Programming the Semantic Web

olivier.corby@inria.fr

Programming the Semantic Web

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

Focus: SPARQL *extension functions*

```
prefix fun: <http://ns.inria.fr/sparql-function/>
select * where {
  ?x us:income ?inc
  filter (?inc >= fun:fac(10))                    # = 3628800
}
```

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

Wish: No compiling, no linking: just write & run

Requirement for Function

1. Call function in SPARQL query
   - Call function whose name is result of expression
2. Execute SPARQL query in function
   - Pass parameter to SPARL query
3. Function objects = RDF terms + triple & graph
4. SPARQL operators and functions available
   - isBlank, isURI, isLiteral, regex, datatype, …

Candidate: SPARQL Filter ?

- Constant: 2.718, true, "1930-01-29", ex:trust
- Datatype: xsd:string, xsd:integer, xsd:boolean
- 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 Language: SPARQL Filter++

- Constant: 2.718, true, "1930-01-29"
- Datatype: xsd:string, xsd:integer, xsd:boolean
- 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) }

Function Language: SPARQL Filter++

```
function                    {

}
```

Function Language: SPARQL Filter++

```
function us:foo(x) {

}
```

Function Language: SPARQL Filter++

```
function us:foo(x) {
        us:bar (x * x)
}
```

Function Language: SPARQL Filter++

```
function us:foo(x) {
        us:bar (x * x)
}

function us:bar(y) {
        1 / y
}
```

Function Language Statement

1. Function Definition
2. SPARQL Filter expression
3. Let, For
4. SPARQL Query
5. Pattern Matching
6. Second Order Function
7. Lambda Expression
8. Linked Function
9. Literal Extension Datatype
10. SPARQL Extension

Let, For

```
let (x = y + 1) {
        us:foo(x)
}

for (elem in list) {
        us:bar(elem)
}
```

Query in Let

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

}
```

Query in Let

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

Query in Let

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

« Same » in Java

```
String q = "select ?x  ?r where {  ?x us:radius ?r }" ;
QueryProcess exec = QueryProcess.create(graph);
Mappings map = exec.query(q);
Datatype dt = map.get(0).getValue("r");
double val = 3.1416 * dt.doubleValue() * dt.doubleValue();
Datatype res = Datatype.create (val);
```

Query in Let

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

Select Query in For

```
for (select * where { ?s ?p ?o } ) {
        us:foo(s, p, o)
}
```

Construct Query in For

```
for (atriple in construct where { ?x rdfs:label ?l } ) {
        us:foo(atriple)
}
```

Construct Query in For

```
for ((s p o) in construct where { ?x rdfs:label ?l } ) {
        us:bar(s, p, o)
}
```

Update in Function

```
function us:myfun() {
        query (insert  data { });
        query (delete data { });
        query (delete { } insert { } where { })
}
```

Function update available in update query,
not available in select query,
because there is a read/write lock

Pattern Matching

```
let ((first | rest . last) = list) {

}
```

Pattern Matching

```
let ((first | rest . last) = @(1 2 3 4)) {
        first = 1 ;
        rest = (2 3) ;
        last = 4
}
```

Second Order Function

- funcall(exp, arg)

eval(exp) = function name or lambda expression

eval(exp)= ex:foo

funcall(exp, arg) = funcall(ex:foo, arg) = ex:foo(arg)

Second Order Function

- funcall(exp, arg)

```
let (select * where { ?x us:function ?name } ) {
        funcall(?name, ?x)
}
```

Second Order Function

- funcall(exp, arg)
- apply(exp, argList) apply(rq:mult, xt:list(2, 3))
- map(exp, list) map(xt:print, list)
  - map, maplist, mapany, mapevery
- reduce(exp, list) reduce(rq:plus, list)

- reduce(rq:plus, maplist(ex:square, xt:iota(10)))

Lambda Expression

- Anonymous function

```
lambda(x, y) { x + y }
```

Lambda Expression

- Anonymous function

```
maplist (lambda(x) { x * x } , list)

let (fun = lambda(y) { 1 / (y * y) } ) {
        funcall(fun, 10)
}
```

Extension Datatype

- dt:graph RDF Graph
- dt:triple RDF Triple
- dt:mappings Query Solution Sequence
- dt:mapping Query Solution

- dt:list, dt:map
- dt:xml, dt:json

Graph Datatype

```
let (g = xt:graph()) {
        datatype(g) = dt:graph ;
        for (t in g) {
                datatype(t) = dt:triple
        }
}
```

Mapping Datatype

```
let (sol = select * where { } ) {
        datatype(sol) = dt:mappings ;
        for (res in sol) {
                datatype(res) = dt:mapping
        }
}
```

XML Datatype

```
let (obj = "<book><title>1984</title></book>" ,
      xml = xt:xml(obj),
      list  = xpath(xml, "/book/title")) {
        dom:getTextContent(xt:first(list))
}
```

XML DOM

- dom:getChildNodes()
- dom:getTextContent()
- dom:getElementsbyTagName()
- …

JSON

```
let (obj = xt:read(<http://example.org/myservice>),
        json = xt:json(obj)) {
          xt:get(json, "slotname");
          for ((key val) in json) {
                    xt:print(key, val)
          }
}
```

List

- xt:list()
- xt:get(list, n)
- xt:first(list)
- xt:rest(list)
- xt:sort(list)
- xt:reverse(list)
- xt:append(l1, l2)
- …

SPARQL Extension

SPARQL Function Statement

```
select * where {
        ?x ex:function ?name
        filter funcall(?name, ?x)
}
```

SPARQL List aggregate

```
select (aggregate(?y) as ?list)
where { ?x foaf:knows ?y }
```

SPARQL Custom aggregate

```
select (aggregate(?y) as ?list)
(xt:sort(?list) as ?sort)
where { ?x foaf:knows ?y }
```

SPARQL Custom aggregate

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

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

SPARQL Values

```
values ?x { 1 2 3 }

values var { unnest(exp) }

values (?s ?p ?o) { unnest(xt:load(URI)) }
```

Named Graph Overloading

```
let (g := construct where { ?x rdfs:label ?l },
      select * where { graph ?g { ?s ?p ?o } } ) {


}
```

Event Driven Programming

- Associate events to SPARQL query processing
- Associate functions to events
- When event occurs, call the function

Event Driven Programming

- Event : before, after, …

```
@before
function us:before(query) {
        xt:print("before:", query)
}
```

Event Driven Overloading

- Overload operator for extension datatype

```
@type us:distance
function us:eq(x, y) {
        us:convert(x) = us:convert(y)
}

x = "100 km"^^us:distance
```

Event Update

```
@update
function us:myupdate(deleteList, insertList) {
        for ((s p o) in deleteList) {
                xt:print("delete:", s, p, o)
        }
}
```

Event Update

```
@update
function us:myupdate(deleteList, insertList) {
        query(
          insert { ?s ?p ?o}
          where {
                values ?deleteList { undef }
                values (?s ?p ?o) { unnest(deleteList) }
          })
}
```

Event Processing

```
@event
select  where {}
```

```
@init       function us:init(dt:query ?q)
@before     function us:before(dt:query ?q)
@after      function us:after(dt:mappings ?map)
@start      function us:start(dt:query ?q)
@finish     function us:finish(dt:mappings ?map)
@statement  function us:statement(URI ?g, dt:statement ?e)
@function   function us:function(dt:expression ?call, dt:expression ?fun)
@produce    function us:produce(URI ?g, dt:triple ?q)
@candidate  function us:candidate(URI ?g, dt:triple ?q, dt:triple ?t)
@result     function us:result(dt:mappings ?map, dt:mapping ?m)
@distinct   function us:key(dt:mapping ?m)
@orderby    function us:compare(dt:mapping ?m1, dt:mapping ?m2)
@limit      function us:limit(dt:mappings ?map)
@timeout    function us:timeout(URI ?serv)
@slice      function us:slice(URI ?serv, dt:mappings ?map)
@join       function us:join    (URI ?g, dt:statement ?e, dt:mappings ?m1, dt:mappings ?m2)
@minus      function us:minus   (URI ?g, dt:statement ?e, dt:mappings ?m1, dt:mappings ?m2)
@union      function us:union   (URI ?g, dt:statement ?e, dt:mappings ?m1, dt:mappings ?m2)
@optional   function us:optional(URI ?g, dt:statement ?e, dt:mappings ?m1, dt:mappings ?m2)
@bgp        function us:bgp    (URI ?g, dt:statement ?e, dt:mappings ?m)
@graph      function us:graph  (URI ?g, dt:statement ?e, dt:mappings ?m)
@service    function us:service(URI ?s, dt:statement ?e, dt:mappings ?m)
@query      function us:query  (URI ?g, dt:statement ?e, dt:mappings ?m)
@values     function us:values (URI ?g, dt:statement ?e, dt:mappings ?m)
@path       function us:path(URI ?g, dt:triple ?q, dt:path ?p, term ?s, term ?o)
@step       function us:step(URI ?g, dt:triple ?q, dt:path ?p, term ?s, term ?o)
@select     function us:select(dt:expression ?e, term ?v)
@aggregate  function us:agg(dt:expression ?e, term ?v)
@bind       function us:bind(URI ?g, dt:expression ?e, term ?v)
@filter     function us:filter(URI ?g, dt:expression ?e, xsd:boolean ?b)
@having     function us:having(dt:expression ?e, xsd:boolean ?b)
```

Usage

```
select where {
}

function us:fun(x) {
}
```

Usage

```
select where {
}

@public
function us:fun(x) {
}
```

Usage

```
@import </home/user/me/myfun.rq>
select where {
}
```

Usage

```
@public @import </home/user/me/myfun.rq>
select where {
}
```

Usage: Java Function Call

```
QueryProcess exec = QueryProcess.create(graph);
exec.compile(fundef);
IDatatype res = exec.funcall(name, arg);
```

Validation: Application

- SHACL Interpreter
- JSON LD Parser
- Turtle Pretty Printer
- Day of date
- Roman to decimal and converse
- Fibonacci, factorial, sort, standard deviation, etc.
- Extensively used with STTL on corese.inria.fr

Example: RDF List to LDScript List

```
# bnode is start of RDF List

function sh:list(bnode) {
    let (select ?bnode (aggregate(?e) as ?list)
        where { ?bnode rdf:rest*/rdf:first ?e } ) {
        return (list)
    }
}
```

Example: RDF List to LDScript List

```
function sh:reclist(bn) {
  let (select ?bn
          (aggregate (if (?b, sh:reclist(?e),
           if (?e = rdf:nil, xt:list(), ?e))) as ?list)
        where {
          ?bn rdf:rest*/rdf:first ?e
          bind (exists { ?e rdf:rest ?a } as ?b)  } ) {
    return (list)
  }
}
```

Example: SHACL

- Validate graph with SHACL

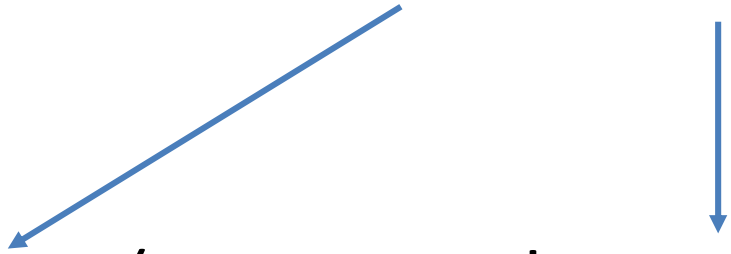
```
select (sh:conform(?g) as ?b) (xt:turtle(?g) as ?t)
where {
        bind (sh:shacl() as ?g)
}
```

SHACL Example

```
us:test a sh:NodeShape ;
sh:targetClass foaf:Person ;
sh:property [
        sh:path foaf:knows ;
        sh:minCount 1;
        sh:class foaf:Person
]
```

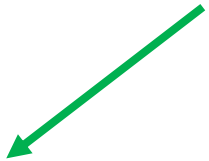
SHACL Extension: Function

```
sh:property [
        sh:path foaf:knows ;
        xsh:function [ us:test (foaf:age 62) ]
]

function us:test(source, node, param) {
    let ((pred val) = param) {
        xt:value(node, pred) >= val
    }
}
```



SHACL Extension: Function

```
xsh:function [ us:test ( [ sh:class foaf:Person ] ) ]

function us:test(source, node, param) {
    let ((shape) = param) {
        sh:eval(shape, node)
    }
}
```



SHACL Extension: Function

```
sh:path [xsh:predicatePath xsh:subject] ;

function xsh:predicatePath(focus, source, node, exp) {
    let (list = xsh:triplePath(focus, source, node, exp)) {
        xt:merge(maplist(xt:predicate, list))
    }
}
```

SHACL Extension: Expression

xsh:evaluate (rq:gt us:length (rq:mult 2 us:width))

SHACL: Validation Report

```
<urn:uuid:2ea80e70-d8d8-423c-a1c5-019dee993b7b>
  a sh:ValidationResult ;
  sh:focusNode _:b30201 ;
  sh:focusNodeDetail [sh:datatype xsd:integer ;
  sh:path sh:maxLength ;
  sh:path sh:minLength] ;
  sh:resultMessage "Fail at: [sh:maxCount 1 ;
  sh:minCount 1 ;
  sh:node <http://www.w3.org/ns/shacl-shacl#PathShape> ;
  sh:path sh:path]" ;
  sh:resultPath sh:path ;
  sh:resultSeverity sh:Violation ;
  sh:sourceConstraintComponent sh:MaxCountConstraintComponent ;
  sh:sourceShape _:b30333 ;
  sh:value 2 .
```

SHACL: RDF Graph Pretty Print

```
sh:property [sh:datatype xsd:string ;
  sh:path (sh:languageIn
  [sh:zeroOrMorePath rdf:rest]
  rdf:first)] ;
sh:property [sh:datatype xsd:integer ;
  sh:path sh:maxLength ;
  sh:path sh:minLength] ;
sh:property [sh:datatype xsd:integer ;
  sh:path sh:maxExclusive ;
  sh:path sh:maxInclusive ;
  sh:path sh:minExclusive ;
  sh:path sh:minInclusive] ;
sh:property [sh:in (sh:IRI
  sh:BlankNodeOrIRI
  sh:IRIOrLiteral
  sh:Literal
  sh:BlankNodeOrLiteral
  sh:BlankNode) ;
  sh:path sh:nodeKind] ;
sh:targetClass sh:NodeShape .
```

Conclusion

- SPARQL Function Language
- Extension of Filter Language
- SPARQL -> Function -> SPARQL
- SPARQL Extension
- « Java Ultra Lite » where objects are RDF terms

- SHACL interpreter written with SPARQL Function
- Corese Semantic Web Factory
  - http://project.inria.fr/corese
  - http://ns.inria.fr/sparql-extension
- Implementation of Core with Jena, Mines St Etienne

Perspective

- Type system, Type checking
- Structured Editor
- Second implementation with Jena
- W3C Member submission