UML for XML Schema Mapping Specification

12/08/99

Grady Booch (Rational Software Corp.)
Magnus Christerson (Rational Software Corp.)
Matthew Fuchs (CommerceOne Inc.)
Jari Koistinen (CommerceOne Inc.)

Abstract
This paper describes a graphical notation in UML for designing XML Schemas. UML (Unified Modeling Language) is a standard object-oriented design language that has gained virtually global acceptance among both tool vendors as well as software developers. UML has been standardized by the Object Management Group (OMG). XML Schema is an emerging standard from W3C. XML Schema is a language for defining the structure of XML document instances that belong to a specific document type. XML Schema can be seen as replacing the XML DTD syntax. XML Schema provides strong data typing, modularization and reuse mechanisms not available in XML DTDs. There is currently no W3C recommendation for XML Schema, although several have been proposed and W3C is actively working on producing a recommendation. This paper describes the relationship between UML and the SOX schema used by CommerceOne. Our intention is, however, to adapt the mapping to the W3C recommendation when that becomes available. W3C discussions up to this point indicate the notation described here will be upward compatible with the eventual recommendation.

1. INTRODUCTION

XML is rapidly establishing itself as the metagrammar for interorganizational communication around the Internet. It is becoming increasingly urgent that business analysts, systems analysts, and software developers be able to:

- model the information to be represented in XML.
- describe the relationships between the XML and the systems to process it.

Having done so, they must also be able to rapidly generate the boilerplate code associated with implementing these processes.

At present there is no tool or tool suite capable of doing this. One path to development is to exploit existing tools using UML to facilitate this. The first step towards doing so is providing a semantically rich mapping from XML into UML. The goal of this paper is to layout such a mapping through XML Schema, a schema language for object-oriented XML. This paper itself does not provide all the information for an end-to-end mapping from UML to XML Schema to programming language-specific data structures, but but such a mapping can be built on the information presented here.

In the immediate, the mapping described in this document serves as a straw man for further discussion. Although we refer to XML Schema in the paper, we are designing the mapping specifically to SOX until a W3C XML Schema recommendation becomes available.

1.1 XML Schema and UML

In developing the mapping between XML Schema and UML we have used the UML extension mechanisms (stereotypes and tagged values) to create new classes of UML objects to explicitly represent XML artifacts. The alternative approach would have been to specify a general mapping from UML classes to XML Schema. Such a mapping would have been applicable to a range of existing UML models. We chose to extend UML for the following reasons:

1. The extension approach allows users to directly model XML Schema in UML in an unambiguous way.
2. An explicit mapping makes it easier to write tools to handle only the XML content of a model and to clearly differentiate XML components from other aspects of a model.
3. Given an existing UML model, there are several issues related to mapping it into XML, including choosing which parts to map, and the existence of potentially several legitimate mappings. Having a set of stereotypes specifically for XML Schema allows for a two-pass mapping, with the first pass applying a straightforward mapping, and the second allowing for a user to edit the results.

1.2 Design Center and Fundamental Issues

The design center of the mapping should be to provide:

- A graphical way of describing all the important aspects of document type design.
- A set of concepts that are familiar and easy to use for an engineer knowledgeable in UML.

The first bullet includes XML Schema document type characteristics such as *required* and *implied* attributes, etc. In addition we need to capture all intrinsic data types as well as provide a mechanism for creating user-defined data types for elements and attributes.

There are a few fundamental issues in achieving these goals. The first issue is that in documents, ordering is significant while for describing the structure of object types it is not. More specifically, a document type may define the order in which data appear within instances of that type. For object types on the other hand we only specify what data an object contains, but not how the data is physically laid out.

2. Mapping Overview

In summary, we map all element and data types in XML Schema to classes annotated with stereotypes. The stereotypes reflect the semantics of the related XML Schema concept. Since ordering for document types is significant for document instances, we need a way of indicating ordering in the UML representation. We do this by including a sequence number for content model elements.

Furthermore, XML Schemas may contain anonymous groups. To represent anonymous groups in UML we need to generate names for the classes that represents such as group in a UML diagram. We introduce special stereotypes indicating that a class represents an anonymous grouping of elements.

The table below lays out the stereotypes being added to the UML to express XML Schema constructs.

Stereotype	UML Construct	SOX Meaning
<<sox>>	Package	Indicates a full Schema
<<elementtype>>	Class	Element type definition
<<sequence>>	Nested Class	Sequence group from a content model
<<choice>>	Nested Class	Choice group from a content model
<<enumeration>>	Class – may be nested	Enumeration datatype – *can be UML enumeration*
<<scalar>>	Class – may be nested	Scalar datatype
<<varchar>>	Class – may be nested	Varchar datatype
<<implied>>	Attribute or Unidirectional Association	Indicates an implied attribute
<<required>>	Attribute or Unidirectional Association	Indicates a required attribute
<<default>>	Attribute or Unidirectional Association	Indicates a default attribute
<<fixed>>	Attribute or Unidirectional Association	Indicates a fixed attribute
<<content>>	Attribute or Aggregation	Indicates an atom in a content model

3. DETAILED MAPPING AND EXAMPLE

1.3 INTRODUCTION

We will use a small example to explaining our XML Schema to UML mapping. The XML Schema for this example is found in section 4, while the corresponding UML diagram is found in section 5. Our immediate goal is to introduce the mapping for further discussion.

There are essentially four new types of class stereotype:

1. Element types. This includes only the <<elementtype>> stereotype.
2. Model groups. These are the <<sequence>> and <<choice>> stereotypes.
3. Various datatype constructors corresponding to the datatype constructors found in XML Schema. These are the <<enumeration>>, <<scalar>> and <<varchar>> stereotypes.
4. Stereotypes associated with XML attributes (<<implied>>, <<required>>, <<default>>, <<fixed>>) and content models (<<content>>).
5. A <<sox>> stereotype to declare a Package to be a XML SCHEMA schema.

Some of these also apply to associations:

- The <<content>> stereotype applies to aggregation associations for parts of XML Schema content models.
- The XML attribute stereotypes can apply to a unidirectional association to delineate XML attributes.

1.4 DEFINING A DATATYPE

Our example contains two different varieties of data types, `scalar` and `enumeration`. A `scalar` always creates an intensive definition of a new number type, while an `enumeration` always provides an extensive definition of a data type. The only other current data type constructor is `varchar`. Each of these constructor has a corresponding stereotype: <<scalar>>, <<enumeration>>, and <<varchar>>.

When defining a `scalar` or `varchar` in XML Schema, there are several (XML) attributes which may require values (including `digits`, `decimals`, `minvalue`, and `maxvalue` for `scalar`, `maxlen` for

varchar). Attributes such as these will appear in the UML compartment as a list of tagged values. An example of this is price in the diagram. They are represented as attributes.

An enumeration also requires a list of values (although that may be empty if the enumeration is extending another enumeration). In the diagram, these values appear as public attributes. CountryCode and LangCode are examples of this. However, the values of an enumeration can be of any kind of string, so these might be better represented as tagged values.

In the diagram, I show datatypes as extending other data types through a generalization association. Since data types are generally *more* specific than their parents (i.e., an enumeration allows less values than the datatype it "extends"), this may not be the best association to use for this relationship. At the type level, it could be seen as an instantiation relationship, i.e., Price instantiates Scalar, and lineItem uses Price.

We assume the existing XML Schema datatypes (see [SOX2.0]) already exist and can be referenced.

1.5 Defining an Element type

Element types are defined with the <<elementtype>> stereotype. Each element type may additionally have:

- UML associations to indicate generalization, XML attributes, and content model.
- Stereotyped attributes for XML attributes and content model constituents.

Element type super types are designated with a generalization relationship. In the diagram, InternatAddress is a generalization of Address.

XML attributes can be indicated as unidirectional associations from the element type to the datatype of the attribute. The association may have a name, which will become the name of the attribute. If it does not have a name, then the name of the attribute will be the name of the datatype. It optionally has a tagged value corresponding to the default or fixed value of the attribute. It also optionally has a stereotype of <<required>> or <<fixed>>. If the <<required>> stereotype is present, then the tagged value is ignored. If the <<fixed>> stereotype is present, then the tagged value is the fixed value. Note that <<implied>> and <<default>> are really only necessary if mixing XML attributes with non-XML attributes in the same class.

Alternatively, XML attributes can be indicated in the attribute compartment like any other attribute, so long as it has one of the four attribute stereotypes (where **value** is only present if the attribute has either a default or fixed value):

```
1. <<stereotype>> attributeName:attributeType = value
```

The target datatype of an attribute may be nested if it is a datatype specified uniquely for that attribute. If the target of an attribute is an element type, then this is an ID/IDREF association (the XML equivalent of a pointer). In that case, the source attribute is of type IDREF and the target must have at least one attribute of type ID. (The value of an ID attribute must be a name unique to the document. This uniquely identifies the element. The value of an IDREF attribute must the value of an ID attribute somewhere in the document.)

In order to build an appropriate inheritance mechanism in an XML Schema, the basic content model of an element type is always either a sequence or a reference to a single datatype. This becomes a semantic constraint.

However, element types may refer to model groups. Model groups are indicated by classes with a stereotype of either:

1. <<sequence>> indicating this will be a sequence.
2. <<choice>> indicating a choice group.

The elements of a content model or model group can be indicated in one of two ways:

1. aggregation associations.
2. attributes with a `<<content>>` stereotype.

The information required to place each item in a content model is:

- A name. As specified by XML Schema, this is required if the target of the association is a datatype, but not if the target is an element type.
- An ordinal, displayed as a tagged value, for sequences (for choices, this ordinal would be 0 if present)
- A cardinality to correspond to the `occurs` attribute in XML Schema. This can take on the obvious values already in the UML.

If the content model is specified as attributes, then the following format is used:

```
<<content>> {ordinal} name:type [cardinality]
```

The name and colon are optional if type is not a datatype.
Because attributes in UML don't nest, model groups need to be described as external types. These consist of classes with stereotypes of `<<sequence>>` or `<<choice>>`. These may have names, but (at least for now) are considered nested within the referencing element type. In the diagram, `PurchaseOrder` has an internal sequence named `lineItem`.

1.6 Namespaces and Packages

The mechanism provided by XML Schema to group sets of definitions together is the schema itself. The schema is named by a Universal Resource Indicator (URI), which is either a URL or a URN. Whenever constructs in a given schema are referenced, they have a name relative to this URI. The exact mechanism for making such references in XML documents is described in [XMLNS], with clarifications in [SOX2.0].

The corresponding UML construct for grouping definitions is the `package`. In the mapping this becomes explicit; the XML Schema itself is mapped to a UML package. The name of the package is the URI of the schema. The resulting package will also have the <<sox>> stereotype to indicate it is based on an XML Schema.

As XML Schema has not defined any visibility constraints on definitions, all definitions in a Schema are required to be public. This will change if visibility constraints are every provided by XML Schema.

XML Schema provides an import mechanism for a schema to refer to definitions in another schema. In SOX this is done with the `namespace` element. These references will be represented in the UML with associations using the `<<import>>` stereotype.

4. AN EXAMPLE

1.7 INTRODUCTION

In this section we will describe a non-trivial example and how it is represented in XML Schema as well as in UML. The example is a data model of a simplified purchase order document.

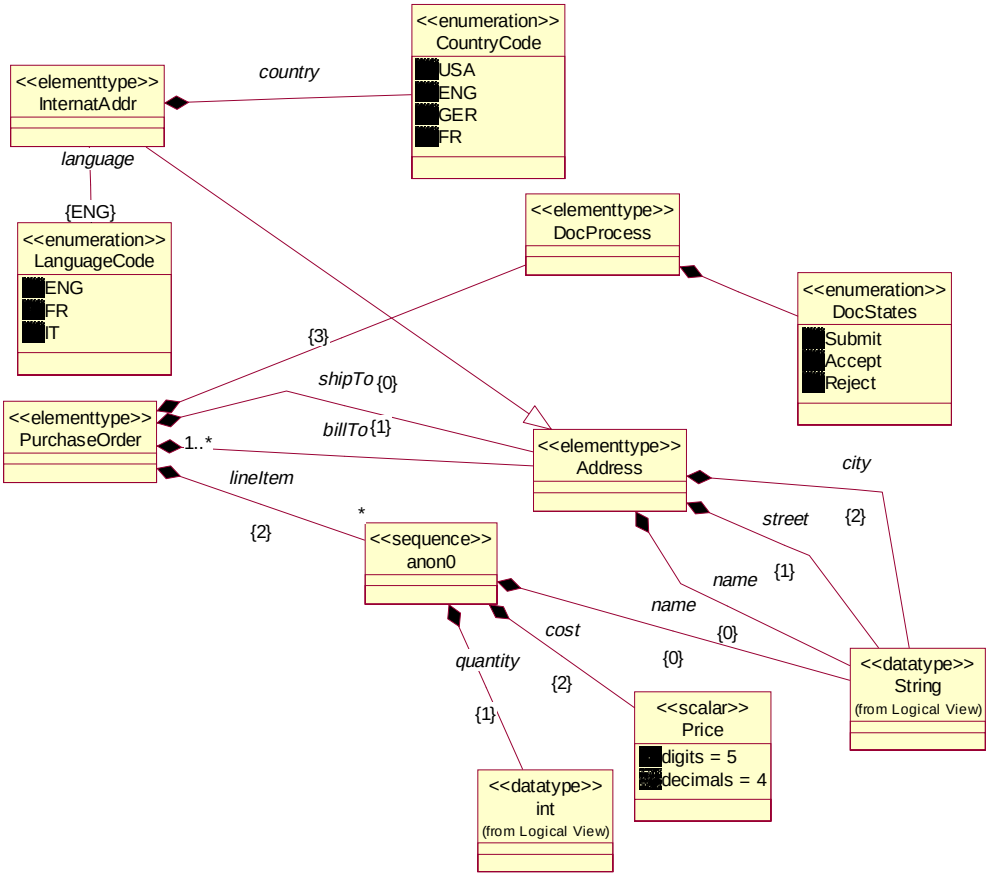
1.8 THE XML SCHEMA

The XML Schema definition below describes the XML document types used to for XML purchase order instances.

```
2.  <schema uri = “urn:document:po.sox”>
3.  <datatype name = “DocStates”>
4.     <enumeration datatype = “NMTOKEN”>
5.        <option>Submit</option>
6.        <option>Accept</option>
7.        <option>Reject</option>
8.     </enumeration>
9.  </datatype>
10.
11. <datatype name = “CountryCode”>
12.    <enumeration datatype = “NMTOKEN”>
13.       <option>USA</option>
14.       <option>ENG</option>
15.       <option>GER</option>
16.       …
17.    </enumeration>
18. </datatype>
19.
20. <datatype name = “Price”>
21.    <scalar digits = “5” decimals = “4”/>
22. </datatype>
23.
24. <elementtype name = “DocProcess”>
25.    <model>
26.       <string datatype = “DocStates”/>
27.    </model>
28. </elementtype
29.
30. <elementtype name = “Address”>
31.    <model>
32.       <sequence>
33.          <element name = “name” type = “string”/>
34.          <element name = “quantity” type = “int”/>
35.          <element name = “cost” type = “Price”/>
36.       </sequence>
37.    </model>
38. <elementtype>
39.
40. <elementtype name = “InternatAddress”>
41.    <extends type = “address>
42.       <append>
43.          <element name = “country” type = “CountryCode”/>
44.       </append>
45.       <attrdef name = “language” type = “LanguageCode”>
46.          <default>ENG</default>
47.       </attrdef>
48.    </extends>
49. </elementtype>
50.
51. <elementtype name = “PurchaseOrder”>
52.    <model>
53.       <sequence>
54.          <element name = “shipTo” type = “Address”/>
55.          <element name = “billTo” type = “Address”/>
56.          <sequence name = “lineItem” occurs = “+”>
57.             <element name = “name” type = “string”/>
58.             <element name = “quantity” type = “int”/>
59.             <element name = “cost” type = “Price”/>
60.          </sequence>
```

```
61.
62.          <element type = “DocProcess”/>
63.       </sequence>
64.    </model>
65. </elementtype>
66. </schema>
```

1.9 THE CORRESPONDING UML SCHEMA DIAGRAM



5. REFERENCES

[UML] Grady Booch, Ivar Jacobson, and James Rumbaugh. Unified Modeling Language. Rational Software Corporation. January 1997. Version 1.0.

[SOX1.1] Matthew Fuchs et. Al. Schema for Object-oriented XML. W3C, 1998, See http://www.w3.org/Sumission/1998/15

[XDR] Charles Frankston and Henry S. Thompson ed. XML Data Reduced. See http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm

[DOM] Document Object Model. See http://www.w3.org/.

[SAX] Simple API for XML. See http://www.megginson.com/SAX and http://www,megginson.com/SAX/SAX2.

[SOX2.0] Andrew Davidson, Matthew Fuchs, Mette Hedin, Mudita Jain, Jari Koistinen, Chris Lloyd, Murray Maloney, and Kelly Schwarzhof . Schema for Object-Oriented XML 2.0. July 1999. See http://www.w3.org/TR/NOTE-SOX

[DCD] Document Content Description for XML (DCD), Tim Bray et al. W3C, 10 August 1998. See http://www.w3.org/TR/NOTE-dcd

[XMLD] XML-Data, Andrew Layman, et al. W3C, 05 January 1998. See http://www.w3.org/TR/1998/NOTE-XML-data-0105

[XML] Extensible Markup Language (XML) 1.0, Tim Bray, et al. W3C, 10 February 1998. See http://www.w3.org/TR/REC-xml

[XSDL] XML Schema Part 1: Structures, David Beech et al. See http://www.w3.org/TR/xmlschema-1/

[XMLNS] Namespaces in XML, Tim Bray, David Hollander, Andrew Layman. See http://www.w3.org/TR/REC-xml-names/