Using SPARQL queries to express integrity constraints in RDF graphs

Final internship report

Leandro Lovisolo

INRA SupAgro and INRIA GraphiK
Montpellier, France

February 25, 2016

Part I

Preliminaries

Problem statement

- We have an RDF graph where we store experimental data extracted from tables in scientific publications.
- The data extraction process is done semi-manually, thus it's very error-prone.
- Therefore, **we want to verify the integrity of the annotated data automatically.**

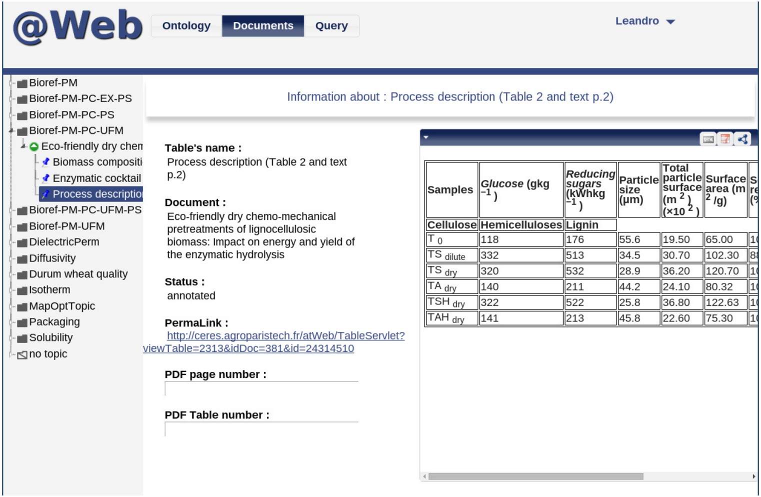
The **@Web** platform
Introduction

- A software platform used to annotate tables from scientific publications in heterogeneous formats (PDF files, Excel spreadsheets, etc.)
- Data is stored in an RDF graph following a predefined OWL ontology.
- Goal of my internship: add integrity constraint checking capabilities to the **@Web** platform[1].

[1] `http://www6.inra.fr/cati-icat-atweb/Web-platform`

The **@Web** platform

Screenshot



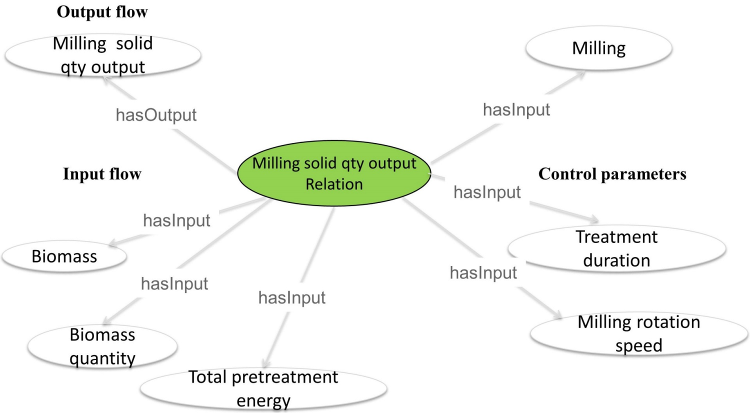
The **@Web** platform

n-ary relation pattern

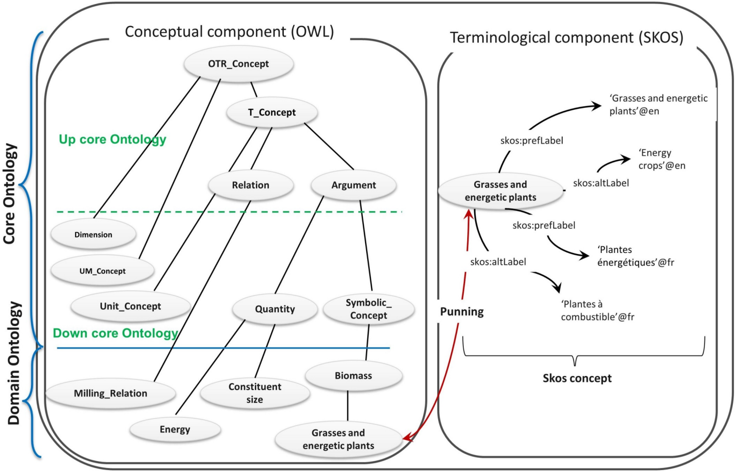
- We're trying to represent experiments composed of many inputs and a single output.
- An OWL ontology is created where OWL classes are defined for each kind of experiment we're interested in representing.
- Instances of each experiment class are connected to their respective input arguments and output argument via OWL object and data properties.
- We're thus defining a pattern for *n*-ary relations.

The **@Web** platform

Example *n*-ary relation



@Web ontology



Annotated tables

Screenshot

n°	Output solid constituent size Unit : mm	Treatment	Experience number Unit : 1	Process step number Unit : 1	Biomass	Biomass quantity Unit : g	Total pretreatment energy Unit : kW.h.kg-1	Water quantity Unit : l	Rotation speed Unit : min-1	Treatment duration Unit : min	Output solid constituent quantity Unit : g	Temperature Unit : oC	Output liquor quantity Unit : l	Salt	Salt quantity Unit : g
1	3.000e+0	Cutting milling	0.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]					
2		Drying	0.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]	6.000e+1			
3		Wet disk milling	0.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	2.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3	[1.800e+1 ; 2.400e+1]	0.000e+0	Salt	0.000e+0
4		Washing and centrifugation	0.000e+0	4.000e+0	Rice straw	1.000e+3	[-inf ; inf]	0.000e+0	9.000e+3	1.000e+1	1.000e+3	[1.800e+1 ; 2.400e+1]	2.000e+1	Salt	0.000e+0
5		Enzymatic hydrolysis treatment	0.000e+0	5.000e+0	Rice straw	[4.000e-2 ; 6.000e-2]			[-inf ; inf]	4.320e+3	[3.400e-2 ; 5.000e-2]	4.500e+1			
6	3.000e+0	Cutting milling	1.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]					
7		Drying	1.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]	6.000e+1			
8		Hot water treatment	1.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	1.000e+1	0.000e+0	6.000e+1	1.000e+3	1.210e+2	0.000e+0	Salt	0.000e+0
9		Wet disk milling	1.000e+0	4.000e+0	Rice straw	1.000e+3	[-inf ; inf]	1.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3	[1.800e+1 ; 2.400e+1]	0.000e+0	Salt	0.000e+0
10		Washing and centrifugation	1.000e+0	5.000e+0	Rice straw	1.000e+3	[-inf ; inf]	0.000e+0	9.000e+3	1.000e+1	1.000e+3	[1.800e+1 ; 2.400e+1]	2.000e+1	Salt	0.000e+0
11		Enzymatic hydrolysis treatment	1.000e+0	6.000e+0	Rice straw	[4.000e-2 ; 6.000e-2]			[-inf ; inf]	4.320e+3	[3.000e-2 ; 4.500e-2]	4.500e+1			
12	3.000e+0	Cutting milling	2.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]					
13		Drying	2.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]	6.000e+1			
14		Hot water treatment	2.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	1.000e+1	0.000e+0	6.000e+1	1.000e+3	1.350e+2	0.000e+0	Salt	0.000e+0
15		Wet disk milling	2.000e+0	4.000e+0	Rice straw	1.000e+3	[-inf ; inf]	1.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3	[1.800e+1 ; 2.400e+1]	0.000e+0	Salt	0.000e+0
16		Washing and centrifugation	2.000e+0	5.000e+0	Rice straw	1.000e+3	[-inf ; inf]	0.000e+0	9.000e+3	1.000e+1	1.000e+3	[1.800e+1 ; 2.400e+1]	2.000e+1	Salt	0.000e+0
17		Enzymatic hydrolysis treatment	2.000e+0	6.000e+0	Rice straw	[4.000e-2 ; 6.000e-2]			[-inf ; inf]	4.320e+3	[2.800e-2 ; 4.200e-2]	4.500e+1			
18	3.000e+0	Cutting milling	3.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]					
19		Drying	3.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]	6.000e+1			
20		Hot water treatment	3.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	1.000e+1	0.000e+0	6.000e+1	1.000e+3	1.500e+2	0.000e+0	Salt	0.000e+0
21		Wet disk milling	3.000e+0	4.000e+0	Rice straw	1.000e+3	[-inf ; inf]	1.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3	[1.800e+1 ; 2.400e+1]	0.000e+0	Salt	0.000e+0
22		Washing and centrifugation	3.000e+0	5.000e+0	Rice straw	1.000e+3	[-inf ; inf]	0.000e+0	9.000e+3	1.000e+1	1.000e+3	[1.800e+1 ; 2.400e+1]	2.000e+1	Salt	0.000e+0

Guidelines

Screenshot

PrefLabel

Milling solid quantity output relation (en)
Quantité de constituant solide issue du broyage (fr)

Hierarchy

Milling solid quantity output relation

AltLabel

ScopeNote

- When the output of a step is a slurry, you need to pick only one output type between « output liquor quantity » and « output solid quantity » depending on which phase is considered to be dominant between solid and liquid. If no indication is given about which phase is major in the slurry, the output will be set as solid by default and described as such in the sequel of the experiment unless other precisions are given. (en)
- The output quantity of a step is equal to the sum of the quantity of water used and the quantity of biomass present in the step. (en)
- la quantité en sortie d'une étape est calculée comme étant la somme de la quantité d'eau utilisée et de la quantité de biomasse présente à l'étape. (fr)
- Lorsque la sortie d'une étape se présente sous forme d'un mélange solide-liquide indissociable, le mélange sera considéré liquide (« quantité de liquide en sortie ») ou solide (« quantité de solide en sortie ») en fonction de la phase prédominante dans le mélange. Si les proportions liquide/solide du mélange ne sont pas connue, on choisira par défaut une « quantité de solide en sortie », que l'on conservera par la suite dans la description de l'expérience, sauf indication contraire donnée par la suite. (fr)

Relation

Result :

- Output solid constituent quantity

Access :

- **Treatment duration**
- **Biomass quantity**
- **Treatment**
- **Rotation speed**
- **Biomass**
- **Total pretreatment energy**
- **Experience number**
- **Water quantity**
- **Process step number**

Example guideline

Guideline

"The output quantity of a step is equal to the sum of the quantity of water used and the quantity of biomass present in the step."

Example guideline

Guideline

"The output quantity of a step is equal to the sum of the quantity of water used and the quantity of biomass present in the step."

$$output = waterInput + biomassInput$$

Example guideline

An annotated row that doesn't fulfill the guideline

n°	Output solid constituent size Unit : mm	Treatment	Experience number Unit : 1	Process step number Unit : 1	Biomass	Biomass quantity Unit : g	Total pretreatment energy Unit : kW.h.kg-1	Water quantity Unit : l	Rotation speed Unit : min-1	Treatment duration Unit : min	Output solid constituent quantity Unit : g
1	3.000e+0	Cutting milling	0.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]	
2		Drying	0.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]
3		Wet disk milling	0.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	2.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3

Example guideline

An annotated row that doesn't fulfill the guideline

n°	Output solid constituent size Unit : mm	Treatment	Experience number Unit : 1	Process step number Unit : 1	Biomass	Biomass quantity Unit : g	Total pretreatment energy Unit : kW.h.kg-1	Water quantity Unit : l	Rotation speed Unit : min-1	Treatment duration Unit : min	Output solid constituent quantity Unit : g
1	3.000e+0	Cutting milling	0.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]	
2		Drying	0.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]
3		Wet disk milling	0.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	2.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3

$$output = waterInput + biomassInput$$

Example guideline

An annotated row that doesn't fulfill the guideline

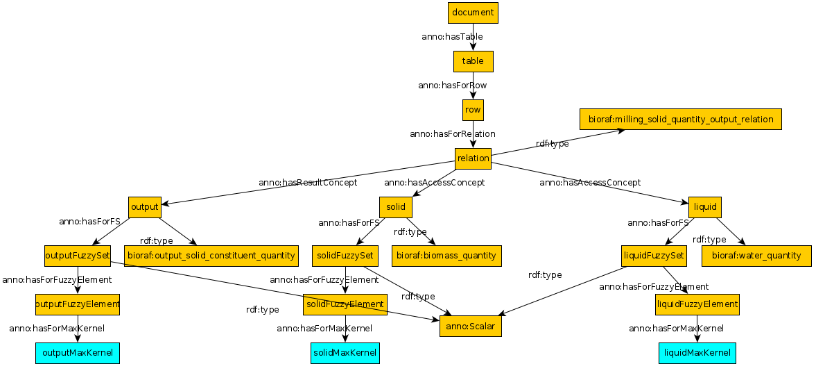
n°	Output solid constituent size Unit : mm	Treatment	Experience number Unit : 1	Process step number Unit : 1	Biomass	Biomass quantity Unit : g	Total pretreatment energy Unit : kW.h.kg-1	Water quantity Unit : l	Rotation speed Unit : min-1	Treatment duration Unit : min	Output solid constituent quantity Unit : g
1	3.000e+0	Cutting milling	0.000e+0	1.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]	0.000e+0	[-inf ; inf]	[-inf ; inf]	
2		Drying	0.000e+0	2.000e+0	Rice straw	[-inf ; inf]	[-inf ; inf]			[-inf ; inf]	[-inf ; inf]
3		Wet disk milling	0.000e+0	3.000e+0	Rice straw	1.000e+3	[-inf ; inf]	2.000e+1	[-inf ; inf]	[-inf ; inf]	1.000e+3

$$output = waterInput + biomassInput$$

$$1000 = 20 + 1000$$

Example guideline

Underlying RDF graph



Part II

RDF data validation: survey of the state of the art

Shape Expressions

Introduction

- It's a validation language for RDF graphs, inspired in regular expressions.
- Allows specifying patterns, or *shapes*, that triples in an RDF graph must conform to.
- Lets one decide whether a RDF graph satisfies all the required shapes.
- Also possible to deduce which triples conform to which shapes (useful for classification.)
- Roughly comparable to what the Data Definition Language (DDL) does for SQL databases, or what XML Schema does for XML documents.

Shape Expressions

Example shape

```
<UserShape> {
  ( foaf:name xsd:string |
    foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string
  ),

  foaf:mbox shex:IRI ?
}
```

Shape Expressions

Example shape

```
<UserShape> {
  ( foaf:name xsd:string |
    foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string
  ),

  foaf:mbox shex:IRI ?
}
```

Valid

```
:Bob
  foaf:givenName "Bob" ;
  foaf:familyName "Smith" ;
  foaf:mbox <mail:bob@example.org> .

:Thompson
  foaf:givenName "Joe", "Joseph" ;
  foaf:familyName "Thompson" ;
  foaf:mbox <mail:joe@example.org> .
```

Invalid

```
# missing :familyName.
:Anna
  foaf:givenName "Bob" ;
  foaf:mbox <mail:bob@example.org> .

# multiple foaf:names.
:Pete
  foaf:name "Peter", "Pete" ;
```

Shape Expressions

Semantic actions

Semantic actions are arbitrary snippets of code that are executed after a rule is evaluated.

- Possible to express more complex validation rules (e.g. arithmetic constraints).
- Multiple programming languages can be supported, depending on the implementation of shape expressions.
- Other uses: transforming RDF triplets into different formats, generating forms for user interfaces automatically, etc.

Shape Expressions

Semantic actions

Semantic actions are arbitrary snippets of code that are executed after a rule is evaluated.

- Possible to express more complex validation rules (e.g. arithmetic constraints).
- Multiple programming languages can be supported, depending on the implementation of shape expressions.
- Other uses: transforming RDF triplets into different formats, generating forms for user interfaces automatically, etc.

Example:

```
:reportedOn xsd:dateTime
    %js{ report = _.o; return true; %},
(:reproducedBy @<EmployeeShape>,
 :reproducedOn xsd:dateTime
    %js{ return _.o.lex > report.lex; %}
    %sparql{ ?s :reportedOn ?rpt . FILTER (?o > ?rpt) %}
)
```

Shape Expressions

Available implementations

- *ShExcala*[2]
  - Implemented in the Scala programming language
  - Can be used from any JVM language (good for **@Web**)
  - **Doesn't implement semantic actions**
- *FancyShExDemo*[3]
  - Implemented in the JavaScript programming language
  - Prototype/proof-of-concept implementation of shape expressions
  - Handles semantic actions
  - Able to generate SPARQL queries

[2] `http://labra.github.io/ShExcala/`

[3] `https://www.w3.org/2013/ShEx/FancyShExDemo`

Shape Expressions

Problems

- Semantic actions are absolutely needed if, in addition to the shape of our RDF graph, we want to validate the data itself (e.g. $output = waterInput + biomassInput$).
- The only available implementation for Shape Expressions that actually supports semantic actions is a proof of concept library built in JavaScript (FancyShExDemo).
- Hard to integrate into a JVM-based app.
- The Shape Expressions draft doesn't specify how semantic actions should be implemented, thus if we move to another Shape Expressions engine in the future, our semantic actions could break.

SHACL
Introduction

- SHACL[4] is a language for describing constraints in RDF graphs.
- Constraints are grouped into *shapes* that apply to nodes in a *data graph*.
- Shapes are described in RDF and stored in a *shapes graph*.
- The simplest interface to a SHACL processor has two inputs:
  - A data graph containing the data to be validated
  - A shapes graph containing shape definitions

[4] `https://www.w3.org/TR/shacl/`

SHACL

An example shape

Shape

```
ex:UserShape
  a sh:Shape ;
  sh:property [
    sh:predicate foaf:name ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
  ] ;
  sh:property [
    sh:predicate foaf:mbox ;
    sh:nodeKind sh:IRI ;
    sh:minCount 1 ;
  ] .
```

SHACL

An example shape

Shape

```
ex:UserShape
  a sh:Shape ;
  sh:property [
    sh:predicate foaf:name ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
  ] ;
  sh:property [
    sh:predicate foaf:mbox ;
    sh:nodeKind sh:IRI ;
    sh:minCount 1 ;
  ] .
```

Valid data

```
:User1
  a foaf:Person ;
  foaf:name "Michel Thomas" ;
  foaf:mbox <mailto:mt@example.org> .

:User2
  a foaf:Person ;
  foaf:name "Bob Lee",
  foaf:mbox <mailto:bl@example.org> ;
  foaf:mbox <mailto:bob.lee@example.org> .
```

Invalid data

```
# More than one foaf:name
:User3
  a foaf:Person ;
  foaf:name "Paul McCartney" ;
  foaf:name "Sir James Paul McCartney" ;
  foaf:mbox <mailto:paul.mccartney@example.org> .

# Missing foaf:mbox
:User4
  a foaf:Person ;
  foaf:name "Donald Knuth".
```

SHACL

Native constraints

```
ex:LanguageExampleShape
  a sh:Shape ;
  sh:scopeClass ex:Country ;
  sh:constraint [
    sh:message "Values must be literals with German language tag." ;
    sh:sparql """
      SELECT $this ($this AS ?subject)
                   (ex:germanLabel AS ?predicate)
                   (?value AS ?object)
      WHERE {
        $this ex:germanLabel ?value .
        FILTER (!isLiteral(?value) || !langMatches(lang(?value), "de"))
      }
      """ ;
  ] .
```

SHACL

Native constraints

```
ex:LanguageExampleShape
  a sh:Shape ;
  sh:scopeClass ex:Country ;
  sh:constraint [
    sh:message "Values must be literals with German language tag." ;
    sh:sparql """
      SELECT $this ($this AS ?subject)
                   (ex:germanLabel AS ?predicate)
                   (?value AS ?object)
      WHERE {
        $this ex:germanLabel ?value .
        FILTER (!isLiteral(?value) || !langMatches(lang(?value), "de"))
      }
      """ ;
  ] .
```

Valid graph

```
ex:ValidCountry
  a ex:Country ;
  ex:germanLabel "Spanien"@de .
```

Invalid graph

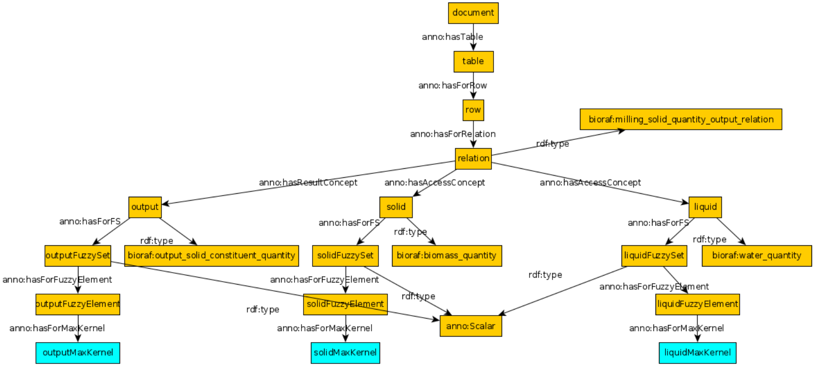
```
ex:InvalidCountry
  a ex:Country ;
  ex:germanLabel "Spain"@en .
```

SHACL

A sample integrity constraint implemented in SHACL (I)

"The output quantity of a step is equal to the sum of the quantity of water used and the quantity of biomass present in the step."

$$output = waterInput + biomassInput$$



SHACL

A sample integrity constraint implemented in SHACL (II)

```
anno:MillingSolidOutputQuantityRelationshipShape
  a sh:Shape ;
  sh:scopeClass bioraf:milling_solid_quantity_output_relation ;

  sh:filterShape [
    sh:inverseProperty [
      sh:predicate anno:hasForRelation ;
      sh:valueShape [
        sh:inverseProperty [
          sh:predicate anno:hasForRow ;
          sh:valueClass anno:Table ;
          sh:minCount 1 ;
          sh:maxCount 1 ;
        ] ;
      ] ;
      sh:minCount 1 ;
      sh:maxCount 1 ;
    ]
  ] ;

  sh:property [
    sh:predicate core:hasAccessConcept ;
    sh:qualifiedValueShape [
      sh:property [
        sh:predicate rdf:type ;
        sh:hasValue bioraf:biomass_quantity
      ]
    ] ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedMaxCount 1 ;
  ] ;

  sh:property [
    sh:predicate core:hasAccessConcept ;
    sh:qualifiedValueShape [
      sh:property [
        sh:predicate rdf:type ;
        sh:hasValue bioraf:water_quantity
      ]
    ] ;
    sh:qualifiedMinCount 1 ;
    sh:qualifiedMaxCount 1 ;
  ] ;

  sh:property [
    sh:predicate core:hasResultConcept ;
    sh:valueClass bioraf:output_solid_constituent_quanti
    sh:minCount 1 ;
    sh:maxCount 1 ;
  ] ;
```

SHACL

A sample integrity constraint implemented in SHACL (IIII)

```
        sh:constraint [
          sh:predicate anno:width ;
          sh:sparql """
            SELECT $this ($this AS ?subject)
                   (CONCAT("Output quantity must be the sum of the solid and liquid input quantities
                            (solid=", STR(?solid_qty),
                           ", liquid=", STR(?liquid_qty),
                           ", output=", STR(?output_qty), ")") as ?message)
            WHERE {
              $this core:hasAccessConcept ?solid ;
                    core:hasAccessConcept ?liquid ;
                    core:hasResultConcept ?output .

              ?solid a bioraf:biomass_quantity ;
                     anno:hasForFS [a anno:Scalar ;
                                   anno:hasForFuzzyElement /
                                   anno:hasForMaxKernel ?solid_qty] .

              ?liquid a bioraf:water_quantity ;
                      anno:hasForFS [a anno:Scalar ;
                                    anno:hasForFuzzyElement /
                                    anno:hasForMaxKernel ?liquid_qty] .

              ?output a bioraf:output_solid_constituent_quantity ;
                      anno:hasForFS [a anno:Scalar ;
                                    anno:hasForFuzzyElement /
                                    anno:hasForMaxKernel ?output_qty] .

              FILTER (xsd:float(?output_qty) !=
                      xsd:float(?solid_qty) + xsd:float(?liquid_qty))
            }
          """ ;
      ] .
```

SHACL

Pros and cons

Pros:

- Constraints are represented as RDF triples; no additional storage medium needed.
- Rich core constraints vocabulary.
- Possible to define arbitrary constraints using SPARQL.
- SHACL implementation readily available (Java language).
- Already being used in the industry (TopQuadrant).

Cons:

- Constraints involving properties from different nodes require describing the graph structure within SPARQL queries, rendering the SHACL shapes redundant.

Plain SPARQL

Idea

- Write SPARQL[5] queries that implement *negative constraints*: only return data that violates a particular constraint.
- In our particular case, we will return relation instances that violate a constraint.

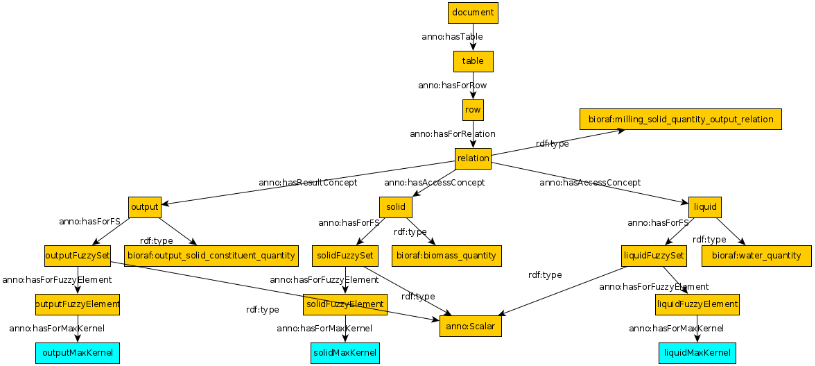
[5] `https://www.w3.org/TR/sparql11-query`

Plain SPARQL

Example constraint (I)

"The output quantity of a step is equal to the sum of the quantity of water used and the quantity of biomass present in the step."

$$output = waterInput + biomassInput$$



Plain SPARQL

Example constraint (II)

```
SELECT ?docid ?doctitle ?tableid ?tabletitle ?rownum  ?solid_qty ?liquid_qty ?output_qty WHERE {
?doc anno:hasForID ?docid ;
     dc:title ?doctitle ;
     anno:hasTable ?table .

?table anno:hasForID ?tableid ;
       dc:title ?tabletitle ;
       anno:hasForRow ?row .

?row anno:hasForRowNumber ?rownum ;
     anno:hasForRelation ?relation .

?relation a bioraf:milling_solid_quantity_output_relation ;
          core:hasAccessConcept ?solid ;
          core:hasAccessConcept ?liquid ;
          core:hasResultConcept ?output] .

?solid a bioraf:biomass_quantity ;
       anno:hasForFS [a anno:Scalar ;
                      anno:hasForFuzzyElement /
                      anno:hasForMaxKernel ?solid_qty] .

?liquid a bioraf:water_quantity ;
        anno:hasForFS [a anno:Scalar ;
                       anno:hasForFuzzyElement /
                       anno:hasForMaxKernel ?liquid_qty] .

?output a bioraf:output_solid_constituent_quantity ;
        anno:hasForFS [a anno:Scalar ;
                       anno:hasForFuzzyElement /
                       anno:hasForMaxKernel ?output_qty] .

FILTER (xsd:float(?output_qty) != xsd:float(?solid_qty) + xsd:float(?liquid_qty)) }
```

Part III

Implementation

Demo

Part IV

Conclusions

Conclusions

- Shape Expressions are well suited for describing the structure of a graph and they can potentially be extended to do more complex kinds of validation via semantic actions. For the moment there are no feature-complete and production-ready implementations.
- SHACL also works well for describing the shape of a graph and is production-ready but the mechanism for expressing native constraints in SPARQL requires describing the shape of the graph again within the SPARQL query, rendering SHACL redundant.
- SPARQL remains the best suited tool for expressing the kind of constraints required in the **@Web** platform.

Future work

- Implement *positive constraints* in **@Web** for classification purposes (i.e. return all relation instances that satisfy a set of conditions)
- Propose a specification for semantic actions in Shape Expressions
- Extend ShExcala (Shape Expressions engine) with a usable implementation of semantic actions

Questions?

Thanks!