MOF and XMI
(version 2.0)

by Philippe Nguyen
March 24, 2005

McGill
School of Computer Science
What should you get from this?

• A clear understanding of:
  – The “big picture” of the MOF 2.0 and XMI 2.0
  – The motivation behind each standard and the role that they play
  – Some important details about each specific standard
Topics Overview

• Introduction
  – Overview of Metamodeling
  – Key Acronyms and Standards

• MOF Details
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• XMI Details
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• Conclusion
Topics Overview

- **Introduction**
  - Overview of Metamodeling
  - Key Acronyms and Standards
- **MOF Details**
  - Goals
  - MOF/UML Relation
  - Capabilities
  - EMOF
  - CMOF
  - Abstract Semantics
- **XMI Details**
  - Motivation and Goals
  - XMI Model
  - Schema and Document Production from MOF
- **Conclusion**
### The Four Level Metamodel Hierarchy

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
</table>
| **M3** | (Meta-metamodel) - Defines a language for specifying a metamodel  
- Example: MOF  
- Typically more compact than the metamodel it describes  
- Can define many metamodels |
| **M2** | (Metamodel) - Defines a language for specifying models  
- Example: UML, CWM  
- Is an instance of a meta-metamodel (every element of the metamodel is an instance of an element of the meta-metamodel) |
| **M1** | (Model) - Defines a language that describe semantic domains  
- Example: model of different problem domains such as software, business, processes, and requirements  
- Is an instance of a metamodel  
- The things that are modeled reside outside the metamodel hierarchy  
- The user model contains both model elements and snapshots of instances of these model elements |
| **M0** | (Instance) - Contains run-time instances of the model elements defined in a model  
- The snapshots modeled at the M1 layer are constrained versions of the M0 run-time instances |
The Four Level Metamodel Hierarchy (cont’d)

• Key metamodelling concepts:
  – Classifiers/Classes ⇔ Instances/Objects

• A metamodelling facility must give the ability to navigate from an instance to its metaobject
Topics Overview

• **Introduction**
  – Overview of Metamodelling
  – Key Acronyms and Standards

• **MOF Details**
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• **XMI Details**
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• **Conclusion**
Links


• MOF 2.0 Core Specification: http://www.omg.org/docs/ptc/03-10-04.pdf

UML

- The Unified Modeling Language

- Upcoming version is 2.0

- OMG standard providing:
  - A framework for specifying, constructing and documenting system artifacts
  - A general-purpose visual modeling language

- Collection of modeling formalisms
  - Most frequently used in Object-Oriented systems is the Class Diagram

- Specification includes *Infrastructure* and *Superstructure*
UML Infrastructure

- Defines basic and more complex modeling constructs that underlie the entire UML architecture
  - Architectural kernel

- Defined by the *InfrastructureLibrary* package

- Basic concept:
  - MOF (EMOF + CMOF) is built upon the merges of certain subpackages defined in *InfrastructureLibrary*
MOF

• The Metadata Object Facility

• Upcoming version is 2.0

• OMG standard that provides a metadata management framework
  – Create, destroy, find, manipulate, and change objects and relationships between those objects as prescribed by metamodels

• Is to be used as the platform-independent metadata management facility for OMG’s Model Driven Architecture (MDA)
  – i.e. build PIMs that are to be transformed to PSMs

• Specification includes the EMOF and the CMOF
EMOF

• The Essential MOF

• “Minimal” subset of the MOF
  – Allows simple metamodels to be defined, using the most basic class diagram concepts

• Serves as a first stepping stone to model driven tool development and tool integration
  – E.g. Eclipse’s EMF is based on Ecore
CMOF

- The Complete MOF
- Used to specify metamodels such as the UML
- Adds more complex constructs to the EMOF
XMI

- The **XML Metadata Interchange**
- OMG standard for serializing MOF-based models to XML format
- Allows tools to exchange model information seamlessly
Topics Overview

• Introduction
  – Overview of Metamodeling
  – Key Acronyms and Standards

• MOF Details
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• XMI Details
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• Conclusion
Goals (1)

• Easier to define and extend models and metamodels
  – Unifying MOF2 and UML2 under a common core should help accomplish this

• Modular and hierarchical models (component-based modeling)
  – Model packages can be imported by other models

• Platform-independence of MOF
  – Interoperability of different tools using XMI
Goals (2)

• Integrate fundamental capabilities directly inside the MOF
  – Model Reflection in MOF as an independent service
  – Model Identity to improve interoperability
  – Model Extension to allow annotation of models

• As a result, we have:
  – Orthogonality between the capabilities and the technology
    • E.g. Reflection is not specific to CORBA
  – A top-down definition of the capabilities
    • All MOF-based metamodels will inherently possess all capabilities
  – MOF capabilities that can be reused at different meta-layers
Topics Overview

• Introduction
  – Overview of Metamodelling
  – Key Acronyms and Standards

• MOF Details
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• XMI Details
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• Conclusion
MOF/UML Relation

- We will look at two aspects of the relation:
  1. Roles of UML Infrastructure
  2. Differences between MOF and UML
Roles of UML Infrastructure

• Defined by InfrastructureLibrary

- **Core**: contains core concepts used when meta-modeling (e.g. classes)
- **Profiles**: defines the mechanisms that are used to customize metamodels (e.g. stereotypes)

• The design of UML Infrastructure into fine-grained packages facilitates the definition of the rest of UML
UML Infrastructure Library::Core

**PrimitiveTypes:** contains a few predefined types that are commonly used when metamodeling.

**Basic:** provides a minimal class-based modeling language on top of which more complex languages can be built. It is intended for reuse by the EMOF.

**Abstractions:** mostly contains abstract metaclasses that are intended to be further specialized or that are expected to be commonly reused by many metamodels.

**Constructs:** mostly contains concrete metaclasses that lend themselves primarily to object-oriented modeling. It is intended for reuse by the CMOF.
Example: Core::Basic::Types

```
Element

NamedElement
- name : String [0..1]

TypedElement

Type
  +type 0..1
```
UML InfrastructureLibrary::Core (cont’d)

• A complete metamodel
  – Designed for high reusability
  – Metamodels at the same metalevel either import or specialize its metaclasses

• InfrastructureLibrary::Core is reused by MOF, UML Superstructure (Kernel package), and UML Infrastructure

• The goal is to reuse the same core modeling concepts between UML, MOF and other emerging OMG metamodels
Differences between MOF and UML

- **MOF**
  - Provides the metadata services
  - Defines the meta-metamodeling language to define other metamodels like UML
    - M3 level: needs to be simpler than UML
  - Defines a model interchange standard (XMI)

- **UML**
  - Provides the modeling (and metamodeling) notation
    - M2 and M1 levels: model elements have added annotations
  - General-purpose modeling language
    - Potentially many target application domains
The Big Picture...

• MOF 2.0 was built on reusing the Core package by the merge, and combine mechanisms

• The advantages are threefold:
  – Simpler rules for modeling metadata, since we only need to learn a subset of UML class diagrams, and no additional constructs
  – Various technology mapping from MOF (e.g. XMI, JMI) now apply to a broader range of UML models, such as UML Profiles
  – Broader tool support for metamodeling, since any UML modeling tool could be also used as a metamodeling tool
Topics Overview

• Introduction
  – Overview of Metamodeling
  – Key Acronyms and Standards

• MOF Details
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• XMI Details
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• Conclusion
MOF Capabilities

• The MOF specifies three capabilities that add-on to the modeling constructs from UML Infrastructure:
  – Reflection: Allows discovery and manipulation of metaobjects and metadata
  – Identifiers: Unambiguously distinguishes objects from each other
  – Extension: Allows dynamic annotation of model elements with additional information

• Each capability is encapsulated in a separate package
  – Technology independent
    • Any MOF-based metamodels will possess the capabilities
  – Can be imported (merged) into other metamodels
Reflection

- **The Object Class**
  - Holds the reflective interface
  - Rationale: used in the production of EMOF, which can then be merged into CMOF to provide reflective capabilities to MOF and all instances of MOF

- Having both MOF and MOF instances be rooted in class *Object*, MOF supports any number of metalayers
Identifiers

- **Applications:**
  - Coordinate model updates
  - Object communication in user interfaces
  - In XMI, object identity can simplify referencing to external objects
  - In MDA, identity is crucial for model (graph) transformations, in order to correlate elements from source and target models
Extension

- Allows dynamic annotation of model elements with additional, and perhaps unanticipated, information
- Provides a simple mechanism to associate a collection of name-value pairs with model elements
Topics Overview

• **Introduction**
  – Overview of Metamodelling
  – Key Acronyms and Standards

• **MOF Details**
  – Goals
  – MOF/UML Relation
  – Capabilities
    – EMOF
    – CMOF
  – Abstract Semantics

• **XMI Details**
  – Motivation and Goals
  – XMI Model
    – Schema and Document Production from MOF

• **Conclusion**
EMOF

- Purposes:
  - Provides the minimal set of elements required to model object-oriented systems
  - Allows simple metamodels to be defined, using the most basic class diagram concepts
  - Gives a fixed modeling base in order to keep the mapping from MOF/UML to XML stable
  - Provides a straightforward framework for mapping MOF models to implementations such as JMI and XMI for simple metamodels
  - Lowers the barrier to entry for model driven tool development and tool integration
EMOF Definition

• EMOF = combine(Basic, Reflection, Identifiers, Extension)

• We would like EMOF to simply extend Core::Basic
  – But, Reflection has to introduce Object in the class hierarchy between Basic::Element and Basic::NamedElement
  – So, we need CMOF’s <<combine>> mechanism

• Described in CMOF
  – But, in order for it to be a usable standalone package, it is also specified in itself by removing all redefinitions and merges
    • using the CMOF’s <<combine>> mechanism

• Reason for specifying EMOF as a complete merged model:
  – Provide a metamodel that can be used to bootstrap metamodel tools rooted in EMOF without requiring an implementation of CMOF and package merge semantics
Topics Overview

• **Introduction**
  – Overview of Metamodelling
  – Key Acronyms and Standards

• **MOF Details**
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• **XMI Details**
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• **Conclusion**
CMOF

• Purposes:
  – Completely define the UML 2.0
  – Define package extending mechanisms
    • Package import
      – model elements contained in the imported package are made visible in the importing package
    • Package merge
      – classes in the merging package specialize similarly named classes in the merged package adding new features
    • Package combine
      – a new package consisting of the model elements of the combined and combining packages is defined
  – These mechanisms are used throughout the MOF and the UML to define metamodels
    • E.g. EMOF = combine(Basic, Reflection, Identifiers, Extension)
CMOF Definition

- CMOF = merge(Constructs, EMOF, CMOFExtension, CMOFReflection)

- Constructs
  - Similar to Basic
  - More complex constructs, e.g. support of user-defined DataType with attributes and operations

- CMOFExtension

- CMOFReflection
  - Extension to Factory to conform to the XMI 2.0 specification
CMOF Definition (cont’d)

- **Classes Diagram**
  - Association and Class are both Classifiers
    - Novelty: Associations can be generalized
  - Classes own Properties and Operations
  - Associations relate Properties of Classes
CMOF Definition (cont’d)

• Constraints Diagram
  – Constraints apply to Elements in a certain context (Namespace)
  – The constraint specification is a ValueSpecification
    • A ValueSpecification identifies values in a model
    • Can be an Expression (e.g. \(a + b = 3\))
    • Can be an OpaqueExpression (e.g. an OCL statement)
• Packages Diagram
  – PackageMerge is a DirectionalRelationship between two Packages
    • “extend” \( \rightarrow \) Package Merge
    • “define” \( \rightarrow \) Package Combine
Package Merge

• Set of transformations where the elements of the merged package are expanded in the merging package

• General idea:
  – Model elements match by name
  – Matching elements are merged together using inheritance and redefinitions
  – Done until there are no more duplicate elements

• At the end of the transformations, the package merge relationship is transformed into a package import relationship, with the same source and target packages
  – The relationship is maintained
Package Merge Example
Package Combine

• Set of transformations where the elements of the combined package are “deeply” copied in the combining package

• General idea:
  – Packages, Classes, Properties match by name
  – Associations match either by name (if any) or by memberEnds
  – Operations match by name and parameters
  – New model elements are born from the combination of matching elements from the combined and combining packages

• At the end of the transformations, the package dependency is removed from the model
Package Combine (cont’d)

• Deep copy:
  – Copy non-matching elements to the combining package
  – Matching packages: combine their classes and associations
  – Matching classes: combine their properties and ignore matching operations
  – Matching properties: find the most specific type and multiplicity
    • Most specific type: the “closest” supertype of both properties, in the combining package
    • Most specific multiplicity: largest lower bound and smallest lower bound
  – Matching association: combine the related classes
MOF Recap

• The MOF (EMOF + CMOF) is described using UML Infrastructure (+ OCL and natural language)

• The EMOF is described using CMOF

• The EMOF is also completely described in EMOF, using CMOF’s combine mechanism

• The CMOF is described using CMOF itself

• The UML2 uses CMOF in its definition
Topics Overview

• **Introduction**
  – Overview of Metamodelling
  – Key Acronyms and Standards

• **MOF Details**
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• **XMI Details**
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• **Conclusion**
• **Instances Diagram**
  – InstanceSpecification represents an instance in a modeled system
    • References a Classifier → its “metaobject”
    • An InstanceValue specifies the value modeled by an InstanceSpecification
    • Has Slots, which specify the value or values for its defining feature, which must be a StructuralFeature of the classifier referenced by the InstanceSpecification owning the Slot
Topics Overview

• Introduction
  – Overview of Metamodelling
  – Key Acronyms and Standards

• MOF Details
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• XMI Details
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• Conclusion
Motivation and Goals

• Define an industry standard for MOF/UML model serialization

• Allow tools to
  – Exchange model information seamlessly
  – Include tool-specific information without affecting the model representation
  – Transmit incomplete metadata, and metadata deltas

• Define production rules for XML Schemas and Documents
  – Schema: validates an XML document, i.e. the metamodel
  – Document: contains actual model information, i.e. the model
  – Backward rules are also defined, but not discussed here!

• Difference from MOF 1.4 XMI:
  – Use of XML Schemas instead of DTDs
XMI Definition

• Two important aspects come into play in the definition of XMI standard:
  – The XMI Model
  – The Production rules

• XMI Model = Instance of MOF
  – Used to describe XMI-specific information
    • i.e. version, documentation, extension, differences
  – XMI can be treated like any other MOF metadata

• Production rules specify as formally as possible the manner in which MOF-based models should be transformed into both XML Schemas and Documents
Topics Overview

• **Introduction**
  – Overview of Metamodeling
  – Key Acronyms and Standards

• **MOF Details**
  – Goals
  – MOF/UML Relation
  – Capabilities
  – EMOF
  – CMOF
  – Abstract Semantics

• **XMI Details**
  – Motivation and Goals
  – XMI Model
  – Schema and Document Production from MOF

• **Conclusion**
XMI Attributes

• XMI defines a set of fixed XML attributes used throughout the production of XMI schemas

• Consistent attributes → consistent architectural structure → consistent object identity and linking across all assets

• Example: Element Identity Attribute & Linking Attributes

```xml
<xsd:attribute name="id" type="xsd:ID" use="optional"/>
<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>

<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>
```
Topics Overview

- **Introduction**
  - Overview of Metamodeling
  - Key Acronyms and Standards
- **MOF Details**
  - Goals
  - MOF/UML Relation
  - Capabilities
  - EMOF
  - CMOF
  - Abstract Semantics
- **XMI Details**
  - Motivation and Goals
  - XMI Model
  - Schema and Document Production from MOF
- **Conclusion**
Schema Production from MOF

• Set of rules that show
  – What declarations must be contained in any well-formed XMI Schema (content)
  – How the information is structured (structure)

• Mapping between any type of MOF model element and a schema declaration
  – E.g. a class maps to a complexType declaration
  – E.g. multiplicities map to minOccurs and maxOccurs indicators

• Formalized using a textual grammar
  – Extended BNF (EBNF)
EBNF Snippet

...  
4.b ClassContents ::= 4d:ClassAttributes 
    4e:ClassReferences 
    4f:ClassCompositions 
    4c:Extension 

4c. Extension ::= ("<xsd:element ref='xmi:extension'/>")* 

4d. ClassAttributes ::= ("<xsd:element name='" /Name of Attribute// "'" ("nillable='true'")? 
    ( 4m:MinOccursAttrib )? ( 4n:MaxOccursAttrib )? 
    ("type=" /Name of type// "/"">") | ("type='xmi:Any'/>")) )* 

4e. ClassReferences ::= ("<xsd:element name='" /Name of Reference// "'" 
    ( 4m:MinOccursAttrib )? ( 4n:MaxOccursAttrib )? 
    ("type=" 4a:ClassTypeName "/"">") | ("type='xmi:Any'/>")) )* 

4f. ClassCompositions ::= ("<xsd:element name='" /Name of Reference// "'" 
    ( 4m:MinOccursAttrib )? ( 4n:MaxOccursAttrib )? 
    ("type=" 4a:ClassTypeName "/"">") | ("type='xmi:Any'/>")) )* 

...  
4m. MinOccursAttrib ::= "minOccurs='" // Minimum // "'" 
4n. MaxOccursAttrib ::= "maxOccurs='" // Maximum // "'" 
...
Overview of Model Representation

GIS Model Example
(from MOF XMI spec)
Document Production from MOF

• Similar to Schema production rules
  – Also in EBNF
  – But less fixed declarations (i.e. namespaces)

• Defines a serialization model:
EBNF Snippet

2:XMIElem ::= 2a:XMIOObjectElement | 2b:XMIValueElement | 2c:XMIReferenceElement

2a:XMIOObjectElement ::= ( "<" 2k:QName 2d:XMIAtribs "/>") |
( "<" 2k:QName 2d:XMIAtribs ">" (2:XMIElem)* "/" 2k:QName ">" )

2b:XMIValueElement ::= ( "<" xmiName ">" value "<!" xmiName ">" ) | ( "<" xmiName "nil='true'/" ")

2c:XMIReferenceElement ::= "<" xmiName (2g:TypeAtrib)? 2l:LinkAtribs "/"

2d:XMIAtribs ::= (1c:StartAtribs)? (2e:IdentityAtribs)? (2g:TypeAtrib)? (2hFeatureAtrib)*

2e:IdentityAtribs ::= ( 2f:IdAtribName ";=" id ";")? ( xmiPrefix "label=" label "")? ( xmiPrefix "uuid=" uuid "")?

2f:IdAtribName ::= xmiPrefix "id" | xmildAtribName

2g:TypeAtrib ::= (1b:XMINamespace | 1g:Namespace) "type=" 2k:QName ""

2h:FeatureAtrib ::= 2i:XMIValueAtrib | 2j:XMIReferenceAtrib

2i:XMIValueAtrib ::= xmiName ";=" value ""

2j:XMIReferenceAtrib ::= xmiName ";=" (refId | 2n:URIref) ""

2l:LinkAtribs ::= 1b:XMINamespace "idref=" refId "" | 2m:Link

2m:Link ::= "href=" 2n:URIref ""
• Composite property serialized as XML elements, the opposite property is not serialized.
Conclusion

• **UML** is a general-purpose modeling notation with a plethora of application domains

• **UML InfrastructureLibrary** defines small subpackages that can be reused by UML and MOF (EMOF + CMOF)

• **MOF** is a metamodeling framework that provides reflection, identity, and extension services

• **EMOF** is a minimal subset of MOF used to define very simple metamodels

• **CMOF** is the complete meta-metamodel used to define UML2 completely

• **XMI** is the OMG standard for serializing MOF-based models
References


• For more information about issues of the standard specifications, see http://www.omg.org/issues