Specification for a Functional View Edit Page

Draft 2017-01-11

Oliver Hopt, Jon Johnson, Wendy Thomas, Joachim Wackerow, Johan Fihn, Olof Olsson

**Time Estimate:** *2-7 days depending on how well you know the flow of the current system and the XMI export. Includes custom interface and some means of capturing this information from the current database.*

# Background

The current position of the Modeling Team is that restrictions of classes as used in Functional Views is to be done through documentation. The reasoning behind limiting restrictions to documentation is to support interoperability. A system supporting two different Functional Views using the same version of a class from the model should be identical in structure, but not necessarily is use.

The edit template for a Functional View currently contains a text box for the name of the Functional View, a text box for a description, and a check list of all available Identifiable classes. All Identifiable classes contained in the Functional View must be checked off to be included in the Functional View. Properties and their data types (primitive or complex) are pulled in during the production of the Functional View.

# Desired Functionality

* Limitation of the check list to first class objects in publishable packages (i.e. remove packages such as Keep, Deprecated, and other packages of non-functional classes)
* Support divisions in the Functional View:
	+ Purpose
	+ Use Cases
	+ Target Audience
	+ Included Classes and noted restrictions per class
		- This section should be auto-generated based on the information captured in the drop-down list
	+ General Documentation
* For each included class:
	+ Drop down of contents (properties and relationships) indicating name, target class, and cardinality of each
	+ For each property and relationship, a Boolean isNotUsed to indicate restriction (non-use) in the particular Functional View
	+ A Functional View specific class level documentation (definition, example, synonyms, explanatory notes). This is documentation that is added to the class as it is used in the Functional View allowing for use case related terminology and examples to be added to the existing class model documentation.

In Drupal, there will be a need to change the form for views. The list of all classes with checkboxes would have to be replaced by a mechanism quite similar to the adding of relationships to classes. The following screen fake shows it in its final state. The lists of used properties and relationships should be filled after choosing the class to be added. If possible, it would be nice to show the properties/relationships cardinalities together with it name. If we choose to allow cardinality restrictions an additional dropdown box should be added to each property/relationship to select new cardinality.



Based on discussions in Modeling Team at Dagstuhl 2015 and “[Restriction of Classes for the Purpose of a Functional View](https://ddi-alliance.atlassian.net/wiki/download/attachments/38240266/restrictions-in-functional-views.docx?version=1&modificationDate=1466708246350&cacheVersion=1&api=v2)” NOTE: include cardinality for each property and relationship; check off box should be a NEGATIVE check off (select to exclude)

# Captured Information per Functional View

Specific information per class in each Functional View will be captured in Drupal. This includes information on used properties and relationships to other classes. This information will be exported to the XMI format in a comment field per the Functional View. The information needs to be in a text format. Line feeds can be used, if CDATA sections in XMI are used. Possible languages for this purpose are Markdown, JSON, or YAML. The information will be used downstream in the production framework for two purposes, documentation and as basis for generation code for validation tools; this could be Schematron (secondary-level validation) in case of XML Schema and SHACL- or ShEx-based tools in case of RDFS/OWL.

The appropriate structure for the captured information per Functional View is probably a nested list. Per class, the used properties and relationships are listed. This simple format could easily be parsed, even in Markdown. Markdown would have the advantage that no additional language is used inside of the XMI comment fields.