

# MOF und XMI

Xun Zhang<sup>1</sup>

<sup>1</sup> TU - Kaiserslautern  
xunzhang@rhrk.uni-kl.de

**Abstract (dt.: “Kurzfassung”).** Meta Object Facility (MOF) ist ein Standard der OMG. Die MOF-Spezifikation beschreibt eine abstrakte Sprache und ein Framework zur Verwaltung der plattformunabhängigen Metamodellen, z.B. UML, Common Warehouse Metamodel (CWM) und sowohl MOF selbst. MOF ist eine entscheidende Grundlage von MDA. XMI (XML Metadata Interchange) ist auch ein Standard der OMG, die die Regeln zur Umwandlung von MOF-Modell ins XML-Dokument und auch von XML-Dokument ins MOF-Modell definiert. Damit kann man MOF-Modelle durch eine einheitliche Methode speichern und umtauschen.

## 1. MOF

MOF ist eine Beschreibungssprache für alle Modellierungssprachen der OMG. Für MDA ist MOF sogar wichtiger als UML, da auch UML durch MOF definiert ist. Über MDA werde ich später erklären.

### 1.1. Was ist MOF?

Die Entstehung von MOF liegt auf einer Basismeinung zugrunde, dass es mehr als eine Art von Modellen gibt und deshalb es auch mehr als eine Modellierungssprache gibt, z.B.: UML und CWM. Um eine Art von Modellen oder eine Modellierungssprache zu beschreiben, braucht man hier eine einheitliche Methode. Und MOF ist diese einheitliche Methode. Mittels MOF werden die Struktur von Datenmodell oder von UML-Klass-Modell beschrieben. Mit MOF kann man auch alle anderen Arten von Modellen beschreiben. Wir können sogar mit MOF eine neue Modellierungssprache definieren.

Eine andere Bedeutung von MOF ist: wenn man Modelle der verschiedenen Arten umtauschen möchte, dann muss man eine Abbildungsregel (Mapping) für jeweils zwei Arten von Modellen (Metamodellen, Metamodel heißt Modell von Modell) definieren. Bei 6 Arten von Modellen muss man 15 Abbildungsregeln definieren. Aber wenn wir MOF benutzen, dann wird die Situation sehr einfach. Man muss nur eine MOF-Abbildung für eine Modellierungssprache definieren. (Siehe Abb. 1.1)

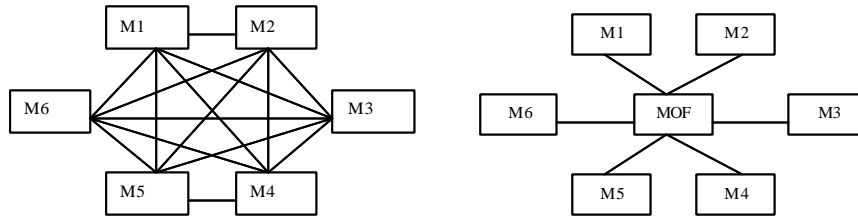


Abb. 1.1

MOF benutzt UML als seine Abstraktsprache, die das Metamodell definieren kann. Deshalb sieht man das MOF-Modell ähnlich wie Klassenmodell von UML. Wir können MOF-Metamodell anhand UML-Modellbildungs-Werkzeug erstellen. In MOF: Modellbildungsstruktur entspricht Klassen-Modell, und Attribute von Struktur entspricht Attribute von Klass. Die Relation zwischen Struktur entspricht Assoziation. OCL(Object Constraint Language) wird hier zur Erhöhung der Genauigkeit der Metamodell benutzt.

Dann wie sieht die Struktur von MOF aus?

### 1.2. 4-Schichten Metamodellarchitektur

Die Struktur von MOF enthält 4 Meta-Ebene, sogenannte M3, M2, M1, M0. Abb. 1.2 zeigt diese 4-Schichten-MOF-Metadaten-Architektur.

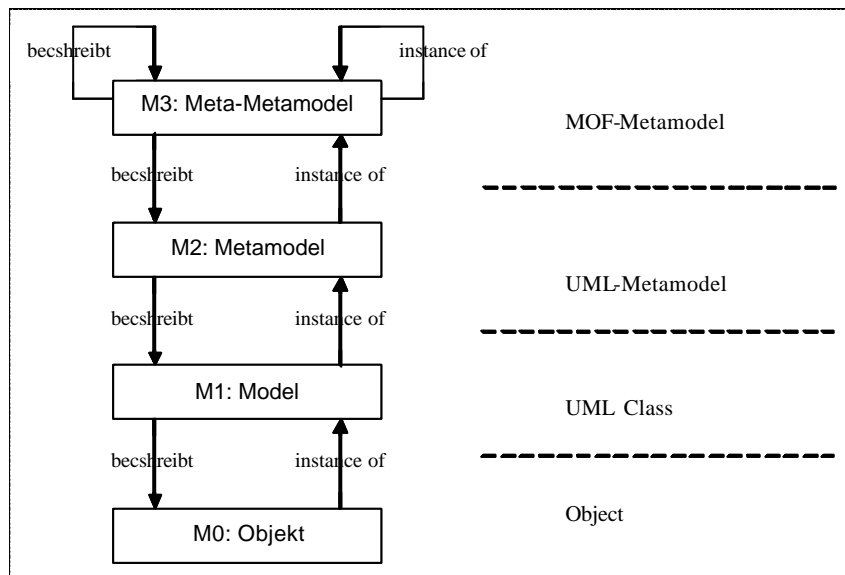


Abb. 1.2 4-Schichten-Architektur

### 1.2.4 M0

M0, Objekte-Schicht, sind eine Instanz von M1 und beschreiben die Ausprägungen eines bestimmten Modells (z.B. Buch: UML2.0, Autor: OMG, usw.).

### 1.2.3 M1

M1, Modell-Schicht, enthält alle Arten von Modellen und ist eine Instanz des M2 und definiert die Sprache zur Beschreibung der Domäne (z.B. Klasse: Buch, Klasse: Autor, usw.).

### 1.2.2 M2

M2, MetaModell-Schicht, ist eine Instanz von M3 und definiert die Sprache zur Beschreibung der Modelle (z.B. Klasse, Attribut, Operation). Diese Schicht ist das zentrale Element der UML und die Konzepte werden bei der UML-Modellierung verwendet (z.B.: UML Class, UML Association, UML Attribute, UML State, usw.). Abbildung 1.3 zeigt die vereinfachte Metamodell von UML

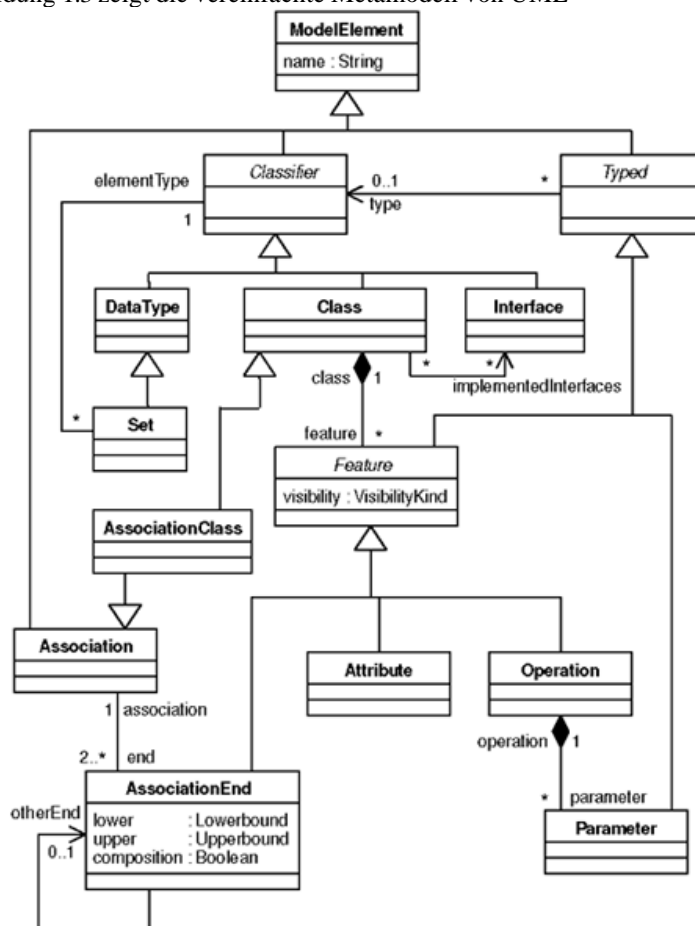


Abb.1.3 MetaModell von UML(vereinfacht)

### 1.2.1 M3

M3-Schicht ist MOF. Es ist die Infrastruktur der Metamodellarchitektur und definiert die Sprache zur Beschreibung von Metamodellen, z.B. die Menge der Konstrukte die zur Definition von Metamodellen genutzt wird (MOF Class, MOF Attribute, MOF

Association, usw).

Man kann es auch als Meta-Metamodell

nennen. Schicht M3 ist die Spitze der Architektur und es keine höhere Schichte gibt,

d.h. MOF definiert MOF selbst. Für

selbst definiert MOF ein Modell, das die

Spezifikation von MOF befolgt. Ab-

bildung 1.4 zeigt ein Fragment von MOF-

Metamodell. Ausführliche Definition finden

Sie in Meta Object Facility (MOF) 2.0

Core Proposal.

In der normalen Soft-

wareentwicklung werden oft nur die Schichten 1 und 2 (M0 und M1) genutzt. Die

Vorteile dieser Architektur gegenüber einer einfachen (flachen) Modellarchitektur

liegen einerseits in der flexiblen Erweiterbarkeit und andererseits im einfacheren und

klaren Informationsaustausch auf Instanzenebene.

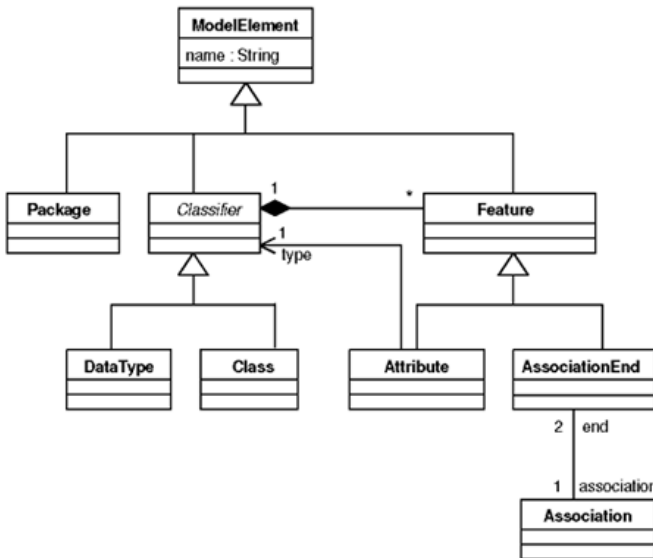


Abb 1.4 MOF-Metamodell(vereinfacht)

### 1.3. Ausrichtung von MOF und UML

Ein großes Problem für UML1.x ist, dass UML und MOF sich nicht gut anpassen.

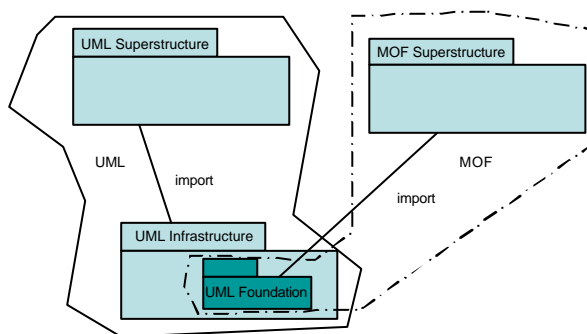


Abb. 1.5

MOF und UML sollen einen gleichen Kern benutzen.

UML1.x und MOF benutzen fast einen gleichen Kern. Aber

nur „Fast“ reicht nicht aus. MOF

spielt eine große Rolle in MDA und

das Problem hast

ein paar Schwierigkeiten für die Entwicklung von MDA verursacht (Über MDA wird später erzählt). Bei UML2.0 wird das Problem gelöst. UML und MOF benutzen das gleiche Basispaket gemeinsam.

#### 1.4. Vorteil von MOF

MOF wird im Jahre 1997 der OMG ratifiziert. Bis heute hat MOF mit XML und Java gut kooperiert ohne irgend eine Veränderung. Und in Zukunft wird MOF stimmt auch mit z.B. C# gut kooperieren. Der Erfolg von MOF verknüpft sich eng mit seinem Vorteil: Plattformunabhängigkeit. Beim Definieren von MOF hat OMG bemerkt, MOF so neutral wie möglich zu behalten. Deshalb enthält MOF gute Plattformunabhängigkeit.

Das bedeutet, dass MOF unabhängig von:

1. Informationsformatierungstechnologien wie XML-DTD oder XMLSchema
2. 3 oder 4GL Programmiersprachen wie Java, C++, C#, VB
3. Component/messaging Middleware wie J2EE, CORBA, .NET

Durch standardisierte Abbildung (Mapping) kann ein auf MOF basierende Generator die Abstraktsprache von Metamodell ins Format von z.B XML DTD, Java oder CORBA umwandeln.

Bei Generierung im Detail existieren verschiedene MOF-Mappings, z.B:

1. MOF zu CORBA
2. MOF zu XML (sogenannte XMI)
3. MOF zu Java (sogenannte Java Metadata Interface, JMI)

#### 1.5. MOF-Mappings

Abb.1.5.2 zeigt noch, dass die Transformation von Metamodellen auf Metadaten Interface eine wichtige Rolle im Metadatenmanagement spielt. OMG hat momentan schon ein paar MOF-Mappings definiert, z.B MOF-CORBA, MOF-XMI, MOF-Java, usw. OMG beschäftigt sich noch MOF auf mehrere Technologie abzubilden wie c#.

##### **MOF-CORBA-Mapping**

Das erste der OMG definierte MOF-Mapping ist MOF-CORBA-Mapping. Die Abbildung von MOF Metamodellen auf CORBA IDL definiert wie IDL-Schnittstellen erstellt werden sollen. Das ermöglicht die Darstellung und die Verwaltung der Metadaten. Diese Abbildung wird danach auf UML-Metamodell angewendet. Dadurch wird der Regel, wie man durch eine Menge von CORBA-IDL-API die UML-Metadaten ins CORBA-Objekt umwandeln soll, definiert.

##### **MOF-XML-Mapping**

Diese Abbildung wird auch als XML-Metadata-Interchange genannt. XMI definiert eine Menge von Regeln zur Darstellung von Metamodell durch XML(Extensible Markup Language).

Anhand dieser zwei Abbildungen kann ein MOF-Generator z.B. durch aus MOF stammende CORBA-API Modelle aus Repository lesen und exportiert es durch XMI, oder MOF-Generator kann auch XMI-Dokument importieren und danach durch CORBA-API die Modelle ins Repository speichern.

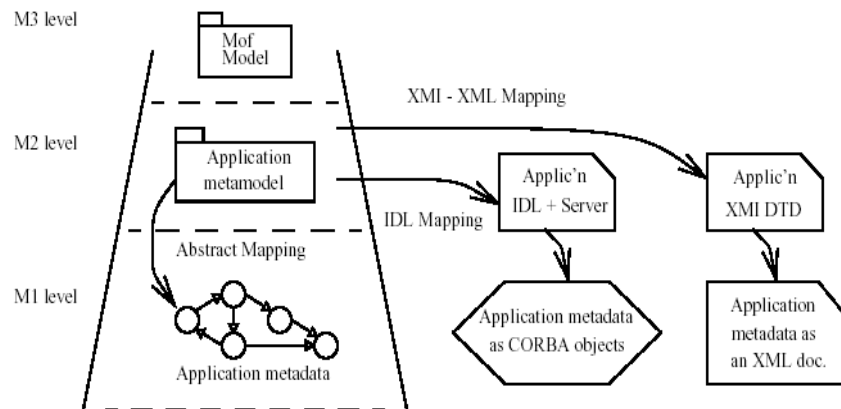


Abb. 1.6

Durch das "abstract Mapping" (M3 nach M2) wird zunächst das abstrakte "Application Metamodel" beschrieben, welches direkt durch IDL und XMI Mapping in applikationsspezifische Schnittstellen und DTDs umgesetzt wird. Dann wird im Rahmen des "abstract Mapping" die logische Struktur weiter auf den M1 Level abgebildet, wobei nun die abstrakte IDL und die XMI DTD durch passende Corba Objekte und XML Dokumente gefüllt werden. Somit kann das "Application Metamodel" leicht angepasst werden, da die IDL und DTD bzw. deren Instanzen leicht automatisch generiert werden können.

### 1.6. Überblick von MDA

Vorher habe ich MDA erwähnt. Hier mache ich eine kurze Vorstellung zu MDA.

MDA (auf Deutsch: Modell getriebene Architektur) ist eine Strategie der OMG zur modellgetriebenen Software-Entwicklung und ist eine einheitliche und effiziente Methode für die Entwicklung der objektorientierten Applikationen.

MDA benutzt Modellierungssprache nicht nur als Entwurfssprache sondern auch als Programmierungssprache. Durch Trennung der Abstraktionsschichten bei der Modellierung erhöht MDA Produktivität, Qualität und Langlebigkeit der Software.

Das Ziel der MDA ist zu verkleinern oder schließen die Lücke zwischen Modellen und ausführbare Code. Das Idee der MDA ist: Aus implementierungsunabhängigen Modell (Platform Independent Model, PIM) werden durch Transformation das implementierungs-plattform-spezifisches Modell (Platform Specific Model, PSM) erstellt und

schließlich durch Transformation die Code generiert. Die Transformation wird durch entsprechende Mapping-Regeln gesteuert.

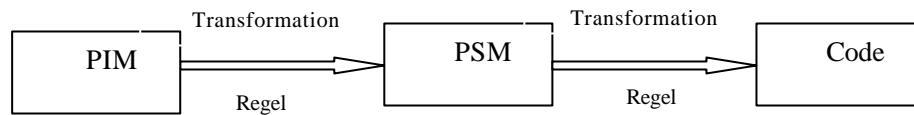


Abb. 1.7

Durch diese Methode realisiert es, dass jede Änderung im Entwicklungsprozess sich nur auf den einzelnen Abstraktionsebenen bezieht. Die Anwendung der MDA ist deshalb eine effizientere Methode für die Programmierer, da sie die Portierungs- und Integrationskosten senken kann. Doch der Stickpunkt dafür ist die Implementierbarkeit von UML. Es braucht noch viel Zeit.

Im Jahre 2002 hat OMG bekanntgegeben, dass MDA ihre strategische Richtung ist. Und momentan gibt es schon die Produkte, die die Eigenschaften von MDA besitzen. Aber MDA ist momentan nur eine Technologie von Zukunft. Durch die Kombination und die Verbesserung der bisherigen Standards der OMG wird MDA realisierbarer. Das ist auch ein Grund der Veröffentlichung von UML 2.0.

### 1.7. Zusammenfassung von MOF

MOF ist wahrscheinlich schwer zu verstehen, da es relativ abstrakt ist. Kurz gesagt, alle Modellierungssprachen der OMG werden von MOF definiert inklusive UML. Und MOF ist die Basis der MDA

## 2. XMI

Vorher haben wir über MOF gesprochen. Dann durch welche Methode kann man MOF-Modell mit anderer Applikation oder mit anderem Benutzer umtauschen? Im zweiten Teil werden wir über das Problem sprechen.

### 2.1. Problem im Metadaten-Austauschen

Beim Metadaten-Austauschen zwischen verschiedenen Applikationen sowie Speichern von Metadaten braucht man für jede Kombination von jeweils zwei Applikationen einen Import- und Exportfilter. Bei 6 Applikationen sind das schon 30 Filter. (Siehe Abb. 2.1) Die Menge der benötigten Filter wächst quadratisch.

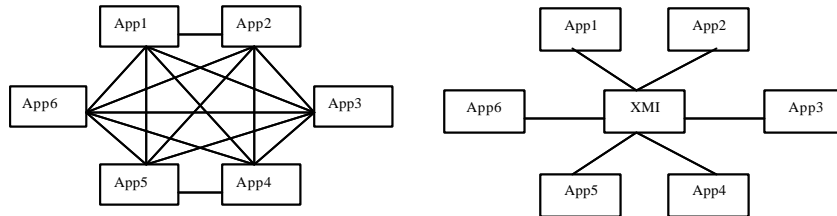


Abb.2.1

Außerdem gibt es noch Versionsproblemen. Wenn es eine neue Version von z.B. App1 gibt, dann müssen alle Filter von oder zu App1 geändert werden. Um das Problem zu beseitigen braucht man natürlich ein einheitliches Format zum Speichern der Daten. XMI liefert die Lösung.

## 2.2. XMI Einführung

Monentan ist XML(eXtensible Markup Language) schon eine oft benutzte Sprache zum Speichern und Umtauschen der Daten, da die Struktur von XML einfach ist und ist leicht zu verstehen.

XMI wird der OMGdefiniert. In XMI-Spezifikaiton wird den Regeln von Abbilung von MOF-Modell zu XML und von XML zu MOF-Modell festgestellt. XMI basiert auf XML. Durch XMI können Metadaten ins Format von XML umwandelt werden und Metadaten wieder aus XML-File auslesen. Damit können wir die Metadaten speichern oder mit anderen Applikationen umtauschen.

XMI wird jetzt schon populär benutzt im Bereich von Modell-Bildung. Viele entsprechende Werkzeuge sind schon seit einiger Zeit verfügbar .

Beispiel: in Websphere System wird Modell ins Foramt von XMI gespeichert und UML2-Plugin für Eclipse benutzt auch solche Technik.

## 2.3. XML Übersicht

XML stammt aus SGML. SGML ist eine einheitliche und verallgemeine Markup-Sprache. Sie entsteht am Anfang der 80er Jahre und ist sehr kompliziert. W3C-Arbeitsgruppe versucht deshalb eine für das Internet optimierte Sprache zu entwickeln. Und Resultat ist XML.

XML-Dokument dient zur Datenhaltung. Es enthält die Dokumentendaten und eventuell zusätzliche Attribute.Die Struktur jedes XML-Dokuments besteht aus einem Prolog und dahinter wird von ineinander geschachtelten Container-Elementen gefolgt. Ein Beispiel:

```
<Proseminar>
    <Thema>UML 2.0<\Thema>
<\Proseminar>
```

Vorteil von XML liegt darin, dass es erweiterbar, meschlesbar ist und es durch Maschine verarbeitet werden kann.

## 2.4. XML-DTD

XML-DTD spezifiziert die Regeln, wie die XML-Elemente definiert sind. Sie ist ein Beispiel für eine Markup-Deklaration und mit ihrer Hilfe werden die Verständnis des Dokuments vereinbart. Der Rest von Beispiel beschreibt die Element von XML und ist wichtigsten Teile von XML.

xmlns steht für Name-Space. Ein XML-namespace definiert den Kontext für die Elemente and Attribute in XML-Dokument. Ein Namespace enthält ein Namespace-präfix und ein URI (Uniform Resource Identifier). Namespace-präfix dient zur Identifizierung von Namespace im XML-Dokumnet. Und URI ist der eindeutige Identifizierer für Namespace.

Beispiel (ein XML-Dokument und sein DTD-Dokument):

```
<?xml version="1.0" encoding="ISO-889-1"?>
<!DOCTYPE Heyne:BooksSYSTEM "sample.dtd">
<!--HerebeginstheXML data-->
<Heyne:Booksxmlns:Heyne="http://www.heyne.de">
<Heyne:Product>Hitchhiker'sguideto thegal-
axy</Heyne:Product>
</Heyne:Books>
```

```
<?xml version="1.0"?>
<!ELEMENT Heyne:Books(Heyne:Product)>
<!ATTLIST Heyne:Booksxmlns:HeyndeCDATA
"http://www.heyne.de">
<!ELEMENT Heyne:Product(#PCDATA)>
```

## 2.5. XML Schemas

XML-Schema ist eine neue Erweiterung für XML. Schema ist mächtiger als DTD. Durch Schema kann man mehrerer Einschränkungen für XML definieren. XML-Schema-Dokument ist genau ein XML-Dokument und die Grammatik ist gleich. Schema liefert einen Mechanismus zur Strukturierung von XML Dokumenten. Schema enthält wohlgeformter Namespaces und Constrains für primitive Daten eines Elements. Ein Beispiel:

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="MyBase">
<xsd:choice>
<xsd:element name="Choice1"/>
```

```

<xsd:element name="Choice2"/>
</xsd:choice>
<xsd:attribute name="baseAttrib" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="MyExtension">
<xsd:complexContent>
<xsd:extension base="MyBase">
<xsd:choice>

<xsd:element name="ExtensionElement1"/>
<xsd:element name="ExtensionElement2"/>
</xsd:choice>
<xsd:attribute name="extensionAttrib"
type="xsd:string"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

Dieser Beispiel zeigt auch, dass man mit Schema noch „Extension-Relation“ darstellen kann. XML-Schema wird jetzt schon populär benutzt im Bereich von z.B Datenaustausch im E-commerce und natürlich Metadata-Interchange.

## 2.6. Entwurf von XMI-Schema

XML DTD oder XML Schema dient zur Übertragung und Prüfung von UML basierte Modelle oder MOF basierte Metamodelle und deren Instanzen. Aber Schema ist neuer und mächtiger als DTD. Deshalb wird hier nur Schema vorgestellt(Ausführliche Anwendung über DTD kann man in XMI2.0 Spezifikation finden).

XMI-Schema benutzt den Definitionsregel von XML-Schemas. Mann kann den Information von Metamodell druch XML-Validierung prüfen. Hier tauchen zwei Begriffe auf, Generierung und Validierung von Schema. Abb. 2.2 zeigt die Unterschiede davon aus.

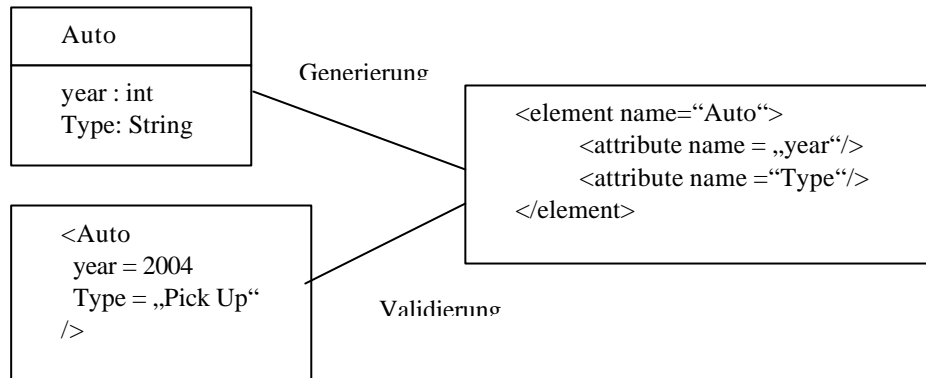


Abb. 2.2

### Prinzip in Entwurf von XMI-Schema:

Um XMI-Schema zu entwickeln muss man einheitliche Prinzipien definieren. Darunter sind die elementare Prinzipien:

#### 1. XML Deklarationen:

Beispiel:

```
<xsd:schema xmlns:xmi="http://www.omg.org/XMI"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  </xsd:schema>
```

Inhalt von Namespace wird hier klar gegeben um Kollision zu verhindern..

#### 2. Klassen eines Metamodells

Klassen werden in XML-Elementen repräsentiert. Und der Name des Elements ist der Klassenname.

#### 3. Extension eines Metamodells

XMI liefert auch den Mechanismus zur Erweiterung eine Klasse.

### Generierung und Struktur von XMI-Schema

XMI-Schema-Productionsregel wird in XMI-Spezifikation festgestellt.

Der Grundregel von Generierung basiert auf die XML-Schema-Generierung. Und Ergebnis ist eine Menge von XML Elementensdefinitionen und Attributendefinitionen. Anhand XML-Namespacespezifikation kann XMI mehrere Metamodelle gleichzeitig nutzen.

Ein XMI-Schema enthält normalerweise solche Elemente:

1. XML Versionierung
2. XML Befehle
3. Schema eines XML Elements
4. Importierte XML Elemente
5. Deklarationen für Metamodelle

Beispiel:

Darunter ist ein UML-Klasse C1 in einem XMI Document. Klasse C1 enthält nur eine Attribute a1.

```
<xmi:XMIversion=„2.0“
  xmlns:UML=„http://schema.omg.org/spec/UML/1.4“
  xmlns:XMI=„http://schema.omg.org/spec/XMI/2.0“>
  <UML:Classname=„C1“>
    <feature xmi:type=„UML:Attribute“
      name=„a1“visibility=„public“/>
  </UML:Class>
</xmi:XMI>
```

## 2.7. Abbildung zwischen MOF-Modell und XMI

Bisher haben wir XML-Schema schon kennengelernt, und wir wissen, dass man durch Schema XML-Dokument prüfen kann. Dann werde ich hier über die Regeln vom Entwurf des XML-Dokuments sprechen.

Die Regeln gelten für von MOF-Modell zu XML und auch von XML zu MOF-Modell. Das heißt, wir mit den Abbildungsregeln die Modelle ins Format von XML umwandeln können und danach aus XML-Dokument wieder erzeugen können. Für das Erstellen von XML-Dokument aus MOF-Modell gibt es eine große Menge von Produktionsregeln. Hier wird nur einige vorgestellt. Die unterliegende Tabelle zeigt diese Abbildung:

XML Construct or Declaration	UML Construct
Element	UML class
Attribute	UML attribute
Nested element	UML association or UML attribute
Text	UML attribute

Abb. 2.3

Darunter ist Codefragment von Klasse Auto (Abb. 2.4):

```
<UML:Class xmi.id = 'sm$18fa85:f9cd1e98da:-7fe6' name = 'Auto'
  visibility = 'public' isSpecification = 'false' isRoot =
  'false' isLeaf = 'false' isAbstract = 'false' isActive =
  'false'>
  <UML:Namespace.ownedElement>
    <UML:Class xmi.id = 'sm$18fa85:f9cd1e98da:-7fc6'
      name = 'String' visibility = 'public'
      isSpecification = 'false' isRoot = 'false'
      isLeaf = 'false' isAbstract = 'false'
      isActive = 'false' />
  </UML:Namespace.ownedElement>
  <UML:Classifier.feature>
    <UML:Attribute xmi.id = 'sm$18fa85:f9cd1e98da:-7fd1'
      name = 'name' visibility = 'public'
      isSpecification = 'false' ownerScope = 'instance'
      changeability = 'changeable'>
    <UML:StructuralFeature.type>
```

```
<UML:Class xmi.idref='sm$18fa85:f9cd1e98da:-7fc6' />
  </UML:StructuralFeature.type>
  </UML:Attribute>
</UML:Classifier.feature>
</UML:Class>
```

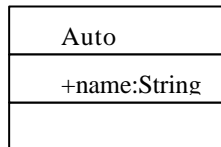


Abb.2.4

Durch das Beispiel werden die Abbildungsregeln von Klasse, Attribute dargestellt. Außer diese gibt es noch andere Beschreibung in ein XML-Dokument, wie z.B Eingebettete Pakete, Abgeleitete Typen und Referenze, usw. Die ausführliche Regeln kann man in XMI-Spezifikation finden.

## 2.8. Diagram Interchange

Ein großer Nachteil von XMI ist, dass durch XMI man Diagraminformation nicht beschreiben kann, z.B wo liegt eine Klasse und wie groß ist sie, usw. Um den Nachteil zu beseitigen hat OMG ein Standard „Diagram Interchange“ als eine Erweiterung von XMI entwickelt. Durch die Feststellung der Beschreibungsweise von z.B Position, Größe, Farbe und Koordinatensystem können jetzt alle Diagraminformation von UML schon in XMI-Dokument enthält werden. Hier werden zwei Methode benutzt, DTD und Style-Sheet. (Siehe Diagram-Interchange-Spezifikation.) Hier ist ein Beispiel (Code-Fragment beschreibt die Diagramstruktur von Klasse Auto, siehe Abb.2.4):

```
.....
<XMI.content>
  <UML:Diagram xmi.id = 'di$100078c:fdb6a1a8d4:-7fa3'
    isVisible = 'true' name = 'Auto' zoom = '1.0'>
    <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>160.0</XMI.field>
      <XMI.field>132.0</XMI.field>
    </UML:GraphNode.size>
    <UML:Diagram.viewport>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
  </UML:Diagram>
</XMI.content>
.....
```

## **2.9. Zusammenfassung von XMI**

Mein Ziel zur Vorstellung von XMI ist nur zu erklären, dass man durch XMI alle Information von MOF-Modell speichern kann und wie die Basisfunktionsweise von XMI ist. Man muss nicht unbedingt die wirkliche XMI-Dokumente lesen oder schreiben kann, da XMI-Dokument relative lang ist wenn auch für eine sehr kleine Klasse. Nötig ist nur zu wissen, dass durch XMI man die Modelle der OMG-Standard(speziell UML-Modelle) speichern und umtauschen kann und es momentan schon viele Werkzeuge vorhanden sind, die XMI unterstützt.

## **3. Literatur:**

- [1] David S.Framkel. Applying MDA to Enterprise Computing. John Wiley & Sons. Inc. 2003
- [2] Anneke Kleppe, Jos Warmer, Wim Bast. MDA Explained: The Model Driven Architecture™: Practice and Promise. Addison Wesley, April 21, 2003
- [3] MOF Spezifikation Version 2.0
- [4] Timothz J.Grose,Gary C.Doney,Stephen A.Brodsky Mastering XMI- Java Programming with XMI, XML und UML, John Wiley & Sons. Inc.
- [5] XMI Spezifikation Version 2.0
- [6] Diagram Interchange for UML2.0 Spezifikation