Markus Nüttgens, Jan Mendling (eds.)

XML4BPM 2004

XML Interchange Formats for Business Process Management

1st Workshop of German Informatics Society e.V. (GI)
in conjunction with the 7th GI Conference "Modellierung 2004"

March 25, 2004 in Marburg (Germany)

Proceedings

Organizer

This workshop is organized by the GI Working Group "Business Process Management with Event-Driven Process Chains (EPC)" within the GI Special Interest Group WI-MobIS (FB-WI) in conjunction with the 7th GI Conference "Modellierung 2004".

Dr. Markus Nüttgens
Email: markus@nuettgens.de

Dipl.-Wirt.-Inf. Dipl.-Kfm. Jan Mendling
Email: jan.mendling@wu-wien.ac.at

XML4BPM 2004 / XML Interchange Formats for Business Process Management. Eds.: Markus Nüttgens, Jan Mendling – Marburg 2004.

OMG's XML Metadata Interchange Format XMI

Mario Jeckle
University of Applied Sciences Furtwangen
mario@jeckle.de

Abstract: Interest in interchange of models and and data conforming to models attracts increasing interest in these days. Especially since incremental and iterative development processes enter the mainstream market chained modeling tools are common in many current projects. Additionally, users are challenged by new developments like generative approaches or OMG's Model Driven Architecture.
In order to cope with the existing and always increasing heterogeneity among the tools deployed vendor neutral interchange formats have turned into a desideratum.
There are some techniques to transfer data and/or models available on the market. But none of them combines a fixed standard's-based approach with the inherent flexibility to support arbitrary models in a uniform way.
This paper provides a brief overview of all concepts found in OMG's XML METADATA INTERCHANGE Format XMI which strives to support exchange of existing metamodel data as well as the creation of XMI compliant schemata for new metamodels.

1 Methodological and Conceptual Background

The basic idea of the XML METADATA INTERCHANGE format (abbreviated XMI thereafter) is it to provide a sound methodological framework for serializing instances of arbitrary models. This framework should be rich enough to support models organized on various meta layers and thus be capable of encoding the structure of whole languages represented by their respective metamodel, arbitrary models represented as instances of the metamodel they adhere to as well as values serving as incarnations of a concrete model.
This section provides an introduction into the background of the XMI format by reviewing the basic concepts of a four-layer metamodel architecture which forms the grounding of the dominant META OBJECT FACILITY established as a unifying metamodel of various languages standardized by the OBJECT MANAGEMENT GROUP (OMG), such as the UNIFIED MODELING LANGUAGE (UML). Additionally, the seminal idea of XMI's schema production principles is introduced. This deterministic formalism allows the preservation of the same methodological principles throughout the deployment of XMI on the various meta levels.
The reminder of this paper is structured as follows. First the methodological framework is introduced. It is represented by OMG's standardized four layer metamodel architecture which relates modeling using the UNIFIED MODELING LANGUAGE, metamodeling using the META OBJECT FACILITIES, and stream-based interchange based on XML using the XML METADATA INTERCHANGE format.

Section two introduces XMI's basic principles. This includes in detail the transformation algorithm which defines how to generate XML schema representations of arbitrary (meta) models. Additionally, related XML standards such as XML Namespaces, XML Links, and XML Schema are sketched.
Section three provides some deployment scenarios of the XMI approach. This includes a discussion of XMI's role interchanging complete and incomplete models and metadata.
Within section four a brief survey of available support offered by commercial CASE tools is given. Also the idea of deploying XMI as native storage format is discussed.
The final section summarizes some experiences in deploying XMI in projects in research and industry. In detail, results of applying XMI's schema production rules for generating custom schemata are discussed. Additionally, an outlook to the UML 2.0 compliant future version of XMI is provided.

1.1 Metamodeling and its Architectures

One of the most prevalent challenges in the design process of a model interchange format is the support of model instances of various abstraction levels. Models may occur in different flavors of abstraction ranging from meta metamodels describing aspects of a whole modeling language like its static structure or the process deployed to create models to instantiable *user level* models which abstract concrete data instances. As a result, a real interchange format has to support all these various abstraction levels.

Intuitively, there is no limit of abstraction levels since *abstraction* is a term commonly referred to in order to describe a surjective one-to-one relation from a real entity to its abstracted (since this is a cognitive process executed by humans it highly depends on the modelers's perception of reality) counterpart. As long as a modeler is able to abstract an element further s/he formally adds an additional meta layer to the model hierarchy.

For reasons of uniformity the OMG has decided to limit the layers of consecutively stacked models strictly to four. The resulting architecture is shown in figure 1.

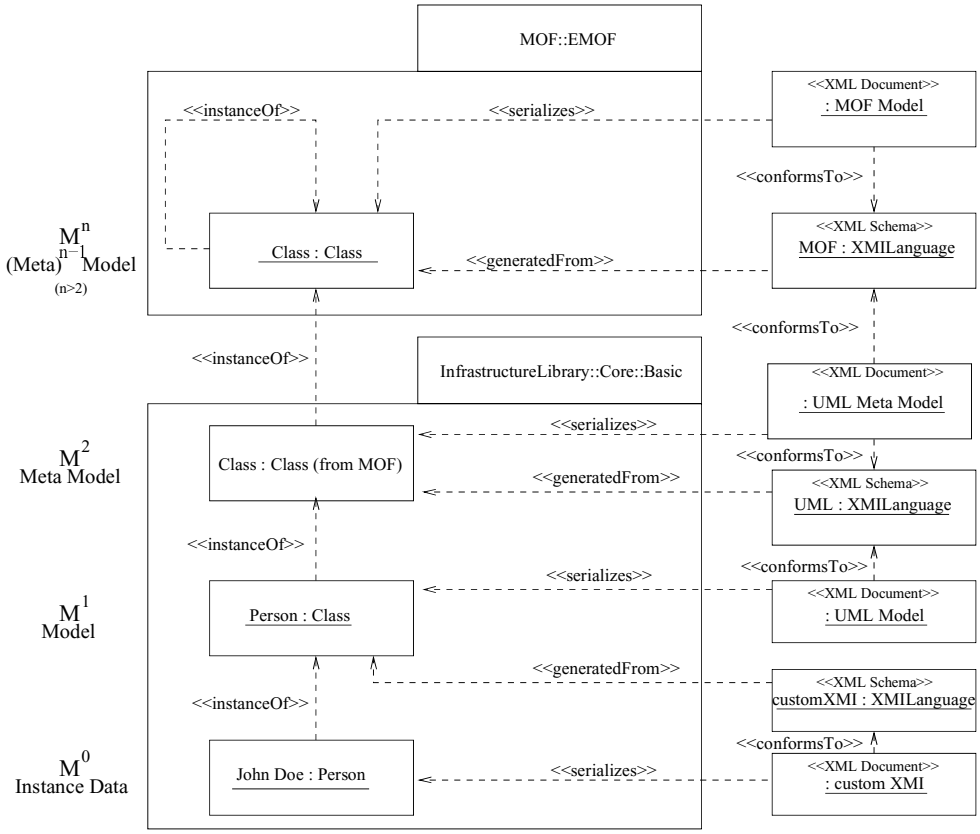

Figure 1: OMG's Four Layer Metamodel Architecture

Restricting the meta layers enables the fixed ascription of an abstraction semantics (i.e. a role adopted by a model layer) to every model within the architecture.
Thus the bottom layer containing concrete data holding objects and association instances representing their interconnections (in UML terms: *links*) is termed the *instance* layer.
Every element of this layer can be directly (more formally: surjectively but not necessarily injectively) mapped onto an element of the *model* layer instantiating the concrete object. This interrelation of the two layers represents the classical *type-instance-dichotomy* [UM01, p. 3-14]. For reasons of brevity, the name of the model layer is shortened to M^1, where the exponent informally denotes the number of occurrences of the letter *m* (which stands for *model*) within the layer's name. Corollary this also clarifies why the instance layer which lacks any occurrences of *m*s is abbreviated as M^0.
The abstraction at the model level, which is normally referred to using the term *class*, both serves as a concrete instance as well as an abstraction of another instance. The relationship between the model level and the *metamodel* level abstraction re-factors the type-instance-dichotomy found at the subjacent architecture layer.
The abstraction is often termed *type*. The connection between a type and its concrete instantiation is represented by an arrow decorated directed edge labeled `instanceOf`, departing from the concrete instance.
This continuous abstraction process is illustrated in figure 1 by introducing `John Doe` as a concrete object typed by the class `Person` which in turn is an instance of the type

entitled `Class` which is predefined by UML's metamodel.
So far, all types and instances are part of a single language framework which is defined but not limited to the UNIFIED MODELING LANGUAGE in figure 1.
By further abstracting the concept *class* as defined by UML, we enter the meta metamodel layer abbreviated to M^3. Since this layer serves as a uniform abstraction of UML and other modeling languages and metamodels (e.g. [IOU+01]) it is addressed by a separate specification titled META OBJECT FACILITY (MOF).

1.2 OMG's Meta Object Facility

The META OBJECT FACILITY [Ob02a] serves as a top level abstraction of various metamodels defined by the OMG. Essentially, MOF is built by abstracting all concepts expressed by different OMG metamodels, e.g. UML, CWM. Furthermore, MOF serves as a termination of an otherwise infinite model stack. This is done by defining all concepts set out by MOF on the concepts defined by the model itself. As a result MOF bootstraps itself and hence terminates the model hierarchy. Formally, MOF is typed by itself as well as it is a valid abstraction of itself.

As shown in figure 1, the class `class` originated by MOF is part of package `MOF::EMOF` which contains MOF's concepts essential for defining metamodels. By definition the prefix *meta* which should precede all concepts defined by the meta metamodel is omitted for brevity [Ob02a, p. 2-5].

Conceptually MOF is organized as a separate model layer which is comprised of the description of the structure and semantics of meta-metadata. Technically, MOF and the metamodel layer use similar concepts. In detail, the essence of the definitions of the concept *class* available in MOF [Ob02a, p. 4] as well as in UML [UM03, p. 6] does not differ. Although the concepts do not match 1:1, the cores of both models are isomorphic and largely share the same concepts w.r.t. the underlying semantics and the naming. This is especially true for MOFs strict subset termed *essential MOF* (EMOF) from which the concept `class` shown in figure 1 originated.

The apparently large conceptual overlap between the metamodel and its metamodel (i.e. the meta metamodel) suggests to consider the organization of the meta metamodel as a subset of the concepts found in the metamodel provided that the metamodel offers enough expressive power to act as a meta metamodel also. On the one hand this would ease the understanding of the metamodel architecture since the concepts found present in the M^3 or higher are identical to concepts found in the modeling language, i.e. the M^2 layer. On the other hand, the selection of a metamodel is crucial. In detail the model has to provide concepts which are mature and semantically rich enough to be propagated to the meta meta-layer without losing expressive power. Otherwise this would brake the abstraction relationship to other metamodels.
Concerning the future development of the four layer metamodel architecture the task force dealing with MOF and UML at the OMG has decided to consolidate and merge the abstract kernel of the UNIFIED MODELING LANGUAGE titled *UML Infrastructure* and the

next generation of the MOF-based meta metamodel by factoring out constructs abstract enough to be re-deployed as meta metamodel from the UML and propagate them to the MOF.
As a consequence of this, the four layer metamodel architecture will be effectively transformed into a hybrid architecture providing four layers of abstraction for all metamodels except for the the seminal UML's one which will reside in a three layer metamodel architecture.

1.3 XML Metadata Interchange

XMI is designed to serve as an interchange format for arbitrary (meta) models. Furthermore, XMI should future proof itself by being capable of also supporting metamodels yet unknown at the time of XMI's definition. Closely related to this is the requirement to offer continuing support by providing compatible encodings for the metamodels changing over time which were initially addressed by the specification. Furthermore, an introduction of interchange formats specific to one version of an existing metamodel would establish a strong dependence of XMI from all metamodel specifications. Changes to these metamodels would in turn imply changes to the interchange format in most cases. Thus XMI would become partly obsolete once a metamodel (specification) is changed.

In order to achieve the highest possible degree of independence the XMI specification does not prescribe concrete schemata supported metamodels. Instead of doing so, XMI sets out a process for automatically generating schemata for arbitrary (meta) models.

OMG's XMI standard defines *schema production rules* which formulate an algorithm to transfer MOF-based (meta) models into XMI-compliant XML grammars which are formulated as instances of the XML SCHEMA vocabulary. The schema production rules can be applied to various models or metamodels as long as they adhere to the structuring principles set out by MOF. As a consequence, the schema production algorithm may be deployed on every model level ranging from pure data models defined as UML class diagrams to meta metamodels. Note that in the context of *XMI* the terms *model* and *metamodel* become interchangeable, likewise do *data* and *metadata*. The various possibilities to create schemata originating from a model are shown in figure 1 by classes connecting to the models by dependency relationships stereotyped with `generatedFrom`.

XMI is MOF-based since it would be virtually impossible to define meaningful production rules that would work on arbitrary models not sharing common characteristics. The approach taken by emphasizing the role of MOF was chosen to provide the commonality among models, allowing the metamodel information to be represented uniformly.

Besides the challenge to define the actual rules driving the process of schema creation which are described by sections 2.3 and 2.2 the schema serialization syntax is crucial to the approach. Since the syntax influences the patterns of interaction with the interchange format and thus promotes or even balks the adoption it could in turn leverage its adoption. The term interaction pattern should refer to the complexity introduced by adding interfaces to existing tools which are capable of importing and exporting the new interchange format.

For reasons of interoperability and market proliferation XMI's group of authors has decided to base the interchange format on the well-known and widely adopted W3C-standardized EXTENSIBLE MARKUP LANGUAGE (XML).
The usage of XML through the XMI specification is two-fold. In essence, the deployment of XML creates the two main constituting parts of XMI.

> **XML DTD/Schema Production Rules**[1] are describing the rules for producing XML Document Type Definitions (DTD) or XML Schema definitions for XMI encoded metadata.[2]
>
> **XML Document Production Rules** are describing the rules for encoding metadata into an XML compatible format.[3]

According to the schemata generated by the schema production principles XMI/XML allows the storage of instances compliant to the metamodel serving as input of schema production. These instances are encoded as XML instance files conforming to the respective XML SCHEMA instance which was derived from the metamodel. By doing so, the XML instances are valid w.r.t. the XML schema which was automatically derived from the metamodel. Thus in turn, provided that the schema production principles governing the generation preserve the structural expressiveness, the XML instances may only store data objects which conform to the metamodel.

1.4 Usage of XMI for Interchange of Business Process Management Data

XMI can be used in two distinct ways to support the interchange of business process management data. First, the XMI format capable of storing arbitrary UML models may be used to encode process management related data which is modeled in a UML compliant manner.
Second, XMI's schema production principles may be applied to arbitrary (MOF compliant) metamodels. This allows the creation of XMI compliant languages for business process management models which are not UML compliant inherently.

Furthermore, provided that a unified business process management metamodel is accepted by an organization XMI may also be deployed to generate an XML vocabulary capable of expressing instances conforming to the business process metamodel. This can be valuable in environments (e.g., a data warehouse storing process descriptions) where process descriptions adhering to diverse metamodels have to be managed

Thus XMI may be used to store business process management data as well as metadata related to these instance data.

[1] All releases of the first version of the specification [UIC 98, UFI 99, Ob02b] use DTDs. This will be changed by the upcoming XMI 2.0 version [Ob03b] which will drop the DTD support in favor of using XML Schema [BLM 01, BM01] for describing structure and content of XMI compliant schemata.

[2] c.f. [Ob02b, p. 1-1]

[3] c.f. [Ob02b, p. 1-1]

2 Deriving XMI Languages from Models

After the introduction of the technical background and the sketch of the basic technical decision to base XMI on XML, this section will provide a more detailed view of the usage of XML techniques by XMI.
Also, an introduction to the basic primitives generated when encoding MOF-based metadata by XMI is given. Finally, the inherent structuring elements present in all schemata which are independent from the source metamodel and generated by deploying XMI's schema production rules are introduced.

2.1 Characteristics of the Language Family

By the use of XML to encode the (meta) models as XML Schema files resp. the (meta) model instances as XML instance files, some elements of the XML technology can be exploited. The reasons for this are two-fold. First, the interoperability of the resulting XML files can be increased by re-using existing approaches and techniques. Second, XML technology provides standardized solutions to classical challenges arising in vocabulary design, such as the unique identification of the vocabulary elements.

In detail, XMI exploits the following XML techniques:

XML Namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references.[4]
Thus, XML namespaces provide the basis for XML-based content syndication and re-use of existing vocabulary and data within new contexts without bearing the danger of conflicting terminology. This also allows usage of XMI fragments within other XML-based languages, but this is beyond the scope of this paper.

XML Links allow elements to be inserted into XML documents in order to create and describe links between resources. They use XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated links.[5]
Thus, XML Links provide the basis for avoiding redundancy within the XML serialization of model data by replacing possible error prone redundant occurencies by references to one single occurrence.

XML Schema is a XML schema language[6] which describe the structure and constraints to the contents of XML documents, including those which exploit the XML namespace facility. The schema language, which is itself represented in XML and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML document type definitions. It also supports definition of dataty-

[4] c.f. [BHL99]
[5] c.f. [DMO00]
[6] Unfortunately, W3C has decided to use the generic name to denote one specific schema language.

pes. These types are to be used in XML Schema as well as other XML specifications. The datatype language, which is itself represented in XML, also provides a superset of the capabilities found in XML document type definitions for specifying data types on elements and attributes.[7]

2.2 XMI Production Rules

Schema production principles are the main cornerstone in striving for independence from the currently addressed (meta) models MOF and UML and in widening the field application to yet unconsidered (and even yet unknown) models. The main ideas for generating XML schemata from static metamodels expressed using MOF's language primitives depicted as UML class diagrams are sketched by this subsection. Afterwards a subset of three basic production rules is discussed. The description of all production rules set out by XMI can be found in the normative documents.

In essence, XMI defines rules to produce XML schema instances from the following basic primitives. This basic primitives are applicable to UML-compliant models as well as to MOF-compliant ones without change.
Note: since XMI version 2.0 will re-factor the production principles this description is based on the available draft [Ob03b] of this upcoming release of the standard. This collection requires some familiarity with the XML techniques and specification introduced in section 2.1.

Class Every class within the metamodel is decomposed into three parts: attributes[8], associations[9], and compositions[10]. A class is represented by an XML element, with an enclosed additional XML element for each attribute, association, and composition. The XML element for the class includes the inherited attributes, associations, and compositions directly since XMI does not utilize XSD's inheritance mechanism.[11]
For example, the representation of the class `person` depicted by figure 1 which does not define any attributes, associations, or containment relationships would be serialized to (simplified form):

```
<xsd:element name="person" type="person"/>
<xsd:complexType name="person">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
   </xsd:choice>
</xsd:complexType>
```

[7] adapted from [BLM 01, BM01]
[8] i.e., slots for containing values within objects instantiating classes
[9] i.e., associating classes
[10] i.e., a special kind of association narrowing the semantics to a physical containment relationship between a whole and its constituting parts
[11] This was chosen since XSD does solely support single inheritance, but MOF and even UML provide support for multiple inheritance.

Attribute
If attributes present in the metamodel are typed by primitive types or enumerations, then by default XML attributes are declared for them as well as XML elements. The reasons for this encoding choice are including: the values to be exchanged may be very large values and thus unsuitable for XML attributes, and may have poor control of whitespace processing with options which apply only on element content.
Assume the class `person` depicted by figure 1 defines an attribute `name` typed by the primitive type `string`. The simplified schema fragment produced would be as follows:

```
<xsd:element name="person" type="person"/>
<xsd:complexType name="person">
   <xsd:choice>
       <xsd:element name="name" type="xsd:string"
                    nillable="true"/>
    </xsd:choice>
</xsd:complexType>
```

For multi-valued attributes, no XML attributes are declared; each value is encoded as an XML element.

Assume `gender` is an attribute with enumerated values, the type used for the declaration of the XML element and the XML attribute corresponding to the attribute of the class within the metamodel is as follows:

```
<xsd:simpleType name="gender">
   <xsd:restriction base="xsd:string">
       <xsd:enumeration value="male"/>
       <xsd:enumeration value="female"/>
    </xsd:restriction>
</xsd:simpleType>
```

Complex typed attributes, i.e. attributes typed with a class present in the model are handled like associations.

Association Each association is represented in an XML element and/or an XML attribute. The element is declared in the content of the `complexType` for the class that owns the reference. This declaration enables any object to be serialized. The attribute declaration which is also included in the `complexType` declaration serves for referencing the element.
Assume the class `person` depicted by figure 1 defines an association to one or more objects of class `company`. Vice versa, the association is also marked with the multiplicity of `1:*`, i.e. a `company` object may also be linked with one or more objects of the `person`. The simplified schema schema resulting would look like:

```
<xsd:element name="person" type="person"/>
<xsd:complexType name="person">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="company"/>
   </xsd:choice>
</xsd:complexType>
```

2.3 Structure of XMI Languages

Besides all variations introduced by schemata derived from arbitrary metamodels XMI's syntactical infrastructure remains the same for all generated schemata which form the *family of XMI languages*.
This section provides a sketch of the infrastructure elements offered by XMI present in all generated models. These elements of the language are the prerequisite of interoperability since they guarantee the structural uniformity of all XMI vocabularies in addition to the uniform production principles.

XMI Header
Every XMI language, regardless of the metamodel it is derived from, defines the fixed start element `XMI`. This XML element serves as root element of all XML elements generated model specific according to the production rules introduced by section 2.2. In order to define a fixed XML schema instance which offers the flexibility to be deployed with schemata generated for arbitrary metamodels, the content model of the element `XMI` is set to `any` with no namespace restriction. This allows, by default, elements of any namespace to be placed as child element of the root element. Additionally, the attribute `processContents` set to `strict` requires the schema processor to validate the embedded content against its respective schema which guarantees the validity of the whole XMI document.

Furthermore, the heading element defines some attributes capable of storing descriptive meta information about the XMI stream serialized. In detail, the concrete metamodel and the version of the XMI specification the XML file is conforming to can be expressed.

```
<xsd:complexType name="XMI">
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:any processContents="strict"/>
    </xsd:choice>
   ...
    <xsd:attribute name="version" type="xsd:string"
                   use="required" fixed="2.0"
                form="qualified"/>
</xsd:complexType>
```

Extension Specification
Opposed to the strictly validated content of the XMI element which constitutes the serialization of the metamodel data the element `extension` allows arbitrary user driven extension be placed within the XMI stream which are serialized purely for tool specific processing.

One use case for introducing these inherently proprietary structures is the intention to store model related data which cannot be covered by instantiating the model's metamodel. Graphical model information like placement and coloring are a well-known example for this since this data cannot be expressed by UML's metamodel prior to version 2.0.

Documentation
Of purely descriptive nature and without any relevance to the encoded metamodel instance are values place inside `documentation` elements within the serialized XMI file. For reasons of identifying the tool which initially created the serialization file the element `documentation` predefines slots for expressing interpreted textual data about the exporter, the concrete version of the tool deployed, and also notices to the human reader of the file.

Linking Attributes
As introduced in section 2.2 every element of a XMI file which represents a metamodel instance can be serialized either *defining* or *referencing* by usage of the predefined linking mechanisms based on XML LINKS.

Every XML element produced from a class within the metamodel is equipped with linking facilities. This is done without introducing redundancy on the meta layer simultaneously which would be the case if multiple definitions of identical linking attributes are generated. Therefore, a set of XML attributes (`LinkAttribs`) is predefined which is referenced from all XML element declarations.

By doing so every XML element produced from a metamodel class offers linking facilities in a uniform way.

```
<xsd:attributeGroup name="LinkAttribs">
   <xsd:attribute name="href" type="xsd:string"
                  use="optional"/>
    <xsd:attribute name="idref" type="xsd:IDREF"
                  use="optional" form="qualified"/>
</xsd:attributeGroup>
```

Identifying attributes
As a prerequisite of general linking XMI defines anchor attributes that links can refer to. These attributes are organized within the re-usage group titled `IdentityAttribs`.

Like the linking attributes discussed before this set of attributes is also referenced by all XML elements produced from classes defined by the metamodel the XMI compliant XML schema is derived from.

```
<xsd:attributeGroup name="IdentityAttribs">
   <xsd:attribute name="label" type="xsd:string"
                  use="optional" form="qualified"/>
    <xsd:attribute name="uuid" type="xsd:string"
                  use="optional" form="qualified"/>
</xsd:attributeGroup>
```

Incomplete Metadata
Besides the transfer of complete XML serializations of metamodel instances (i.e., concrete models), XMI also supports the encoding of arbitrary parts of a model. Serializations of these type are termed transfer of *incomplete meta data* by the specification since the resulting stream does necessarily represent a valid instance of the metamodel.

Serialization of incomplete meta data is therefore not intended to be used by modeling tools for storing models but can be deployed for integrating heterogeneous tools. Since XMI defines three kinds of incomplete meta data this variant of XMI is suitable for coupling tools without relying on a published API.

> **Add** named elements embrace XMI content which is merged with existing content into a new model. The model fragment embraced by `add` has to conform to the schema a.k.a. metamodel of the including content.
>
> **Replace** named elements embrace XMI content which replaces existing content. This is done by replacing model elements by those bearing the same identifying attributes as the ones conveyed in the model fragment.
>
> **Delete** named elements embrace XMI content which deletes existing content from the model. This is done by deleting model elements bearing the same identifying attributes as the ones conveyed in the model fragment.

This three elements could be sent over a network connection as fully legal XML document (e.g., using Web services). CASE tools implementing this will in turn offer an interoperable API for coupling.

3 Deployment Scenarios

This section summarizes usage scenarios which are already common in XMI deployment or which may be promising for the future.

3.1 (Meta) Model Interchange

The native usage of XMI is the Interchange of XMI serialized models. As figure 1 shows, instances of every layer of OMG's four layer metamodel architecture may be expressed

by XMI compliant XML streams. Especially, concerning interchange scenarios involving heterogeneous tool-sets XMI has established itself as the only existing vendor neutral standard providing interoperability.

Furthermore, XMI encoded models are used for tool external storage within revision control systems. As a byproduct of the chosen XML serialization, tools operating on clear text input such as the well-known *diff* may be used to verify consistency of models produced by drawing tools.

3.2 Transfer of Incomplete Model Data

The same is true regarding dynamic tool integration scenarios like those present in incremental and iterative processes as commonly found in agile development. These patterns of development inherently require a tighter level of integration as could be accomplished by integration mechanisms transferring whole models. Since XMI's fragment interchange allows to virtually cut the hot spot of recently changed model elements it is best-suited for this kind of processes.

Moreover, the approach to transfer model fragments instead of complete models conserves bandwidth and avoids whole models be locked by a tool during synchronization.

3.3 Model Driven Architecture

Perhaps the most important application of model interchange in general is formed by the so called MODEL DRIVEN ARCHITECTURE (MDA) which was introduced by OMG in 2001. MDA serves merely as a philosophy and an *umbrella standard* and does not actually introduce new technical ideas by itself. In essence the MDA is made up of a suite of standards including (among others) UML, MOF, and XMI. [Si01, Ob01]

MDA enhances the metamodel architecture depicted by figure 1 by splitting a model residing on a single model layer above the M^0 level into two distinct kinds of models. These models are termed either *platform-specific models* (PSM) or *platform-independent models* (PIM). In terms of MDA PIM denotes a view of a system from the platform independent viewpoint. A PIM exhibits a specified level of platform independence so as to be suitable for use with a number of different platforms of similar type.[12] In contrast, a PSM denotes a view of a system from the platform specific viewpoint. The PSM combines the specifications in a PIM with the details that specify how that system uses a particular platform.[13] In order to achieve this combination within the MDA models of the two kinds mentioned are interconnected by relationships stating that the PSM is generated from a PIM ideally without manual interaction.

Putting MDA into practice requires either modeling tools capable of describing and hand-

[12] c.f. [Ob03a, p. 2-6]
[13] c.f. [Ob03a, p. 2-6]

ling both models and model transformations or tool chains where specific tools interact. Especially the latter requires sufficient interfaces eligible to transfer whole models without loss of information. XMI naturally fits into this scenario since PIM as well as PSM are in most cases expressed using the UNIFIED MODELING LANGUAGE or a customized version.[14] Also, due to the applicability of XMI's schema production rules to arbitrary MOF-based[15] metamodels XMI can serve as a connecting element between any PIM and PSM incarnations.

4 Tool Support

This section briefly summarizes the current state of tools supporting XMI. It should be noted that all tools solely support usage of fixed formats for storing model information. In detail, XMI solely capable of storing UML models. There is currently no tool (neither commercial nor in the academic field) available which fully implements XMI's schema generation production rules.

4.1 Import/Export Interfaces

Most of the UML-supporting CASE and drawing tools currently dominating the market[16] support XMI for exporting UML models and also for importing them back into the tool.

Unfortunately, no overview of the deployment of XMI-based vocabularies produced from MOF-compliant but non UML compliant metamodels can be given since vendors do not provide these information.

In detail both RATIONAL ROSE/XDE[17] (recently acquired by IBM) and TOGETHER[18] (recently acquired by BORLAND) support saving and reading back UML models as XMI-compliant streams. Since the standardization of UML 2.0, which will add capabilities for interchanging the visual representation of a model also, is not finalized yet both tools do not allow to encode diagrams in an interoperable manner. For instance RATIONAL ROSE deploys XMI's extension mechanism sketched in section 2.2 to encode data expressing the graphical placement of model elements.

The general situation shows that approximately 25 percent of the UML supporting tools available on the market currently offer XMI reading and writing capabilities. Up-to-date details of various UML tools and their specific level of support offered for XMI are available at `http://www.jeckle.de/umltools.html`.

[14] The process of customization by extending the predefined modeling primitives is termed *profiling* within the context of UML. Consequentially, the resulting customized UML is termed to be a *UML profile*.

[15] Note: This forms a prerequisite of the applicability of XMI's schema production rules. Hence, only MOF-compliant metamodels can be used a source of the generation process.

[16] A analysis of current market shared can be found in: [Ri02].

[17] `http://www-306.ibm.com/software/awdtools/developer/rosexde/`

[18] `http://www.borland.com/together/index.html`

Slow mass market adoption of XMI may be due to the limitations of XMI documented in section 5.1 and 5.3.

4.2 Native Format Support

Besides the support of XMI by additional filters added to an existing tool XMI may also serve as native internal format. Specifically XML schemata produced from M^2 level metamodels could be deployed in this way. Given a sufficient in-memory representation of XML structures, XMI can also be used as a repository storing all model information at runtime.

Furthermore, data structures for accessing the stored data may be generated directly from the language's metamodel. By doing so a virtual XMI API is established which allows direct operation on the stored model data.

The only commercially available tool currently adhering to this approach is POSEIDON[19] which also offers an early access implementation of the upcoming diagram interchange facilities which will be introduced by UML 2.0's metamodel.

4.3 Meta CASE Tools

In addition to the XMI support by a given tool offering fixed functionalities XMI-based formats can be deployed for encoding custom models which are not directly supported by available tools. A promising application is formed by *meta CASE tools* which are in essence drawing tools enriched by a rule enforcing component.

Instead of hand craft tool support and developing proprietary storage formats for custom languages (i.e. metamodel) meta CASE tools like METAEDIT+[20] could be used. Tools of this type often offer direct support for reading and writing XML compliant data representing metamodel instances (i.e., concrete models) which are stored in the tool's on-line repository.

5 Experiences and Future Work

Finally, we want to provide some documentation of existing experiences and practical problems arising in today's application of XMI. Additionally, an outlook of further developments of the XMI standard is given.

[19] `http://www.gentleware.com`
[20] `http://www.metacase.com`

5.1 XMI's Inherent Heterogeneity Problem

The flexible nature of XMI's schema production rules outlined in section 2.2 inherently introduces an heterogeneity problem concerning the serialization of data instances conforming to a given metamodel. In detail, the schema production rule chosen for the transformation of associations connecting metamodel classes preserves the general net-compliant structure in the inherently strictly hierarchical (i.e. tree-oriented) structures of XML schema.

On the one hand, this offers an XMI export filter the possibility to traverse the metamodel instances in an arbitrary manner provided that it is compliant to the metamodel.
On the other hand, even if the resulting stream of serialized instances is compliant to the XML schema describing the metamodel instances, it may differ vastly from other serializations of the same model.

It is clearly a point of hindrance in market adoption that the OMG standard does not define or even recommend a default encoding. The notion of *default encoding* should refer to predefined paths for fully traversing a given networked model. Due to the lack of these predefined paths any traversal are standard compliant but lead to fairly distinct lexical results. Such an encoding style should include recommended paths which should be followed to access model data.
Also complete abandonment of serialized hierarchical structures would be a valid option for the benefit of interoperability. A flattened structure could serve as equivalent alternative. This would not lower expressiveness since all structures could still be expressed by using XMI's predefined linking mechanisms outlined in section 2.3.

5.2 Schema Production for Custom Languages

Deployment of XMI's schema production rules is currently not widely adopted but as evidence [Je01b] has shown implementation of the schema production algorithm is feasible.

Additionally, industry projects [Je01a] successfully demonstrated the usage of MOF-compliant metamodels generate XMI-compliant XML representations for data interchange.

5.3 Interchange of Visual Models

The revised metamodel introduced by UML 2.0 will add support for storing data describing the visual representation of a metamodel instance, i.e. a UML model. Consequentially, the altered metamodel will result in an update to the XMI serialization of UML created by the schema production rules described in section 2.2. In detail, also the parts of the metamodel which describe the structure of the visual model representation are subject to production rules just like any other metamodel entity.

Besides, the obvious capability of interchanging model data combined with its visual re-

presentation the inclusion of graphical meta data offers to process the visual information separately. An example of such an approach is the generation of Web presentable vector graphics directly from UML models described by [BJMF02].

References

[BHL99] Bray, T., Hollander, D., und Layman, A. (Hrsg.): *Namespaces in XML*. World Wide Web Consortium. Boston, USA. Januar 1999. W3C Recommendation. URL: `www.w3.org/TR/REC-xml-names`.

[BJMF02] Boger, M., Jeckle, M., Müller, S., und Fransson, J.: Diagram Interchange for UML. In: Jézéquel, J., Hußmann, H., und Cook, S. (Hrsg.), *Proceedings of the 5th International Conference on the Unified Modeling Language*. S. 398–411. Oktober 2002. Lecture Notes in Computer Science 2460.

[BLM 01] Beech, D., Lawrence, S., Maloney, M., Mendelsohn, N., und Thompson, H. S. (Hrsg.): *XML Schema Part 1: Structures*. World Wide Web Consortium. Boston, USA. 2001. W3C Recommendation. URL: `http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/`.

[BM01] Biron, P. V. und Malhotra, A. (Hrsg.): *XML Schema Part 2: Datatypes*. World Wide Web Consortium. Boston, USA. 2001. W3C Recommendation. URL: `http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/`.

[DMO00] DeRose, S., Maler, E., und Orchard, D. (Hrsg.): *XML Linking Language (XLink) Version 1.0*. World Wide Web Consortium. Boston, USA. 2000. W3C Recommendation. URL: `http://www.w3.org/TR/2001/REC-xlink-20010627/`.

[IOU 01] International Business Machines Corporation, Oracle Corporation, Unisys Corporation, UBS AG, NCR Corporation, Genesis Development Corporation, Hyperion Solutions Corporation, und Dimension EDI: *Common Warehouse Metamodel (CWM) Specification*. Object Management Group. Framingham, USA. 2001. OMG Document ad/2001-02-01.

[Je01a] Jeckle, M.: Practical usage of W3C's XML-Schema and a process for generating schema structures from UML models. In: *Proceedings of the Second International Conference on Advances in Infrastructure for E-Business, Science, and Education on the Internet*. August 2001. Rome, Italy, 6.-11. August, 2001.

[Je01b] Jeckle, M.: Using XSLT to derive schemata from UML. In: Rahtz, S. und Pawson, D. (Hrsg.), *Proceedings of the International Conference on XSLT*. S. 51–102. April 2001. Oxford, UK.

[Ob01] Object Management Group (Hrsg.): *Model Driven Architecture*. Framingham, USA. Juli 2001. OMG Document ormsc/2001-07-01.

[Ob02a] Object Management Group (Hrsg.): *Meta Object Facility (MOF) Specification*. Object Management Group. Framingham, MA, USA. April 2002. Version 1.4.

[Ob02b] Object Management Group (Hrsg.): *XML Metadata Interchange (XMI)*. Object Management Group. Framingham, MA, USA. Januar 2002. Version 1.2.

[Ob03a] Object Management Group (Hrsg.): *MDA Guide*. Object Management Group. Framingham, USA. 2003. Version 1.0.1, OMG docuemnt omg/2003-06-01.

[Ob03b] Object Management Group (Hrsg.): *XML Metadata Interchange (XMI) Specification*. Object Management Group. 2003. OMG Document formal/2003-05-02.

[Ri02] Rikki Kirzner: *Worldwide Analysis, Modeling, and Design Tools Forecast and Analysis 2002–2006*. IDC. 2002. IDC document no. 24809.

[Si01] Siegel, J. Developing in OMG's Model-Driven Architecture. Object Management Group. November 2001. White Paper, Revision 2.6.

[UFI 99] Unisys Corporation, Fujitsu, International Business Machines Corporation, Softeam, Cooperative Research Centre for Distributed Systems Technology, Recerca Informatica, Oracle Corporation, DaimlerChrysler AG, und Platinum Technology, Inc.: *XML Metadata Interchange (XMI) Version 1.1*. Object Management Group. 1999. OMG Document ad/1999-10-02.

[UIC 98] Unisys Corporation, IBM, Cooperative Research Centre for Distributed Systems Technology, Oracle Corporation, Platinum Technology, Inc., Fujitsu, Softeam, Recerca Informatica, und DaimlerChrysler: *XML Metadata Interchange (XMI). Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF)*. Object Management Group. Framingham, MA, USA. 1998. OMG Dokument ad/98-10-05.

[UM01] UML Partners: *OMG Unified Modeling Language Specification*. Framingham, MA, USA. Februar 2001. Version 1.4, OMG Dokument ad/2001-02-14.

[UM03] UML 2 Partners: *Unified Modeling Language: Superstructure*. Object Management Group. Framingham, MA, USA. April 2003. 3rd Revised Submission. OMG Dokument ad/2003-04-01.