XMI Flavor for DDI

A Robust Representation in XMI of the DDI UML Model

Joachim Wackerow | 25 November 2015

**Table of contents**

[Goal](#h.ukd7l83v86py)

[Business Cases](#h.skgzkt6z9uil)

[Solution](#h.eg5bem5d32h)

[Restricted Set of UML Constructs](#h.faozk2r20zbo)

[Syntax Flavor of XMI](#h.6nyyjmxlj0nh)

[List of Common UML Tools](#h.84h2bobz6e4m)

[Version of UML and XMI](#h.j9t5ri5suqdx)

[Canonical XMI](#h.bo1okvpw4nf)

[Identification of XMI Elements](#h.rhn4atwiv5nj)

[Diagrams](#h.o5o2r4onzjr1)

[Test Suite](#h.6wr7e75lo9xk)

# Goal

The goal is to define a XMI flavor for DDI which …

* confirms to the standards UML and its representation XMI,
* is robust, and
* can be imported into major UML tools.

The background is to have representations of the DDI model (PIM and PSM) which can be imported into UML tools. This way the DDI model can be exposed to users who are familiar in using an UML tool. This broadens the accessibility and usage of the DDI model.

# Business Cases

The DDI model can be exposed to additional audiences by providing the model in a XMI flavor which can be imported to common UML tools. This way modelers and developers can use their tool of choice for looking at the DDI model. Additionally, UML tools provided often the function to generate program code on the basis of a model.

The DDI model can be validated by UML tools. This can be done by DDI people and also by any person who is looking at the DDI model by UML tools. This way, there would be an additional validation of the DDI model possible from multiple perspectives.

Validation with UML tools by us and by community.

# Solution

Two levels should be considered for this:

1. Restricted set of UML constructs
2. Syntax flavor of XMI

XMI representation of the DDI model is the basis of:

* Further transformations down in the production framework
* Import into UML tools
* Possible flavors of XMI for specific UML tools.

## Restricted Set of UML Constructs

Only a subset of UML constructs is used. This supports the idea of a simple and robust model.

UML has two levels, the formal definition of classes and their relationships, and the definition of related diagrams. The following subsections focuses only on the formal definitions. The diagrams are discussed in a separate section below.

## Syntax Flavor of XMI

XMI is a well-defined language. The issue is that UML tool vendors tend to use different versions and flavors of XMI. The goal is to support most popular UML tools in the community. This comprehends the free and open-source UML tools of Eclipse, and the commercial tools Enterprise Architect and Altova UModel.

The Eclipse UML tools use the file format uml2 for representing the models. This is a specific flavor of XMI. Some commercial UML tools support this format for import and/or export to be compatible with this set of tools.

The way to achieve this is to use the validation rules of Canonical XMI (http://www.omg.org/spec/XMI/CanonicalXMI/Beta2/). This can be understood as a subset of the syntax of XMI. There exists a tool from NIST which can be used to test an XMI instance against UML/XMI 2.4.1 and the validation rules of Canonical XMI.

## List of Common UML Tools

* NIST XMI Validator (<http://validator.omg.org/se-interop>)
  + Tests according to UML 2.4.1. Tests according to Canonical XMI.
* Eclipse Modelling Tools (https://eclipse.org/modeling/tools.php).  
  Eclipse tools are popular. Therefore they play a similar role nowadays as years ago IBM Rational Rose.
  + MDT UML2 (<https://projects.eclipse.org/projects/modeling.mdt.uml2>)
    - Files with extension „uml2“ are a flavor of XMI.
    - List of UML2-compatible UML Tools (http://wiki.eclipse.org/MDT-UML2-Tool-Compatibility)
  + Modelio (https://www.modelio.org/)
  + Epsilon (http://www.eclipse.org/epsilon/), maybe useful
* Enterprise Architect (popular in the DDI and GSIM community, http://www.sparxsystems.com/products/ea/)
* IBM Rhapsody (successor of Rational Rose, http://www-03.ibm.com/software/products/en/ratirhapfami)
* Altova UModel (http://www.altova.com/umodel.html)
* MagicDraw (http://www.nomagic.com/products/magicdraw.html)

## Version of UML and XMI

UML and the related XMI had a major change with version 2.x. The most popular (in terms of implementations) version is 2.4.1. UML and XMI will be used in this version.

As we are using only a subset of UML it is not expected that future version changes of UML/XMI have a large influence on our flavor of XMI. It will be robust in this regard.

## Canonical XMI

The goal for canonical XMI is to find an easier way to validate results from interchange testing

## Identification of XMI Elements

* Each class and package (including Functional Views) can be identified by three ways in parallel. Attributes:
* name (UML)[1]. The value is a unique name in the scope of the DDI model.
* xmi:id[2]. The value is an identifier used for identification in a XML/XMI file. The value of this identifier should be a combination of the class name and its version. An xmi:id can be referenced by xmi:idref[3] in the same XML/XMI file. The usage of xmi:id for referencing in a XML/XMI file is mandatory according to Canonical XMI[4].
* xmi:uuid. The value is a universally unique identifier. It should be a resolvable URI according to OMG rules. This attribute can be used for identification beyond a single XML/XMI file.

The name is a mandatory attribute. Both attributes, xmi:id and xmi:uuid are mandatory according to the rules of Canonical XMI[4].

Releases have no identification attributes. They are only identified by an URI with a related release date. The URI resolves to the related XMI file of this release.

URIs for Releases, packages, Functional Views, and classes could be organized in this way:

* http://ddialliance.org/Specification/DDI-Lifecycle/Views/Release/<date>
* http://ddialliance.org/Specification/DDI-Lifecycle/Views/Package/<package\_name>/<version>.
* http://ddialliance.org/Specification/DDI-Lifecycle/Views/FunctionalView/<functional view\_name>/<version>.
* http://ddialliance.org/Specification/DDI-Lifecycle/Views/Class/<classname>/<version>.

# Diagrams

UML class diagram definitions have two levels, the formal definition of classes and their relationships, and the description of the diagrams. The discussion above focused on the formal definition.

It would be great to find a way to define the diagrams as well. The drawback is that UML tools have multiple ways to do this. Some exploration is required which way is practical.

The goal should be that Eclipse tools can import diagrams.

A general issue is the layout of a diagram. A workaround is that only a diagram with small complexity is defined like a diagram per class and its direct neighbors. After import of such a diagram, the auto-layout might be useful in an UML tool.

# Test Suite

A test suite of used UML constructs (in XMI) should be developed. This could comprehend a test for each construct and combinations of several constructs. This can be used for testing the whole production framework including the XMI representation(s) of PIM and PSM, and the representations in XML Schema and OWL.

The result of the test can be a table with tests in one dimension and UML tools in another dimension.

[1] P. 93. OMG Unified Modeling Language, Infrastructure. Version 2.4.1.<http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF>.

[2] P. 13-14, OMG MOF 2 XMI Mapping Specification. Version 2.4.1.<http://www.omg.org/spec/XMI/2.4.1/PDF>.

[3] P. 15, OMG MOF 2 XMI Mapping Specification. Version 2.4.1.<http://www.omg.org/spec/XMI/2.4.1/PDF>.

[4]<http://www.omg.org/spec/XMI/CanonicalXMI/Beta2/PDF/>