

Abstract

In this paper we study the use of SPARQL queries to verify a set of integrity constraints in an ontology dedicated to the annotation of experimental data in biorefinery, packaging and other domains.

Keywords: Semantic Web, RDF, OWL, SPARQL, Integrity constraints.

1 Introduction

The *@Web platform*¹ is a Semantic Web application that allows domain experts to annotate experimental data in scientific documents, and researchers to explore and query those annotations via a graphical user interface. The annotations are stored in a publicly accessible *RDF*² graph using a vocabulary predefined in an *OWL*³ ontology, and shared with the research community.

Given the error-prone nature of the data annotation process, a set of integrity constraints has been identified that all annotated data must fulfill. It is desired to validate these constraints automatically and report any validation errors to the domain expert during the data annotation process.

To this end, we first surveyed the current W3C recommendations for constraining the contents of RDF graphs, and the available tools implementing these recommendations. We decided to focus our analysis on *SPARQL*⁴, *Shape Expressions*⁵ and *SHACL*⁶. We then implemented a set of test constraints using each of the available tools and compared them according to expressiveness, verbosity, readability, etc.

Our analysis shows that none of the libraries implementing the Shape Expressions recommendation fully support all our use cases. We also observe that certain constraints expressed in SHACL require nesting SPARQL queries that are comparable in length to stand-alone SPARQL queries implementing those same constraints, thus defeating the purpose of an alternate constraint language.

We finish our analysis by comparing the running times of a set of test constraints implemented as SPARQL queries against different triple stores supported by *Jena*⁷, a Java library for building Semantic Web applications which is already used in @Web for other purposes.

1.1 The @Web platform

The **@Web** platform is a software system that allows researchers to extract heterogeneous experimental data from tables in scientific publications (such as papers, articles, CSV files, etc.) and store it in an RDF graph following a uniform structure.

¹<https://www6.inra.fr/cati-icat-atweb>

²<http://www.w3.org/TR/rdf11-primer/>

³<http://www.w3.org/TR/owl-primer/>

⁴<http://www.w3.org/TR/sparql11-query/>

⁵<http://www.w3.org/2013/ShEx/Primer>

⁶<http://www.w3.org/TR/shacl/>

⁷<https://jena.apache.org/>

Do we really need to mention the comparison we did between SPARQL, SHACL and Shape Expressions?

Shouldn't we focus on SPARQL instead and leave out the details on why we chose SPARQL over the other alternatives?

The platform provides access to the RDF graph and also graphical tools to explore and query the data.

Some typical examples of data extracted from scientific publications using the @Web platform include input and output parameters associated to a milling operation in a multi-step biorefinery experiment, the kind of biomass associated to that experiment, chemical properties such as glucose rate, oxygen permeability, etc.

1.2 Ontology

Internally, @Web stores data in an RDF graph following a vocabulary defined in an OWL ontology. In this ontology, a model for n -ary relations in quantitative experimental data is established. Under this model, n -ary relations have one or more input parameters controlling various aspects of the experiment, and exactly one output parameter.

Conceptually, the ontology can actually be thought of as two separate ontologies:

- a core ontology, where a *Relation* OWL class is defined, along with object properties *hasInput* and *hasOutput*, cardinality constraints (each relation has one or more input parameters and exactly one output parameter), etc.
- a domain ontology with *Relation* subclasses for each kind of experimental data relevant for a particular domain. Some examples in the biorefinery domain include milling processes, extrusion processes, enzymatic treatments, etc.

A number of additional concepts are defined in both ontologies such as magnitudes and units of measurement, but for the sake of brevity we won't elaborate any further.

Figure 1 shows an example of an n -ary relation representing a milling step in an experiment in the biorefinery domain.

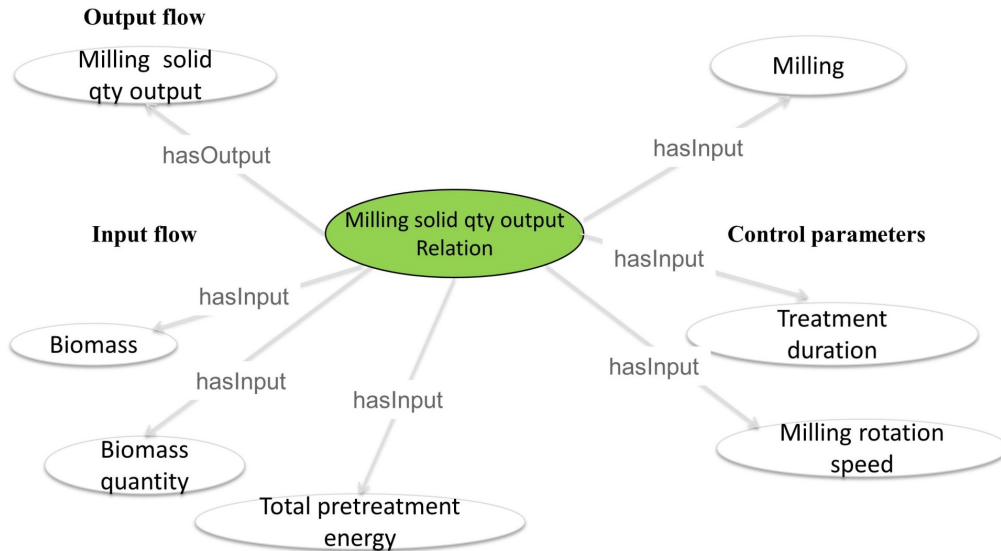


Figure 1: *Milling solid quantity output* relation: an example of an n -ary relation in @Web.

Finally, the **@Web** platform uses concepts defined in an auxiliary OWL ontology to store metadata associated to scientific publications such as document title, authors, table title and number, etc.

1.3 Annotated tables

The screenshot in Figure 2 illustrates a typical annotated table in the **@Web** platform after it's been extracted from a scientific publication, which in this case belongs to the biorefinery domain.

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 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]	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	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	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]	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	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	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	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]	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	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	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	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]	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	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	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	0.000e+0

Figure 2: An example of an annotated table in **@Web**.

The table contains data from 4 different experiments (see column *Experience number*), each composed of many steps (see column *Process step number*). Each row in this table represents a step in an experiment, and for each row there is an instance of an n -ary relation in the underlying RDF graph. Rows 1 and 3, for example, correspond to cutting milling and wet disk milling steps, respectively, and both are associated with instances of the *milling solid quantity output* relation.

1.4 Guidelines

In the **@Web** platform, each relation is associated with a set of guidelines written in natural language. These guidelines explain the kinds of experiments a relation is meant to represent and in many cases provide a number of rules that all instances of a relation are supposed to fulfill.

Figure 3 shows capture of the screen in **@Web** where an annotator can read the guidelines associated to the *milling solid quantity output* relation, introduced in subsection 1.2.


PrefLabel	Hierarchy
Milling solid quantity output relation (en) Quantité de constituant solide issue du broyage (fr)	L  Milling solid quantity output relation
AltLabel	
ScopeNote	
<p>- 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)</p> <p>- 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)</p> <p>- 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)</p> <p>- 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 connues, 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)</p>	
Relation	
<p>Result :</p> <ul style="list-style-type: none"> Output solid constituent quantity <p>Access :</p> <ul style="list-style-type: none"> Treatment duration Biomass quantity Treatment Rotation speed Biomass Total pretreatment energy Experience number Water quantity Process step number 	

Figure 3: Guidelines associated to the *milling solid quantity output* relation.

1.5 Integrity constraints

The guideline highlighted in yellow in Figure 3 represents a rule that must be valid for all instances of the *milling solid output quantity* relation. We call this kind of rule an *integrity constraint*.

In most cases, integrity constraints can be stated formally as mathematical equations or pseudocode. To show this, the guideline in Figure 3 highlighted in yellow is transcribed below, followed by a possible formalization as a mathematical equation.

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.”

Equation:

$$output = waterInput + biomassInput$$

Once a guideline is stated formally, it should be possible to verify automatically whether it is being fulfilled by an instance of its associated relation.

1.6 Problem statement

The goal of this work is, therefore, to identify a technology or technique that allows expressing integrity constraints formally, and automatically verifying whether an n -ary relation instance satisfies an integrity constraint.

2 Proposed solution

We propose using SPARQL queries for expressing and verifying integrity constraints. Specifically, given an integrity constraint c and its associated relation r , we propose to express c as a SPARQL query that will select all instances of r that do not fulfill c .

Our decision to focus on SPARQL was based on the following facts:

- most general tool
- mature specification
- well supported, efficient implementations available
- strong community around it

Refine this list.

In the following sections we explain in detail our implementation of the proposed solution, we give examples of actual integrity constraints expressed this way, and we show what the implementation looks like from the **@Web** user's point of view.

2.1 Integrity constraints expressed as SPARQL queries

In order to make sense of the result set returned by the execution of a SPARQL query implementing an integrity constraint, some assumptions have to be made.

Let c be an integrity constraint associated to a relation r , and let q be a SPARQL query implementing c . We make the following assumptions about q :

Should we mark this as a definition?

- Each solution in a result set obtained by evaluating q corresponds to exactly one instance of r , which we will call i .
- Each solution in the result set must include a variable **relation** which contains the IRI of i . All other variables will be ignored.
- i must be in violation of c .
- There's at least one solution in the result set corresponding to each instance of r in violation of c .

SPARQL queries implementing integrity constraints will therefore have a structure similar to the example in Listing 1.

Decide if it's worth talking about the structure of queries instead of showing a concrete example.

Listing 1: Structure of a SPARQL query implementing an integrity constraint.

```

1 SELECT ?relation
2 WHERE {
3   ?relation a example:MyRelationClass .
4   # ... restrictions go here
5 }

```

Let's illustrate this with an example. Let r be a relation representing ...

Add a concrete toy example. Idea: input parameter must be greater than output parameter.

2.2 An example of a real integrity constraint implemented as a SPARQL query

In this section we will show how the integrity constraint mentioned in subsection 1.5 is implemented in the @Web platform.

Recall from subsection 1.5 the integrity constraint in question and its mathematical formulation:

“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.”

$$output = waterInput + biomassInput$$

In order to express this integrity constraint as a SPARQL query we must, first, select the nodes in our RDF graph corresponding to the *output*, *waterInput* and *biomassInput* variables in the equation above. Figure 4 shows what a subset of our RDF graph containing an instance of the *milling solid quantity output* relation would look like. We're therefore interested in selecting the nodes painted in a light blue color.

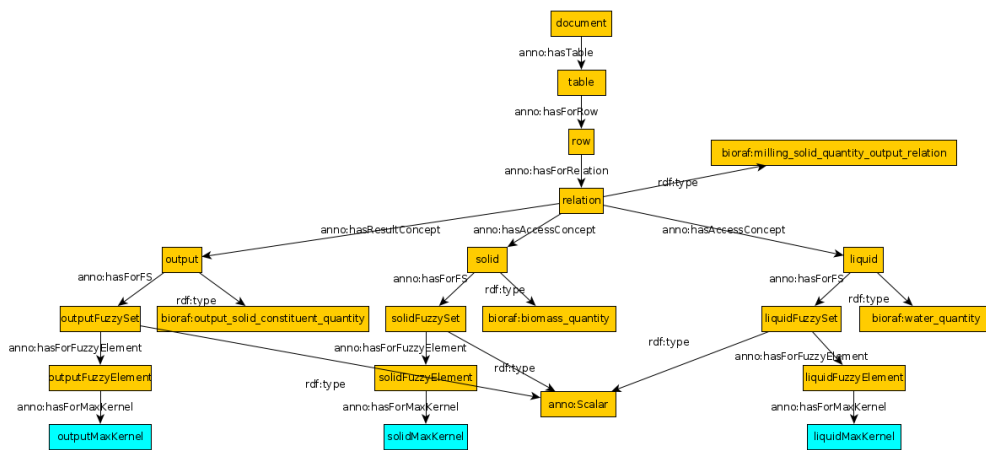


Figure 4: Subset of an RDF graph containing an instance of the *milling solid quantity output* relation.

We're now ready to introduce the SPARQL query implementing this integrity constraint, which you can see in Listing 2.

Should we explain nodes and edges relative to the annotation ontology? (hasForTable, hasForRow, etc.) Should we explain fuzzy sets?

Listing 2: A SPARQL query implementing an integrity constraint.

```

1 SELECT ?relation ?solid_qty ?liquid_qty ?output_qty
2       ?docid ?doctitle ?tableid ?tabletitle ?rownum
3 WHERE {
4   ?doc anno:hasForID ?docid ;
5       dc:title ?doctitle ;
6       anno:hasTable ?table .
7
8   ?table anno:hasForID ?tableid ;
9         dc:title ?tabletitle ;
10        anno:hasForRow ?row .
11
12   ?row anno:hasForRowNumber ?rownum ;
13        anno:hasForRelation ?relation .
14
15   ?relation a bioraf:milling_solid_quantity_output_relation ;
16            core:hasAccessConcept ?solid ;
17            core:hasAccessConcept ?liquid ;
18            core:hasResultConcept ?output] .
19
20   ?solid a bioraf:biomass_quantity ;
21         anno:hasForFS [a anno:Scalar ;
22                       anno:hasForFuzzyElement /
23                       anno:hasForMaxKernel ?solid_qty] .
24
25   ?liquid a bioraf:water_quantity ;
26         anno:hasForFS [a anno:Scalar ;
27                       anno:hasForFuzzyElement /
28                       anno:hasForMaxKernel ?liquid_qty] .
29
30   ?output a bioraf:output_solid_constituent_quantity ;
31         anno:hasForFS [a anno:Scalar ;
32                       anno:hasForFuzzyElement /
33                       anno:hasForMaxKernel ?output_qty] .
34
35   FILTER (xsd:float(?output_qty) != xsd:float(?solid_qty) + xsd:float(?liquid_qty))
36 }
```

A few things to note about this query:

- A variable **relation** is selected, which is bound on line 15 to an instance of the *milling solid output quantity* relation, as per the convention explained in subsection 2.1. Other variables are also selected for debugging purposes, such as the title of the scientific publication and the table from which this relation instance was extracted.
- Nodes for water, biomass and output quantities are selected on lines 23, 28 and 33, respectively. These are the nodes painted in light blue on Figure 4.
- Line 35 discards instances of the *milling solid output quantity* relation that satisfy the equation $output = waterInput + biomassInput$, selecting only those instances that are in violation with the integrity constraint.

2.3 Extending the ontology to represent integrity constraints and validation errors

We decided to extend the **@Web** core ontology to make it possible to store integrity constraints in the ontology itself, and also to store validation errors after running an integrity constraint.

2.3.1 Integrity constraints

A new OWL class **Constraint** was added, along with the following properties:

- **hasRelationClass**, which links an instance of the **Constraint** class with a **Relation** subclass.
- **hasSparqlQuery**, which links an instance of the **Constraint** class with a string literal containing the SPARQL query associated to the integrity constraint.

Both properties have cardinality constraints of exactly one.

2.3.2 Validation errors

Similarly, an OWL class **ConstraintValidationError** was added, along with the properties below:

- **hasConstraint**, which links a **ConstraintValidationError** instance with an instance of the **Constraint** class representing the integrity constraint that produced the validation error.
- **hasRelation**, which links an instance of the **ConstraintValidationError** class with the relation violating the integrity constraint (which is an instance of a **Relation** subclass.)

Both properties also have cardinality constraints of exactly one.

2.4 Experimentation

Pending.

2.5 Related work

Pending.

3 Conclusions

Pending.

4 Future work

Pending.