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

1.1.1 Categories

Categories are a way to group unit operation relation instances 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.)

Categories are characterized by the two following points:

- Each unit operation relation instance must belong to exactly one category.
- Each category has clearly defined rules to decide whether an experiment (i.e. a set of unit operation relation instances) belongs to it or not.

Thus, given a unit operation relation instance and the category 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 category. 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 that represent unit operations in this domain with the purpose of linking a relation instance with the category (i.e. `ProcessType` subclass) it belongs to.

The `Constraint` instances that verify the category 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 category constraints.

2 Uploading constraints in CSV format

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:

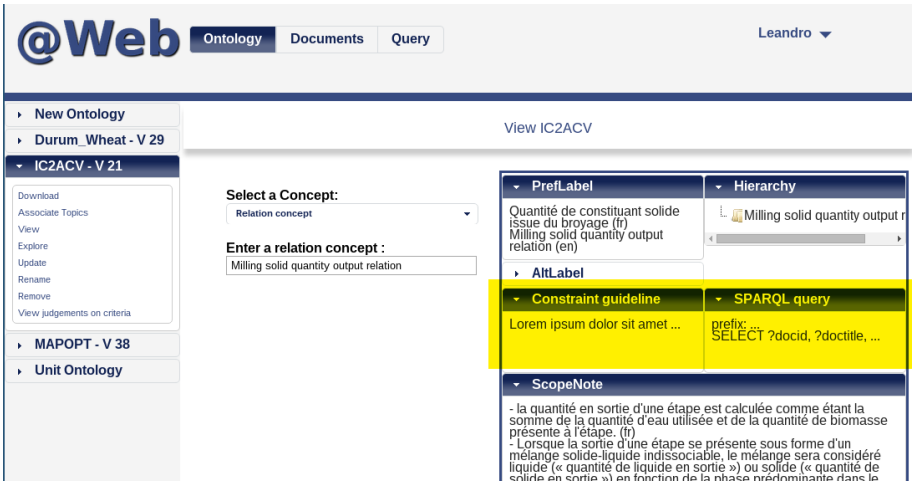
- `prefLabel=EN` (e.g. *Milling*)
- `altLabel=EN` (e.g. *PM*)
- `scopeNote=EN` (textual guideline)
- `Relation_Concept`
- `SPARQL_query`

3 User interface

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

3.1 Constraint visualization

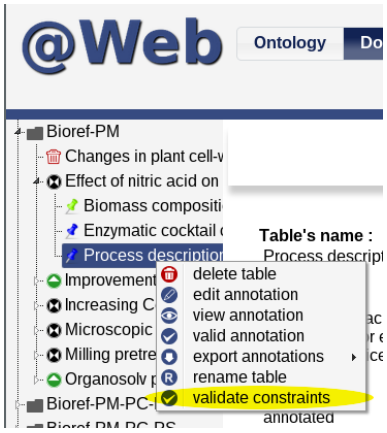
Constraints are displayed next to the scope notes when exploring relations, as shown below.



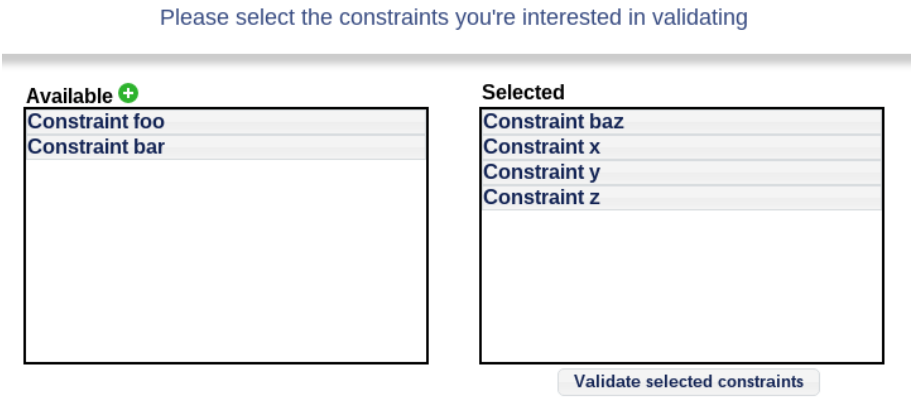
Note that this visualization is read-only.

3.2 Constraint verification

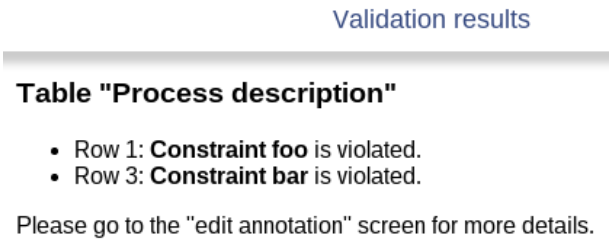
The constraint verification process is done on a per-table basis, and is launched by right-clicking the target table and selecting *validate constraints*, as shown below.



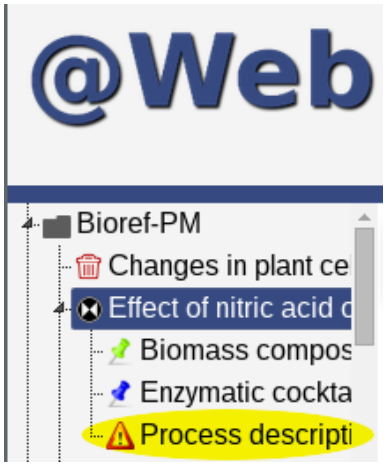
The user is then taken to a screen where they can select the constraints to verify. This list is compiled by finding all constraints linked to the categories associated with all relation instances in the table annotations.



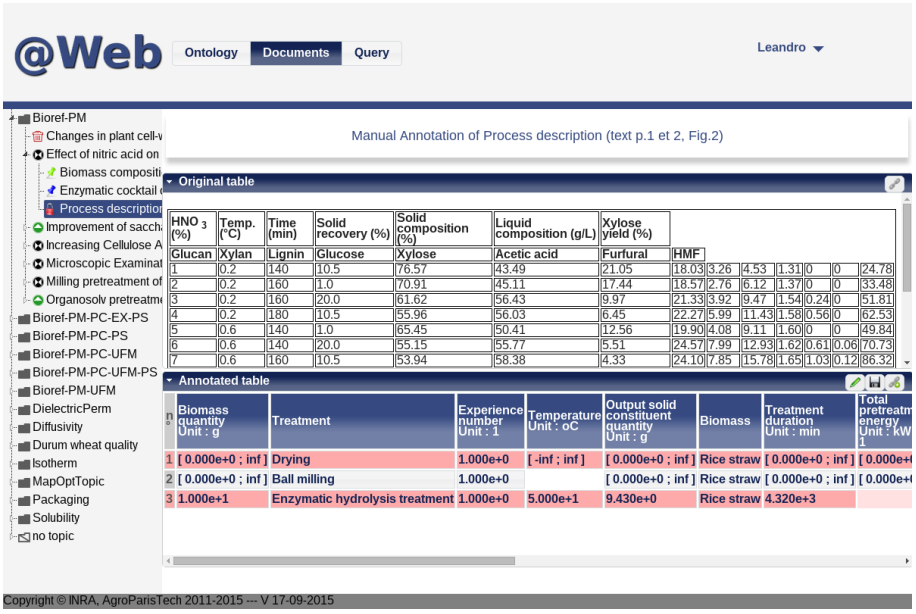
After clicking the *Validate selected constraints* button, a loading indicator is shown. When the queries finish running the user is taken to a results screen sketched below, where a summary of the errors is provided.



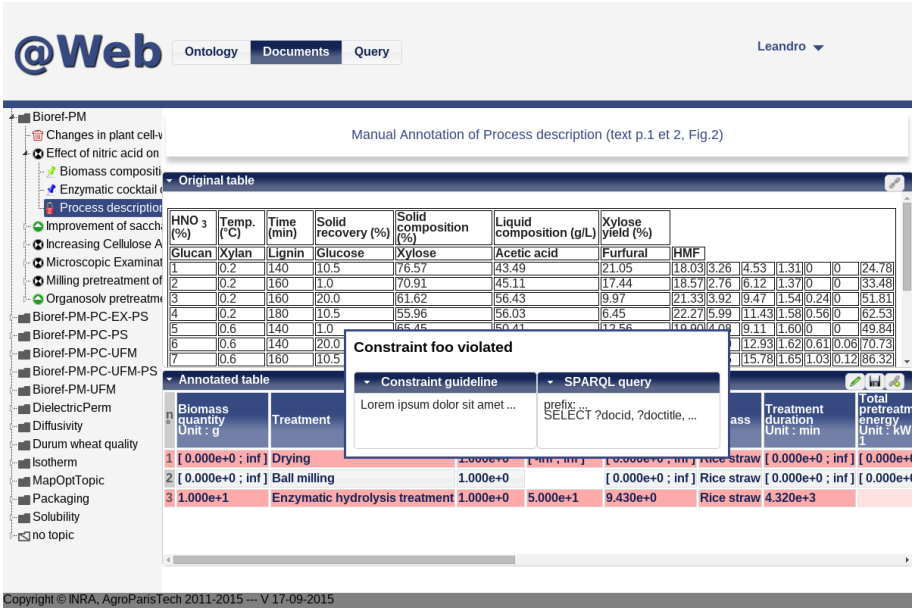
The table icon is updated to reflect the presence or absence of errors.



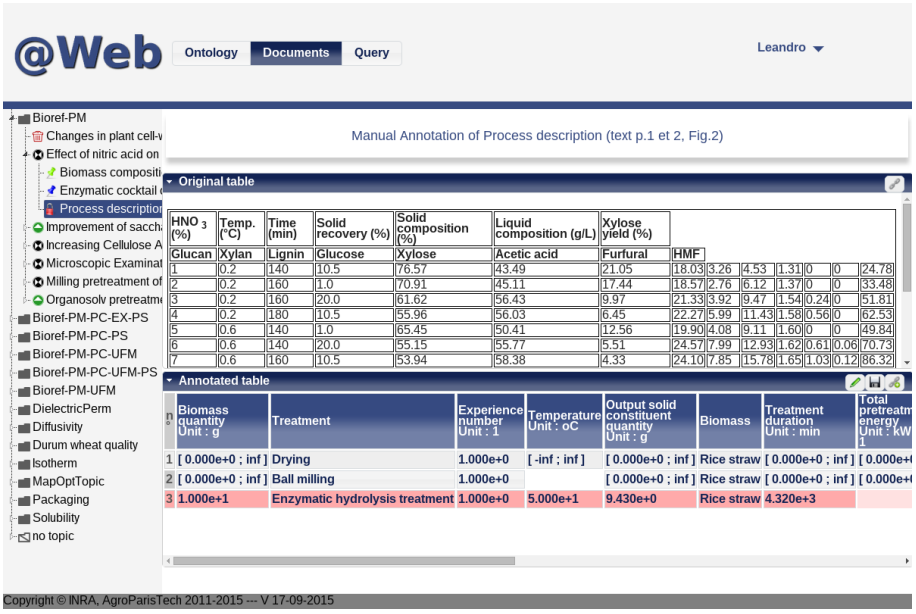
In the case of errors, when the user enters the table edition screen they will see the rows in a table containing errors highlighted in red.



The user can get additional information (such as the name of the constraint violated) by hovering the mouse pointer over the affected row.



After performing modifications on an affected row, the row color goes back to gray.



The table icon also changes color to reflect this new state.