\ d_2)))\ \} \\
 & | & \textbf{at least}\ i\ \{\ \lambda d_1.\lambda d_2.(\textbf{atleast}\ i\ (\textbf{and}\ d_1\ d_2))\ \} \\
S & \rightarrow & \textbf{for}\ \mathit{NP}\ ,\ S\ \{\ np\ \lambda x.s\ \}
\end{array}
$$

"for [$_{NP}$every publication ?X], [$_{S}$an author of ?X work-s]"

$$
\begin{array}{lcl}
 & | & \textbf{there is/are}\ \mathit{NP}\ \{\ np\ \lambda x.\textbf{true}\ \}
\end{array}
$$

"there are [$_{NP}$at least 3 person-s that know A]"

Subordination and n-ary predicates with reification

$$
\begin{array}{lcl}
\textit{NP} & \rightarrow & \textbf{that}\ \textit{S}\ \ \{\ \lambda d.\lambda \overline{a}.\lambda g.(s\ ()\ \lambda t.(d\ t\ \overline{a}\ g))\ \} \\
 & & \text{“that } [_{S}\text{A know-s B]”} \\
\textit{PP} & \rightarrow & \textbf{at}/\textbf{in}\ \textit{Prep NP}\ \ \{\ \lambda s.(np\ \lambda z.(prep\ z\ s))\ \} \\
 & & \text{“at } [_{Prep}\text{place] } [_{NP}\text{the city Rennes]”} \\
 & \mid & \textbf{at}/\textbf{in}\ \textit{Det Prep AR} \equiv \textbf{at}\ \textit{Prep Det}\ \textbf{thing}\ \textit{AR} \\
 & & \text{“at } [_{Det}\text{some] } [_{Prep}\text{venue] } [_{AR}\text{whose place is Rennes]”} \\
\textit{Rel} & \rightarrow & \textbf{at}/\textbf{in}\ \textbf{which}\ \textit{Prep AR S} \\
 & & \{\ \textbf{init}\ \lambda x.(\textbf{and}\ (ar\ x)\ (prep\ x\ s))\ \} \\
 & & \text{“in which } [_{Prep}\text{graph] } [_{S}\text{A work-s]”} \\
\textit{Prep} & \rightarrow & \textbf{graph}\ \ \{\ \textbf{graph}\ \}\ \mid\ \textit{URI}\ \ \{\ \textbf{arg}\ \textit{uri}\ \}
\end{array}
$$

- Prepositional phrases (PP) can occur at any position of a sentence among the subject, verb, and object.

Built-in predicates and aggregations

$$
\begin{array}{lcll}
\textit{P1} & \rightarrow & \textit{Pred1URI}\ \{\ \lambda x.(\textbf{pred1}\ \textit{uri}\ x)\ \} & \text{“Monday”} \\
\textit{P2} & \rightarrow & \textit{Pred2URI}\ \{\ \lambda x.\lambda y.(\textbf{pred2}\ \textit{uri}\ x\ y)\ \} & \text{“match”} \\
\textit{NG1} & \rightarrow & \textit{AggregURI}\ \textbf{of}\ \textit{AP}\ (\textbf{per}\ \textit{AP}_i^+)? & \\
 & & \{\ \lambda x.(\textbf{aggreg}\ \textit{uri}\ x\ \textit{ap}\ (\textit{ap}_i)_i)\ \} & \\
 & & \text{“count of [}_{AP}\text{the publication ?P] per [}_{AP}\text{the year of ?P]”} &
\end{array}
$$

Translation to SPARQL (principle)

Starting with the λ-term produced when parsing a SQUALL sentence:

1. replace some constants by their definition
   - ex., **count** $= \lambda d.$(**select** $\lambda x.$(**aggreg** `COUNT` x d ()))
2. perform all β-reductions on the result
3. inductively generate SPARQL code
   - 4 generators for: sentences, updates, queries, and graph patterns
   - [**ask** f] $=$ `ASK` $\{$ $[f]_G$ $\}$
   - [**select** d] $= [?x \mid d\ ?x]_Q$
   - [**forall** d_1 d_2]$_G$ $=$ [**not** (**exists** (**and** d_1 (**not** d_2)))]$_G$
   - [**the** d_1 d_2]$_G$ $=$ [**exists** (**and** d_1 d_2)]$_G$
   - [**the** d_1 d_2]$_U$ $=$ [**forall** d_1 d_2]$_U$

Translation to SPARQL (example)

The SQUALL sentence

> *for which researcher-s ?X, in graph DBLP every publication whose author is ?X and whose year $\geq$ 2000 has at least 2 author-s*

is parsed as

> “[$_S$for [$_{NP}$[$_{Det}$which] [$_{NG1}$[$_{P1}$researcher-s] [$_{AR}$[$_{App}$?X]]]],
> [$_S$[$_{PP}$in [$_{Prep}$graph] [$_{NP}$DBLP]] [$_S$[$_{NP}$[$_{Det}$every]
> [$_{NG1}$[$_{P1}$publication] [$_{AR}$[$_{Rel}$[$_{Rel}$whose [$_{NG2}$[$_{P2}$author]] [$_{VP}$is
> [$_{AP}$?X]]] and [$_{Rel}$whose [$_{NG2}$[$_{P2}$year]] [$_{VP}$[$_{P2}\geq$] [$_{NP}$2000]]]]]]]
> [$_{VP}$has [$_{Det}$at least 2] [$_{P2}$author-s]]]]]”

Translation to SPARQL (example)

The SQUALL sentence

> *for which researcher-s ?X, in graph DBLP every publication whose author is ?X and whose year $\geq$ 2000 has at least 2 author-s*

is parsed as

> “[$_S$for [$_{NP}$[$_{Det}$which] [$_{NG1}$[$_{P1}$researcher-s] [$_{AR}$[$_{App}$?X]]]],
> [$_S$[$_{PP}$in [$_{Prep}$graph] [$_{NP}$DBLP]] [$_S$[$_{NP}$[$_{Det}$every]
> [$_{NG1}$[$_{P1}$publication] [$_{AR}$[$_{Rel}$[$_{Rel}$whose [$_{NG2}$[$_{P2}$author]] [$_{VP}$is
> [$_{AP}$?X]]] and [$_{Rel}$whose [$_{NG2}$[$_{P2}$year]] [$_{VP}$[$_{P2}\geq$] [$_{NP}$2000]]]]]]]
> [$_{VP}$has [$_{Det}$at least 2] [$_{P2}$author-s]]]]]”

Translation to SPARQL (example)

...whose internal representation is:

(**select** X (**and**
(**triple** X rdf:type :researcher)
(**graph** :DBLP
(**forall** (**exists** x_3 x_5 (**and**
(**triple** x_3 rdf:type :publication) (**triple** x_3 :author X)
(**triple** x_3 :year x_5) (**pred2** $\geq$ x_5 2000)))
(**exists** x_6 (**and**
(**aggreg** COUNT x_6 x_8 x_3
(**exists** x_8 (**triple** x_3 :author x_8)))
(**pred2** $\geq$ x_6 2)))))))

Translation to SPARQL (example)

...which translates to the SPARQL query:

```
SELECT ?r
WHERE {
    ?r rdf:type :researcher .
    BIND (?r AS ?X)
    GRAPH :DBLP {
        FILTER NOT EXISTS {
            ?p rdf:type :publication .
            ?p :author ?X .
            ?p :year ?y .
            FILTER (?y >= 2000)
            FILTER NOT EXISTS {
                { SELECT COUNT(?a) AS ?n
                  WHERE { ?p :author ?a . } }
                FILTER (?n >= 2) } } } }
```

Implementation

- available as a Web form
  - See `http://www.irisa.fr/LIS/softwares/squall`
- developed in OCaml (functional, type safe, concise)
  - syntax (367 loc), semantics (295 loc), sparql (198 loc)
- extends the paper with
  - semantic validation: variable scope and binding, semantic restrictions (e.g., no built-ins in assertions)
  - arithmetic expressions and function calls (*NP*, *P2*, *AP*)
    - "what is the height * width of the rectangle-s whose color is red ?"
  - quoted NPs as verbs (P1, P2)
    - "a publication has author a person ?"
    - "a publication has 'which rdf:Property' a person ?"

Perspectives as a language

- covering 100% of SPARQL 1.1 (nothing hard)
  - `CONSTRUCT` and `DESCRIBE` forms
  - query modifiers (`ORDER BY`, `LIMIT`, `OFFSET`)
  - conditional expressions
  - concise notation of RDF collections (list) and membership test
  - + type checking in expressions
- adding more natural constructs
  - anaphoras: e.g., “some man ... this man” instead of “some man ?X ... ?X”
  - comparatives and superlatives
  - adjectives and adverbs (as RDF classes)
  - more forms of aggregations
    - e.g., “what is the average age of the researcher-s ?”

Perspectives as a user interface

- even a CNL is not simple enough
- need for guided input
  - syntax-based auto-completion (like in Ginseng)
  - query-based faceted search (like in Sewelis)

Thanks!

Questions ?