LDScript :
a Linked Data Script Language

Olivier Corby, Catherine Faron-Zucker, Fabien Gandon

Université Côte d'Azur
Inria, I3S, UNS, CNRS
http://wimmics.inria.fr
olivier.corby@inria.fr

Context

- **RDF**: Semantic Web of Linked Data
- **SPARQL**: Query RDF graph

Focus: SPARQL *extension functions* for computing

Computing ?

- Find resources whose income is greater that n!
  - ?income >= factorial(?n)
- Find a function name associated to a resource and call it
  - funcall(?fun, ?x)
- Define a formula and share it among queries
  - area = width * length
- Deal with exotic datatype
  - "XIX"^^dt:roman + "I"^^dt:roman
- Define extended aggregate
  - aggregate(?income) as ?list
  - median(?list) as ?median

```
prefix fun: <function://fr.inria.sparql.Extension>
select * where {
  ?x us:income ?inc
  filter (?inc >= fun:fac(10))
}
```

```
package fr.inria.sparql;
class Extension {
  public Datatype fac(Datatype dt) {
     return Datatype.create(fac(dt.longValue()));
  }


}
```

```
prefix fun: <function://fr.inria.sparql.Extension>
select * where {
  ?x us:income ?inc
  filter (?inc >= fun:fac(10))
}
```

```
package fr.inria.sparql;
class Extension {
  public Datatype fac(Datatype dt) {
     return Datatype.create(fac(dt.longValue()));
  }
  long fac(long n) {
     if (n == 0) { return 1;} else { return n * fac(n-1);}
  }}
```

Needs to be compiled and linked

```
prefix fun: <http://ns.inria.fr/sparql-function/>
select * where {
  ?x us:income ?inc
  filter (?inc >= fun:fac(10))
}
```

```
function fun:fac(?n) {
  if (?n = 0, 1, ?n * fun:fac(?n – 1))
}
```

No Compiling, nor Linking

Requirements for LDScript Language

- Define SPARQL extension function
- SPARQL compliant as much as possible
- Manipulate RDF and SPARQL entities
- Execute SPARQL query in function
- Facilitate SPARQL extension
- No API, no cast, no compiling, no linking

Research Questions

- Is it possible to define a Linked Data Script Language on top of SPARQL ?
- Is it possible to share the same core SPARQL interpreter ?

Solution

- LDScript language on top of SPARQL Filter

SPARQL Filter

- Constant: 2.718, true, "1930-01-29"^^xsd:date
- Variable: ?x, ?y, ?z, ?t
- Expression: ?x + 1, ?x = (10 * ?y) / ?z, ?t >= 3.14
- Connector: ! (?x < 0 || ?y = "Keynes") && …
- Graph match: exists { ?x a us:ScriptLanguage }
- If then else: if (?deficit <= .03, true, true)
- Function call: regex(?uri, ".*cnrs"), us:fac(?n)

SPARQL Filter LDScript Extension

- Constant: 2.718, true, "1930-01-29"^^xsd:date
- Variable: ?x, ?y, ?z, ?t
- Expression: ?x + 1, ?x = (10 * ?y) / ?z, ?t >= 3.14
- Connector: ! (?x < 0 || ?y = "Keynes") && …
- Graph match : exists { ?x a us:ScriptLanguage }
- If then else: if (?deficit <= .03, true, true)
- Function call: regex(?uri, ".*cnrs"), us:fac(?n)
- Function: **function** us:fun(?x) { 1 / (?x * ?x) }

LDScript Function

function

LDScript Function

```
function us:fac
```

LDScript Function

```
function us:fac(?n)
```

LDScript Function

```
function us:fac(?n) {

}
```

LDScript Function

```
function us:fac(?n) {
    if (?n = 0, 1, ?n * us:fac(?n – 1))
}
```

LDScript Function

```
function us:fac(?n) {
    if (?n = 0, 1, ?n * us:fac(?n – 1))
}
```

Manage variables in a stack

LDScript

- SPARQL Filter
- Function Definition

LDScript Statements

- SPARQL Filter
- Function Definition

1. Let
2. For
3. Query
4. Pattern Matching
5. Second Order Function
6. Lambda Expression

Let

```
let (?y = us:foo(?x)) {
  us:bar(?y)
}
```

For

```
for (?x in xt:list(1, 2, 3, 4, 5)) {
        xt:display(?x)
}
```

Query in Let

```
function us:circleArea(?x) {
   let (query) {
        body
   }
}
```

Query in Let

```
function us:circleArea(?x) {
   let (select ?x  ?r where { ?x us:radius ?r } ) {
        body
   }
}
```

Query in Let

```
function us:circleArea(?x) {
   let (select ?x  ?r where { ?x us:radius ?r } ) {
         (3.1416 * ?r * ?r)
   }
}
```

Query in Let: variable binding

```
function us:circleArea(?x) {          # ?x = v
  let (select ?x  ?r where {
        ?x us:radius ?r } ) {
        (3.1416 * ?r * ?r)
  }
}
```

Query in Let: variable binding

```
function us:circleArea(?x) {          # ?x = v
  let (select ?x  ?r where {  values ?x  { v }
        ?x us:radius ?r } ) {
        (3.1416 * ?r * ?r)
  }
}
```

Query in For

```
for (query) {
        body
}
```

Query in For

```
for (select ?x ?y where { ?x foaf:knows ?y } ) {
        xt:display(?x, ?y)
}
```

Iterate ?x and ?y on every query solutions

Query in For

```
for (?t  in  construct where { ?x foaf:knows ?y } ) {
        xt:display(?t)
}
```

Iterate ?t on triples of the result graph

Pattern Matching

- for ((?s, ?p, ?o) in ?graph)
- for ((?var, ?val) in ?map)
- for ((?a, ?b) in ?listOfList)

Literal Extension Datatypes (New)

prefix dt: <http://ns.inria.fr/sparql-datatype/>

1. dt:list
2. dt:graph
3. dt:triple
4. dt:mappings
5. dt:mapping

Second order functions

Function whose first argument is an expression that returns a function

1. funcall — funcall(?fun, ?x, ?y)
2. apply — apply(?fun, ?list)
3. map — map(?fun, ?list)
4. reduce — reduce(rq:plus, ?list) (New)

Lambda Expression (New)

Declaration of an anonymous function

```
lambda(?x) { 1 / (?x * ?x) }
```

Call of an anonymous function

```
maplist( lambda(?x) { 1 / (?x * ?x) } , ?list)
```

SPARQL Extension with LDScript

- Custom Aggregate
- Values with Expression
- SPARQL with Recursion

SPARQL Custom aggregate

```
select (aggregate(?x) as ?list)
(ag:median(?list) as ?med)
where { … }

function ag:median(?list) {
    xt:get(xt:sort(?list), xt:size(?list) / 2)
}
```

SPARQL Values with Expression

```
construct {
  ?s ?p ?o
}
where {
  values (?s ?p ?o) { unnest(xt:load(?uri)) }
}
```

SPARQL with Recursion

- Count descendants on the woman side

SPARQL with Recursion

```
select ?x (count(?y)                                    as ?c)
 where { ?x us:child ?y . ?y a us:Woman }
 group by ?x
```

SPARQL with Recursion

```
function us:pattern(?x) {
  let (select ?x (count(?y)                                    as ?c)
        where { ?x us:child ?y . ?y a us:Woman }
        group by ?x) {
        ?c
  }
}
```

SPARQL with Recursion

```
function us:pattern(?x) {
  let (select ?x (count(?y) + sum(us:pattern(?y)) as ?c)
        where { ?x us:child ?y . ?y a us:Woman }
        group by ?x) {
        ?c
  }
}
```

Validation: Implementation

- Implementation: Corese Semantic Web Factory
- http://wimmics.inria.fr/corese
- http://corese.inria.fr

Validation: Application

- SHACL validator (with STTL)
- Day of date
- Roman to decimal and converse
- Fibonacci, factorial, sort, standard deviation, etc.

Validation: Benchmark

LDScript

```
function us:fib(?n) { if (?n<=2, 1, us:fib(?n-1) + us:fib(?n-2)) }
```

Java

```
IDatatype fib(IDatatype dt) {
   if (dt.le(Datatype.TWO).booleanValue()){
      return Datatype.ONE;
   } else { return
         fib(dt.minus(Datatype.ONE)).plus(fib(dt.minus(Datatype.TWO)));
   }
}
```

fib(35) = 9 227 465

nb function call: 18 454 930

Java : 0.721 s — 25,6 M funcall / s

LDScript : 1.783 s — 10,3 M funcall / s

LDScript = 2.5 Java

Conclusion

- Script Language on top of SPARQL filter
- SPARQL extension function
- Second order function
- Query in function
- Recursive query
- Pattern matching
- Custom aggregate

http://ns.inria.fr/sparql-extension

Perspective

- Linked functions
- Method attached to class, inheritance
- Type checking
- Two implementations
- Leverage performance

Perspective

- Linked functions (done)
- Method attached to class, inheritance
- Type checking
- Two implementations
- Leverage performance

Perspective

- Linked functions (done)
- Method attached to class, inheritance (done)
- Type checking
- Two implementations
- Leverage performance