Design XML schemas using UML

Translating business concepts into XML vocabularies

Ayesha Malik

February 01, 2003

Unified Modeling Language (UML) is an industry standard that is used in modeling business concepts when building software systems in an object-oriented manner. Recently, XML has gained ground in becoming a key enabler of these systems in terms of transport of information and commands. XML schemas, which are used to define and constrain the nature of XML exchanged, have consequently come into the limelight. This article discusses the use of UML in designing XML schemas and gives a hands-on approach for using the UML framework to create your XML vocabularies.

When using the UML framework for constructing XML schemas, you must consider three issues:

- The complementarities between UML and XML schemas
- How to extend UML to capture all the functionalities provided by schemas
- The ability to engineer XML schemas from UML diagrams

To help discuss the two frameworks in this context, I will use an example of a fictitious company: BALTIC Shipping.

**BALTIC Shipping** is an international shipping company that specializes in transporting shipments from the U.S. to Eastern Europe. It seeks to create a mechanism for tracking shipments from its headquarters in New York to its regional offices such as the one in Tallinn, Estonia (see Figure 1). When products are shipped, the head office sends information electronically in XML about the shipment. Once the shipment has reached its destination, the confirmation is electronically sent back to headquarters.

All the order and confirmation data is exchanged in XML documents and schemas have to be designed to outline the structure of the documents. The business constructs used to model shipping orders are also used to exchange information with the Inventory Tracking System which knows which packages the company is holding for delivery at any time. This article discusses the efficacy of using UML when constructing the XML schemas that define these business constructs for data transport in XML.
Complementary frameworks

UML and its object-oriented modeling can be complementary to building XML schemas. You can easily represent business concepts with the graphical notation in UML and begin designing your XML schemas.

The value of modeling

A discussion on the advantages of UML when creating XML schemas presumes that the value of object-oriented modeling is a given. In my last article, "Create flexible and extensible XML schemas," I discussed the importance and value of building XML schemas using an object-oriented approach. Apart from the technical advantages of using UML to design an object-oriented system, UML provides a common medium in which the business and technical teams can easily communicate ideas. Business analysts are key collaborators in a software system -- particularly one that contains domain-specific information. Since business analysts are involved in the process of designing XML documents, the ease with which the software architect and the business analyst can collaborate becomes important for a successful project. UML's graphical notation makes it easy for technical and non-technical people to agree on business concepts, such as the definition of a Shipping Order, and therefore expedites and facilitates the completion of the project.

Complementarities

Imagine that the business manager of BALTIC Shipping comes to you and asks you to model the XML schema that will formalize the information that is transmitted between different systems in the company. He sits down with you to discuss the business concepts of the domain. You could make some rough sketches on paper, but UML provides a better formal methodology for modeling these concepts with diagrams and notations.
In the UML diagram in Figure 2, the business definition of a Shipping Order is outlined. BALTIC Shipping defines a Shipping Order as consisting of a ShippingId, an Origin, a Destination, and an Order. It considers this imperative information whenever any data regarding a Shipping Order is exchanged. In addition, the UML diagram is used to represent what constitutes an Origin or an Order. Origin and Destination types are shown to be the same as type Address, and BALTIC Shipping stores an Address in its database with the following characteristics: Name, Street, City, and Country. These are business concepts and they have been used in the database models, in the software programs, and in the documents that are read by managers and business partners. These concepts also include cardinality (an Order can consist of many Items), inheritance (Origin inherits all the characteristics of an Address), and dependency relationships (an Order depends on the details of its Items); all of these relations are captured by the UML diagram. Since you want your XML documents to carry the Shipping Order information, the next step is to design XML schemas that conform to the sketched UML diagrams. The following schemas represent the mapping of the UML diagram (see Figure 2) to XML schemas.
Listing 1. ShippingOrder.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:include schemaLocation="DataTypes.xsd"/>
  <xs:element name="shippingOrder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="shippingId" type="int"/>
        <xs:element name="origin" type="Origin"/>
        <xs:element name="destination" type="Destination"/>
        <xs:element name="order" type="Order"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

As you can see in Listing 1, the Shipping Order class in UML is represented by the complex type shippingOrder in the schema. As the business recommended, a Shipping Order consists of a shippingId, an Origin, a Destination, and an Order. One thing to note is that I put the Origin type along with other generic types in a DataTypes schema (see Listing 2). A DataTypes library is convenient for storing reusable types, such as the definition of an Address, which are used in XML documents across the firm for different projects.

In the UML diagram (see Figure 2), Address is an abstract type as indicated by the italics in which the word "Address" is written. The types Origin and Destination inherit the characteristics Name, Street, City, and Country from Address. Creating blueprints for reusable types is considered good object-oriented design. In XML schemas (see Listing 2), I have put the type Address as an abstract type by using the keywords abstract="true". The types Origin and Destination mimic the design I originally outlined in my UML diagram when I use extension base="Address" to show that they inherit the characteristics of Address. In addition, I have captured the business model that an Order can consist of many Items with the code type="Item" maxOccurs="unbounded".

Listing 2. DataTypes.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:complexType name="Address" abstract="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Origin">
    <xs:complexContent>
      <xs:extension base="Address"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Destination">
    <xs:complexContent>
      <xs:extension base="Address"/>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```
If you had started designing the XML schema from scratch, it would have been difficult to just write down the object types in XML. In addition, it would be nearly impossible to explain them to the business manager who is not familiar with the terminology of XML schemas. Based on the UML diagrams, you could effectively translate the business concepts of the company and then create the XML schemas with this visual representation in front of you. The following is a resulting instance document for a parcel of strawberry jam that is shipped from New York to Tallinn given the schemas you created.

**Listing 3. ShippingOrder.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<shippingOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\schemas\ShippingOrder.xsd">
  <shippingId>09887</shippingId>
  <origin>
    <name>Ayesha Malik</name>
    <street>100 Wall Street</street>
    <city>New York</city>
    <country>USA</country>
  </origin>
  <destination>
    <name>Mai Madar</name>
    <street>Liivalaia 33</street>
    <city>Tallinn</city>
    <country>Estonia</country>
  </destination>
  <order>
    <item>
      <description>Ten Strawberry Jam bottles</description>
      <weight>3.141</weight>
      <tax>7.60</tax>
    </item>
  </order>
</shippingOrder>
```

**Fundamentals**

While it's easy to see the advantages of using UML in this example, does the BALTIC Shipping example really typify the complementarities between UML and XML schemas? I have already shown that XML schemas can capture the information represented in a UML diagram. But now let's turn our analysis on its head and ask whether UML diagrams can capture all the functionalities provided by XML schemas. XML schemas provide a great deal of richness in formalizing XML documents and a preliminary analysis shows that most of the fundamental functionalities of XML schemas can be represented by UML diagrams. Consider some important tenets of XML schemas:
1. Types
XML schemas have both built-in data types, such as int and double, and types that are constructed, such as a complex type representing the components of an Address. The generation of types is an important facet of object-oriented design and is necessary for modularity, flexibility, and encapsulation. In UML, built-in data types are present and new types can be built using the structure of classes.

XML schemas also have user-defined types which, according to the W3C, "allow creation of user-defined data types, such as data types that are derived from existing data types and which may constrain certain of its properties (e.g., range, precision, length, format)." For special or user defined types, UML allows extensions, known as stereotypes, to its basic profile. Stereotypes are discussed at length in Extend UML with stereotypes.

2. Attributes
Attributes in XML schemas are useful for two things:

- To capture associations. For instance, an order can contain many Items. This is written as maxOccurs="unbounded" in an XML schema. In UML, an association relates two or more classes in a model and is indicated by an arrow on one end along with a number implying the multiplicity of that association.
- To show additional information that may be linked intrinsically to an element. One example of such an element and attribute pairing in an XML schema is the tax element with a currency attribute. An example of adding the attribute to the element tax is shown in Listing 4.

Listing 4. Adding attributes

```xml
<xs:element name="tax">  
  <xs:complexType>  
    <xs:simpleContent>  
      <xs:extension base="xs:double">  
        <xs:attribute name="currency" type="xs:string" use="required"/>  
      </xs:extension>  
    </xs:simpleContent>  
  </xs:complexType>  
</xs:element>
```

An attribute can be captured in the UML diagram as an attribute of the class. However, it is necessary to create a mechanism by which an element and an attribute can be differentiated in a UML class diagram. This is achieved by creating a special prototype called attribute in the UML diagram.

3. Namespaces
Namespaces are a very important concept in XML schemas. They represent a separation of business concepts into buckets. In UML diagrams, namespaces can be represented by keeping the UML modules in different packages.

4. Multiplicity
I have already shown that associations such as the one between order and Item (an order can contain several Items) can be shown in a UML diagram. In XML schemas, the maxOccurs
and minOccurs keywords are used to show the multiplicity association with an element or an association.

5. Deriving types
XML schemas allow types to be derived by extension and by restriction. Inheritance by extension is a common object-oriented design and is easily expressed through showing abstract classes in UML. Abstract classes are written in italics (see Figure 2). However, there is no construct in UML to represent derivation by restriction.

Gap analysis
It is clear that UML fails to capture all of the richness in XML schemas. This means that you should manually scan and tweak the XML schemas even after they have been modeled in UML. Some issues that you should address are:

- **Ordering:** In XML schemas, ordering matters whereas the traditional domain of UML does not have any use for ordering.
- **Attributes:** As discussed earlier in Fundamentals, attributes are not clearly demarcated from child elements in a UML diagram.
- **Derivation by restriction:** While UML can show abstract classes and their implementation, the traditional UML diagrams have no formal notation for showing restriction which is part of the XML schema generalization techniques.
- **Keys:** Keys are used to link documents together in XML schemas, however key representation is missing in UML notation.

The gap analysis points to the shortfalls of UML notation when capturing all of the functionalities provided by XML schemas. UML profiles provide a generic extension mechanism for building UML models in particular domains. In the domain of XML schema design, the UML profile is extended to make up for the gaps in the basic UML model. Dave Carlson has already created one such XSD profile in his book, *Modeling XML Applications with UML*.

Extend UML with stereotypes

**Gap analysis** lists areas where UML was unable to clearly capture what you wanted in your XML schema and it was decided to extend the UML profile to fill the gap. A UML profile has three key items: stereotypes, tagged values (properties), and constraints. A stereotype allows you to attach a new meaning to a UML foundation class, such as Class, and it is represented on a UML diagram as a name surrounded by double brackets (<< >>).

Suppose you have a business requirement that the tax amount must carry information regarding the denomination of its currency as well. In other words, the XML looks like this:

```xml
<tax currency=USD>7.60</tax>
```

In XML schemas, the tax element carries the attribute currency to represent this requirement (see Listing 4). To create a corresponding diagram in UML, two new stereotypes are created: <<xsd:attribute>> to represent an XML schema attribute, and <<xsd:simpleType>> to represent a
double. I have used the XSD extension to the UML profile created by Dave Carlson. The extension to represent an attribute is shown here:

```xml
<<XSDAttribute>> on a UML attribute or association end
use (prohibited | optional | required | fixed)
```

The `complexType` `Tax` is based on the `simpleType` `double` and contains the attribute `currency`. By making the stereotypes, you have now accurately captured the new business requirements in the UML diagram and can again use the UML diagram to create the XML schema. Just as I made an attribute `currency` for the element `tax`, I can make an attribute `unit` for the element `weight` in the UML diagram using my `<<XSDAttribute>>` stereotype. In similar vein, UML extensions need to be created for many of the XML schema constructs to provide complete bi-directional mapping between the two frameworks.

**Figure 3. UML diagram with stereotypes**

The corresponding schemas are below:
Listing 5. ShippingOrder.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:include schemaLocation="DataTypes2.xsd"/>
  <xs:element name="shippingOrder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="shippingId"/>
        <xs:element name="origin" type="Origin"/>
        <xs:element name="destination" type="Destination"/>
        <xs:element name="order" type="Order"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 6. DataTypes2.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:complexType name="Order">
    <xs:sequence>
      <xs:element name="item" type="Item" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Item">
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="weight" type="Weight"/>
      <xs:element name="tax" type="Tax"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Address" abstract="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Origin">
    <xs:complexContent>
      <xs:extension base="Address"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Destination">
    <xs:complexContent>
      <xs:extension base="Address"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Tax">
    <xs:simpleContent>
      <xs:extension base="xs:double">
        <xs:attribute name="currency" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="Weight">
    <xs:simpleContent>
      <xs:extension base="xs:double"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```
The XML instance document now contains the currency each time it exchanges information about the tax on an item.

Listing 7. ShippingOrder2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<shippingOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\schemas\ShippingOrder.xsd">
  <shippingId>09887</shippingId>
  <origin>
    <name>Ayesha Malik</name>
    <street>100 Wall Street</street>
    <city>New York</city>
    <country>USA</country>
  </origin>
  <destination>
    <name>Mai Madar</name>
    <street>Liivalaia 33</street>
    <city>Tallinn</city>
    <country>Estonia</country>
  </destination>
  <order>
    <item>
      <description>Ten Strawberry Jam bottles</description>
      <weight unit="kg">3.14</weight>
      <tax currency="US">7.16</tax>
    </item>
  </order>
</shippingOrder>
```

Generate XML schemas from UML models

It naturally follows from the discussion on UML and XML schemas to ask whether there is a standard way of mapping UML to XML schemas. If so, automatic generation of schemas from UML diagrams is possible and vendor and open source tools can provide this functionality. Ideally, a tool takes the UML meta model and converts it to the XML schema. XML Metadata Interchange (XMI) was designed to enable easy interchange of metadata between modeling tools and can be further used to generate XML DTDs from UML diagrams using a set of conversion rules.

Using XMI (XML Metadata Interchange)

XMI is sponsored by the vendor-neutral Object Management Group (OMG) and has three main standards:

- **XML**: eXtensible Markup Language, a W3C standard
- **UML**: Unified Modeling Language, an OMG modeling standard
- **MOF**: Meta Object Facility, an OMG modeling and metadata repository standard

By combining these three standards, XMI represents a new way of transferring metadata from one repository to another.
I've included a little snippet of XMI to illustrate what happens at the back end when the tool automatically generates the meta information in XML and then translates it into XML schema. In Listing 8, the Foundation Core Model Element is `ShippingOrder` and you can see that XMI, which is actually fairly verbose, writes each one of the leaves that are present in the UML model.

**Listing 8. Part of the XMI generated by hyperModel for Figure 2 UML model (created in ArgoUML)**

```xml
<Foundation.Core.ModelElement.name>ShippingOrder</Foundation.Core.ModelElement.name>
<Foundation.Core.ModelElement.isSpecification xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isRoot xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
```

Most models, including TogetherJ, Rational Rose, and ArgoUML, create XMI files. I used ArgoUML (which is open source) to create the UML diagram in Figure 2. ArgoUML created a project file that included an XMI file in it. When imported into hyperModel (developed by Dave Carlson, who began the discussion on modeling XML schemas using UML and wrote a book on the subject), it generated the appropriate schema constructs. I downloaded the 30-day evaluation model for hyperModel but you can choose another tool. Just make sure that your encoding for XML and the UML model is correct; for example, hyperModel works with UML Model 1.3. Also, always check the XML schema generated to validate that the conversion was accurate, particularly if new types are used.

The XMI generated by ArgoUML above and put in hyperModel returned the XML schema constructs in Listing 9. On close inspection, it can be confirmed that the basic UML model in ArgoUML in Figure 2 was captured by hyperModel. Since the UML was in a single package, the XML schema generated is in a single namespace. Types such as `Origin` are referenced when called under `ShippingOrderType`.

**Listing 9. ShippingOrder element type generated by using XMI**

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <!-- Class: ShippingOrderType  -->
  <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
  <xs:element name="ShippingOrder" type="ShippingOrderType"/>
  <xs:complexType name="ShippingOrderType">
    <xs:sequence>
      <xs:element name="shippingId" type="xs:int"/>
      <xs:element name="origin">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="Origin"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="destination">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="Destination"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
### Summary

Many large conglomerates -- such as SWIFT, which provides the electronic infrastructure for trading and settlement for 7,000 financial institutions around the world -- are using UML-to-XML schema conversion to design their XML documents. UML represents the easiest way of modeling business concepts, especially when they are domain-specific. It is natural to want to extrapolate and automate the process so that the transformation is clean and complete. For this purpose, I have discussed the use of XMI and the ability of products such as hyperModel to generate the XML schema from the XMI describing the UML meta model. However, the reader is cautioned to always refine and double-check the validity of the model. Even though the ability to completely map UML to XML schemas has not yet been perfected, UML is a good way to start the modeling of XML schemas in an object-oriented manner. If the trend towards creating tools -- both open source and vendor managed -- for automatic generation of XML schemas continues, UML class diagrams might become a standard way of incorporating business concepts into XML vocabulary.

As XML becomes intrinsic to all parts of a software system -- from data exchange to Web services messages to description of build scripts -- a clean, concise way of modeling XML schemas becomes imperative. UML is a tried and tested modeling tool for object-oriented systems, and it is attractive for developers, business analysts, and vendors as a medium for designing XML schemas. I believe we will see increasing use of UML as industries and consumers begin to develop their ontologies and services using XML.
Related topics

- Check out the site on XML modeling which discusses many applications of UML-to-XML modeling.
- Read Dave Carlson's three-part series of articles on "Modeling XML Vocabularies with UML."
- Explore Will Provost's article on XML.com, "UML for W3C Schema Design."
- Learn how one organization uses UML and XMI to create software in "XMI and UML combine to drive product development" by Cameron Laird (developerWorks, October 2001).
- IBM trial software for product evaluation: Build your next project with trial software available for download directly from developerWorks, including application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Find out how you can become an IBM Certified Developer.