From UML models to XML schemas

Dennis Pagano, Anne Brüggemann-Klein
Technische Universität München

Balisage Conference · Montréal, 8/11/2009

Agenda

- Problem statement
- What is XMI and how can it help?
- XMI unleashed
- Realizing the Layer Shift
- Conclusion and related work

The Setting

- In software engineering we model persistent data of software systems using UML class diagrams
- There are patterns how to map object models to relational database schemas

➡ What happens if the target platform are XML documents?

Problem Statement

- What are data models in an XML world?
- If we persist data as XML documents, the obvious counterparts to data models are XML Schemas

➡ How can we map object models to XML Schemas?

Existing Approaches

- Currently we design object models for XML applications using
  - XML-specific modeling languages and methodologies
  - Graphical editors for schema languages
  - UML profiles for XML Schema

➡ What are problems here?

Problems of Existing Approaches

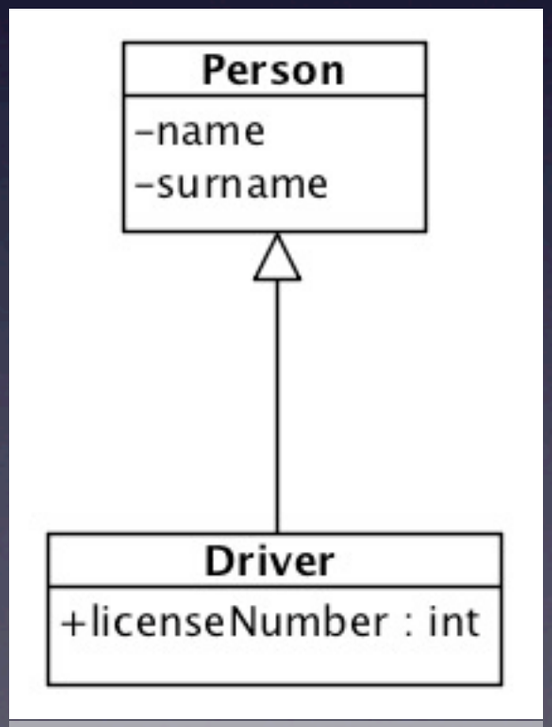
- Major drawbacks:
  - Modeling depends on schema technology
  - Modeling language is close to implementation language
  - Document modeling is separated from general system modeling

➡ We do not want to model XML Schemas

➡ We need a standardized transformation for any object model.

The Goal

- A transformation from UML object models to XML Schema that leaves a model decoupled from its implementation technology




```
<xs:complexType name="Person">
   <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="surname" type="xs:string"/>
   </xs:sequence>
</xs:complexType>

<xs:complexType name="Driver">
   <xs:complexContent>
      <xs:extension base="Person">
         <xs:sequence>
            <xs:element name="licenseNumber" type="xs:int
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

XMI - XML Metadata Interchange

- Standard data format for interchange of models (OMG)
- Aims at interoperability of UML modeling tools
- Supports any MOF compliant model
- Based on XML.

What‘s inside XMI?

- XMI defines a mapping from
  - MOF compliant metamodels to XMI schemas (such as the UML metamodel)
  - Instances of these metamodels to XMI documents (such as an arbitrary UML model)
- XMI schemas are special XML Schemas
- XMI documents are special XML documents

➡ XMI enables validation of models.

How to use XMI?

- UML models are mapped to XML documents?
- From perspective of object persistency they should be mapped to XML Schemas!

➡ Then how can XMI help us?

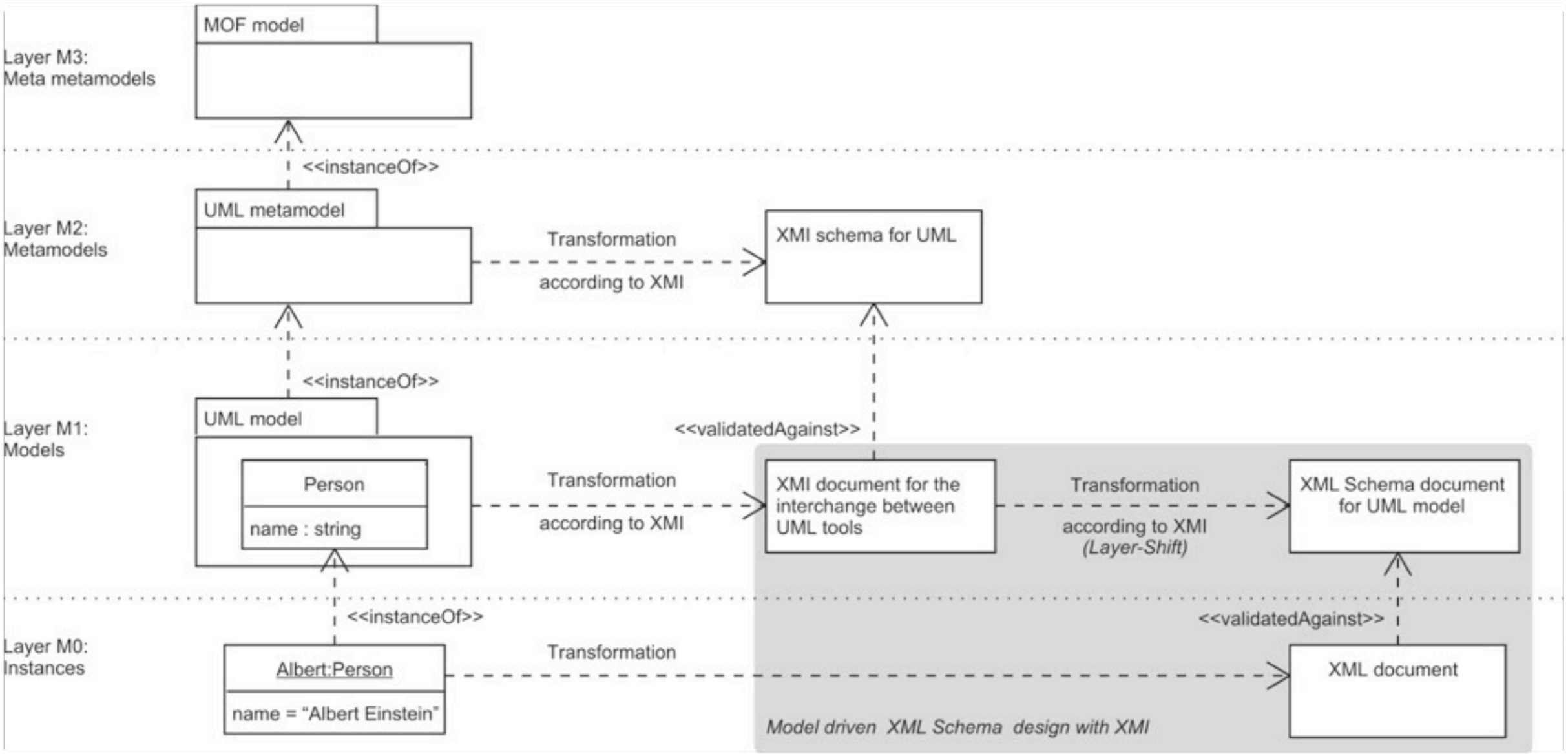
Using the XMI mapping

- UML class diagrams and MOF metamodels are defined using the same concepts (architectural alignment by OMG)

- Thus we may interpret UML class diagrams as MOF compliant metamodels

➡ This allows us to „abuse“ the XMI mapping to generate XMI schemas from conceptual models: *layer shift*.

The layer shift



A closer look

- How does XMI map UML class diagrams to XML Schemas?
- What problems arise?
- Can we configure the mapping?

The XMI mapping

- Packages become elements with locally defined complex types
- Classes become elements with globally defined complex types
- Properties become elements / attributes within a choice in the containing type.

The XMI mapping II

- Packages with different target namespaces are mapped to multiple schemas
- Association ends become elements of type “Any” resp. attributes of type IDREFs
- Copy-down-inheritance.

Major drawbacks

- Element declaration for each class
- Realization of associations
- Content model too generous
- No multiplicities by default
- Inheritance mechanism
- Lack of abstraction.

Tailoring the mapping

- XMI can be configured using tagged values on the model elements themselves
- There are many tags one may use to influence the schema generation process
- nsPrefix, nsURI, element, attribute, sequence, enforceMinimumMultiplicity...

Configuration drawbacks

- Model elements have to be annotated
- This surmises domain knowledge about XML Schema

➡ General set of tagged values reflecting best practice would be desirable.

Realizing the Layer Shift

- We use XSLT to transform XMI documents to XML Schema
- Stylesheet aligned with OMG's translation rules
- Performs the MOF mapping but interprets UML models in particular.

Additional Features

- Substitution groups usable for derived model elements
- Generation of model based name prefixes
- Realization of associations via XLink
- Default tagged values may be set in the parameters
- ...

Conclusions

- Document modeling independent of implementation technology
- Forward engineering for schemas
- Based on OMG standard
- Implementation that works with XMI capable CASE tools

Conclusions II

- CASE tools realize XMI in different ways
- XMI leaves its marks in resulting schemas
- Promising proof of concept, easily extendable to a powerful mechanism
- Could melt into the UML profile approach.

Related Work & Literature

- W.E. Kimber, J.D. Heintz: *Using UML To Define XML Document Types*. Presentation at Extreme Markup Languages 2000.
- D. Carlson: *Modeling XML Applications with UML: Practical E-Business Applications*. Redwood City, Addison Wesley Longman Publishing, 2001.
- A. Brüggemann-Klein, Th. Schöpf, K. Toni: *Principles, Patterns and Procedures of XML Schema Design — Reporting from the XBlog Project*. Extreme Markup Languages 2007
- D. Pagano: *Modeling and defining XML applications with UML and XML Schema*. Diploma Thesis, Technische Universität München, 2008.

Thank you

for your attention!

Backup Slides

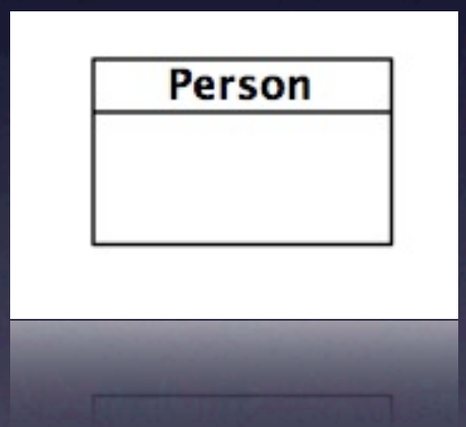
MOF - Meta Object Facility

- The MOF defines a metamodel hierarchy consisting of 4 layers (OMG)
  - The concepts on a layer are the model for the concepts on the layer beneath
  - The UML metamodel e.g. is an instance of the MOF metamodel.

Tagged Values

- In UML a model element can be annotated with tags that have values (e.g. XSDComplexType = “true”)
- Tagged values are part of stereotypes
- Stereotypes are defined within a UML profile.

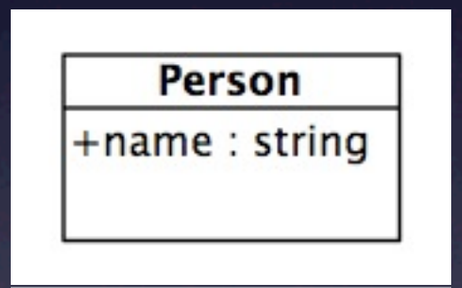
XMI mapping: Classes




```
<xsd:element name="Person" type="Person"/>

<xsd:complexType name="Person">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
         <xsd:element ref="xmi:Extension"/>
      </xsd:choice>
      <xsd:attribute ref="xmi:id"/>
      <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
</xsd:complexType>
```

XMI mapping: Properties




```
<xsd:element name="Person" type="Person"/>

<xsd:complexType name="Person">
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="name" type="xsd:string"/>
         <xsd:element ref="xmi:Extension"/>
      </xsd:choice>
      <xsd:attribute ref="xmi:id"/>
      <xsd:attributeGroup ref="xmi:ObjectAttribs"/>
      <xsd:attribute name="name" type="xsd:string"
                     use="optional"/>
</xsd:complexType>
```