@Web Constraint Checking: Functional Specification

Leandro Lovisolo <leandro.lovisolo@supagro.inra.fr>,
INRA SupAgro and INRIA GraphiK,
Montpellier, France

December 2015

Abstract

In this document I summarize the changes proposed to the **@Web** platform in order to implement automatic constraint checking using SPARQL queries.

1 Core ontology changes

The following list provides a high level view of the changes proposed to the core ontology.

- A new OWL class `Constraint` is added to the core ontology, which represents constraints expressed as SPARQL queries.
- Instances of the `Constraint` class are associated to their respective relation classes via a new object property `hasForConstraint`.
- A new data property `hasForSPARQLQuery` is added, which connects instances of the `Constraint` class with a string literal holding the actual SPARQL query.
- Constraints are described in natural language with a textual guideline associated to `Constraint` instances via a SKOS scope note.

1.1 Biorefinery domain ontology

This particular domain requires the notion of *topics*, which have associated constraints that require additional information. In the following sections, the concept of topics is explained, and then some changes to the domain ontology are proposed to support expressing the required constraints as SPARQL queries.

1.1.1 Topics

Topics are a way to group relations according to the kind of experiment they model. This grouping is currently done in the **@Web** platform via document topics (e.g., Bioref-PM, Bioref-PM-UFM, Bioref-PM-PC-EX-PS, etc.)

Each experiment must belong to exactly one topic.

Each topic has clearly defined rules to decide whether an experiment (i.e. a set of relation instances) belongs to it or not.

Given an experiment and the topic it belongs to, it is required to check said rules automatically. To this end, rules are encoded as constraints written as SPARQL queries.

1.1.2 Changes to the Biorefinery domain ontology

A new symblic concept `ProcessType` is created, with one subclass for each supported topic. Each such subclass is listed below, with its proposed alternate label between parentheses:

- `Milling` (PM)
- `Milling_PhysicoChemical_Extrusion` (PM-PC-EX-PS)
- `Milling_PhysicoChemical` (PM-PC-PS)
- `Milling_PhysicoChemical_UltraFineMilling` (PM-PC-UFM)
- `Milling_PhysicoChemical_UltraFineMilling_PressSeparation` (PM-PC-UFM-PS)
- `Milling_UltraFineMilling` (PM-UFM)

A new argument is added to the n-ary relations in this domain with the purpose of linking a relation instance to the topic (i.e. `ProcessType` subclass) of the experiment it belongs to.

The `Constraint` instances that verify the topic inclusion rules are associated to the `Relation` class in order to make them available to all subclasses (i.e. all relations).

In a future version of **@Web** there should be a mechanism for ontology-level constraints that would allow a more correct way of expressing topic constraints.

2 Ontology uploading via CSV files

A new screen is added to the **@Web** management UI with the purpose of uploading constraints as CSV files. Such files would have the following columns:

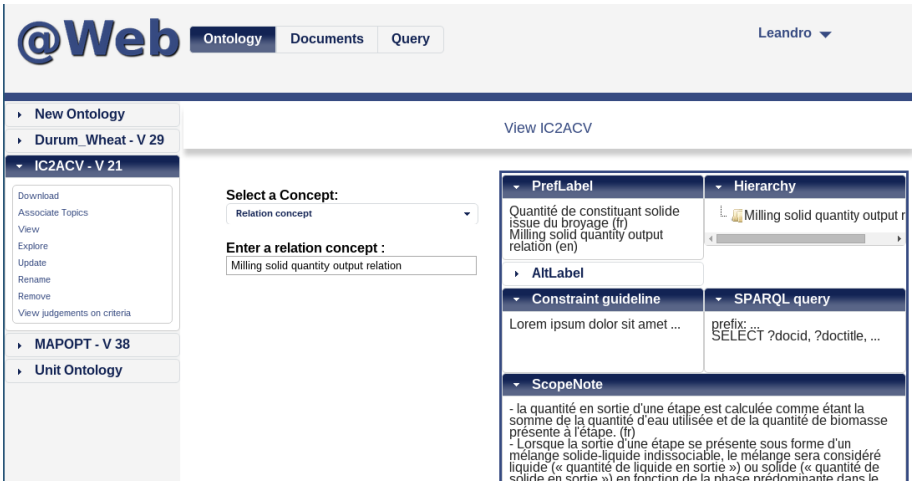
- `prefLabel=EN` (e.g. *Milling*)
- `altLabel=EN` (e.g. *PM*)
- `scopeNote=EN` (textual guideline)
- `Relation_Concept`
- `SPARQL_query`

3 User interface

The constraint visualization and verification user interface is described in the following sections.

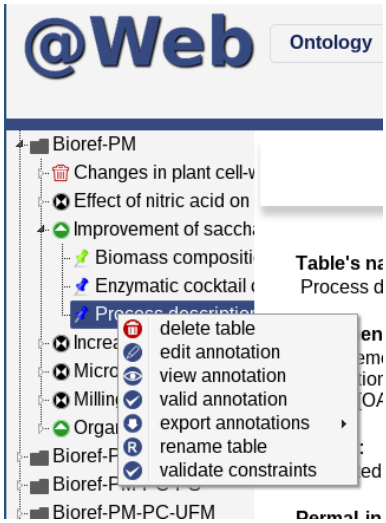
3.1 Constraint visualization

Constraints are displayed next to the scope notes when exploring relations, as shown below.



3.2 Constraint verification

The constraint verification process is done on a per-table basis, and is launched by rigth-clicking on the target table and selecting *validate constraints*, as shown below.



The user is then taken to a screen where they can select the constraints they're interested in verifying. This list is built by compiling the constraints linked to all relations that show up in the table.

[Pending screen sketch]

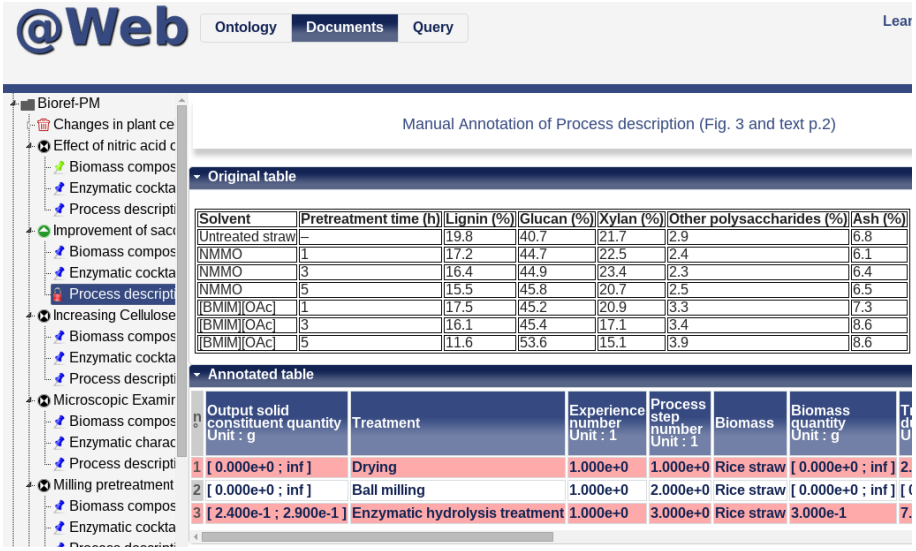
After clicking the *Validate* button, a loading indicator is shown. When the queries finish running the user is taken to a results screen, where a summary of the errors is provided.

[Pending screen sketch]

The table icon is updated to reflect the presence or absence of errors.

[Pending screen sketch]

In the case of errors, when the user enters the table edition screen they will see the rows in a table containing errors highlighted in red. The user can get additional information (such as the name of the constraint violated) by hovering the mouse pointer over the affected row.



It should be noted that we're also interested in running the constraints not only at the table label, but also at the document and topic level. Although this specification doesn't cover how this will be done, it should be taken into consideration in the event of architectural changes to support constraint checking.