XML Application Development using UML

Goran Lj. Janackovic[1] and Zvonko R. Milosevic[2]

Abstract- **XML Schema has richer structure and semantics that can be expressed as compared to DTD, but it is also more complex. Resulting schemas are difficult to share with users and business partners. UML is standard for system specification and design, effective for specifying vocabularies and sharing definitions with users. These two previously mentioned standards are complementary, and can work together, as it is shown in this paper.**

Keywords – **XML Schema, UML, Development Process.**

I. INTRODUCTION

For a large system problem need to be divided and conquered as a set of alternate models and views, each of them ignores some irrelevant aspects of the system. Different stakeholder groups have different needs with respect to abstraction level [1].

Simplified UML activity diagram for schema development process is shown in Fig. 1.

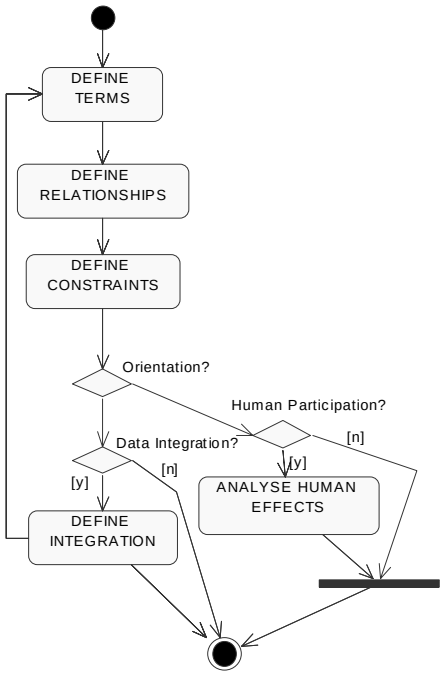

Fig. 1. UML Activity diagram for schema development process

[1]Goran Lj. Janackovic is with the Faculty of Occupational Safety, Carnojevica 10a, 18000 Nis, Yugoslavia, E-mail: jagor@ptt.yu

[2]Zvonko R. Milosevic is with the Faculty of Electronic Engineering, Beogradska 14, 18000 Nis, Yugoslavia, E-mail: zvonko@ptt.yu

It includes three decision points that determine the final definition, regardless of which schema language is used [2]. A data-oriented system can be optimized for serialization of objects or database query results and its constraints are connected to the data-types and referential integrity constraints of its sources. These documents may never be viewed by humans, other than by developers testing the application.

A text-oriented vocabulary often has human users who need to edit the XML documents, with the assistance of GUI editing tools. Its structure must be easily understood by people who write stylesheets that transform and present the documents' content.

II. XML CONCEPTUAL MODEL

UML class diagrams can be applied to a larger XML vocabulary design, and multi-schema support [3]. The purchase order vocabulary is defined in two modules, corresponding to the core PurchaseOrder type and a separate reusable Address module specification. In UML, these modules are called packages. The first package specification is shown as a UML class diagram in Fig. 2. The PurchaseOrder class has two attributes and three associations that define its structure. Several of these attributes include a multiplicity specification of `[0..1]`, which means that those attribute values are optional, either 0 or 1 occurrences.

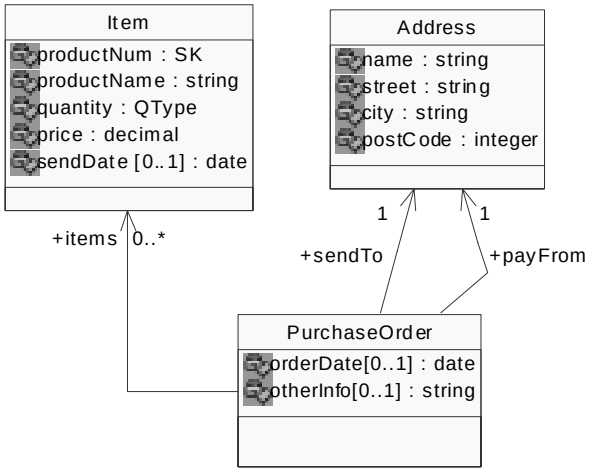

Fig. 2. Conceptual model of purchase order vocabulary

The Address class plays a sendTo and payFrom role in association with a PurchaseOrder. The multiplicity of 1 means that a PurchaseOrder must have exactly one of each address role. On the Item class, a quantity is of type QType. This type is defined as another class in the UML model. In the same

diagram, QType is defined as a subclass of positiveInteger, which is coming from the XSD_Datatypes package in this UML model. Thus, a quantity is a specialized kind of positive integer.

III. Design Models of XML Schemas

According to the schema development process illustrated in Fig. 1, the purchase order vocabulary is data-oriented. The remaining design decisions relate to deployment issues: developer conventions for using XML attributes or child elements, data type alignment with other sources and destinations of data to be exchanged using this vocabulary, and anticipated future requirements for extending this vocabulary or combining it with other XML namespaces.

If this were a text-oriented application, then content managers and authors would have further input on design choices. In is prefered the XML document structures that avoids excessive use of container elements to group related content elements, and the order of elements in a document important to human authors and readers.

IV. Mapping UML Models to XML Schema

By using UML to capture a conceptual model, it is possible to clarify the essential terms and relationships. A primary goal guiding the specification of this mapping is to allow sufficient flexibility to encompass most schema design requirements, while retaining a smooth transition from the conceptual vocabulary model to its detailed design and generation. A related goal is to allow a valid XML schema to be automatically generated from any UML class diagram, even if the modeller has no familiarity with the XML schema syntax, enabling a rapid development process and reuse in different deployment languages or environments.

A class in UML defines a complex data structure and associated behavior that maps to a complex type in XSD. As a first step, the PurchaseOrder class and its UML attributes produce the following XML Schema definition:

```
<xs:complexType name="PurchaseOrder">
  <xs:all>
    <xs:element name="orderDate" type="xs:date"
            minOccurs="0" maxOccurs="1"/>
    <xs:element name="otherInfo" type="xs:string"
            minOccurs="0" maxOccurs="1"/>
  </xs:all>
</xs:complexType>
```

An XSD <xs:all> element is used to create an unordered model group. A UML class creates a distinct namespace for its attribute names. Both of UML attributes are optional, and mapped to minOccurs and maxOccurs attributes in the XSD. The UML attributes are defined using primitive data types from the XSD specification, so these are written directly to the generated schema using the appropriate namespace prefix. If other data types are used in the UML model, then an XSD type library can be created to define these types for use in a schema.

The PurchaseOrder type is specified by its UML attributes and by its associations to other classes in the model. Fig. 1 includes three associations that originate at PurchaseOrder, which is designated by navigation arrows at the opposite ends. Each association has a role name and multiplicity that specifies how the target class is related. These associations are added to the model group of the XSD complexType along with the elements created from the UML attributes.

The ability to include stereotypes is an integral part of the UML standard and is used to specify additional model characteristics that are unique to XML schema design. Using the stereotype, the schema generator knows to create the following definition for SK:

```
<xs:simpleType name="SK">
  <xs:annotation>
    <xs:documentation>SK is a special code
        for identifying every product  </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{8}"/>
  </xs:restriction>
</xs:simpleType>
```

A UML model may also include documentation for any of its model elements, which is passed through to the XML schema definition as shown in this example. The UML generalization relationship indicates which existing simple datatype should be used as the base for this user-defined type. A fundamental and pervasive concept in object-oriented analysis and design is generalization from one class to another. The specialized subclass inherits attributes and associations from all of its parent classes. This is easily represented in W3C XML Schema, although it requires more indirect mechanisms when producing other XML schema languages.

V. UML Profile for XML Schema

UML provides a foundation for modeling structure and behavior of most software systems, but there are domain-specific situations that require additional model information to be captured by the analyst beyond what is possible with UML [4]. This issue is solved through the use of UML extension profiles. 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, which is XML schema design in our case.

Three stereotypes are introduced here along with a value properties, each of them assigned to one or more UML constructs. Each stereotype can be further specified by adding one or more properties that refine its meaning or impact on a model. A stereotype assigned to a UML class extends the meaning of a "class" within the profile's domain and the stereotype's properties are added to the specification of that class in the model. Three stereotypes from the UML Profile for XML Schema are summarized as follows:

<<XSDcomplexType>> on a UML class

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

<<XSDelement>> on a UML attribute or association end

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

<<XSDattribute>> on a UML attribute or association end

- use (prohibited | optional | required | fixed)

Other stereotypes can be used to modify the meaning of those structures in the XML schema without specifying additional properties [5]. After applying these profile extensions, the following schema is produced for the PurchaseOrder class and its associations:

```
<xs:element name="purchaseOrder"  type="ipo:PurchaseOrder"/>
<xs:complexType name="PurchaseOrder">
  <xs:sequence>
    <xs:element name="sendTo"  type="ipo:Address"/>
    <xs:element name="payFrom" 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>
```

The use of a sequence model group raises a new issue when mapping from UML to XML schemas. UML attributes and associations are inherently unordered within their owning class. So each UML attribute and association end that is part of a sequence group must be annotated with a profile property that specifies its position. These position property values are shown as annotations in Fig. 1. The procedure for adding profile stereotypes and property values is different in each UML tool, although any tool that claims compliance with the UML specification must provide some means for adding them.

Because the items role on the association to the Item class is *not* specified as an anonymousType, its definition in the schema shown above retains the role's container element to hold elements for the related class. The document instance for purchase order items is as follows:

```
<ipo:purchaseOrder>
   <ipo:items>
      <ipo:item partNum="23251">
         <ipo:productName>C# Unleashed</ipo:productName>
         <ipo:quantity>1</ipo:quantity>
         <ipo:YuPrice>5000,00</ipo:YuPrice>
         <ipo:comment>Simply the best!</ipo:comment>
         <ipo:shipDate>2002-15-04</ipo:shipDate>
      </ipo:item>
   </ipo:items>
</ipo:purchaseOrder>
```

In the class diagram, if an association end is marked as an anonymousType, then the name of the associated class is anonymous when its instances appear in XML documents, regardless of which schema language is actually used to define those documents. The concept of anonymous types is realized differently in different schema languages.

There is an difference between the XML document elements for purchaseOrder and item appear with a lower-case first character, but the default mapping from UML creates these element names equal to the class names, which begin with upper-case letters, commonly used in object-oriented models and languages.

This issue can be resolved by adding an property to a UML class along with the appropriate stereotype. This profile property allows an XML schema designer to choose a preferred naming convention when modeling the schema details. Like many other profile properties, this value can be set as a default for the entire model so that all class names will be mapped to XML element names in the same way.

VI. Schema Modularity and Reuse

One of the benefits gained by using UML as part of our XML development process is that it enables a thoughtful approach to modular, maintainable, reusable application components. UML includes package and namespace structures for making these modules explicit and also specifying dependencies between them.

When used in a schema definition, each package produces a separate schema file. The implementation of dependencies varies among alternative schema languages. For DTDs they might become external entity references. For the W3C XML Schema, these package dependencies create either <include> or <import> elements, based on whether or not the target namespaces of related packages are equal. A dependency is shown from the PO package to the XSD_Datatypes package, but an import element is not created because this datatype library is inherently available as part of the XML Schema language.

This object-oriented approach to XML schema design facilitates modular reuse, just as one would do when using languages such as Java or C++. A new vocabulary module could import current Address package and define a new subclass of Address or further specialize adresses with a new subclass. For example, BusinessAddress might be created with a new attribute. When transformed to XML Schema, this new subtype would automatically become available as valid content for the sendTo or payFrom elements in a purchaseOrder.

This is conceptually similar to the way one would create a new Java subclass within a new application-specific package;

other libraries are reused by importing, and possibly extending, their classes.

VII. Successful Application Development

In the following paragraphs are presented some ideas to enable better e-business project development.

- Plan for conceptual models of developed system that are reusable in several different deployment contexts, i.e. W3C XML Schema, DTD, relational DBMS, Java or EJB, or for using in standard and also web applications. Alternative UML profiles can be used to transform the common business model to alternative platforms. Full realization of this goal is beyond the capabilities of many current UML tools.
- Pre-existing UML models might be specialized to their deployment platform, platform libraries, and datatypes (Java, .NET, etc.). Isolate the platform independent domain model to enable its reuse and to generate XML schemas for data interchange.
- Use consistent modeling guidelines for naming and structure, both within a single vocabulary and across a set of related models. It is necessary to use some architecture specification that provides clear guidelines for writing DTDs that are easily transferred to UML models using some previously mentioned transformation rules or any other object-oriented framework.

The following list summarizes several goals that guide XML application development work [6].

- It is necessary to create a valid XML schema from any UML class structure model, as previosly described in this paper.
- Refine the conceptual model to a design model specialized for XML schema by adding stereotypes and properties that are based on a customization profile for UML.
- Support a bi-directional mapping between UML and XSD, including reverse engineering existing XML schemas into UML models.
- Design and deploy XML vocabularies by assembling reusable modules.
- Integrate XML and non-XML information models in UML; to represent, for example, both XML schemas and relational database schemas in a larger system.

The previous introduction to a UML profile for XML adds a critical step toward all of these goals. These extensions to UML allow schema designers to satisfy specific architectural and deployment requirements, analogous to physical database design in a RDBMS. The same extensions are necessary when reverse engineering existing schemas into UML because we must map arbitrary schema structures into an object-oriented model. In most cases, a few well-defined stereotypes and properties will achieve major design objectives.

VIII. Conclusion

The default mapping rules described in this paper can be used to generate a complete XML schema from any UML class diagram. This might be application model that now must be deployed within an XML web services architecture, or it might be a new XML model intended as a data interchange standard. The default schema provides a usable first iteration that can be immediately used in an initial application deployment, although it may require refinement to meet other architectural and design requirements.

Text-oriented schemas, and any other schema that might be authored by humans and used as content for HTML portals, often must be refined to simplify the XML document structure.

Stereotypes and their associated property values are part of a UML profile for XML Schemas. Using stereotypes one can customize the generated schema. A web-based tool that implements the complete UML profile for schema design and transforms any UML class model to a W3C XML Schema can be developed using these transformation rules.

To support an iterative modeling process, it is easy to developed a web application that creates XML schemas from UML models. A key enabling technology is the XML Metadata Interchange (XMI) specification from the OMG that defines a standard for serializing UML models as XML documents. Many UML tools now support this standard import-export format, and some use it as their native file format.

The Microsoft .NET framework is becoming well known for its integration of XML into nearly all data-manipulation tasks, and using possibility to develop XML code from UML specification of system elements will unify the whole development process.

References

[1] A. Bergholz, “Extending Your Markup: An XML Tutorial”, *IEEE Internet Computing*, pp. 74-79, July/Aug. 2000.

[2] R. Khare, A. Rifkin, “XML: A Door to Automated Web Applications”, *IEEE Internet Computing*,pp. 79-87, July/Aug. 1997.

[3] Elisa Bertino, Elena Ferrari, “XML and Data Integration”, *IEEE Internet Computing*, pp. 75-76, Nov/Dec. 2001.

[4] D. Connolly and J. Bosak, “Extensible Markup Language (XML)”, 1997. http://www.w3.org/XML/

[5] B. Box, “The XML Data Model,” 1997. http://www.w3.org/XML/ Datamodel.html

[6] D. Carlson, *Modeling XML applications with UML*, Addison-Wesley, 2001.