

XMI – Theorie und Praxis

XMI in 45 Minuten

14.10.04

Eike Frost

Übersicht

- Umfeld (XML, UML)
- XMI Übersicht
- XMI Beispiel
- Implementationen von XMI
- Mögliche Ansätze für P-Umlaut
- Ausblick

Umfeld - XML

- XML (eXtensible Markup Language)
- DTD (Document Type Definition)
- XML Schema
- Sowie weitere vom W3C-Konsortium vorgeschlagene oder verabschiedete Standards (XSL, XPointer, XLink, etc.)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tag>
  <subtag attribute="value">Content</subtag>
  <anothertag>Content</anothertag>
</tag>
```

```
<!ELEMENT tag (subtag, anothertag)>
<!ELEMENT subtag #PCDATA>
<!ATTLIST subtag attribute CDATA #REQUIRED>
<!ELEMENT anothertag #PCDATA>
```

```
<schema xmlns=
"http://www.w3.org/2001/XMLSchema">
  <element name="tag" type="tagtype"/>
  <element name="subtag" type="string"/>
  <element name="anothertag" type="string"/>
  <complexType name="tagtype">
    <sequence>
      <element ref="subtag"/>
      <element ref="anothertag"/>
    </sequence>
  </complexType>
</schema>
```

XML Validation

- XML well-formedness
- XML validity
- DTD / XML Schema gibt Struktur valider Dokumente an
- ggf. auch semantische Informationen (applikationsspezifisch, selten komplett)

Umfeld - UML

- UML (Unified Modelling Language)
 - Modellierung von (Software)-Systemen
 - Grosse Vielfalt an Diagrammtypen und -semantiken
 - Traditionell „nur“ grafisch dargestellt
 - Ständige Weiterentwicklung (UML 1.5/2.0/etc.)
 - Siehe Vorlesung Software Engineering

Was ist XMI ?

- Ein von der OMG (Object Management Group) verabschiedeter Standard
- Ein auf XML basierendes Format zur Speicherung von (Objekt)-Metadaten
- Basiert auf „4-Layer Meta-Model Architecture, MOF
- Nicht ausschliesslich auf UML abgestimmt

Was ist XMI - MOF

- MOF ?

- Meta Object Facility
- OMG Standard
- Beschreibt Information
- Kann auch Meta-Informationen beschreiben
- Oft mit 4 Layern angegeben (allerdings nicht notwendigerweise)

M3 : meta-metamodel

MOF-Model

M2 : meta-metadata/metamodel

UML Metamodel

M1 : metadata/model

UML Model <-- hier interessant

M0 : data

modelliertes System

XMI – MOF Model

- Beschreibt ein MOF-Metamodell (vgl. UML-Metamodell -> UML Modell)
- Klassen, Assoziationen, Packages
- Datatypes, Constraints
- Enger Zusammenhang mit CORBA (IDL-Mappings, etc.)
- XMI-Spezifikation = Austauschformat für MOF-Metadaten

XMI & UML

- UML-Metamodell definiert als MOF-Meta-Model
- Somit ist XMI automatisch auch als Austauschformat für UML zu benutzen
- MOF-Metamodelling-Konzepte sind einfach auf UML-Modelling-Konzepte projizierbar (teilweise gleiche Diagrammtypen, siehe auch ebige Folie)
- Für uns der interessanteste Aspekt

XMI 1.2, 2.0

1.2

- Definiert XMI mittels DTD

2.0

- Definiert XMI mittels XML Schema

- XML Namespaces

Im Grossen und Ganzen äquivalent – lediglich andere Mechanismen zur Validation

XMI & XML

- XMI besteht aus 2 Teilen
 - Format von XMI-gültigen XML DTDs
 - Format von XMI-gültigen XML Streams
- Im Kontext von UML
 - XMI DTD beschreibt das UML-Modell (z.B. Klassen, Attribute, etc.)
 - XMI Stream beschreibt eine Ausprägung dieses Modells (z.B. explizit Objekte, Beziehungen, etc.)

XMI & XML (DTD)

- Einige Elemente fest vorgegeben
- Rest hängt von den modellierten Daten/Modellen ab
- EBNF-Produktionen zur Erzeugung von DTDs aus MOF-Modellen
- XMI-Standard definiert bereits eine DTD für UML 1.x-Dokumente

XMI, XMI.header, XMI.content,
XMI.extensions, XMI.extension,
XMI.documentation, XMI.owner, XMI.contact,
XMI.longDescription, XMI.shortDescription,
XMI.exporter, XMI.exporterVersion,
XMI.exporterID, XMI.notice, XMI.model,
XMI.metamodel, XMI.metametamodel,
XMI.import, XMI.difference, XMI.delete,
XMI.add, XMI.replace, XMI.reference,
XMI.field, XMI.seqItem, XMI.sequencem,
XMI.enum

```
<!ELEMENT XMI (XMI.header?,  
XMI.content?,  
XMI.difference*,  
XMI.extensions*) >  
<!ATTLIST XMI  
xmi.version CDATA #FIXED "1.2"  
timestamp CDATA #IMPLIED  
verified (true | false) #IMPLIED  
>
```

...

XMI & XML (Vorgegebenes)

- In XMI-Dokumente können Referenzen über XLink und XPointer angegeben werden
- Es werden Elemente zur Definition von Änderungen bereitgestellt :
- DTDs und XML Schemas können die Semantik eines XML-Streams nicht zweifelsfrei validieren – hierzu sind zu wenig Informationen vorhanden.
- Jedes modellierte Element muss eine unique ID haben

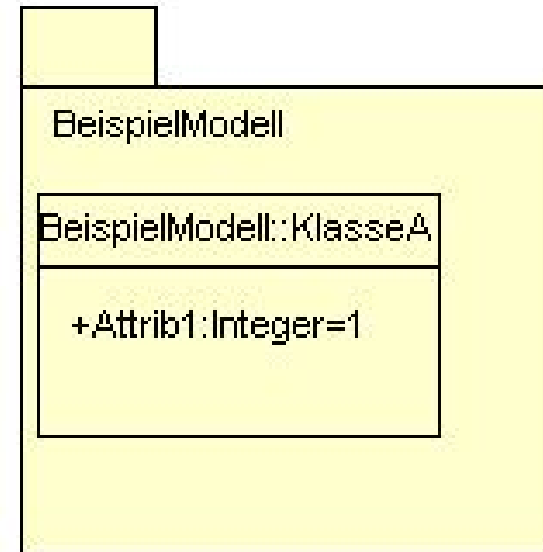
```
<XMI.content>
<XMI.difference href="original.xml">
  <XMI.delete href="original.xml|ccc"/>
  <XMI.add href="original.xml|ppp">
    <Class xmi.label="c2"/>
  </XMI.add>
  <XMI.replace href="original.xml|ppp"/>
  <Package xmi.id="ppp" xmi.label="p2"/>
</XMI.replace>
</XMI.difference>
</XMI.content>
```

XMI & XML (XML Generation)

- Generell 2 Formen der Streamgenerierung
 - Production by Object Containment
 - Production by Package Extent
- Ersteres für UML interessanter
- Wie bei DTD werden EBNF-Regeln zur Überführung eines Modells in einen XML-Stream angegeben
- (theoretisch) entsteht für eine gleiche Modellierung der gleiche XML-Stream

XMI Beispiel

- UML Klassendiagramm
- Package BeispielModell
- Klasse KlasseA
- Attribut Attrib1
- XML-Stream genügt DTC des UML Metamodells



XMI Beispiel - Code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<XMI xmi.version="1.1" xmlns:UML="//org.omg/UML/1.3">
  <XMI.header><XMI.metamodel xmi.name="UML" xmi.version="1.3"/></XMI.header>
  <XMI.content>
    <UML:Model xmi.id="model:BeispielModell" name="BeispielModell"
      visibility="public" isSpecification="false" isRoot="false"
      isLeaf="false" isAbstract="false">
      <UML:Namespace.ownedElement>
        <UML:Package xmi.id="package:BeispielModell" name="BeispielModell"
          visibility="public" isSpecification="false" isRoot="false"
          isLeaf="false" isAbstract="false"
          namespace="model:BeispielModell">
          <UML:Namespace.ownedElement>
            <UML:Class xmi.id="class:KlasseA" name="KlasseA" visibility="public"
              isSpecification="false" isRoot="true" isLeaf="true"
              isAbstract="false" isActive="false"
              namespace="package:BeispielModell">
```

...

XMI Beispiel – Code 2

```
<UML:Classifier.feature>
  <UML:Attribute xmi.id="attribute:Attrib1:class:KlasseA" name="Attrib1"
    visibility="private" isSpecification="false"
    ownerScope="instance" changeability="changeable"
    targetScope="instance" type="dataType:Integer">
    <UML:Attribute.initialValue>
      <UML:Expression language="" body="1"/>
    </UML:Attribute.initialValue>
  </UML:Attribute>
</UML:Classifier.feature>
</UML:Class>
</UML:Namespace.ownedElement>
</UML:Package>
<UML:DataType xmi.id="dataType:Integer" name="Integer" visibility="public"
  isSpecification="false" isRoot="false" isLeaf="false"
  isAbstract="false"/>
</UML:Namespace.ownedElement>
</UML:Model>
</XMI.content>
</XMI>
```

XMI Beispiel - Ausprägung

- Vorheriger XML Stream genügt UML DTD (M2)
- Man könnte für dieses Modell auch eine M1-DTD erzeugen; der XML-Stream wäre dann eine zu diesem Modell konforme Ausprägung.
- DTD hier aus Platzgründen weggelassen; Erzeugung erfolgt programmatisch
- Nächste Folie : Eine zur DTD dieses Modells konforme Ausprägung in XML

XMI Beispiel – Ausprägung 2

```
<?xml version="1.0"?>
<!DOCTYPE XMI SYSTEM "BeispielModell.dtd">
<XMI>
  <XMI.header />
  <XMI.content>
    <BeispielModell>
      <KlasseA>
        <KlasseA.Attrib1> 1 </KlasseA.Attrib1>
      </KlasseA>
    </BeispielModell>
  </XMI.content>
</XMI>
```

XMI Implementationen

- Rational Rose
 - Exportiert XMI[UML] nach UML 1.3 DTD
- IBM XMI Toolkit -> Eclipse Modeling Framework
 - transformiert Rational Rose, Java, XMI[UML], XMI-DTD
- UMMF (Perl)
 - Code Generation, transformation, etc.
- Jede XML-parse-fähige Sprache
- ...

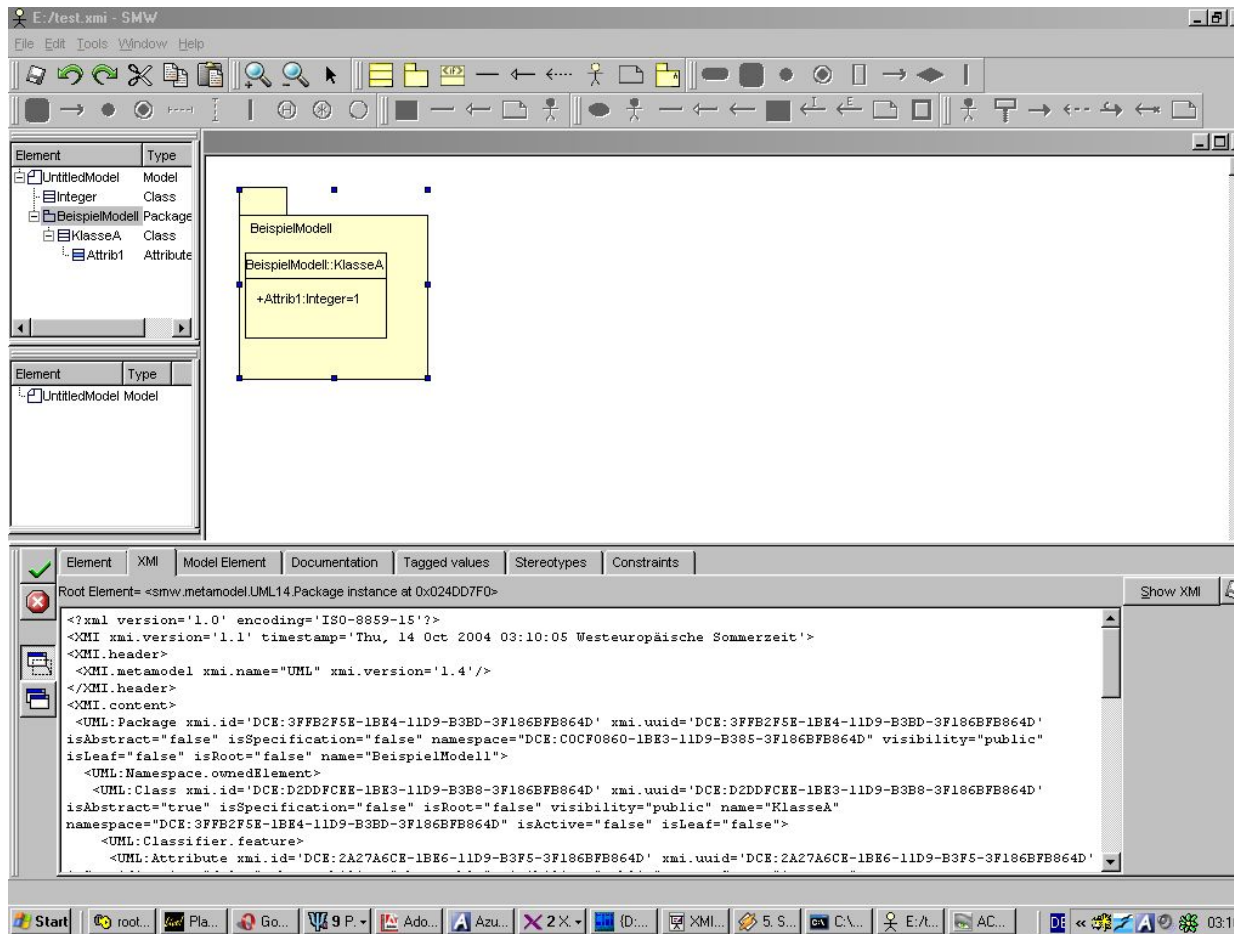
XMI Implementationen - SMW

System Modeling
Workbench

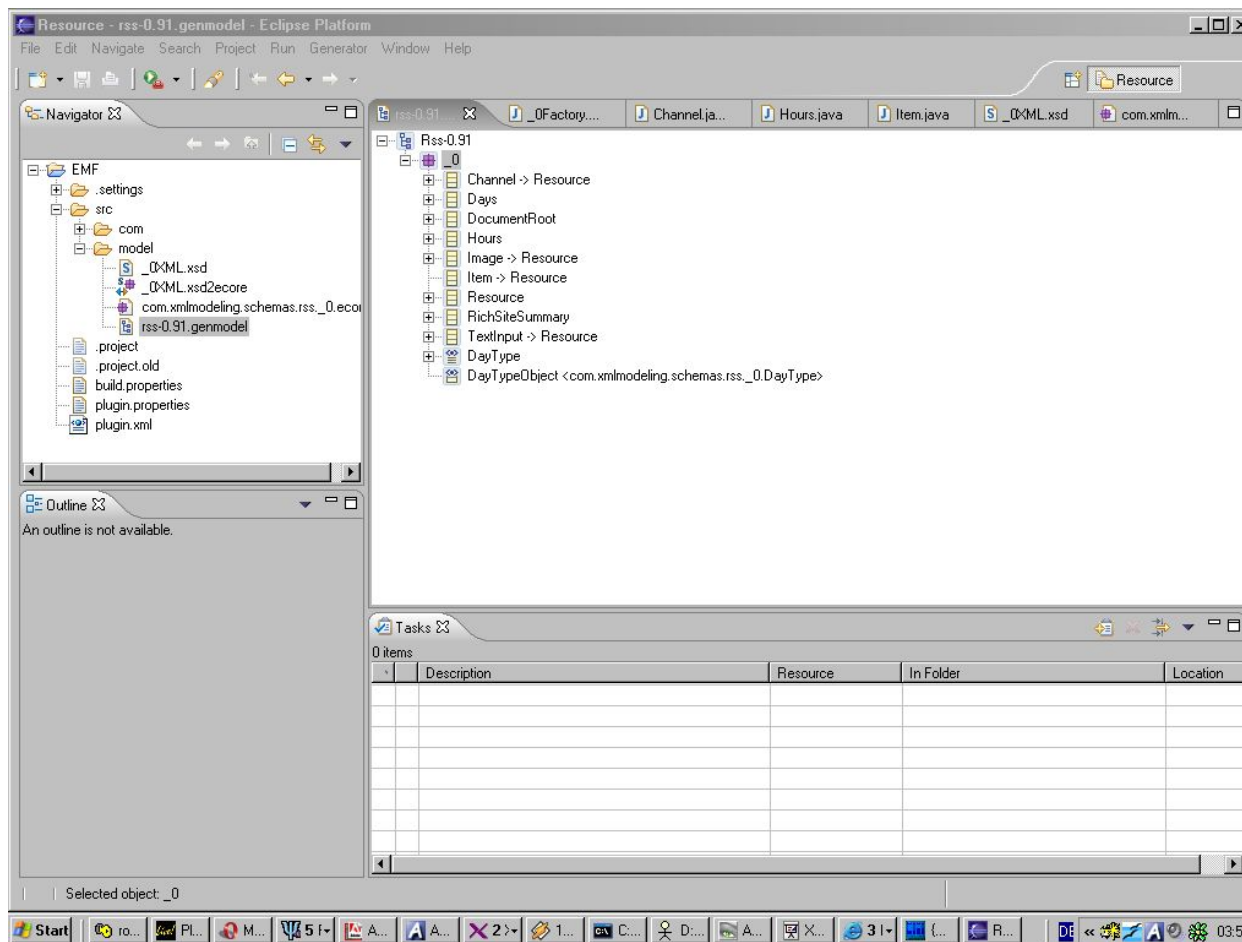
Grafischer UML-Editor

Modellierungswerkzeuge

Direkter XMI-Viewer für
markiertes Objekt



XMI Implementationen - EMF



Geht aus IBM's XMI Toolkit hervor

Java-Codegeneration aus XSD, Rational Rose-MDLs, etc.

vollständige Integration in Eclipse 3.0.1

Generiert auch aus annotiertem Java-Code XMI XML Schemas

Mögliche Ansätze für P-Umlaut

- XMI ist Input für das zu simulierende Modell
- Vor- oder Nachbearbeitung der XML-Streams durch geeignete XSL-Stylesheets
 - XSLT allerdings keine vollständige Skriptsprache
 - Gewisse Transformationen schwierig
 - Vorteil : Es existieren XML-DTDs für Petrinetze

Ansätze continued

- Programmatische Transformation
 - In allen Sprachen mit XMI-Bibliotheken, zur Not reicht aber auch XML
 - Eclipse Modeling Framework (Java)
 - UMMF (Perl)
 - SMW (System Modeling Workbench) (Python)
 - ...

Ausblick

- Zukünftige Modellierungssprachen bringen eigene normative XMI DTD/XML Schema-Definitionen ihres Metamodells mit (-> UML 1.4, 2.x)
- Vielzahl von Tools (de-facto Austauschformat)
- Zwar Provisionen für Differences, jedoch reines Austauschformat (für interaktive Simulation also ohne andere Hilfsmittel ungeeignet)

Abschliessendes

- Folien ab Samstag auf Webpage (jeweils mit Links und Quellen unter den einzelnen Folien)
- Fragen ?
- Diskussion ?