Report: UML model exchange using XMI

<table>
<thead>
<tr>
<th>TUT Institute of Softw. Syst.</th>
<th>Authors: Juan-Juan Jiang and Tarja Systä</th>
<th>Title: UML model exchange using XMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note: this is a revised report, the original one was written for the purposes of the ART research project. Any ART related information has been removed from the document.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status of the document: Revised version</td>
<td>Modified: 31.10.2002</td>
<td></td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

1. INTRODUCTION ............................................................................................................... 5
2. PARTNERS AND TOOL SUPPORT .................................................................................. 7
3. PRODUCTION RULES OF XMI ....................................................................................... 8
4. UML DIAGRAMS AND XMI .......................................................................................... 10
5. THE EXTENSION MECHANISM OF XMI .................................................................... 10
6. XML TECHNOLOGIES .................................................................................................... 12
7. XMI AND LAYOUTS FOR UML MODELS .................................................................... 13
8. CHALLENGES AND DOWNSIDES OF XMI ................................................................. 16
9. OMG’S LATEST PROPOSAL FOR XMI ...................................................................... 17
11. SUMMARY ..................................................................................................................... 18
12. REFERENCES .................................................................................................................. 19
1. INTRODUCTION

XMI (XML Metadata Interchange) [XMI] intends to provide a standard way for users to exchange any kind of metadata that can be expressed using the MOF (Meta-Object Facility) [MOF] specification by the Object Management Group (OMG) [OMG]. It integrates three industry standards: MOF (OMG), UML (OMG), and XML (W3C). The fact that the UML specification defines the UML meta-model as a MOF meta-model directly implies that XMI serves as an interchange format for UML. In addition, XMI can be used to exchange information among, e.g., data warehouses that support the MOF-based Common Warehouse Metamodel (CWM). Figure 1 shows various possible partners among which XMI can be used as an exchange format. Due to the incorporation of XML into the leading Internet browsers such as Netscape Navigator and Internet Explorer, XMI gets the capability to convey information through Internet. Specifically, XMI intends to help UML CASE-tools to exchange their data models with each other.

![Figure 1. Information exchange with XMI [IBM XMI]](image)

Relations between an OMG Metadata architecture and XMI are depicted in Figure 2. At the lowest level (M0) we have instances of MOF models, namely, the systems to be modeled. The second lowest level (M1) represents a model for the level M0; models of the example systems can be represented as XMI documents. The M2 level, in turn, is a meta-level for M1. From the point of view of UML, the actual UML models belong to the M1 level, while the UML metamodel belongs to the M2 level.
The DTD or XML Schema specifications of the XMI language thus also represent M2 level information. At the highest M3 level we have the MOF model itself.

<table>
<thead>
<tr>
<th>Meta-level</th>
<th>MOF</th>
<th>MOF examples</th>
<th>DTD/XMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>M3</td>
<td>Meta-metamodel</td>
<td>“The MOF model” (i.e., an abstract syntax for defining metamodels)</td>
<td>MOF DTD /</td>
</tr>
<tr>
<td>M2</td>
<td>Metamodel, meta-metadata</td>
<td>UML metamodel, CWMI metamodel</td>
<td>UML DTD /</td>
</tr>
<tr>
<td>M1</td>
<td>Model, metadata</td>
<td>UML models, Warehouse Schemas</td>
<td>/ UML models as XMI documents</td>
</tr>
<tr>
<td>M0</td>
<td>Instances</td>
<td>Modeled systems, warehouse databases</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2. The relationships between the OMG metadata architecture and XMI.

Basing XMI on the XML metalanguage by W3C [W3C] has several advantages. First, XML standard is programming language-neutral and API-neutral. Commonly used choices for an API include SAX and DOM. Second, XML is metamodel neutral and can represent metamodels compliant with OMG’s meta-metamodel, namely MOF. Third, XML allows expressing rules for the structure (i.e., grammar) of a document, which, in turn, is defined by tags encapsulating the content of the document. This allows automatic separation of data and metadata and enables usage of generic tools to validate an XML document against its grammar.

XMI was designed to provide the means to interchange both data (models and model fragments) and metadata. In particular, XMI allows automatic generation of a transfer syntax for a model, based solely on the model’s metamodel. The full validation of a model is not required for metadata interchange. The other interesting features of XMI include the following:

1. XMI can transfer metadata using multiple metamodels after the adoption of the namespace specification by W3C.
2. XMI can transfer the difference of the documents so that the overall document needs only to be transferred once. The new model can be obtained by adding the old model and the difference.
3. An XMI document can refer to an XMI element from another XML document using the Xlink technology.
4. XMI provides extension mechanism so that different tools can modify the model freely without information loss and confusion.
5. XMI supports the ability to transfer incomplete metadata in the model, that is, XMI can transfer subset of one model.
2. PARTNERS AND TOOL SUPPORT

OMG is an open membership, not-for-profit consortium that produces and maintains internationally recognized standards for interoperable enterprise applications. The consortium includes most of the large computer industry companies, as well as hundreds of smaller ones. The Model Driven Architecture (MDA), based on MOF, UML, XMI, and CWM, has become the flagship of OMG. OMG is also known from its CORBA standard.

XMI is an OMG technology, based on the XML standard from the World Wide Web Consortium (W3C) [W3C]. W3C is a consortium that develops common technologies (protocols and tools) to support interoperability. OMG and W3C are collaborative partners leveraging work developed by their peer organizations. For example, XML standard of W3C is used in the OMG’s XMI and the DOM specification of W3C uses the OMG IDL.

XMI is currently supported by most of the UML CASE-tools. For instance, IBM is incorporating XMI in several products. Some tools, such as Rational Rose, support XMI by add-ins usable with the tool. Novosoft UML (NSUML) [NS00], in turn, is a Java library, which offers an implementation of a complete UML metamodel, a reflective API, XMI support, etc. NSUML is used with various CASE-tools, e.g., in ArgoUML [Arg02].
3. PRODUCTION RULES OF XMI

Being an XML-based language, XMI consists of two parts: the DTDs (Document Type Declarations) and XML documents. The XML documents contain information as a set of tags, while the DTD defines what tags can be used in the XMI document and how.

XMI’s XML document production process is defined as a set of production rules. An XML document can be generated by applying these rules to a model or to a model fragment. The inverse of these rules can be applied to an XML document to reconstruct the model or model fragment. In both cases, the rules are implicitly applied in the context of the specific metamodel for the metadata being interchanged. In XMI specification, production rules are defined in Extended Backus-Naur Form (EBNF) notation. Although this grammar provides a definition of conforming XMI documents, it does not specify how a model is transformed into a document. The Object Constraint Language (OCL) is employed to specify that. Figure 2 shows the XML DTD and XML document’s production using production rules.

Using XMI, the XML DTDs for a metamodel are obtained by defining the metamodel in MOF (e.g., corresponding to a UML metamodel) and then applying the XMI generation rules. The generation approach ensures that a given metamodel will always map to the same set of XML DTDs regardless of which vendor implemented the MOF and the XMI stream protocol.

The XMI specification also contains rules for stream production based on the MOF metamodel. These rules can be used to automatically generate XML import and export tools for instances of a metamodel, removing a source of errors and reducing the cost of developing the software needed to support a new metamodel.

![Figure 2. Production Rules of XMI](image)

(1) A set of XML DTD production rules (in EBNF) for generating XML DTDs from a MOF-based meta model.
(2) XML document production rules for encoding/decoding MOF-based models.
Two examples to describe the production rules of XMI DTD and XMI document are presented next. Production rules (in EBNF), shown in Example 1, are used to generate the XML DTDs.

**Example 1:**

\[
\text{<DTD> ::= <1b:FixedContent><1d:XMIAttList>?<2:PackageDTD> +}
\]

The resulting DTD consists of exactly one \text{<FixedContent>} element, zero or one \text{<XMIAttList>} elements, and at least one \text{<PackageDTD>} element. \text{<FixedContent>} means that some XMI DTD elements are fixed. They should be included in the DTD and may be used all XMI documents. These fixed elements provide a default set of data types and the document structure. For more details on the XMI DTD’s fixed elements, the reader is referred to Section 4.4 (Fixed Element DTD) in the XMI specification.

In Example 2, production rules for generating XML documents are presented.

**Example 2:**

\[
\text{<XMI> ::= "<XMI" <2a:Namespaces>
"version=" //XMI version//
("timestamp=" //timestamp//)?
("verified=" //verified//)? ">"}
( <3:Header> )? ( <6:Content> )?
( <4:Differences> )? ( <5:Extensions> )?
"</XMI>"
\]

This example shows how the root of XMI documents should be defined. The root is defined to contain zero or more \text{<Namespace>} elements, \text{version} information, an optional \text{timestamp}, an optional \text{verified} information, and zero or one \text{<Header>}, \text{<Content>}, \text{<Differences>}, \text{<Extensions>} elements.
4. UML DIAGRAMS AND XMI

We next discuss mappings between UML class diagrams and XMI in more detail. For a specific UML model, one can generate an XML document using her own metamodel DTD or Schema based on MOF or using the standard DTD or Schema. The other users of the XML document should use corresponding DTDs/Schemas to validate the XML document and to understand the syntax of the document.

OMG provides a normative DTD: UML DTD, which represents UML1.1 metamodel, and MOF DTD, which represents MOF 1.1 metamodel.

A simple class diagram is shown in Figure 3 and the corresponding XMI file is depicted in Figure 4. The examples are modified from the ones presented in the XMI 1.2. specification [XMI1.2]. The header of the XMI document conforms to the production rules presented in Example 2. The header of the XMI document contains the fixed elements (e.g., XMI.header) and the user defined model elements are included in the content part of the document.

```
<XMI version="1.1" xmlns:UML="org.omg/UML1.3">
  <XMLheader>
    <XMI.model xmi.name="Department" href="Department.xml"/>
    <XMI.metamodel xmi.name="UML" href="UML.xml"/>
  </XMLheader>
  <XMLcontent>
    <UML:Class name="Department" xmi.id="Department"/>
    <UML:Class name="Instructor" xmi.id="Instructor"/>
    <UML:Class name="Professor" xmi.id="Professor" generalization="Instructor"/>
    <UML:Association>
      <UML:Association.connection>
        <UML:AssociationEnd name="instructors" type="Instructor"/>
        <UML:AssociationEnd name="memberOf" type="Department"/>
      </UML:Association.connection>
    </UML:Association>
  </XMLcontent>
</XMI>
```

Figure 3. A UML Model.

Figure 4. XMI document representing the model.
Next we consider an instantiation of the UML model in Figure 3. The example shows the usage of namespaces and referencing to other XMI documents. The models are referred to using the XMI.metamodel tags, and the namespace declarations are used in the XMI element. Figure 5 shows a fraction of such an instantiation.

```xml
<XMI version="1.1" xmlns:Department="tut/Department">
  <XMI.header>
    <XMI.model xmi.name="CompSci" href="CompSci.xml"/>
    <XMI.metamodel xmi.name="Department" href="Department.xml"/>
  </XMI.header>
  <XMI.content>
    <Department:Department name="CompSci">
      <Department:Department.instructors>
        <Department:Professor name="Kai Koskimies" xmi.id="kk"/>
      </Department:Department.instructors>
    </Department:Department>
  </XMI.content>
</XMI>
```

**Figure 5.** Instantiation of the model.

1. `<XMI.header>` contains the XMI elements of documentation, model, metamodel, metametamodel, and import. Here, the example just lists two XMI elements in XMI.header.
2. `<XMI.content>` contains the model what we actually want to exchange.
5. THE EXTENSION MECHANISM OF XMI

XMI uses ‘xmi.extensions’ elements to extend the XMI metamodel. The `xmi.extensions` elements have a content model of ANY, namely providing the freedom of using the elements the designer wants to include. The elements put in the xmi.extensions element must be declared in either an external DTD or internal DTD. An external DTD is a separate file containing the DTD declarations, while an internal DTD is embedded to the XMI document. The top level XMI element may contain zero to more xmi.extensions elements. The corresponding XMI DTD specification is

```xml
<!ELEMENT XMI (XMI.header?, XMI.content?, XMI.difference*,XMI.extensions*) >
```

In addition, XMI.extensions can exist in any place that includes the XMI.extensions in DTD.
For example, in a UML DTD, a declaration

```xml
<!ELEMENT UML:Class(UML:ModelElement.Name|XMI.extension|…
```

allows the classes (UML:Class elements) to be extended. This can be done, for instance, in the following way:

```xml
<UML:Class xmi.id="u1.1"…>
  <XMI.extension …>
```

XMI’s extension mechanism provides a lot of freedom in expanding the specified metamodel. One typical usage scenario of the extension mechanism of XMI in the UML CASE-tools is to include representation information (e.g., layouts). Figure 10 shows an example of using the extension mechanism to represent the self-defined graph information. The elements placed in the xmi.extension field are defined in the internal DTD.
Attributes and references can be employed to specify how the model elements connect with the extension. For example:

```xml
<DiagramElement xmi.id="d1.1" owner="s1.2" xpos="120" ypos="220">…
```

or

```xml
<DiagramElement xmi.id="d1.1" xpos="120" ypos="220"/>
```

```xml
<DiagramElement.Owner xmi.idref="s1.2">…
```

Figure 10. An example of using the extension mechanism of XMI.
6. XML TECHNOLOGIES

The XML family grows very rapidly. Many new technologies have been developed. Some of them will be used with XMI when stabilized. The new technologies include:

1. RDF (Resource Description Framework) is a specification for infrastructure to support web information based on the entity-relationship model. Support for converting XMI to RDF exists.
2. RDF-Schema provides types for XML. XML-Data is a note to the W3C for public comment on providing schemas and types for XML.
3. XSL (Extensible Style Language) specification of W3C can be used to create visual layouts for the underlying XML data and metadata.

GXL is an XML-based language, providing the ability to exchange data using graphs between graph-based tools. The difference between GXL and XMI is that GXL is used to exchange any kinds of graphs, but XMI is used to exchange models including metadata. A graph has no meaning of its own, the meaning lies in the context in which they are used and exchanged. In the ART project, we need a mapping between hierarchical graphs and UML. This mapping is discussed in Section 10.1 in more detail.
7. XMI AND LAYOUTS FOR UML MODELS

XMI does not specify how to represent the XMI elements graphically. Different tools can use their own methods to represent the layout information in XMI. This is typically done using the extension mechanism of XMI. XSL provides another way to implement layouts for the models represented in XMI. Figures 11-13 illustrate an example for representing a class diagram in HTML. Figure 11 shows an XMI file capturing the UML model (a class diagram). Figure 12 shows how the class diagram can be mapped with the representation information defined using XSL. The resulting HTML file is depicted in Figure 13.

Figure 11. An XMI document generated by Rose.
Add the UML namespace

Define the format of the HTML presentation using a table.

Get the.uml:Class.

Call the “attribute” template, and list the attributes of classes in the table.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!- Edited with XML Spy v4.2 -->
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:UML="/org.omg/UML/1.3"
version="1.0">
ownedElement">
<h2 align="center">XMI Elements Representation
In HTML</h2>
<table align="center" border="1">
<tr bgcolor="color:green">
<th align="center">Index of Class</th>
<th align="center">ID of Class</th>
<th align="center">Name of Class</th>
<th align="center">Attributes</th>
<th align="center">Operations</th>
</tr>
<xsl:for-each select="UML:Class">
<tr>
<td align="center"> <xsl:value-of select="position()"/></td>
<td><xsl:value-of select="@xmi.id"/></td>
<td><xsl:value-of select="@name"/></td>
<xsl:call-template name="attribute"/>
<xsl:call-template name="operation"/>
</tr>
</xsl:for-each>
</table>
<xsl:apply-templates />
</xsl:template>
<xsl:template name="attribute">
<td> <xsl:for-each select="./UML:Classifier.feature/UML:Attribute">
<xsl:value-of select="@name"/>
<xsl:text>,</xsl:text>
</xsl:for-each>
</td>
</xsl:template>
<xsl:template name="operation">
<td> <xsl:for-each select="./UML:Classifier.feature/UML:Operation">
<xsl:value-of select="@name"/>
<xsl:text>,</xsl:text>
</xsl:for-each>
</td>
</xsl:template>
</xsl:stylesheet>
```

**Figure 12.** An XSLT style sheet.
Figure 13. An HTML view of the XMI elements shown in Figure 11, combined with the representation information defined with XSL illustrated in Figure 12. The HTML view is visualized with the Internet Explorer 6.
8. CHALLENGES AND DOWNSIDES OF XMI

From the point of view of UML, XMI specifies how to represent model information but does not define how the view information is to be presented. In practice, this means that XMI is not a full-fetched solution for information exchange among UML CASE-tools. For storing the view information, the tool vendors typically extend XMI. Some extensions are naturally not readable by the other tools. However, if the XMI import/export facilities are implemented correctly, such extensions should not restrict the other tools to be able to manage the model parts of the XMI files. The lack of support for storing layout information has been acknowledged by OMG, which is currently working on the extended metamodel for diagramming.

XMI is a very verbose language. Because of this and the heavy usage of tags for representing different elements (the names of which can be quite long), the XMI files tend to grow very large even for a moderate size models. Parsing large XMI files and loading the model to the CASE-tool can be very slow. We constructed a case study, in which the same class diagram consisting of about 100 classes was stored in XMI using different UML CASE-tools. The tools used were Rational Rose, Together/J, FUJABA, and IDEA. The sizes of the XMI files for the same model varied from ~500 kB (IDEA, which stores the model information only) to over 8 MB (Rose).

XMI is not good for accessing information instantly. A possible solution for speeding it up would be to parse the XML file and store it in a repository first, and then use the repository API to access the information instantly.
9. OMG’S LATEST PROPOSAL FOR XMI

The usage of namespaces in the specification was added to XMI after XML Namespace was added as a W3C recommendation. This facility can exchange several models together and reduce the need for long XML element names [XMI1.1 RTF]. Another enhancement is to allow the generation of XMI documents from both XML Schemas and XML DTDs. Compared to XMI 1.1, the latest XMI specification, namely XMI 1.2, contains two interesting enhancements: (1) XMI incorporate technology neutral data types introduced by MOF 1.4, and (2) XMI 1.2 removes the obsolete OCL and pseudo-code. [XMI1.2 RTF]
10. SUMMARY

XMI is used to exchange data and metadata between different tools in distributed heterogeneous environments. In an ideal case, development teams are split into several groups that work in multiple locations using different tools from multiple vendors, and access the metadata with interoperability and consistency. The developer can access the model through the Internet regardless of platforms, tools, and programming languages. The practical problems of XMI do not currently support this ideal picture.

In the ART project, we will use the standard XMI without extending it. Our goal is to gather practical experience (e.g., performance and scaling) on using XMI for UML model interchange. The XMI import and export components open the xUMLi platform to any outside XMI-compliant UML case tools or reverse engineering tools, requiring no further implementations.
11. REFERENCES


