Binding the DDI UML profile to RDF Schema

This document sketches a binding of the UML profile defined for DDI 4 to RDF Schema.

Some modelling constructs are also mapped to OWL constructs, where the expressivity of RDF Schema is insufficient.

Table of Contents

Overview 1

Modules and namespaces 2

Skeleton 2

Mappings to other vocabularies 2

Instance packaging and Named Graphs 2

Classes 3

Associations 3

Compositions 4

Generalization 4

Source cardinality 4

Target cardinality 4

Literal properties (UML attributes) 5

Cardinalities on literal properties 5

# Overview

We divide the UML profile into a number of modelling constructs (classes, associations, cardinalities, etc.). An implementation of this binding would perhaps include a rule for each of these constructs that extracts informationfrom the XMI file, and generates some RDF triples.

The complete RDF output is obtained by merging all the generated RDF triples.

Some additional RDF triples may be merged from additional files, to address metadata for the RDF schema itself, mappings to other RDF vocabularies, and similar things.

# Modules and namespaces

We assume that the DDI Information Model will be organized as a number of packages. To derive an RDF Schema representation, a namespace URI and mnemonic namespace prefix has to be assigned. For example, the DDI Core module could be:

|  |  |
| --- | --- |
| **Title** | DDI Core |
| **Namespace URI** | http://rdf-vocabulary.ddialliance.org/core# |
| **Namespace prefix** | ddicore: |

# Skeleton

The following snippet provides some skeleton triples for describing the vocabulary generated from a module:

@prefix ddicore: <http://rdf-vocabulary.ddialliance.org/core#>.

<http://rdf-vocabulary.ddialliance.org/core> a owl:Ontology;

rdfs:label “DDI Core Vocabulary”;

rdfs:comment “””This is the DDI Core Vocabulary, an RDF

Schema vocabulary that defines foundational concepts

For describing the domain of statistics.”””.

# Mappings to other vocabularies

Additional RDF triples may be merged into the resulting RDF file in the process. For example, an additional RDF file could be written with mappings to other RDF vocabularies.

Such mappings could take the following forms (where one term is from a DDI module, the other term from some third-party vocabulary):

ns1:SomeClass rdfs:subClassOf ns2:OtherClass.

ns1:SomeClass owl:equivalentClass ns2:OtherClass.

ns1:someProperty rdfs:subPropertyOf ns2:otherProperty.

ns1:someProperty owl:equivalentProperty ns2:otherProperty.

# Instance packaging and Named Graphs

Here we will briefly discuss one important difference between this RDF Schema binding and previous versions of DDI-XML.

It is often desirable to package parts of a DDI description, with the intent of re-using it in multiple places, or with the intent of versioning it.

In XML, this is typically done by giving an identifier to a subtree, and referencing that subtree from all relevant places.

Many elements in DDI-XML exist mainly for the purpose of allowing the grouping of information into such re-usable form, or the referencing of such groups. They do not form part of the domain that DDI describes. They are merely artefacts of the requirement to package bits of DDI information for re-use or versioning.

In RDF, a slightly different approach is used. Information expressed in XML is a tree. Information in RDF is a graph, or more often a *set* of multiple graphs, where each graph is named (with a URI, and thus can be referenced). To package parts of an RDF graph, usually the following approach is used:

* The triples to be packaged are put into a separate named graph
* @@@ … TODO

The following sections will walk through a number of UML modelling constructs, showing how they are translated.

# Classes

## Input

A class in a package with a class name

## Output

ns:MyExample a rdfs:Class, owl:Class;

rdfs:label “my example”@en;

rdfs:isDefinedBy <http://rdf-vocabulary.ddialliance.org/core>.

## Comments

* There is a correspondence between packages and namespace prefixes. In the example, everything in the same package as our class will use the ns: prefix. If we reference terms in other namespaces, they’d get other prefixes. As always in RDF, the prefix stands for some full URI. The mapping between packages, prefixes and full URIs needs to be configured somewhere.
* By convention, class identifiers in RDF start with an uppercase letter.
* The label is intended to be human-readable, therefore the camel-case capitalization is removed. This may either be done algorithmically, or the label could be injected from the

# Associations

## Input

An association between two classes with an association name

## Output

ns:relatedTo a rdf:Property, owl:ObjectProperty;

rdfs:label “related to”@en;

rdfs:domain ns:ClassOne;

rdfs:range ns:ClassTwo;

rdfs:isDefinedBy <http://rdf-vocabulary.ddialliance.org/core>.

## Comments

* By convention, property identifiers in RDF start with a lowercase letter.
* See *Classes* above for notes on namespaces and labels

# Compositions

## Input

A composition that relates a parent class to a child class with a composition name

## Output

ns:child a rdf:Property, owl:ObjectProperty;

rdfs:label “child”@en;

rdfs:domain ns:Person;

rdfs:range ns:Person;

rdfs:isDefinedBy <http://rdf-vocabulary.ddialliance.org/core>.

## Comment

* See *Classes* above for notes on namespaces and labels
* The output is identical to the generated RDF for associations. We do not make a distinction between them.

# Generalization

## Input

A superclass related to a subclass

## Output

ns:Subclass rdfs:subClassOf ns:Superclass.

## Comments

* None

# Source cardinality

## Input

A minimum and maximum cardinality for a named association or composition, where the minimum may be 0 or 1, and the maximum 1 or \*

## Output

TODO

## Comments

* TODO

# Target cardinality

## Input

A minimum and maximum cardinality for a named association or composition, where the minimum may be 0 or 1, and the maximum 1 or \*

## Output

TODO

## Comments

* TODO

# Literal properties (UML attributes)

## Input

An attribute associated with a class, consisting of a name and datatype

## Output

ns:numberOfMissingLimbs a rdf:Property, owl:DatatypeProperty;

rdfs:label “number of missing limbs”@en;

rdfs:domain ns:OwningClass;

rdfs:range xsd:integer;

rdfs:isDefinedBy <http://rdf-vocabulary.ddialliance.org/core>.

## Comments

* See *Classes* above regarding namespaces and comments.

## Supported datatypes

* See <http://www.w3.org/TR/rdf11-concepts/#xsd-datatypes> and following sections for the default datatypes in RDF; these include a large selection of XML Schema types, as well as rdf:HTML, rdf:XMLLiteral, and rdf:langString.
* Additional user-defined datatypes are possible.
* TODO: Table with mappings from UML types to RDF-compatible types.

# Cardinalities on literal properties

TODO