Rule-Based Generation of XML Schemas from UML Class Diagrams

Tobias Krumbein, Thomas Kudrass
Leipzig University of Applied Sciences,
Department of Computer Science and Mathematics, D-04251 Leipzig
{tkrumbe|kudrass}@imn.htwk-leipzig.de

Abstract. We present an approach of how to automatically extract an XML document structure from a conceptual data model that describes the content of the document. We use UML class diagrams as the conceptual model that can be represented in XML syntax (XMI). The algorithm we present in the paper is implemented as a set of rules using XSLT stylesheets that transform the UML class diagram into an adequate XML Schema definition language (XSD). The generation of the XML Schema from the semantic model corresponds with the logical XML database design based on the fact that the XML Schema is the database schema description. Therefore we discuss many semantic issues and how to express them in XML Schema to minimize the loss of information.

1 Motivation

Conceptual modeling of information is a widely accepted method of database design. It improves the quality of the databases, supports an early recognition of design errors and reduces the cost of the development process. Analogous to the relational database design we must embrace a 3-level information architecture for XML databases, also known as document viewpoints [1]. This architecture allows the data modeler to start by focusing on conceptual domain modeling issues rather than implementation issues. At the conceptual level, the focus is on data structures, semantic relationships between data and integrity constraints (information viewpoint). The information of an XML document can be arranged in a logical structure (logical level) and is stored dependent on the type of the document (physical level).

Currently, DTDs are traditionally the most common way to specify an XML document schema which corresponds with the logical structure of the document. XML Schema is the successor of DTD and provides strong data typing, modularization und reuse mechanisms which are not supported in DTD. The textual description of an XML Schema allows for communication in the WWW and the processing with XML parsers. There is a number of tree-based graphical tools for developing the document structure, such as XML Spy or XML Authority. But there are almost no established methods that explicitly model the information of an XML document at the conceptual level. The more complex the data is, the harder is it for the designer to produce the correct document schema. UML makes it easier to visualize the conceptual model and to express the integrity constraints.

There are only a few publications on the automatic generation of XML document schemas from conceptual models [2]. Conrad et al. [3] propose a set of transformation rules for UML class diagrams into XML DTDs but don't provide a complete transformation algorithm and UML associations are only translated into XLinks. Another approach is to extract semantic information from the relational database schema as it is proposed in [4]. The authors ignore lots of semantic issues like cardinality or key constraints. In [5] the authors propose an algorithm for the automatic generation of XML DTDs from an (Extended) Entity Relationship Diagram. Another interesting approach is presented in [6] that describes a mapping between UML class diagrams and XML Schema using the 3-level design approach. They represent the logical design level by UML class diagrams which are enhanced by stereotypes to express the XML Schema facilities. Jeckle also presents an interesting approach that has been implemented [7]. EER schemas and UML class diagrams have much in common which makes it possible to adapt mapping procedures from both source models for the generation of XML Schemas. On one hand, there is a variety of mapping strategies for the logical XML database design. On the other hand, there are almost no reports on working implementations. This paper contributes a mapping algorithm for the automatic generation of XML Schemas using stylesheets to represent the transformation rules. Our approach is open since the algorithm is adaptable by changing rules. In the past we also presented and implemented such an complete algorithm for DTDs in [8].

This paper is organized as follows: Section 2 gives an overview of UML class diagrams that are used for modeling data structures. For every diagram element, different mapping strategies are discussed which can be expressed in transformation rules to generate an adequate XML Schema representation. In Section 3 we give an overview about the complete algorithm for the generation of XML Schemas from UML class diagrams that we have implemented as rules. We illustrate this algorithm on a sample model. After it our implementation with XSLT - based on XMI (*XML Metadata Interchange*) - and the rules of the XSLT stylesheet are described. We discuss options and limitations of the mapping approach in section 4. As a conclusion, we give an assessment of our experiences in section 5.

2 Mapping UML Class Diagrams into XML Structures

2.1 Elements of UML Class Diagrams

The primary element of class diagrams is the class. A class definition is divided into three parts: class name (plus stereotypes or properties), attributes and operations of the class. A class can be an abstract one. Attributes can be differentiated into class attributes (underlined) and instance attributes. An attribute definition consists of: visibility (public, protected, private), attribute name, multiplicity, type, default value and possible other properties. Derived attributes can be defined, i.e. their values can be computed from other attribute values. They are depicted by a '/' prefix before the name. UML types can be primitive or enumeration types or complex types. Classes can be arranged in a generalization hierarchy which allows multiple inheritance.

Associations depict relationships between classes in UML which are represented by lines, for example an association between classes A and B. The multiplicity r..s at the

B end specifies that an instance of A can have a relationship with at least r instances and at most s instances of B. Associations can comprise more than two classes. Those n-ary associations are represented by a rhomb in the diagram. Associations can be marked as navigable which means that the association can be traversed only along one direction. Yet the default is a bidirectional association. In order to specify attributes of an association, an association class has to be defined additionally.

Besides general associations UML provides special types of associations. Among them is the aggregation representing a part-of semantics (drawn by a small empty diamond in the diagram). The composition as another type is more restrictive, i.e., a class can have at most one composition relationship with a parent class (exclusive) and its life span is coupled with the existence of the superclass. It is represented by a black diamond at the end of the composite class. Qualified association is a special type of association. Qualifiers are attributes of the association, whose values partition the set of instances associated with an instance across an association.

The elements of an UML model can be modularized and structured by the usage of packages.

2.2 Mapping of Classes and Attributes

UML classes and XML elements have much in common: Both have a name and a number of attributes. Hence a class is represented by an element definition; operations do not have an XML equivalent. The generated XML element has the same name as the UML class. The elements need to be extended by an ID attribute in order to refer them from other parts of the document. Note that the object identity applies only within the scope of one document. Abstract classes are mapped to an abstract element definition. Additionally, a global complex type with the name of the class is defined for all classes. It supports the reuse of their definitions by their subclasses.

Classes with the stereotype `enumeration`, `choice` or `any` are separately handled. For all enumeration classes a simple type with an enumeration list is defined with the attribute names of the enumeration class as values. Choice classes are handled as normal classes but with the type constructor choice, so all subelements of the class element are a choice list. Classes with the stereotype `any` are also handled as normal classes but additionally an *any* element and an *any* attribute are defined in the class element. Other stereotypes are represented as an attribute of the element.

UML attributes can be transformed into XML attributes or subelements. A representation as an XML attribute is restricted to attributes of primitive datatypes and therefore not applicable to complex or set-valued attributes. A workaround solution is the usage of the NMTOKENS type or other list datatypes for XML attributes, although this excludes attribute values containing blanks. A default value, a fixed value and a value list in XML Schema can be assigned to attributes as well as to elements. XML Schema provides also data typing. So the mapping of the UML datatypes to XML Schema is possible. For detail information see [7] or [9].

UML element	XML attribute	XML element
primitive datatypes	supported	supported
complex datatypes	not supported	supported
multiplicity	[0..1] and [1..1] or all by the use of a list datatype (values don't contain blanks)	all
property string	not supported	supported
default value	default property	default property
fixed value	fixed property	fixed property
value list	enumeration supported	enumeration supported
scope of definition	local	local or global

Table 1: Attributes vs. elements at XML Schema generation

The decision of the automatic transformation of UML attributes into elements or attributes is dependent on the intended use of the XML Schema. If the fact is considered that UML attributes can be multiple or complex, UML attributes should be mapped into elements.

There are some UML constructs which cannot be translated into an adequate document type definition: The visibility properties of UML attributes cannot be transformed due to the lack of encapsulation in XML. The property {*frozen*} determines that an attribute value can be assigned once and remains static, which cannot be mapped properly to an equivalent XML construct. The only workaround solution is to define an initial value as default with the property *fixed* in an XML Schema. Class attributes are not supported in XML as well; they can be marked by naming conventions in the automatic transformation. An adequate transformation of derived attributes into XML would require the access to other document parts which implies the transformation of the derivation expression into an XPath expression. We propose to ignore derived attributes because they do not carry information.

2.3 Mapping of Associations

2.3.1 Approaches for Binary Associations

The most crucial issue of the transformation algorithm is the treatment of UML associations. There are different procedures how to represent associations in an XML Schema but all of them result in some loss of information regarding the source model. There are four approaches which are subsequently discussed.

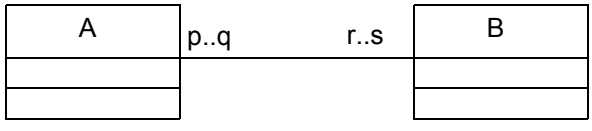

Figure 1: Mapping of non-hierarchical relationships

- nested elements (hierarchical relationship)
- Key/Keyref references of elements
- references via association element
- references with XLink and XPointer

Hierarchical relationship

The hierarchical relationship is the "natural" relationship in XML because it corresponds with the tree structure of XML documents. Elements are nested within their parent elements which implies some restrictions. The existence of the subelement depends on the parent element. If B is represented as subelement of A, the upper bound of its multiplicity q is restricted to 1. Usually p must also be 1. Otherwise, alternative mappings have to be defined, e.g. the definition of B as subelement of the root element. The main obstacle for the nesting of elements is the creation of redundancies in case of many-to-many relationships. It depends on the application profile how far redundancy in the document can be tolerated. For example, read-only applications may accept redundancy within a document because it fastens the access to related information. From the viewpoint of the logical XML database design the hierarchical approach appears inappropriate.

Regarding hierarchical representation it is also difficult to deal with recursive associations or relationship cycles between two or more classes. The XML documents have a document tree of indefinite depth. This can be avoided by treating each association as optional - regardless of the constraint definition in the class diagram.

Key/Keyref references

The Key/Keyref relationship is expressed by adding an ID attribute to referenceable elements and a key that contains a selector and a field which includes an XPath expression each. The selector selects all elements of a class and the field selects the ID attribute of each selected element. The references are implemented by reference elements with an attribute of type IDREF and a keyref (key reference). This keyref references the key of the target class. Additionally, the keyref selects all reference elements with the selector XPath expression and the field XPath expression selects the IDREF attribute of each selected reference element. So the schema validator compares the IDREF attribute of all the reference elements with the ID attribute of the target class element. If these attributes do not match the process stops with an error. The multiplicity p..q is defined in the reference element. This approach guarantees type safety. There are restrictions which obstruct a semantically correct mapping. Bidirectional associations are represented by two Key/Keyref references in the XML Schema. However, this approach cannot guarantee a mutual reference between two element instances that take part in a bidirectional association.

References via association elements

For each association an association element is introduced that references both participating elements using IDREF attributes (analogous to relations for many-to-many relationships in RDBMS). The association elements are included as subelements of the document root. There are no references in the class elements. The association element

gets the name of the association, the references are labeled according to the association roles. The approach produces XML documents with minimal redundancy because every instance needs to be stored only once within the document.
The multiplicity values cannot be expressed adequately by association elements. We can merely define how many elements are related by an association instance. This does not consider participation constraints for the element instances. Association elements are particularly useful for n-ary associations and attributed associations only because of their limitations.

References with XLinks
XLinks have been invented for hyperlink documents that are referencing each other which makes it possible to reference different document fragments. We consider the extended features provided by XLinks. The association element is represented as *extended* link. A *locator* element is needed for each associated element to identify it. The association itself is established by *arc* elements that specify the direction. The use of XLinks has been explored by [3]. However this approach has no type safety.

2.3.2 Association Classes
An association class is an association with class features. So the transformation has to consider the mapping of both a class and an association. Therefore, the four mapping approaches for associations, as sketched above, apply to association classes as well:
The association class is mapped to an association element that is nested inside the parent element in the hierarchical approach (for functional relationships only). The association attributes and the child element in the hierarchical approach are added to the association element.
Using Key/Keyref references requires the introduction of two references to consider bidirectional relationships. Thus the attributes of the association class would be stored twice. It could not be guaranteed that those attributes are the same in two mutually referencing elements. Hence, the mapping has to be enhanced by association elements.
The association elements contain the attributes of the corresponding association class. Associations of each multiplicity are dealt with the same way.
Regarding the last approach that uses extended XLinks is comparable with association elements one can draw the same conclusion as mentioned above.
It is also possible to resolve the association class and represent it as two separate associations. Note that the semantics of bidirectional associations cannot be preserved adequately with that mapping.

2.3.3 N-ary Associations
N-ary associations can also be mapped by using one of the four mapping approaches for associations. Simple hierarchical relationships or Key/Keyref references are not appropriate; they support binary associations at best. Better mappings are association elements and extended XLinks which can contain the attributes of n-ary associations. Alternatively, the n-ary association can be resolved into n binary associations between every class and the association element.

2.3.4 Other Properties of Associations / Limitations

Each end of an association can be assigned the *{ordered}* property to determine the order of the associated instances. It is not possible to define the order of element instances in an XML Schema.

The direction of an association cannot be preserved by mapping approaches that represent just bidirectional associations. This applies to: hierarchical relationships, association elements and extended XLinks.

UML provides association properties regarding changeability: *{frozen}* and *{addonly}*. Addonly allows an instance to join more associations instances without deleting or changing existing ones. Both properties cannot be expressed in XML.

There are no means to represent access properties of associations in XML.

In UML, a qualifier can be defined at an association end to restrict the set of instances that can take part in the association. Of the described approaches only Key/Keyref references can represent qualifier associations. For these qualifier associations the reference element, key and keyref are extended by the qualifier attributes. So the parser not only compares the ID attributes but also the qualifier attributes.

XOR constraints between associations cannot mapped into XML Schema because XOR constraints are not completely preserved when they are exported into XMI with *Unisys Rose XML Tools* [10].

2.4 Mapping of Generalization

There is no generalization construct in the XML Schema. The most relevant aspect of generalization is the inheritance of attributes of the superclass. There are two reasonable approaches to represent the inheritance in the XML Schema: the type inheritance by type extension and the reuse of element and attribute groups. A complex type is defined for all class elements. If a class is a subclass the complex type is defined as an extension of the complex type of the superclass element. So the subclass element inherits all properties of the superclass element. This approach supports the substitution relationship between a superclass and its subclasses, but it supports only single inheritance. Alternatively, an element and an attribute group can be defined for the subelements and attributes of each class element which can be reused in the complex type of the corresponding class element. Additionally, the element and attribute groups of all superclasses of a class are reused in the complex type of this class. So all elements and attributes of the superclasses are assigned to the subclasses. This approach supports multiple inheritance, but doesn't support the substitution relationship between a superclass and its subclasses. To express the substitution relationship between a superclass and its subclasses, the use of a superclass element is substituted by a choice list that contains the superclass element and all its subclass elements.

2.5 Further Mapping Issues

The aggregation relationship of UML embodies a simple part-of semantics whereas the existence of the part does not depend on the parent. Therefore aggregations are treated like associations.

Compositions can be mapped through hierarchical relationships according to our previous proposal for associations, because nested elements are dependent on the existence of their parent elements and therefore represent the semantics of compositions.
Packages are represented as elements without attributes. The name of the element is the package name. All elements of the classes and packages are subelements of their package element. Alternative packages can be represented as namespaces.

3 Generation of XML Schemas from Class Diagrams

3.1 Algorithm

Among different alternatives discussed in the section above we give an overview about the transformation methods we have implemented as rules (for further details see [9]). We dont represent a formal algorithm, because XSLT consists of interdependent rules that are difficult to describe procedurally.

UML Element	XML Schema
class	element, complex type, with ID attribute, and key
abstract class	abstract element and complex type, with ID attribute
attribute	subelement of the corresponding class complex type
stereotype	attribute of the corresponding element
package	element without attributes
association aggregation	reference element, with IDREF attribute referencing the associated class and keyref for type safety (key/keyref references)
association class	association class element and an additional IDREF references to the association class element and a keyref in the corresponding reference elements in the associated classes
qualified association	extension of the reference element, keyref and key of the target class with the qualified attributes
composition	reference element, with subordinated class elem. (hierarch. rel.)
generalization	complex type of the subclass is defined as an extension of the complex type of the superclass
association constraint	currently not mapped
n-ary association	association element with IDREF references to all associated classes (resolution of the n-ary association)

Table 2: Mapping of UML elements to XML Schema

3.2 Sample Model

The following UML example (figure 2) illustrates our transformation algorithm. There is an abstract superclass `Person` as a generalization of `Employee`, both belonging to the package `People`. The model contains a bidirectional 1..n association between `Department` and `Employee`. The association between `Company` and `Employee` is an attributed one-to-many relationship that is represented by the association class `Contract`. Furthermore, a `Company` is defined as a composition of 1..n `Departments`.

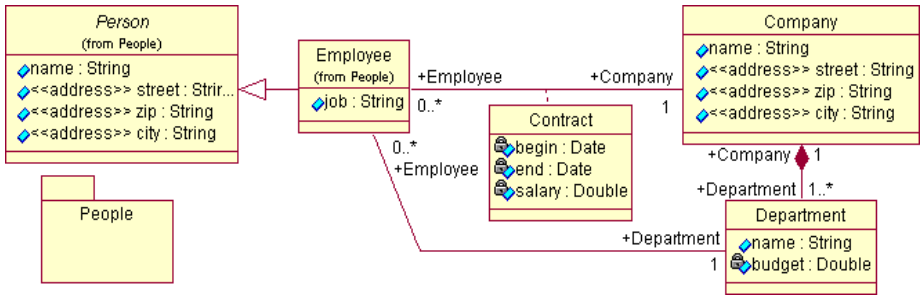

Figure 2: UML class diagram of sample model

```
<!-- define the model structure that consists of packages
and classes - root element = model name -->
<xsd:element name="sample"> <xsd:complexType>
<xsd:sequence> <xsd:element name="People" minOccurs="0">
 <xsd:complexType> <xsd:sequence>
  <xsd:element name="Person" type="People.Person"
    minOccurs="0" maxOccurs="unbounded" abstract="true"/>
  <xsd:element name="Employee" type="People.Employee"
   minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence> ... </xsd:complexType> </xsd:element>
 <xsd:element name="Company" type="Company" ... />
 <xsd:element name="Contract" type="Contract" ... />
 </xsd:sequence> </xsd:complexType>
<!-- define a key for every class -->
 <xsd:key name="People.Person">
  <xsd:selector xpath="People/Person | .//Person |
   .//Employee"/> <xsd:field xpath="@id"/> </xsd:key>
 <xsd:key name="People.Employee"> ... </xsd:key>
 <xsd:key name="Company"> ... </xsd:key>
 <xsd:key name="Department"> ... </xsd:key>
 <xsd:key name="Contract"> ... </xsd:key> </xsd:element>
<!-- define a complexType for every class -->
<xsd:complexType name="People.Person"> <xsd:sequence>
 <!-- define an element definition for every attribute
  of a class -->
 <xsd:element name="name"> <xsd:complexType>
  <xsd:simpleContent> <xsd:extension base="xsd:string"/>
  </xsd:simpleContent>...</xsd:complexType> </xsd:element>
 <xsd:element name="street"> ... </xsd:element>
 <xsd:element name="zip"> ... </xsd:element>
 <xsd:element name="city"> ... </xsd:element>
 </xsd:sequence> <xsd:attribute name="id" type="xsd:ID"
  use="required"/> </xsd:complexType>
<xsd:complexType name="People.Employee">
```

```
 <xsd:complexContent> <xsd:extension base="People.Person">
 <xsd:sequence><xsd:element name="job"> ... </xsd:element>
 <xsd:element name="Department"> ... </xsd:element>
 <!-- definition of the association between Employee and
  Company -->
 <xsd:element name="Company" minOccurs="1" maxOccurs="1">
  <xsd:complexType> <xsd:sequence/> ...
  <xsd:attribute name="idref" type="xsd:IDREF"
   use="required"/>
  <xsd:attribute name="Contract" type="xsd:IDREF"
   use="required"/> </xsd:complexType>
  <xsd:keyref name="People.Employee.Company"
   refer="Company"> <xsd:selector xpath="."/>
   <xsd:field xpath="@idref"/> </xsd:keyref>
  <xsd:keyref name="People.Employee.Company.Contract"
   refer="Contract"> <xsd:selector xpath="."/>
   <xsd:field xpath="@Contract"/> </xsd:keyref>
</xsd:element> </xsd:sequence> </xsd:extension>
</xsd:complexContent> </xsd:complexType>
<xsd:complexType name="Company"> <xsd:sequence> ...
 <xsd:element name="Employee" minOccurs="0"
  maxOccurs="unbounded"><xsd:complexType> <xsd:sequence/>
  ... <xsd:attribute name="idref" type="xsd:IDREF"
  use="required"/> <xsd:attribute name="Contract"
  type="xsd:IDREF" use="required"/> </xsd:complexType>
  <xsd:keyref name="Company.Employee"
   refer="People.Employee"> ... </xsd:keyref>
  <xsd:keyref name="Company.Employee.Contract"
   refer="Contract"> ... </xsd:keyref> </xsd:element>
 <xsd:element name="Department" minOccurs="1"
  maxOccurs="unbounded"> <xsd:complexType> <xsd:sequence>
  <xsd:element name="Department" type="Department"/>
   </xsd:sequence> ... </xsd:complexType> </xsd:element>
  </xsd:sequence> <xsd:attribute name="id" type="xsd:ID"
  use="required"/> </xsd:complexType>
<xsd:complexType name="Department"> ... </xsd:complexType>
<!-- define the association class -->
<xsd:complexType name="Contract"> ... </xsd:complexType>
```

3.3 Implementation

The XMI format (*XML Metadata Interchange*) makes it possible to represent an UML model in an XML format. Our implementation is based on the XMI version 1.1 [11]. The XMI standard describes the generation of DTDs from a meta model as well as the

generation of an XMI document from any model, provided that they are MOF compliant (*Meta Object Facility*).

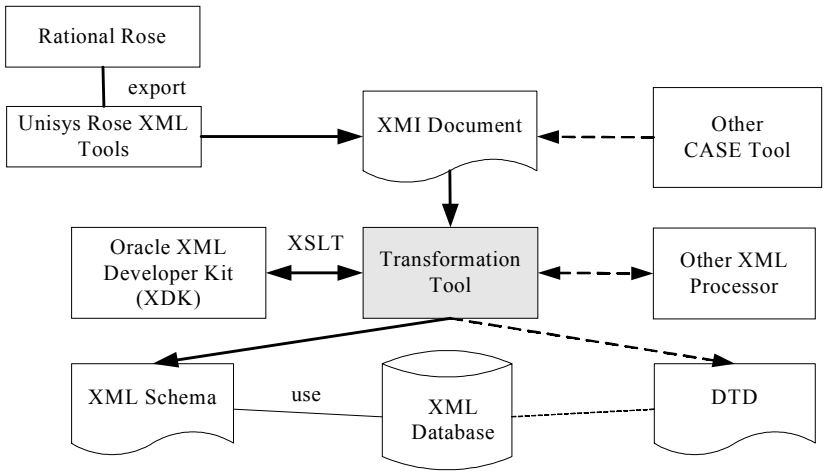

Figure 3: Overall structure of the DTD generation

We edit UML class models with the CASE tool Rational Rose. The model information can be stored in XMI documents using *Unisys Rose XML Tools* (version 1.3.2) [10] as an extension. Since XMI is a standard, the original tool is not relevant for the next transformation steps.

The actual transformation is implemented with XSLT (*eXtensible Stylesheet Language Transformation*) that can process the XMI document as syntactically it is just an XML document. XSLT is a language to transform XML documents into other XML documents or even other formats. The stylesheet document consists of rules that specify how the document tree of the source document has to be transformed into the target tree. The rules called template rules have two parts: a search pattern (source tree) and a template applied for matching patterns.

In our implementation, we have two categories of template rules: Some template rules have patterns that must match with certain XMI elements that are relevant for the conceptual data model. One can find the `UML:Class` template among them. It transforms a UML class description into the corresponding element definition in the XML Schema. Some other templates are just auxiliary templates without matching XMI elements. Instead, they are invoked by other templates that use their functionality. The transformation program starts with the root template rule. Subsequently, the template rules are shown as they are currently implemented.

/ (Root Template)

The root template is called first. It checks the XMI version, determines the first model element and calls the next matching template.

UML:Model

The `UML:Model` element is the root element of the UML model and comprises all other elements like UML packages. This template defines the XML structure of the XML

documents by the creation of an element definition tree of all packages, classes (not parts of another class in a composition), association classes and n-ary associations. The name of the elements corresponds with the corresponding UML element name. Additionally, a key is defined for each class.

UML:Package
For each package an element type was created in the template `UML:Model`. For each subelement of the package element the appropriate template is activated.

UML:Class | UML:AssociationClass
Our algorithm treats association classes like classes. If the stereotype of this class like `enumeration`, `choice` or `any` this class is separately handled by calling the template with the same name. For each `UML:Class` element a complex type is defined in the XML Schema. The name of the type corresponds with the full class name. The name of a possible stereotype appears as an attribute in the complex type. If the class is a subclass the complex type of this class is an extension of the superclass complex type. Next, the content of the class type is governed by all attributes and associations. They are determined by an XPath query on the XMI document. For example, the attributes are represented in the `UML:Attribute` element. An element with the name of the attribute and a simple or complex datatype is defined for all attributes. The associations of a class are processed by calling the template `Association`. In the third step, all properties of a class are defined. Each class receives an ID attribute to make it a potential target of element references.

UML:Association
This template is exclusively called by n-ary associations because only these associations are embedded in a package element. It defines a complex type for the n-ary association with the name of this association and associations for each associated class involved in it.

Association
In XMI, the associations of a class cannot be found within the class element. Instead, they have to be queried throughout the whole XMI document where they are represented as association elements. Once an association of a class has been found it is processed by calling one of the templates `createAssociation`, `createComposition` or `createNaryAssociation` corresponding his type.

createAssociation
This template creates a reference element with the rolename of the association, and an IDREF attribute and a keyref to the key of the target class into this reference element. If this association is an association class an IDREF attribute with the name of the association class and a keyref fitting to the association class key is additionally defined in the reference element. If the association is a qualifier association the reference element, keyref and the key of the target class are extended by the qualifier attributes.

createComposition
This template creates a reference element with the rolename of the association and a choice list in the reference element that contains the subordinated class element and all its subclass elements.

createNaryAssociation
This template creates a reference element with the rolename of the association, an IDREF attribute and a keyref to the key of the n-ary association element in this reference element.

Enumeration
A simple type with the name of the enumeration class is defined. The attributes of this class are defined as enumeration values in this type. The type can be used for attribute elements as datatype.

Choice
Choice classes are treated as normal classes with the type constructor choice, so all subelements of the class element are a choice list.

Any
Classes with the stereotype `any` are also handled as normal classes but an `any` element and an `any` attribute are additionally defined in the class element.

Annotation
If an UML element that is transformed into an element and has a commentary, this commentary is copied into XML Schema as a commentary of the element.

Stereotype
The `Stereotype` template checks for stereotypes for all UML elements. Those elements are referenced by the `stereotype` element via object IDREFS in XMI.

Name
This template determines the name of the current UML element. The name is stored either in the `name` attribute of the element or in the `UML:ModelElement.name` subelement in the XMI definition.

4 Options and Limitations

A number of options is available when mapping the document definition from the conceptual level to the logical level. Section 2 has already outlined alternatives for most UML elements. It just requires the change of template rules to vary certain transformation steps. For example, by changing the template rules the mapping of UML attributes can be modified. In the same way rules can be substituted to implement alternative mappings for the generalization relationship: Instead of the type inheritance by type extension the use of the reuse of element and attribute groups can be a viable alternative for an adequate representation in the XML Schema.

In order to assess the quality of the transformation the loss of information has to be determined. This can be done by a reverse transformation of the generated XML Schema. The following UML elements could not be represented in the XML Schema. Therefore they are not considered at the reverse transformation:

- stereotypes of associations, aggregations, compositions, generalizations
- name of associations, aggregrations, compositions, generalizations
- dependencies

Dependencies have not been transformed because their definition bases mainly on the class behaviour which cannot be expressed in XML Schema. In our implementation, the full syntax of the XML Schema has not been used yet. Among the elements that also should be included are `unique` and all elements which can be used for the definition of simple data types.

Also Rational Rose has some limitations. Thus it is not possible to define attributes with a multiplicity greater one and n-ary associations. On the other hand, the multiplicity of the aggregate end of an aggregation or composition can exceed one in Rational Rose.

5 Conclusion

This paper presents a very flexible method for the logical XML database design by transforming the conceptual data model represented in UML. The UML was primarily chosen because of its widespread and growing use. Yet it would also be possible to use the extended ER model to describe the XML document at the conceptual level. In our approach, we strictly separate the conceptual model and the XML representation of the document content. Therefore, we do not involve XML specific constructs in the conceptual model as they can be found, e.g., in DTD profiles for UML [12] or XML extensions of the ER model [13]. Our methodology is well-suited for the storage of data-centric and semi-structured documents exchanged among different applications. Vendors of XML database systems are able to process document schemas when storing the XML documents in the database. So the result of our transformation can easily be combined with an XML DBMS which accepts XML Schemas as a document schema. Tamino (by Software AG) does not support the full syntax of XML Schema. Therefore the combination of XML Schema with Tamino is not recommended because much information go lost.

The design of the transformation stylesheets has to consider the interplay of the templates when modifying some of the mapping rules to implement a different strategy. A well-designed set of templates as presented in our paper is the precondition to adapt our transformation tool to other target models as well.

Acknowledgement

This work has been funded by the Saxonian Department of Science and Art (Sächsisches Ministerium für Wissenschaft und Kunst) through the HWP program.

References

[1] H. Kilov, L. Cuthbert: A model for document management, Computer Communications, Vol. 18, No. 6, Elsevier Science B.V., 1995.

[2] M. Mani, D. Lee, R. Muntz: Semantic Data Modeling using XML Schema, Proc. 20th Conceptual Modeling Conference ER2001, Yokohama, Springer Verlag, 2001.

[3] R. Conrad, D. Scheffner, J.C. Freytag: XML Conceptual Modeling Using UML, Proc. 19th Conceptual Modeling Conference ER2000, Salt Lake City, Springer Verlag, 2000.

[4] G. Kappel, E. Kapsammer, S. Rausch-Schott, W. Retschitzegger: X-Ray - Towards Integrating XML and Relational Database Systems, Proc. 19th Conference on Conceptual Modeling (ER2000), Salt Lake City, 2000.

[5] C. Kleiner, U. Liepeck: Automatic generation of XML-DTDs from conceptual database schemas (in German), Datenbank-Spektrum 2, dpunkt-Verlag, 2002, pp. 14-22.

[6] N. Routledge, L. Bird, A. Goodschild: UML and XML Schema, Proc. 13th Australasian Database Conference (ADC2002), Melbourne, 2002.

[7] M. Jeckle: Practical usage of W3C's XML-Schema and a process for generating schema structures from UML models, http://www.jeckle.de, 2001.

[8] T. Kudrass, T. Krumbein: Rule-Based Generation of XML DTDs from UML Class Diagrams, Proc. 7th East-European Conference on ADBIS, Dresden, 2003

[9] T. Krumbein: Logical Design of XML Databases by Transformation of a Conceptual Schema, Masters Thesis (in German), HTWK Leipzig, 2003.

[10] Unisys Comporation: Unisys Rose XML Tools V.1.3.2, http://www.rational.com/support/downloadcenter/addins/media/rose/UnisysRoseXMLTools.exe

[11] OMG: XML Metadata Interchange, http://www.omg.org/cgi-bin/doc?formal/00-11-02.pdf, 2000.

[12] D. Carlson: Modeling XML Applications with UML: Practical E-Business Applications, Boston, Addison Wesley, 2001.

[13] G. Psaila: ERX - A Conceptual Model for XML Documents, Proc. of the ACM Symposium of Applied Computing, Como, 2000.