Rule-Based Generation of XML DTDs from UML Class Diagrams

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

Abstract. We present an approach of how to extract automatically an XML document structure from a conceptual data model that describes the content of a 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 that transform the UML class diagram into an adequate document type definition (DTD). The generation of the DTD from the semantic model corresponds with the logical XML database design with the DTD as the database schema description. Therefore, we consider many semantic issues, such as the dealing with relationships, how to express them in a DTD in order to minimize the loss of semantics. Since our algorithm is based on XSLT stylesheets, its transformation rules can be modified in a very flexible manner in order to consider different mapping strategies and requirements.

Keywords: UML, DTD, XML, Schema Generation

1 Introduction

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. A conceptual schema facilitates the communication with the domain expert since it abstracts from the implementation. Due to the analogy of 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 the most common way to specify an XML document schema, which corresponds with the logical structure of the document. The textual description of a DTD facilitates the 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 cor-

rect 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. Conrad et al. [2] propose a set of transformation rules for UML class diagrams into XML DTDs. In this paper, UML associations are only translated into XLinks and there is no complete algorithm available. Another approach is to extract semantic information from the relational database schema as it is proposed in [3]. The authors ignore many semantic issues like cardinality or key constraints. In [4] the authors propose an algorithm for the automatic generation of XML DTDs from an (Extended) Entity Relationship Diagram. They intend an implementation by reconstructing the ER schema from a relational database schema. Another interesting approach is presented in [5] that describes a mapping of UML class diagrams into XML Schema definitions 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. 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 DTDs. 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 DTDs using stylesheets to represent the transformation rules. Our approach is open since the algorithm is adaptable by changing rules. In the same way, the rules can be applied to the generation of another target schema, such as XML Schema or even a relational schema.

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 DTD representation. Section 3 is an overview about the complete algorithm for the generation of DTDs from UML class diagrams that are implemented as rules. This algorithm is illustrated on a sample model. Then the implementation with XSLT - based on XMI - and the rules of the XSLT stylesheet are described. Options and limitations of the mapping approach in section 4 are discussed. As a conclusion, the assessment of the experiences are given 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 possibly 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 describe 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 be comprised 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 in 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 super class. 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 should be mapped to parameter entities to support the reuse of their definitions by their subclasses. It is also possible to define an element for an abstract class without declaring it within the package element.

Classes with the stereotype enumeration are separately handled as enumeration datatypes. For all attributes that have an enumeration class as a datatype, an enumeration list is defined with the attribute names of the enumeration class as values. Other stereotypes are represented as prefixes of the element name or attribute name.

UML attributes can be transformed into XML attributes or subelements. A representation as 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 for XML attributes, although this excludes attribute values containing blanks. A default value, a fixed value and a value list in a DTD can be assigned to attributes which is not possible for elements.

UML element	XML attribute	XML element
primitive datatypes	supported	supported
complex datatypes	not supported	supported
multiplicity	[0..1] and [1..1]	all

Table 1: Attributes vs. elements at DTD generation

UML element	XML attribute	XML element
property string	not supported	supported
default value	default property	not supported
fixed value	#FIXED 'value'	not supported
value list	enumeration supported	not supported
scope of definition	local	global

Table 1: Attributes vs. elements at DTD generation

The last entry of table 1 highlights a serious problem for an automatic transformation of UML attributes into elements. XML attributes are always defined within a certain element, whereas elements are globally defined within the whole document. Therefore name conflicts may occur when UML attributes are transformed into XML 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 a DTD. Class attributes are also not supported in XML; they can be marked by naming conventions in the automatic transformation. An adequate transformation of derived attributes into XML would require access to other document parts which implies the transformation of the derivation expression into an XPath expression. Derived attributes are ignored 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 on how to represent associations in a DTD but all of them result in some loss of information regarding the source model. There are four approaches, which are subsequently discussed.

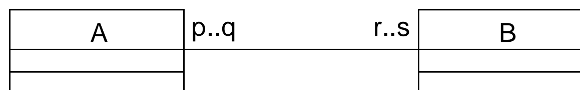

Figure 1: Mapping of non-hierarchical relationships

- nested elements (hierarchical relationship)
- ID/IDREF 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 par-

ent 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 as to 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.

ID/IDREF references

The ID/IDREF relationship is expressed by adding an ID attribute to elements to be referenced. The references are implemented by attributes of type IDREF (single reference) or IDREFS (multiple reference). Depending on the multiplicity p..q the reference attribute ref of B is defined as follows:

p	q	Definition of the reference attribute of B
0	1	<!ATTLIST B ref IDREF #IMPLIED>
0	*	<!ATTLIST B ref IDREFS #IMPLIED>
1	1	<!ATTLIST B ref IDREF #REQUIRED>
1	*	<!ATTLIST B ref IDREFS #REQUIRED>

Table 2: Mapping of multiplicity constraints

There are serious restrictions, which obstruct a semantically correct mapping. The "type" IDREFS accepts duplicate reference whereas the multiplicity in UML denotes the number of distinct instances of an association. For a better representation of multiple references, it is also possible to define an element with an IDREF attribute and with the multiplicity in the parent element. The main drawback is the lacking type safety in the ID/IDREF representation. IDREF can reference elements of any type. The type information could by expressed by naming conventions for the reference attributes without enforcing the integrity. Bidirectional associations are represented by two ID/IDREF references in the DTD. 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. The extended features provided by XLinks have been considered. 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 [2]. 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 for association classes as well.

The association class is mapped to an association element that is nested towards 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 ID/IDREF 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. Thus the mapping has to be enhanced by an association element.

The association elements contain the attributes of the corresponding association class. Associations of each multiplicity are dealt with the same way.

References with extended XLinks is comparable with association elements with 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 treated by using one of the four mapping approaches for associations. Simple hierarchical relationships or ID/IDREF references are not appropriate; they support binary associations at best. Better mappings are association elements and extended XLinks, because they can contain the attributes of n-ary associations and represent an association with references to all association ends. 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 a DTD.

The direction of an association cannot be preserved by mapping approaches that represent just bidirectional associations. This applies to: hierarchical relationships, association elements, 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. The described mapping procedures do not support qualifiers as they cannot guarantee type safety.

XOR constraints specify the participation of an instance in one of many possible associations in UML. In a DTD, one can define alternative subelements, listed by |. When exporting the UML class diagram into XMI with the *Unisys Rose XML Tool* the XOR constraints are represented only as comments in XMI, which are split up among different elements. So the information about the UML elements related by the constraint cannot be preserved during the transformation.

2.4 Mapping of Generalization

There is no generalization construct in the DTD standard. The most relevant aspect of generalization is the inheritance of attributes of the superclass. There are two reasonable approaches to represent inheritance in the DTD: parameter entities and embedded elements. Parameter entities are defined for attributes and subelements of superclasses. They are inherited by the subclass using parameter entities in the definition of the corresponding element in XML. Alternatively, the superclass element can be embedded completely into the subclass element. 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. Another solution is the embedding of the subclasses into the superclass. The best way to represent a superclass with different subclasses is to use a choice list in the element definition of the superclass. So the subclass element is nested within the superclass element giving up its identity.

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 the previous proposal for associations, because nested elements are dependent on the existence of their parent elements and, hence, 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.

3 Generation of DTDs from Class Diagrams

3.1 Algorithm

Among different alternatives, discussed in the section above, an overview is given about the transformation methods which have been implemented as rules instead of a conventional algorithm (for further details see [6]).

UML Element	XML DTD
class	element, with ID attribute
abstract class	element but not subelement of the parent element
attribute	attribute of the corresponding class element
stereotype	prefix of the element name or attribute name
package	element without attributes
association	reference element, with IDREF attribute referencing the associated class
association class	association class element with IDREF references to both associated classes (resolve the association class)
qualified association	currently not mapped
aggregation	like association
composition	reference element, with subordinated class element (hierarchical relationship)
generalization	superclass is nested within subclass element
association constraint	currently not mapped
n-ary association	association element with IDREF references to all associated classes (resolve the n-ary association)

Table 3: Mapping of UML elements to DTDs

3.2 Sample Model

The following UML example (figure 2) illustrates the transformation algorithm. There is an abstract superclass `Person` as generalization of `Employee` and `Manager`, all of them belong to the package `People`. The model contains several bidirectional associations: a one-to-one relationship between `Manager` and `Department`, a one-to-many relationship between `Department` and `Employees`, a many-to-many relationship between `Employees` and `Projects` as well as a unidirectional relationship between `Department` and `Project`. The association between `Company` and `Employees` 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`.

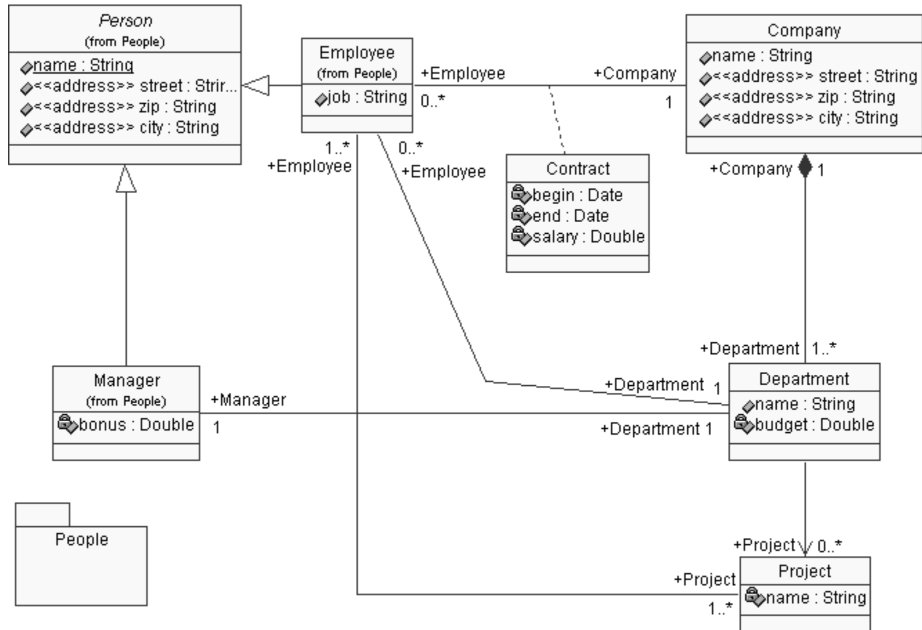

Figure 2: UML class diagram of sample model

```
<!ELEMENT sample (People, Company*, Project*, Contract*)>
<!ELEMENT People (Employee*, Manager*)>
<!ELEMENT Person EMPTY>
<!ATTLIST Person
  id ID #REQUIRED
  name CDATA #REQUIRED
  address.street CDATA #REQUIRED
  address.zip CDATA #REQUIRED
  address.city CDATA #REQUIRED>
<!ELEMENT Employee (Person, Ref_Employee.Department,
  Ref_Employee.Project+, Ref_Employee.Company)>
<!ATTLIST Employee
  id ID #REQUIRED
  job CDATA #REQUIRED>
    <!ELEMENT Ref_Employee.Department EMPTY>
    <!ATTLIST Ref_Employee.Department
      Department IDREF #REQUIRED>
    <!ELEMENT Ref_Employee.Project EMPTY>
    <!ATTLIST Ref_Employee.Project
      Project IDREF #REQUIRED>
    <!ELEMENT Ref_Employee.Company EMPTY>
    <!ATTLIST Ref_Employee.Company
      Contract IDREF #REQUIRED>
```

```
<!ELEMENT Manager (Person, Ref_Manager.Department)>
<!ATTLIST Manager ... >
    <!ELEMENT Ref_Manager.Department EMPTY>
    <!ATTLIST Ref_Manager.Department
      Department IDREF #REQUIRED>
<!ELEMENT Company (Ref_Company.Employee*,
  Ref_Company.Department+)>
<!ATTLIST Company ... >
    <!ELEMENT Ref_Company.Employee EMPTY>
    <!ATTLIST Ref_Company.Employee
      Contract IDREF #REQUIRED>
    <!ELEMENT Ref_Company.Department (Department)>
<!ELEMENT Department (Ref_Department.Employee*,
  Ref_Department.Manager, Ref_Department.Project*)>
<!ATTLIST Department ... >
    <!ELEMENT Ref_Department.Employee EMPTY>
    <!ATTLIST Ref_Department.Employee ... >
    <!ELEMENT Ref_Department.Manager EMPTY>
    <!ATTLIST Ref_Department.Manager ... >
    <!ELEMENT Ref_Department.Project EMPTY>
    <!ATTLIST Ref_Department.Project ... >
<!ELEMENT Project (Ref_Project.Employee+)>
<!ATTLIST Project ... >
    <!ELEMENT Ref_Project.Employee EMPTY>
    <!ATTLIST Ref_Project.Employee ... >
<!ELEMENT Contract (Ref_Contract.Company,
  Ref_Contract.Employee)>
<!ATTLIST Contract ... >
    <!ELEMENT Ref_Contract.Company EMPTY>
    <!ATTLIST Ref_Contract.Company
      Company IDREF #REQUIRED>
    <!ELEMENT Ref_Contract.Employee EMPTY>
    <!ATTLIST Ref_Contract.Employee
      Employee IDREF #REQUIRED>
```

3.3 Implementation

The XMI format (*XML Metadata Interchange*) makes it possible to represent an UML model in an XML format. This implementation is based on the XMI version 1.1 [7]. 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 they are MOF compliant (*Meta Object Facility*).

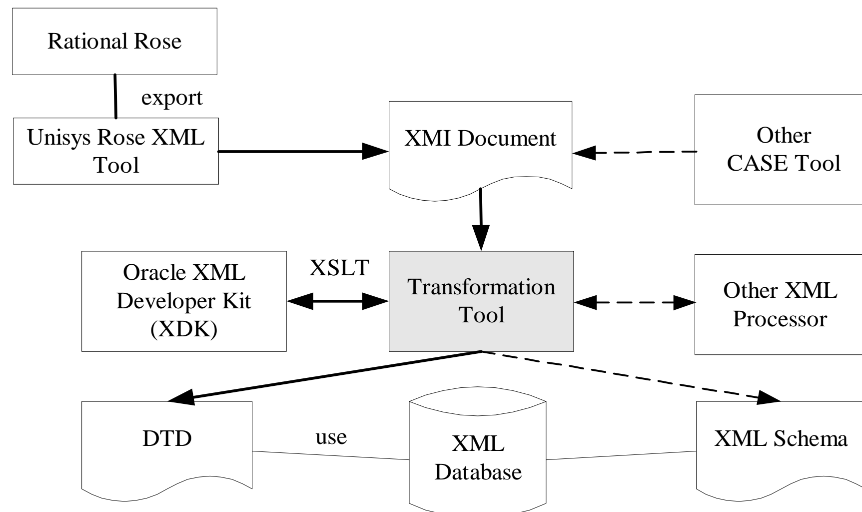

Figure 3: Overall structure of the DTD generation

UML class models are edited with the CASE tool Rational Rose. The model information can be stored in XMI documents using the *Unisys Rose XML Tool* 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 any other 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 this implementation, there are two categories of template rules: Some template rules have a pattern that must match with certain XMI elements that are relevant for the conceptual data model. Among them is the `UML:Class` template that transforms a UML class description into the corresponding element definition in the DTD. Some other templates are just auxiliary templates without matching XMI elements. Instead, they are invoked by other templates that make use of 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 | UML:Package

Because `UML:Model` and `UML:Package` elements have the same structure and are transformed the same way, they are combined in the same template. The `UML:Model`

element is the root element of the UML model and comprises all other elements like UML packages. For each package an element type is created. The name of the element corresponds with the complete package name. Possible stereotypes appear as a prefix before the package name. The definition of the package element is completed by the list of names of all packages, classes (that are not abstract and not parts of another class in a composition), association classes and n-ary associations that are the topmost level within the package. Afterwards, for each subelement of the package element the appropriate template is activated.

UML:Class

For each `UML:Class` element an element type is defined in the DTD. The name of the element corresponds with the class name. The name of a possible stereotype appears as prefix before the class name. Next, the content of the class element - i.e., all superclasses, complex attributes and associations - are extracted. They are determined by an XPath query on the XMI document. For example, the superclasses are represented in the `UML:Generalization.parent` element. The superclass is determined by the reference represented by the `xmi.idref` attribute. The superclass appears as a subelement of the current element. An element with the name of the attribute and the class name as a prefix before the attribute name is defined for all complex attributes. 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 the `Multiplicity` template. This template needs the name of the association element and the cardinality as parameters. The chosen naming convention for association elements is: `Ref_`*`Classname`*`.`*`Rolename`*. In the third step, all simple datatype attributes of a class are defined. Each class receives an ID attribute to make it a potential target of element references. The attributes of a class can be extracted from the `UML:Attribute` elements. Finally, the templates `createComplexAttributeElements` and `createAssociationElements` are called to define the necessary complex elements and reference elements with the IDREF attribute. Those reference elements have been included into the class definition and are defined by this template.

UML:AssocationClass

This algorithm transforms an association class into a class and two associations. So it works the same way as the `UML:Class` template. In addition, two or more associations have to be defined for each associated class involved in it. The attributes of association classes are treated like attributes of usual classes.

UML:Association

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

Multiplicity

Unlike the other templates above, this template does not process the content of the XMI document but the content of the three received parameters that control the transformation. They are: the name of the reference element of the association, the lower and the upper bound of the cardinality. The transformation of the cardinality is based on the rules represented in table 4. The repeated generation of `ref` attributes is realized by recursive calls of the template.

Cardinality	Result of Transformation
0..n	`ref*`
1..n	`ref+`
2..n	`(ref, ref+)`
0..1	`ref?`
1..1	`ref`
0..2	`(ref?, ref?)`
1..2	`(ref, ref?)`
m..n	`(ref, ref, ref?, ref?)` Example: m=2, n=4

Table 4: Transformation of cardinality constraints into DTD

createComplexAttributeElements

This template is called from the `UML:Class` and the `UML:AssociationClass` templates. It defines an element with the name of the attribute and the class name as a prefix before the attribute name for all complex attributes of a class. The content of this element is the element of the complex datatype.

createAssociationElements

This template is also called from both the `UML:Class` and the `UML:Association-Class` templates. It determines all associations of a class and defines the reference elements of an association. Those reference elements have been included into the DTD before (cf. `UML:Class` template). For each reference element an IDREF attribute is defined. The name of the attribute is composed of the name of the target class to be referenced. At association classes two more reference elements are generated for the newly created association.

Stereotype

The `Stereotype` template checks for stereotypes for all UML elements. Those 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 are 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 nesting elements, the use of parameter entities can be a viable alternative for an adequate representation in the DTD.

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 DTD. The following UML elements could not be represented in the DTD. Therefore they are not considered at the reverse transformation:

- stereotypes of associations, aggregations, compositions, generalizations
- name of associations, aggregations, compositions, generalizations
- dependencies
- type integrity of associations
- qualified associations
- data type of attributes

Dependencies have not been transformed because their definition bases mainly on the class behavior, which cannot be expressed in a DTD. In this implementation, the full syntax of the DTD has not yet been used. Among the elements that also should be included are entities and notations.

When transforming a conceptual data model into a DTD two fundamental drawbacks inherent to the DTD have to be dealt with:

DTD supports weak typing only. So only the CDATA type for strings is available. Numeric or other data types cannot be expressed adequately. Accordingly, the right type of document content cannot be guaranteed by an XML DBMS using DTDs.

Another serious drawback is the lacking type safety of references. Neither ID/IDREF nor XLink can define the target type of a reference. The only workaround used was a naming convention for elements and attributes to denote the original relationship. DTD cannot define elements with subelements in an arbitrary order. Furthermore, there are no object-oriented constructs such as generalization or any semantic relationships. The uniqueness constraint for key values cannot be enforced by a definition in a DTD.
Also Rational Rose has some limitations. Therefore it is not possible to define attributes with a multiplicity greater than 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. 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 this approach, the conceptual model and the XML representation of the document content were strictly separated. Therefore, XML specific constructs in the conceptual model are not involved as they can be found, e.g., in DTD profiles for UML [8] or XML extensions of the ER model [9]. This methodology is well-suited for the storage of data-centric 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 this transformation can easily be combined with an XML DBMS such as Tamino (by Software AG), which accepts DTDs as document schema.

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 this paper is the precondition to adapt this transformation tool to other target models as well. Currently we are working on mapping algorithms that produce an XML Schema definition as an output of the transformation process. For that purpose, some of these transformation rules have to be rewritten to consider the extended semantic capabilities beyond those of DTDs.

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] R. Conrad, D. Scheffner, J.C. Freytag: XML Conceptual Modeling Using UML,
Proc. Conceptual Modeling Conference ER2000, Salt Lake City, USA, Springer Verlag, 2000, pp. 558-571.

[3] 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.

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

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

[6] T. Krumbein: Logical Design of XML Databases by Transformation of a Conceptual Schema, Masters Thesis (in German), HTWK Leipzig, 2003, available at tkrumbe@imn.htwk-leipzig.de.

[7] OMG: XML Metadata Interchange,
http://www.omg.org/cgi-bin/doc?formal/00-11-02.pdf, 2001.

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

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