XML Binding from the DDI Model

1. Introduction

This document presents the mapping rules from the XML export of the Enterprise Architecture tool to the XML schema deliverables.

1. High-Level XML Constructs

For each package in the model, we will have an XML namespace declared. The XMI of the DDI model must place the name of the UML package (its URI) into the name attribute of the <packagedElement> which represents it. Configuration information will be provided at the time of the transformation. (This could be a parameter passed into an XSLT transformation, for example.)

* Small XML file will provide prefixes and identifiers for each importable namespace, for declaration in the root element of the schema, and for xsd:import statements. The transformation will identify which are needed, and input the appropriate xmlns:[prefix] declarations, and the correct import statements. The content will look like this:  
  <Namespace name="Core" prefix="ddic" location="ddi-core.xsd"/>  
  <Namespace name="Util" prefix="ddiu" location="ddi-util.xsd"/>  
  <Namespace name="Foundational" prefix="ddif" location="ddi-foundation.xsd">  
   <Import>ddic</Import>  
   <Import>ddiu</Import>  
  </Namespace>
* XML schemas for import must live within a predictable physical structure, for mapping to namespace identifiers with xsi:schemaLocation. The namespace identifiers will exist in the model.
* The namespace for which a schema is to be generated must be passed into the transformation as a parameter. The XSI file will contain the description of all model packages, even though some may not be needed.

1. Rules
2. No use of substitution groups – these should not appear in the bindings.
3. There will be a utility package in the model which provides a “universal” document type, having as its children all the root-level elements found in the functional DDI namespaces.
4. The same utility package described in (2) will contain an “Ur-document” class, which has no properties or associations.
5. The document types created in functional namespaces will be extensions of the Ur-document class mentioned in (3). The name of these extending classes will be used to provide the name of the root element for each functional schema.
6. Object-Level Mappings
7. **Packages:** <packagedElement xmi:type=”uml:Package”> corresponds to the declaration of the document type if it contains <packagedElement xmi:type=”uml:class”>. The name attribute is used to get the value of the xmlns attribute from the XML file mentioned above.
8. **Elements:** <packagedElement xmi:type=”uml:class”> corresponds to the generation of several things:
   1. The root element of the schema is generated by determining which class is a generalization of the Ur-document class. This can be detected in the UMI by finding the <packagedElement> which contains a <generalization> element where the generalization attribute holds the ID of the Ur-document class. The name of this class becomes the name of the root element for the document type.
   2. A global element with the same name as the XMI name attribute, and of the complex type declared in the next step. If there is an attribute isAbstract with value true the global element will be defined as abstract.
   3. A complex type using a concatenation of the name attribute and the string “Type”. Within it complexContent tis will contain at first an extension declaration with the name of the Ur-class, derived from <generalization>, via the xmi:id attribute, as base and then a sequence with an element definition for each <ownedAttribute xmi:type=”uml:Property”> following the next step. If no <generalization>is available, the base will be ddic:DDIObjectType
9. **Associations to other DDI Elements:** If a child <ownedAttribute xmi:type=”uml:Property”> has an aggregation or association attribute, then resolve the ID in the association attribute, and take the value of the name attribute from the resolved XMI element. Concatenate the value of the name with the string “Reference”, and place an element declaration of type “Reference” there. The DDI referencing elements will always take their minOccurs and MaxOcciurs values from the value attribute of the <lowerValue> and <upperValue> XML elements (a value of “-1” means the maxOccurs has a value “unbounded”.
10. **Properties:** If a child <ownedAttribute xmi:type=”uml:Property”> has no aggregation and association attribute the type for the generated element has to be determined whether it is part of extended primitive definitions in DDI Core or Utility by the name attribute or one of the primitives already defined in XML schema, using the <type xmi:type="???"/> child of ownedAttribute. Cardinalities are handled the same way as for associations.