An introduction to Shape Expressions

And how they can be used to implement integrity and classification constraints within the **@Web** platform

Leandro Lovisolo
leandro.lovisolo@supagro.inra.fr

INRA SupAgro and INRIA GraphiK
Montpellier, France

October 5, 2015

Outline of the presentation

- Introduction to Shape Expressions
- Examples of integrity and classification constraints for the **@Web** platform written as Shape Expressions
- Comparision with equivalent constraints written as SPARQL queries
- Survey of the available libraries implementating of Shape Expressions

Shape Expressions
Definition

- It’s a validation language for RDF graphs.

Shape Expressions
Definition

- It’s a validation language for RDF graphs.
- Allows specifying patterns, or *shapes*, that triples in an RDF graph must conform to.

Shape Expressions

Definition

- It's a validation language for RDF graphs.
- 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.

Shape Expressions
Definition

- It's a validation language for RDF graphs.
- 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.)

Shape Expressions
Definition

- It’s a validation language for RDF graphs.
- 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.

Motivating example (I)

We're going to model an issue tracking system.

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,
  - is reported on some date,

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,
  - is reported on some date,
  - can be reproduced by an user on some date,

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,
  - is reported on some date,
  - can be reproduced by an user on some date,
  - is either assigned or unassigned,

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,
  - is reported on some date,
  - can be reproduced by an user on some date,
  - is either assigned or unassigned,
  - is related to other issues.

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,
  - is reported on some date,
  - can be reproduced by an user on some date,
  - is either assigned or unassigned,
  - is related to other issues.
- A user:
  - has one full name, **OR**

Motivating example (I)

We're going to model an issue tracking system.

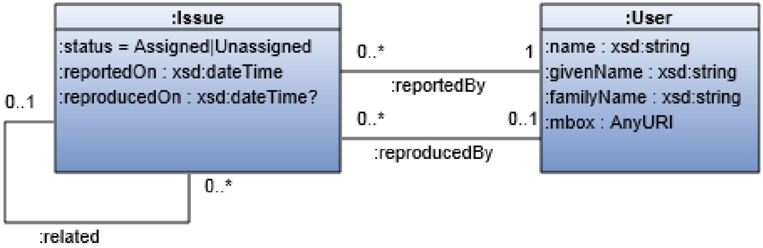
- An issue:
  - is reported by a user,
  - is reported on some date,
  - can be reproduced by an user on some date,
  - is either assigned or unassigned,
  - is related to other issues.
- A user:
  - has one full name, **OR**
  - has several given names and one family name,

Motivating example (I)

We're going to model an issue tracking system.

- An issue:
  - is reported by a user,
  - is reported on some date,
  - can be reproduced by an user on some date,
  - is either assigned or unassigned,
  - is related to other issues.
- A user:
  - has one full name, **OR**
  - has several given names and one family name,
  - can have one mbox.

Motivating example (II)



Example data (Turtle syntax)

Valid

```
:Issue1
  :status :unassigned ;
  :reportedBy :Bob ;
  :reportedOn "2013-01-23"^^xsd:date ;
  :reproducedBy :Thompson ;
  :reproducedOn "2013-01-23"^^xsd:date .

: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

```
:Issue2
  :status :checked ;  # invalid :status.
  :reportedBy :Bob ;
  :reportedOn "2013-01-23"^^xsd:date ;
  :reproducedOn "2013-01-20"^^xsd:date ;
     # missing :reproducedBy.
     # :reproducedOn earlier than
     # :reportedOn.

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

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

Shape Expression: User

```
<UserShape> {
  ( foaf:name xsd:string |
    foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string
  ),
  foaf:mbox shex:IRI ?
}
```

Shape Expression: Issue

```
<IssueShape> {
  :status ( :unassigned
            :assigned ),
  :reportedBy @<UserShape>,
  :reportedOn xsd:date,
  ( :reproducedBy @<UserShape>,
    :reproducedOn xsd:date
  )?,
  :related @<IssueShape>*
}
```

The Shape Expressions language (I)

- Using this language, we define a *schema*.

The Shape Expressions language (I)

- Using this language, we define a *schema*.
- A schema is just a set of *shape expressions*.

The Shape Expressions language (I)

- Using this language, we define a *schema*.
- A schema is just a set of *shape expressions*.
- A shape expression is a labelled *pattern*, e.g.:

```
<label>: {
  ... pattern ...
}
```

The Shape Expressions language (I)

- Using this language, we define a *schema*.
- A schema is just a set of *shape expressions*.
- A shape expression is a labelled *pattern*, e.g.:

```
<label>: {
  ... pattern ...
}
```

- A pattern is a *conjunction* (denoted with a comma) of several *expressions*, e.g.:

```
<IssueShape>: {
  :status ( :unassigned :assigned ), # an expression
  :reportedBy @<UserShape>,          # another expression
  :reportedOn xsd:date,              # yet another expression
  ...
}
```

The Shape Expressions language (II)

An expression can be:

- An *arc rule*, which is a *name definition* followed by a *value definition*, e.g.:

```
# name definition  # value definition
:reportedOn        xsd:date
```

The Shape Expressions language (II)

An expression can be:

- An *arc rule*, which is a *name definition* followed by a *value definition*, e.g.:

```
# name definition  # value definition
:reportedOn        xsd:date
```

- A *group rule* which groups several rules as a single rule (useful for cardinality constraints), e.g.:

```
( :reproducedBy @<EmployeeShape>,
  :reproducedOn xsd:dateTime )?
```

The Shape Expressions language (II)

An expression can be:

- An *arc rule*, which is a *name definition* followed by a *value definition*, e.g.:

```
# name definition  # value definition
:reportedOn        xsd:date
```

- A *group rule* which groups several rules as a single rule (useful for cardinality constraints), e.g.:

```
( :reproducedBy @<EmployeeShape>,
  :reproducedOn xsd:dateTime )?
```

- A list of *alternatives* such that the expression matches any of the alternatives in the list, e.g.:

```
( foaf:name xsd:string |
  foaf:givenName xsd:string+, foaf:familyName xsd:string )
```

The Shape Expressions language (III)

The possible *cardinality constraints* are:

The Shape Expressions language (III)

The possible *cardinality constraints* are:

- `?` for rules with cardinality 0 or 1, e.g.:

```
( :reproducedBy @<UserShape>,
  :reproducedOn xsd:dateTime )?
```

The Shape Expressions language (III)

The possible *cardinality constraints* are:

- `?` for rules with cardinality 0 or 1, e.g.:

```
( :reproducedBy @<UserShape>,
  :reproducedOn xsd:dateTime )?
```

- `*` for rules with cardinality at least 0, e.g.:

```
:related @<IssueShape> *
```

The Shape Expressions language (III)

The possible *cardinality constraints* are:

- `?` for rules with cardinality 0 or 1, e.g.:

  ```
  ( :reproducedBy @<UserShape>,
    :reproducedOn xsd:dateTime )?
  ```

- `*` for rules with cardinality at least 0, e.g.:

  ```
  :related @<IssueShape> *
  ```

- `+` for rules with cardinality at least 1, e.g.:

  ```
  foaf:givenName xsd:string+
  ```

The Shape Expressions language (III)

The possible *cardinality constraints* are:

- `?` for rules with cardinality 0 or 1, e.g.:

  ```
  ( :reproducedBy @<UserShape>,
    :reproducedOn xsd:dateTime )?
  ```

- `*` for rules with cardinality at least 0, e.g.:

  ```
  :related @<IssueShape> *
  ```

- `+` for rules with cardinality at least 1, e.g.:

  ```
  foaf:givenName xsd:string+
  ```

- $\{m, n\}$ for rules with cardinality between m and n.

The Shape Expressions language (IV)

A name definition matches predicate names. Can be either:

- an IRI, e.g. `foaf:name`

The Shape Expressions language (IV)

A name definition matches predicate names. Can be either:

- an IRI, e.g. `foaf:name`
- an IRI prefix, e.g. `foaf:~`

The Shape Expressions language (IV)

A name definition matches predicate names. Can be either:

- an IRI, e.g. `foaf:name`
- an IRI prefix, e.g. `foaf:~`
- any predicate except some from a list, e.g. “ `- foaf:name`” matches any predicate but `foaf:name`

The Shape Expressions language (IV)

A name definition matches predicate names. Can be either:

- an IRI, e.g. `foaf:name`
- an IRI prefix, e.g. `foaf:~`
- any predicate except some from a list, e.g. “ `- foaf:name`” matches any predicate but `foaf:name`
- any predicate at all, denoted by the symbol dot (`.`)

The Shape Expressions language (V)

A value definition places a constraint on the kinds of nodes that can be referenced by a predicate. Can be either:

- A *value type*, e.g. `xsd:date`

The Shape Expressions language (V)

A value definition places a constraint on the kinds of nodes that can be referenced by a predicate. Can be either:

- A *value type*, e.g. `xsd:date`
- A *value set* matching any nodes within a set, e.g. `(:assigned, :unassigned)`

The Shape Expressions language (V)

A value definition places a constraint on the kinds of nodes that can be referenced by a predicate. Can be either:

- A *value type*, e.g. `xsd:date`
- A *value set* matching any nodes within a set, e.g. `(:assigned, :unassigned)`
- Any value except those in a particular set, e.g. " `- :checked`"

The Shape Expressions language (V)

A value definition places a constraint on the kinds of nodes that can be referenced by a predicate. Can be either:

- A *value type*, e.g. `xsd:date`
- A *value set* matching any nodes within a set, e.g. `(:assigned, :unassigned)`
- Any value except those in a particular set, e.g. “ `- :checked`”
- A node that has a particular shape, e.g. `@<UserShape>`

The Shape Expressions language (VI)

Shape inheritance can be achieved by using the ampersand symbol (&), e.g.:

```
<PersonShape> {
    ( foaf:name xsd:string
      | foaf:givenName xsd:string+,
        foaf:familyName xsd:string
    ),
    foaf:mbox IRI
}

<UserShape> {
    & <PersonShape>
}

<EmployeeShape> {
    & <PersonShape>,
    foaf:phone IRI+
}
```

The Shape Expressions language (VII)

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

The Shape Expressions language (VII)

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

- Possible to express complex validation rules using.

The Shape Expressions language (VII)

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

- Possible to express complex validation rules using.
- Multiple programming languages can be supported, depending on the implementation of shape expressions.

The Shape Expressions language (VII)

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

- Possible to express complex validation rules using.
- 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.

The Shape Expressions language (VII)

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

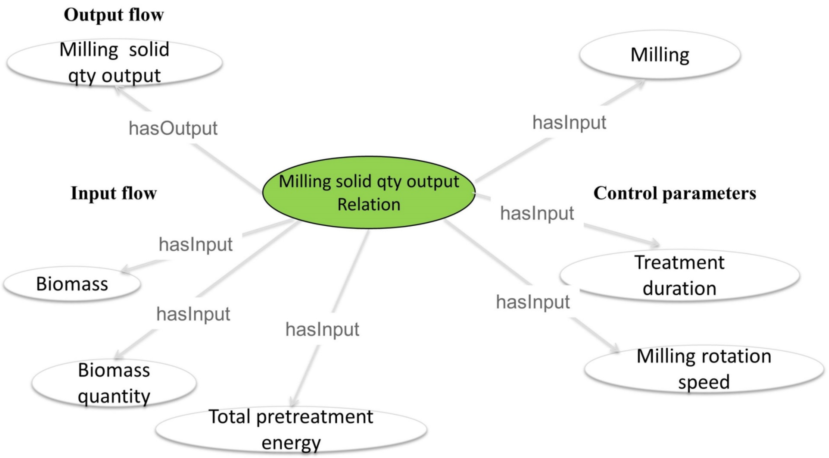
- Possible to express complex validation rules using.
- Multiple programming languages can be supported, depending on the implementation of shape expressions.
- Other uses: transforming RDF triplets into different formats, generating forms for user interfaces automatically, etc.

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) %}
)
```

An integrity constraint

Milling solid quantity output relation



An integrity constraint
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.”

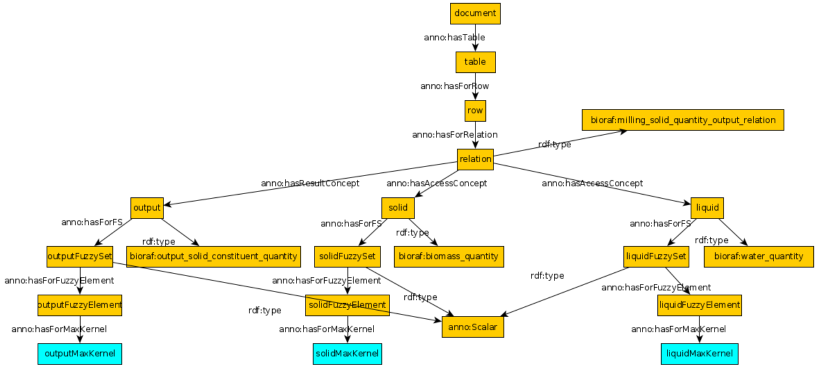
An integrity constraint

SPARQL query

```
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 [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))
}
```

An integrity constraint

Graph view of the SPARQL query



An integrity constraint

Shape Expression

```
<DocumentShape> { rdf:type anno:Document, anno:hasTable @<TableShape> }
<TableShape> { anno:hasForRow @<RowShape> }
<RowShape> { anno:hasForRelation @<MillingSolidQuantityOutputRelationShape> }

<MillingSolidQuantityOutputRelationShape> {
  rdf:type bioraf:milling_solid_quantity_output_relation,
  core:hasAccessConcept @<SolidAccessConceptShape>,
  core:hasAccessConcept @<LiquidAccessConceptShape>,
  core:hasResultConcept @<OutputResultConceptShape>
}

<SolidAccessConceptShape> {
  rdf:type bioraf:biomass_quantity,
  anno:hasForFS @<FuzzySetShape>
}

<LiquidAccessConceptShape> {
  rdf:type bioraf:water_quantity,
  anno:hasForFS @<FuzzySetShape>
}

<OutputAccessConceptShape> {
  rdf:type bioraf:output_solid_constituent_quantity,
  anno:hasForFS @<FuzzySetShape>
}

<FuzzySetShape> {
  rdf:type anno:Scalar,
  anno:hasForFuzzyElement @<FuzzyElementShape>
}

<FuzzyElementShape> { anno:hasForMaxKernel xsd:string }
```

A classification constraint
Guideline

“Topic Bioref-PM: it contains experiments with only one milling followed by the enzymatic hydrolysis (Pre-Milling). It does not include a physico-chemical step but it can include a washing and separation step. All control experiments should be indexed in this topic. (en) ”

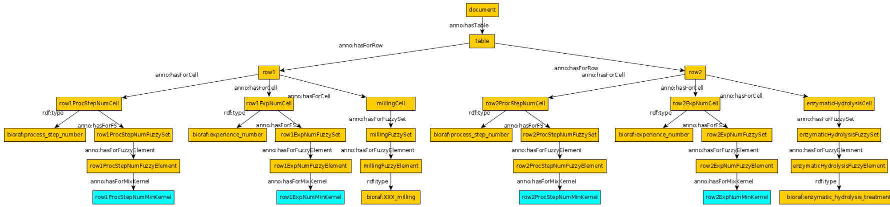
A classification constraint

SPARQL query

```
SELECT ?docid ?doctitle ?tableid ?tabletitle
       ?rownum1 ?expnum1 ?stepnum1 ?milling
       ?rownum2 ?expnum2 ?stepnum2
WHERE {

VALUES ?milling { bioraf:ball_milling
                  bioraf:wet_disk_milling
                  ... }

?doc anno:hasForID ?docid ;
     dc:title ?doctitle ;
     anno:hasTable [anno:hasForID ?tableid ;
                    dc:title ?tabletitle ;
                    anno:hasForRow ?row1, ?row2] .

?row1 anno:hasForRowNumber ?rownum1 ;
      anno:hasForCell [a bioraf:experience_number ;
                       anno:hasForFS /
                       anno:hasForFuzzyElement /
                       anno:hasForMinKernel ?expnum1] ;
      anno:hasForCell [a bioraf:process_step_number ;
                       anno:hasForFS /
                       anno:hasForFuzzyElement /
                       anno:hasForMinKernel ?stepnum1] ;
      anno:hasForCell /
      anno:hasForFS /
      anno:hasForElement /
      a ?milling .

?row2 anno:hasForRowNumber ?rownum2 ;
      anno:hasForCell [a bioraf:experience_number ;
                       anno:hasForFS /
                       anno:hasForFuzzyElement /
                       anno:hasForMinKernel ?expnum2]
      anno:hasForCell [a bioraf:process_step_number ;
                       anno:hasForFS /
                       anno:hasForFuzzyElement /
                       anno:hasForMinKernel ?stepnum2]
      anno:hasForCell /
      anno:hasForFS /
      anno:hasForElement /
      a bioraf:enzymatic_hydrolysis_treatment .

FILTER (?expnum1 = ?expnum2 &&
        xsd:float(?stepnum1) < xsd:float(?stepnum2))
}
```

A classification constraint

Graph view of the SPARQL query



A classification constraint

Shape Expression

```
<DocumentShape> {
  rdf:type anno:Document,
  anno:hasTable @<TableShape>
}

<TableShape> {
  anno:hasForRow @<PreMillingRowShape>,
  anno:hasForRow @<EnzymaticHydrolysisRowShape>
}

<PreMillingRowShape> {
  & <RowShape>,
  anno:hasForCell @<PreMillingCellShape>
}

<PreMillingCellShape> {
  & <RowShape>,
  anno:hasForFS @<PreMillingFuzzySet>
}

<PreMillingFuzzySet> {
  anno:hasForElement (bioraf:ball_mlling
                      bioraf:wet_disk_milling ...)
}

<EnzymaticHydrolysisRowShape> {
  anno:hasForCell @<EnzymaticHydrolysisCellShape>
}
```

```
<EnzymaticHydrolysisCellShape> {
  anno:hasForFS @<EnzymaticHydrolysisFuzzySet>
}

<EnzymaticHydrolysisFuzzySet> {
  anno:hasForElement bioraf:enzymatic_hydrolysis_treat
}

<RowShape> {
  anno:hasForRowNumber xsd:integer,
  anno:hasForCell @<ExperienceNumberCellShape>,
  anno:hasForCell @<ProcessStepeNumberCellShape>
}

<ExperienceNumberCellShape> {
  rdf:type bioraf:experience_number,
  anno:hasForFS @<FuzzySetShape>
}

<ProcessStepNumberCellShape> {
  rdf:type bioraf:experience_number,
  anno:hasForFS @<FuzzySetShape>
}

<FuzzySetShape> {
  rdf:type anno:Scalar,
  anno:hasForFuzzyElement @<FuzzyElementShape>
}

<FuzzyElementShape> {anno:hasForMinKernel xsd:string}
```

Libraries implementing Shape Expressions

- *ShExcala*

Libraries implementing Shape Expressions

- *ShExcala*
  - Implemented in the Scala programming language

Libraries implementing Shape Expressions

- *ShExcala*
  - Implemented in the Scala programming language
  - Can be used from any JVM language (good for **@Web**)

Libraries implementing Shape Expressions

- *ShExcala*
  - Implemented in the Scala programming language
  - Can be used from any JVM language (good for **@Web**)
  - **Doesn't implement semantic actions**

Libraries implementing Shape Expressions

- *ShExcala*
  - Implemented in the Scala programming language
  - Can be used from any JVM language (good for **@Web**)
  - **Doesn't implement semantic actions**
- *FancyShExDemo*

Libraries implementing Shape Expressions

- *ShExcala*
  - Implemented in the Scala programming language
  - Can be used from any JVM language (good for **@Web**)
  - **Doesn't implement semantic actions**
- *FancyShExDemo*
  - Implemented in the JavaScript programming language

Libraries implementing Shape Expressions

- *ShExcala*
  - Implemented in the Scala programming language
  - Can be used from any JVM language (good for **@Web**)
  - **Doesn't implement semantic actions**
- *FancyShExDemo*
  - Implemented in the JavaScript programming language
  - Prototype/proof-of-concept implementation of shape expressions

Libraries implementing Shape Expressions

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

Libraries implementing Shape Expressions

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

Statistics for the two SPARQL constraints

Statistics for the two SPARQL constraints

Integrity constraint (milling solid output quantity relation):

Statistics for the two SPARQL constraints

Integrity constraint (milling solid output quantity relation):

- Detected **22** inconsistencies out of **79** instances of the relation.

Statistics for the two SPARQL constraints

Integrity constraint (milling solid output quantity relation):

- Detected **22** inconsistencies out of **79** instances of the relation.

Classification constraint (BIOREF-PM topic):

Statistics for the two SPARQL constraints

Integrity constraint (milling solid output quantity relation):

- Detected **22** inconsistencies out of **79** instances of the relation.

Classification constraint (BIOREF-PM topic):

- Found **342** candidate experiments in **30** different documents.

Thanks!