*6th International Conference on Applied Informatics*
*Eger, Hungary, January 27–31, 2004.*

# Standardized interchange of application design models

**Ágnes Papp**

University of Debrecen
e-mail: agi@delfin.unideb.hu

**Abstract**

Unified Modeling Language is a standard for modeling an application that is developed in object-oriented manner. XML is a key enabler of these systems in terms of transport of information. XML schemas are used to define and constrain XML documents. This paper discusses the use of UML in designing XML schemas.

**Keywords:** MOF, UML, XMI, XML schemas, Model Driven Architecture.

## Introduction

Application systems usually maintain difficult data structures. There are several application development tools but data conversion is needed when they want to make accessible their data for each other. UML has been widely accepted as an object oriented analysis and design method. An application-neutral interchange format allows UML models to be interoperable between development tools and developers. The XML is an appropriate format for transferring data via the Internet. The XML based XMI standard allow for different types of applications to interchange their data or models in a standardized way. There is a new way of developing applications, the Model Driven Architecture. The MDA specification consists of a platform-independent UML based model (PIM), and one or more platform-specific models (PSM). The MDA also will take advantage of XMI when it defines the mapping from PIM to XML.

## 1. The UML metamodel architecture

Specifications by OMG [1] summarize principles of data storing and modeling in a four level architecture [2]. The first level is the meta-meta model that defines UML at metamodel level. The second level is the metamodel that describes the

UML syntax. In the third level there are the models created by the users, and in the fourth level there are the object instances or records.

## 1.1. MOF

The MOF (Meta Object Facility) specification [3] includes describing the OMG metamodel concept. The MOF is designed to support many different kinds of meta-information. This is achieved by treating the meta-information as information and formally modeling each distinct kind of meta-information. These formal models are expressed using the metamodeling construct provided by the MOF model [4]. The MOF model is based on the concepts of entity-relationship modeling. The three kinds of elements for a meta-information model are objects, links that connect objects and data values. These constructs are organized as MOF packages. A MOF Class defines the type of an object but not its implementation. A MOF Association defines a class of links between MOF objects. Links are always binary and directed. The third construct in the MOF model is the Package. A MOF Package serves as the unit of modularization and reuse of meta-information models like UML packages.

## 1.2. UML

UML (Unified Modeling Language) [5] is a graphical language for visualizing, specifying, constructing, and documenting the elements of an object system. It has been widely accepted as an object-oriented analysis and design method [6]. UML is created as a union of the previous leading object modeling languages and methods and became standard for object-oriented modeling. It is a graphical language that represents the model by diagrams.

UML specification defines:

- Notation: Graphical notations for the visual representation for model elements.
- Semantics: Semantics for the object modeling concepts.

The UML metamodel defines the language using the following elements. Every kind of diagrams uses combination of them:

- Modeling elements: Structural elements (class, interface, collaboration, use case, component, node), Behavioral elements (interaction, state machine), Grouping elements (package, subsystem), Misc. (note).
- Relationships - Edges connecting modeling elements as nodes: Dependency, Association, Generalization, Realization.
- Extensibility Mechanism - Support for making UML extensible: Stereotype, Tagged value, Constraint.

- Diagrams: Structural Diagrams: Class Diagram, Object Diagram, Behavioral Diagrams: Use Case Diagram, Sequence Diagram, Collaboration diagram, State Transition Diagram, Activity Diagram, Implementation Diagrams: Component Diagram, Deployment Diagram.

While UML defines the constructs above and their interchangeable semantics it does not provide the explicit format to exchange the model information.

## 1.3. XMI

Az XMI (XML Metadata Interchange) [7] is a widely used interchange format for sharing objects using XML (Extensible Markup Language). XMI based standards are in use for integrating tools, repositories, applications and data warehouses. It provides rules by which a schema can be generated for any valid MOF-based metamodel.

XMI defines most important aspects that describe objects in XML [8].

- The representation of objects is made by terms of elements and attributes.
- XMI includes standard mechanism to link objects within the same file or across files.
- Identifying objects allow references from other objects.
- The XMI model handles versioning of objects and their definitions.
- XMI documents can be validated using DTD-s or other schemas.
- Support for XML Schema. The mapping is parameterized, giving users more flexibility when apply the rules. For example, in earlier discussions there was a debate about whether a class attribute should map to an element or attribute in the XML document.

XMI defines production rules:

- Production of DTDs starting from an object model.
- Production of XML documents starting from objects.
- Production of XML Schemas starting from an object model.
- Production of XML Documents compatible with XML Schemas.
- Reverse engineering from XML to an object model.

MOF is the foundation technology for describing object models, which cover the wide range of object domains: analysis (UML), software (Java, C++), components (EJB, IDL, CCM), and databases (CWM). XMI is applicable to all levels of objects and metaobjects.

## 2. UML and XML Shemas

This section discusses the use of UML in designing XML schemas. XML schemas are used to define and constrain XML document instances. An example will be shown to represent the connection between UML class diagrams and XML schema definitions [9]. The first package specification is shown as a UML class diagram. The PurchaseOrder class has two attributes and three associations. A few of these attributes are has [0..1] multiplicity, which means that these attribute values are optional. The Address class plays both a ShipTo and BillTo role in association with the PurchaseOrder class. The multiplicity of 1 means, that PurchaseOrder must have exactly one of each address role. On the Item class there are QuantityType and SKU types. Both types are user-defined data types and defined as other classes in the UML package. (Figure 1)

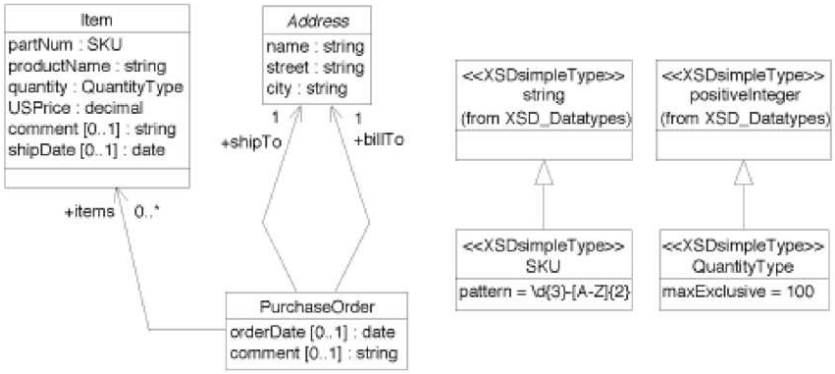


Figure 1. The PurchaseOrder package

In the Address package specification both USAddress and UKAddress are specialized subtypes of the Address class. The class name Address is shown in italics, which means that it is an abstract class. (Figure 2)

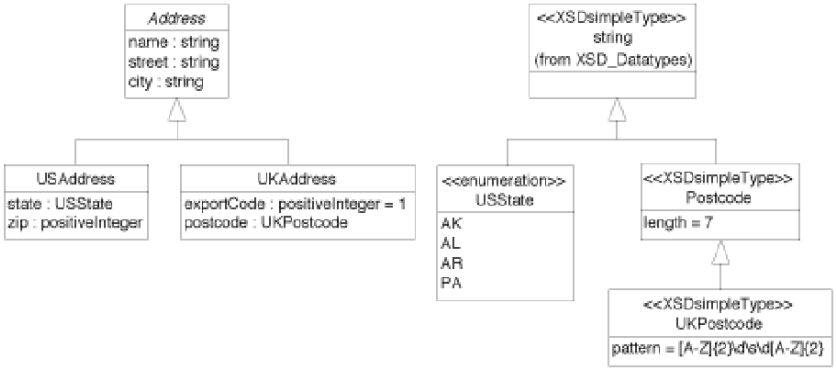


Figure 2. The Address package

We need a mapping specification between UML class diagrams and XML schema definitions, in this example the W3C XML Schema Definition Language [10] (XSD).

Goals of mapping specification from UML to XML schemas are:

- to allow sufficient flexibility to accomplish most schema design requirements
- smooth transition from conceptual model to a detailed design model
- XML schemas can be generated from any UML class diagram
- reuse of the model in different deployment languages and environments.

The first schema definition is produced using the default mapping rules. These defaults are aligned with XMI specification for using XML as a model interchange format. A class in UML defines a complete data structure that maps by default to a complexType in XSD.

- <xs:all> element is used to create an unordered model group.
- A UML class creates a distinct namespace for its attribute names. The attributes in UML classes are produced as local element definitions.
- When a UML attribute is optional it is expressed with minOccurs and maxOccurs attributes in the XSD.
- Primitive data types from the XSD specification can be written directly to the generated schema.
- A top level element is created for each complexType in the schema. The default name for this element is the name of the class.

The associations are also added to the complexType XSD elements. Each association has role name and multiplicity.

- The default mapping for associations creates a wrapper element in XSD with the role name in UML.
- This element contains the instances of the associated class, which the schema refers to.

```
<xs:complexType name="PurchaseOrder">
  <xs:all>
    <xs:element name="orderDate" type="xs:date" minOccurs="0" maxOccurs="1"/>
    <xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="1"/>
    <xs:element name="shipTo">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Address"/>
        </xs:sequence>
      </xs:complexType>
```

```
    </xs:element>
    <xs:element name="billTo">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Address"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="items" minOccurs="0" maxOccurs="1">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="Item" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:all>
</xs:complexType>
```

The default mapping to XSD would produce a complexType definition for SKU and QuantityType , but we want these to become user-defined simple data types. This is easily achieved by addig the «XSDsimpleType» stereotype to these two classes.

```
<xs:simpleType name="SKU">
  <xs:annotation>
    <xs:documentation>Stock Keeping Unit, a code
         for identifying products</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-[A-Z]{2}"/>
  </xs:restriction>
</xs:simpleType>
```

A fundamental concept in object-oriented analysis and design is generalization from one class to another. The specialized subclass inherits attributes and associations from its parent class. This is easily presented in W3C XML schema. The complexType definitions for Address and USAddress are produced as follows.

- The top-level element and complexType definitions for Address include the XSD attribute abstract=“true”.
- The USAddress element includes substitutionGroup=“Address”, which means that whenever the Address element is required as a content element, then USAddress may be substituted in its place.

```
<xs:element name="Address" type="Address" abstract="true"/>
<xs:complexType name="Address" abstract="true">
  <xs:all>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="street" type="xs:string"/>
    <xs:element name="city" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

```
<xs:element name="USAddress" type="USAddress"
            substitutionGroup="Address"/>
<xs:complexType name="USAddress">
  <xs:complexContent>
    <xs:extension base="Address">
      <xs:all>
        <xs:element name="state" type="USState"/>
        <xs:element name="zip" type="xs:positiveInteger"/>
      </xs:all>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The next step is refining the conceptual model to a design model specialized for XML schema. This is achieved by adding stereotypes and properties, which are based on a customization profile for UML. A UML profile has three key items: stereotypes, tagged values (properties) and constraints. A profile provides a definition of these items and explains how they extend the UML in a particular domain, in this case XML schema design. A stereotype assigned to a UML class extends the meaning of a class definition in the model.

Three stereotypes from UML Profile for XML Schemas are summarized as follows: «XSDcomplexType» on a UML class

- modelGroup ( all | sequence | choice )
- attributeMapping ( element | attribute )
- roleMapping ( element | attribute )
- elementNameMapping ( upperCamelCase | lowerCamelCase | . . . )

«XSDelement» on UML attribute or association end

- position ( integer value ) within a sequence model group
- anonymousType ( true | false )

«XSDattribute» on UML attribute or association end

- use ( prohibited | optional | required | fixed )

Other stereotypes e.g. «XSDsimpleType» or «XSDfacet» can be used without adding properties. Many of the profile property can be set as defaults for an entire UML model or for a package in the model.

Design requirements divided into the next categories:

- Should the attributes of a UML class be produced as XML attributes or child elements in the schema?
- Which kind of model group (all, sequence or choice) should be used to validate an element content?

- Should we choose to include or exclude XML element tags that represent class names and roles in the UML associations?
- How do we map UML class name to XML element names?

The PurchoseOrder class diagram now includes profile extensions that resolve these design choices. (Figure 3)

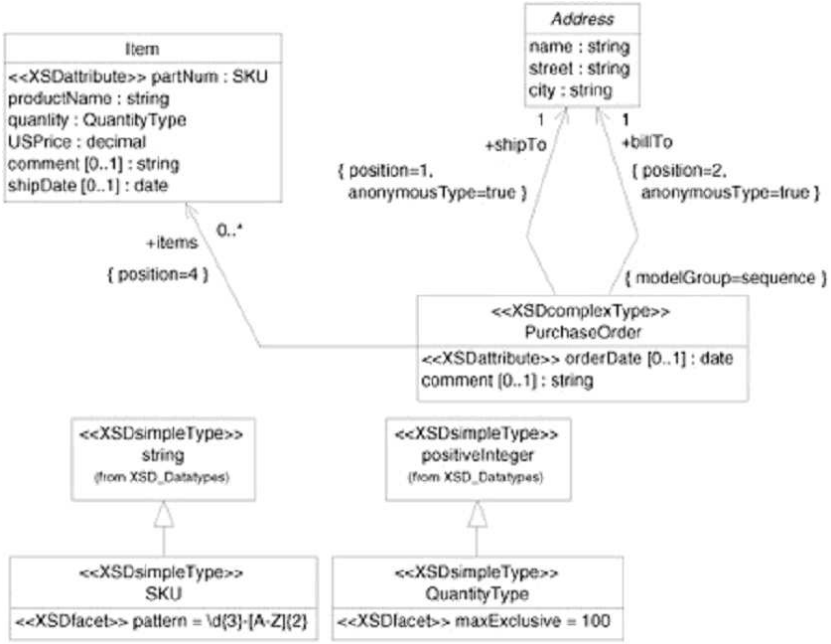


Figure 3. The PurchoseOrder package with profile extensions

The following schema produced for the PurchoseOrder class and its associations:

```
<xs:element name="purchaseOrder" type="ipo:PurchaseOrder"/>
   <xs:complexType name="PurchaseOrder">
      <xs:sequence>
         <xs:element name="shipTo" type="ipo:Address"/>
         <xs:element name="billTo" type="ipo:Address"/>
         <xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="1"/>
         <xs:element name="items" minOccurs="0" maxOccurs="1">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="ipo:item" minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="orderDate" type="xs:date"/>
   </xs:complexType>
```

- By assigning «XSDattribute» stereotype on the orderDate attribute in UML, we specify that it should be represented as an attribute in XML. The comment UML attribute follows the default mapping to an element in the schema.
- We assigned the «XSDcomplexType» stereotype to the PurchoseOrder class and set the modelGroup property to "sequence". The default mapping uses <all>. Each UML attribute and association end must be annotated with a profile property that specifies its position.
- The «XSDelement» stereotype is assigned to the association ends connected to the Address class and the anonymousType property is set to "true". It means that the instance document omits the Address tag and embeds its element and attribute content directly within the role tag.
- Because the item role on the association is not specified as an anonymousType, its definition in the schema retains the role container element to hold elements for the related class.
- The default mapping from UML creates element names equal to the class names. By adding the elementNameMapping property to a UML class along with the «XSDcomplexType» stereotype can be set other name convention.

There are different types of UML modeling tools that create XML schemas for UML models. Many UML tools support the standard XMI format as an import-export format and some use it as their native file format. When an XMI file is imported it is transformed into other representations e.g. HTML.

## 3. The Model Driven Architecture

There is a new way of developing applications, the Model Driven Architecture [11]. The MDA specification consists of a platform-independent UML based model (PIM), and one or more platform-specific models (PSM). With MDA, an application system is modeled once and only once. The MDA also will take advantage of XMI when it defines the mapping from PIM to XML.

In MDA there is a distinction between application architecture and systems architecture [12]. Application architecture includes the components and structural relationships that specify the functional purpose of the application. Systems architecture consists of the lower level components and structural relationships that allow the application architecture to execute. This separation is fundamental for MDA. A PIM is a complete application specification that is independent of technology platforms. A PIM is mapped onto a PMS, which provide the systems architecture infrastructure. This mapping implements the PIM on a specific platform, turning it into an executable application. The PIM let us model a solution visually at higher level of abstraction. Re-writing applications when a new technology comes will be unnecessary. We just simply regenerate the application using a mapping into the newer environment.

There are standard PIM to PSM mappings for many of the popular technology platforms such as COBRA, J2EE, .NET. OMG standards - MOF, XML, XMI and CWM - work in concert to make the MDA a complete approach to software development.

# References

[1] Object Management Group http://www.omg.org
[2] http://protege.stanford.edu/plugins/xmi/background.html
[3] Meta Object Facility http://www.omg.org/technology/documents/formal/mof.htm
[4] http://www.iam.unibe.ch/ schlpbch/XMIforMoose/reportHTMLized/node3.html
[5] Unified Modeling Language
http://www.omg.org/technology/documents/formal/uml.htm
[6] Vég Csaba: Alkalmazásfejlesztés az UML szabványos jelöléseivel, Logos 2000, 1999
[7] XML Metadata Interchange
http://www.omg.org/technology/documents/formal/xmi.htm
[8] UML to XML Design Rules Project
http://xml.coverpages.org/uml2xmlDesignRules.html
[9] Modeling XML Vocabularies with UML: Part I, II, III by Dave Carlson
http://www.xml.com/pub/a/2001/08/22/uml.html
[10] W3C XML Schema Definition Language http://www.w3.org/XML/Schema
[11] OMG Model Driven Architecture http://www.omg.org/mda
[12] Executable UML: Diagrams for the Future http://www.devx.com/uml/Article/10717

# Postal address

**Ágnes Papp**
*Health College*
*University of Debrecen*
*4027 Debrecen*
*Egyetem sgt. 13.*
*Hungary*