RDF Binding Specification and Validation

26 November 2015

*Arofan Gregory*

In-line documentation in the transformation code should reference the rules that are being applied.

|  |  |  |
| --- | --- | --- |
|  | Purpose | Platform Independent Model to Platform Specific Model for RDF |
|  | Description | Flattening the Inheritance out of the Platform Independent Model |
|  | File Location |  |
|  | File Name |  |
|  | Specification | Validation Criteria | Step |
| 1 | Name conventions for classes, properties and relationships must follow the identified naming rules | Properties and relationships: lowerCamelCaseClasses : UpperCamelCase | Validate PIM |
| 2 | The following list of classes, properties and relationships must contain documentation stubs which are not empty | Property: Cardinality;DataType,DescriptionRelationship:TargetObject;Description;Source Cardinality;TargetCardinality;RelationshipType | Validate PIM |
| 3 | PIM will only contain those packages and functional views flagged for publication |  | Validate PIM |
| 4 | Abstract classes which function as extension bases will be deleted | if (PIM abstract.class) = DELETE | TransformPIM to PSM |
| 5 | All properties and relationships are moved down the chain of inheritance to concrete classes | 1. if (PIM abstract class properties) = PSM concrete class properties down down the chain of inheritance
2. if (PIM abstract class relationships) = PSM concrete class properties down down the chain of inheritance
 | TransformPIM to PSM |
| 6 | When non-abstract classes extend non-abstract classes properties and relationships of the extension base are duplicated on the extending class | 1. if (PIM non-abstract class that extends another non-abstract class) = PSM properties are duplicated down the chain of inheritance
2. if (PIM non-abstract class that extends another non-abstract class) = PSM non-extension relationships are duplicated down the chain of inheritance
 | TransformPIM to PSM |

|  |  |  |
| --- | --- | --- |
|  | Purpose | Platform Specific Model transformation to RDFS/OWL specification |
|  | Description | Production of an RDFS/OWL specification  |
|  | File Location |  |
|  | File Name |  |
|  | Specification | Validation Criteria | Step |
| 1 | Normative RDFS/OWL specification for the library for all packages will be produced | Existence of RDFS/OWL file | TransformPSM to RDFS/OWL specification |
| 2 | Convenience (non-normative) RDFS/OWL specification for each functional view will be produced | Existence of RDFS/OWL file | TransformPSM to RDFS/OWL specification |
| 3 | Generated RDFS/OWL specification must be compliant with the RDFS and OWL standards | Use a validation tool | PSM Validation |
| 4 | RDFS/OWL specifications must be valid XML according to the RDF XML schema | Check using XML parser | PSM Validation |
| 5 | All schemas will be declared in the DDI namespace | Check the namespace declaration in each specification | PSM Validation |
| 6 | The library schema will be named DDI\_Library\_OWL.xml | Check the file is correctly named | PSM Validation |
| 7 | Each functional view schema will be named DDI\_[functional\_view\_name]\_OWL.xml | Check the files are correctly named | PSM Validation |
| 8 | The version of the specification[s] will be indicated in their content | Check the schema is correctly versioned | PSM Validation |
| 9 | Each enumeration class should be declared using owl:oneOf |  | PSM Validation |
| 10 | Non-xsd primitive data types will be mapped to xsd primitive data types (UML) | Check simple types in specifications to ensure that no UML primitives are used | Transform PSM to RDFS/OWL specification |
| 11 | The skeleton declarations for each specification will be generated, with a label containing the name of the functional view or the string “library” as appropriate | Check each specification and make sure the label exists with the correct value for each Functional View and the library | Transform PSM to RFS/OWL specification |
| 12 | Each class in the PSM, from the complex types package will become a blank node  | Compare the complex datatypes in the PSm against declarations in the specifications for the correct correspondence | Transform PSM to RDFS/OWL specification |
| 13 | For each class in the PSM, from the non-complex datatypes package declare a class and assign a URL | Compare the non-complex datatypes in the PSM against declarations in the schema for the correct correspondence | Transform PSM to RDFS/OWL specification |
| 14 | For each class in other packages (non-complex datatype; these are classes which inherited from AnnotatedIdentifiable in the PIM) of the PSM create a class, generate a URL, declare properties which are expressed as literals and relationships which are expressed as associations | Compare non-complex datatype classes in the PSM against each schema for the correct correspondence | Transform PSM to RDFS/OWL specification |
| 15 | Declare any classes which are equivalent to those in the PIM from non-DDI namespaces | [QUERY: How are synonyms expressed in the XMI?]e.g. DDI Individual = foaf personMay need to use an extension | Transform PSM to RDFS/OWL specification |
| 16 | All cardinalities in the specifications on declared associations and properties should match those in the PIM | Compare the cardinalities in the generated specifications with those in the PSM. | PSM Validation |
| 17 | Objects inside each Functional View are all declared in the corresponding specifications. | Compare the objects in each PSM Functional View for the correct correspondence in the non-normative corresponding schema | PSM Validation |
| 18 | Objects inside Library Packages are included in the library specification | Compare the objects in the PSM library for the correct correspondence in the library specification | PSM Validation |