Producing XML Documents with Guaranteed “Good” Properties

David W. EMBLEY {embley@cs.byu.edu}
Department of Computer Science, Brigham Young University
Provo, Utah 84602, USA

and

Wai Y. MOK {mokw@email.uah.edu}
Department of Accounting and Information Systems, University of Alabama in Huntsville
Huntsville, Alabama 35899, USA

ABSTRACT

Many XML documents are being produced, but there are no agreed-upon standards formally defining what it means for complying XML documents to have “good” properties. Without guidance, users are likely to make poor choices and needlessly produce problematic specifications for XML documents. We therefore proposed a normal form for XML documents called XNF [2], which simultaneously guarantees that complying XML documents have maximally compact connectivity and that the data in complying XML documents cannot be redundant with respect to a set of common constraints. This paper builds on this work and shows show that although XNF allows alternative schematic specifications, we can generate XNF-compliant specifications that maintain the properties of XNF and also provide pleasing XML documents. The resulting XML documents are thus not only compact and free of data redundancy, but are elegant as well.

Keywords: XML Documents, Document Type Definition (DTD), XML Normal Form (XNF), Nested Normal Form (NNF), redundancy.

INTRODUCTION

Many DTDs (Document Type Definitions) for XML documents are being produced (e.g. see [6]), and soon many XML-Schema specifications [5] for XML documents will be produced. With the emergence of these documents, we should be asking the question, “What constitutes a good DTD?”[1] We argue that a “good” DTD should guarantee that all complying XML documents are in an agreed-upon form that has two desirable properties: (1) the DTD should have as few hierarchical trees as possible rooted just below the top-level node, and (2) at the same time, the DTD should not allow any of these trees to have redundant data values in XML documents that comply with the DTD. Intuitively, this should ensure that complying XML documents are compactly connected in as few hierarchies as possible while simultaneously ensuring that no data value in any complying document can be removed without loss of information.

Here, we build on our previous work in which we formalized these ideas as XNF, a normal form for XML documents [2].[2] In particular, we specify an algorithm for converting XNF scheme trees into sophisticated and pleasing DTDs. Naive conversions are straightforward, as we showed in [2], but these conversions are flat, nonintuitive, and make essentially no use of the richness provided by the DTD specification language.

We assume that the DTDs produced are for XML documents representing some aspect of the real world—those for which conceptual modeling makes sense.[3] Under this assumption, we argue that to produce a “good” DTD for an application A, we should (1) produce a conceptual-model instance for A, (2) apply a transformation guaranteed to produce an XNF scheme tree, and (3) convert the XNF scheme tree into an XNF-compliant DTD.[4]

[1]Since we do not address issues beyond hierarchical structure in this document, we discuss the issues in terms of DTDs rather than XML-Schemas. Further, since proposed specifications require XML-Schema to include full DTD expressibility, we do so without loss of generality.

[2]We based the original idea of XNF on nested normal form as defined in [4], which is also related to other similar nested normal forms such as those surveyed in [3].

[3]The class of documents we are considering is certainly large, but also certainly not all-inclusive. We do not, for example, consider text documents where the order of textual elements is important.

[4]Because of space limitations, we depend on examples to explain the first two steps and refer the interested reader to [2] for precise definitions and theorems guaranteeing correct generation of XNF scheme trees. Further, we focus here only on the most common and widely used case in practice—conceptual-model instances that include only binary relationship sets connecting object sets with mandatory or optional participation. For conceptual-model instances that include n-ary relationship sets ($n > 2$) and ISA hierarchies with union, mutual-exclusion, and partition constraints, see [2].

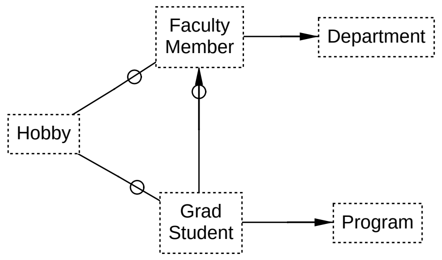

Figure 1: Simple Faculty-Student-Hobbies Diagram

We present our contribution as follows. Section provides examples that illustrate the definition and implications of XNF. Besides showing that we can produce DTDs that yield only XNF-complient XML documents, we also provide examples to show that even for simple applications, it is easy to produce (and thus nontrivial to avoid) DTDs that have redundancy and have more hierarchical clusters than necessary. Section then shows (1) that a straightforward but naive algorithm exists for converting XNF scheme trees into DTD specifications, but (2) that, with some effort, we can produce sophisticated and elegant DTD specifications. Having achieved our objective in this paper, we conclude in Section with a few short remarks.

XNF—BY EXAMPLE

As an example that illustrates the central idea of XNF, consider the conceptual-model diagram in Figure 1. Alghough based on [1], the conceptual modeling approach we present here is generic. Users model the real world by constraint-augmented graphs, which we call *CM graphs* (conceptual-model graphs). Vertices of CM graphs are object sets denoted graphically as named rectangles. The object set *Hobby* in Figure 1, for example, may denote the set {*Hiking*, *Sailing*, *Skiing*}. Edges representing functional relationships have arrowheads on the range side. In Figure 1, a graduate student enrolls in only one program (e.g. *PhD*, *MS*) and has only one faculty-member advisor. A connection between an object set and a relationship set may be optional or mandatory, denoted respectively by an "O" on the edge near the connection or the absence of an "O." A faculty member need not have hobbies and need not have advisees, but must be in a department.

An XML document has a hierarchical structure with a single root.[5] The set of structures immediately below the single root constitutes a forest of hierarchical trees. It is this forest of trees we wish to derive from a conceptual-model instance. We abstractly represent each tree in this forest as a *scheme tree*.

[5]We do not concern ourselves with XML documents that have no DTD or XML documents rooted at a DTD subelement.

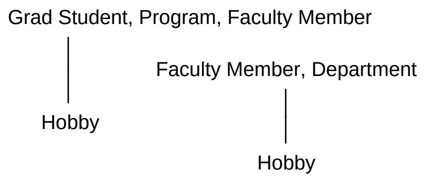
Grad Student, Program, Faculty Member
Hobby
Faculty Member, Department
Hobby

Figure 2: Sample Scheme-Tree Forest for the Diagram in Figure 1

Example 1 The two scheme trees in Figure 2 are for the model instance in Figure 1. Textually written, the scheme trees in Figure 2 are *(Grad Student, Program, Faculty Member, (Hobby)*)** and *(Faculty Member, Department, (Hobby)*)**. □

To explain XNF, we now give several more possible scheme trees that cover the CM graph in Figure 1. Some are in XNF and some are not.

Example 2 As examples of possible scheme-tree forests generated from the CM graph in Figure 1, consider the following.

1. *(Department, (Faculty Member, (Hobby)*, (Grad Student, Program, (Hobby)*)*)*)**
2. *(Faculty Member, Department, (Hobby)*, (Grad Student, Program, (Hobby)*)*)**
3. *(Hobby, (Faculty Member)*, (Grad Student)*)*; (Grad Student, Program, Faculty Member)*, (Department, (Faculty Member)*)**
4. *(Hobby, (Faculty Member, Department)*, (Grad Student, Program, Faculty Member)*)**
5. *(Faculty Member, Department, (Hobby, (Grad Student)*)*)*;*
   *(Grad Student, Faculty Member, Program)** □

The first two sample scheme-tree forests in Example 2 are in XNF—have a minimal number of scheme trees (only one) and can never have a redundant data value in any complying XML document. The third scheme-tree forest, as well as the scheme-tree forest in Figure 2, are not in XNF—although neither allows redundancy in any complying XML document, both have more than one scheme tree and thus neither is as compact as the first two sample scheme trees. The fourth sample scheme-tree forest is not in XNF because it allows redundancy—since faculty members and grad students are listed repeatedly for each additional *Hobby* in which they participate, department values for faculty members and program values as well as faculty advisors for grad students can be redundant. The fifth scheme-tree forest is also not in XNF both because it contains more than the minimum number of scheme trees needed to cover the CM graph and also because the first scheme tree allows redundancy—whenever faculty members have the same hobbies, all the graduate students that also share these hobbies are listed.

Example 2 illustrates that although we can guarantee XNF compliance, we cannot guarantee uniqueness. (Both (1) and (2) in Example 2 are in XNF.) In general, several "good" scheme trees, correspond to any given conceptual-model instance. Selecting the best depends on usage requirements and viewpoints that are "in the eye of the beholder." Sometimes these usage requirements or viewpoints should even cause the principles of XNF to be violated, but most of the time XNF-compliance should be compatible with usage requirements and viewpoints. Heuristic "rules of thumb" can go a long way toward resolving this problem of nonuniqueness and can often produce results that are highly satisfactory. We believe, however, that the ultimate resolution should be synergistic. Given heuristic rules and the principles of XNF, a system should work with a user to derive a suitable application scheme tree. The system can automatically derive reasonable XNF-compliant scheme trees. The user may adjust, reject, or redo any of the generated suggestions. The system can check the revisions and report any violations of XNF so that the user is aware of the consequences of the revisions. Iterating until closure is reached, the user can further revise the scheme trees, and the system can evaluate and provide feedback.

DTD GENERATION

Straigtforward But Naive

In addition to being able to derive multiple scheme-tree forests from a given CM graph, we can also derive a DTD from a forest of scheme trees in several ways. One straightforward approach is to use the scheme trees directly as DTD specifications as follows.

1. Select a name N for the root and generate $<!$ $DOCTYPE$ N [$<$ *scheme trees* $><$ *data elements* $>$] $>$.

2. Make each object-set name that appears more than once in the scheme-tree forest unique, for example, by appending a "_2" to the second, a "_3" to the third, etc.

3. Replace $<$ *scheme trees* $>$ by $<$ $!$ $ELEMENT$ N ($<$ *the generated scheme − tree forest* $>$) $>$ where N is the selected name for the root and $<$*the generated scheme-tree forest*$>$ is a comma-separated list of textual representations of the scheme trees in the scheme tree forest.

4. Replace $<$ *data elements* $>$ by a sequence of $<$ $!$ $ELEMENT$ N_i ($\#PCDATA$) $>$, one for each object-set name N_i including object-set names with appended numbers (or with other uniqueness alterations).[6]

Example 3 Figure 3(a) shows a DTD generated directly from the two scheme trees in Figure 2, and Figure 3(b) shows a sample complying XML document.[7] □

Sophisticated and Elegant

Unfortunately, this straightforward approach of directly using scheme trees as DTD specifications does not make good use of nesting, grouping, XML attribute declarations, optionals, or renaming possibilities. Observe in Figure 3, for example, that all elements are on the same level of nesting since neither *Grad_Student* elements nor *Faculty_Member_2* elements enclose their respective subcomponents. We therefore provide a more satisfactory derivation, as follows.[8]

1. For each node of each tree:
   (a) Find the candidate keys of the node.
   (b) Choose a noncomposite primary key, if one exists; otherwise create an object set and make it the primary key of the node.
   (c) Create a name for the node.
   (d) The node now has the form N P N_1 ... where N is the name created for the node, P is the primary key, and N_1 ... is the list of additional object-set names, if any.
      i. Generate $<!ELEMENT\ N\ (P)^* >$.
      ii. If the list N_1 ... is not empty or if the node has children, generate $<$ $!ELEMENT$ P (N_1, ..., C_1, ...) $>$ where C_1, ... is the list of names created for the children nodes. Further, generate $<!ATTLIST$ P $value$ $CDATA$ $\#REQUIRED$ $>$.[9] Finally, for each N_i generate $<!ELEMENT$ N_i ($\#PCDATA$) $>$.

[6]Here, and throught our discussion, when we generate object-set names as DTD elements, we replace spaces by underscore characters. We also appropriately replace any other illegal symbols.

[7]Although not in XNF, we nevertheless use the two scheme trees in Figure 2 in order to illustrate the wider variety of possibilities in generating DTDs without having to introduce a new, larger example. We also use this example in our sophisticated DTD generation for the same reason.

[8]Other derivations are also possible, but this one appears to be the most elegant among those we have investigated.

[9]If P was created and has no natural string value, we can always use simple integers as OIDs for values.

```
<!DOCTYPE University[
<!ELEMENT University (
          (Grad_Student, Program, Faculty_Member,
               (Hobby)* )*,
          (Faculty_Member_2, Department,
               (Hobby_2)* )* )>
     <!ELEMENT Grad_Student (#PCDATA)>
     <!ELEMENT Program (#PCDATA)>
     <!ELEMENT Faculty_Member (#PCDATA)>
     <!ELEMENT Hobby (#PCDATA)>
     <!ELEMENT Faculty_Member_2 (#PCDATA)>
     <!ELEMENT Department (#PCDATA)>
     <!ELEMENT Hobby_2 (#PCDATA)>
]>
```

(a) Sample DTD Specification

```
<University>
     <Grad_Student>Pat</Grad_Student>
          <Program>PhD</Program>
          <Faculty_Member>Kelly</Faculty_Member>
          <Hobby>Hiking</Hobby>
          <Hobby>Skiing</Hobby>
     <Grad_Student>Tracy</Grad_Student>
          <Program>MS</Program>
          <Faculty_Member>Kelly</Faculty_Member>
          <Hobby>Hiking</Hobby>
          <Hobby>Sailing</Hobby>
     <Grad_Student>Chris</Grad_Student>
          <Program>MS</Program>
          <Faculty_Member>Kelly</Faculty_Member>
     <Faculty_Member_2>Kelly</Faculty_Member_2>
          <Department>CS</Department>
          <Hobby_2>Hiking</Hobby_2>
          <Hobby_2>Skiing</Hobby_2>
     <Faculty_Member_2>Noel</Faculty_Member_2>
          <Department>Math</Department>
          <Hobby_2>Sailing</Hobby_2>
</University>
```

(b) Sample XML Document

Figure 3: Directly Generated DTD with a Complying XML Document

iii. Otherwise generate $<$ $!ELEMENT\ P\ (\#PCDATA) >$.

2. For names of nodes and object sets that appear more than once:
   (a) If the names have the same structure, do nothing (except discard any duplicate DTD specifications).
   (b) Otherwise, rename at least all but one to distinguish the cases.
3. If there is more than one tree:
   (a) Generate $<!DOCTYPE\ N\ [\ <\ !ELEMENT\ N\ (N_1,\ ...\ )\ >$ $<$ *DTD specification* $>$] $>$ where N is a uniquely chosen name, N_1, ... is the list of created (possibly renamed) root-node names for the trees, and $<$ *DTD specification* $>$ is the generated DTD specification.
   (b) Otherwise generate $<!DOCTYPE\ N$ [$<$ *DTD specification* $>$] $>$ where N is the name created for the root node and $<$ *DTD specification* $>$ is the generated DTD specification.

Example 4 Figure 4(a) shows a DTD generated in this more sophisticated way from the scheme trees in Figure 2, and Figure 4(b) shows a sample complying XML document. In generating this DTD, (1) we took the only candidate key from each node as the primary key, namely *Grad Student*, *Faculty Member*, and *Hobby*, (2) we created the node names *Grad Students*, *Faculty Members*, and *Hobbies* using plurals of the primary-key names, (3) we renamed *Faculty Member*, which appears in both subtrees in Figure 2, to be *Advisor* in the subtree for *Grad Student*, and (4) we noted that the hobby structures are identical in both trees so that we needed to declare the structure only once. □

CONCLUDING REMARKS

We have previously proposed and formally defined XNF (XML Normal Form), which guarantees that complying XML documents have no redundant data values and have maximally compact connectivity [2]. We have also developed conceptual-model-based algorithms to generate DTDs to ensure that complying documents are in XNF, and we have proved that these algorithms are correct [2]. Conversion to elegant XML documents, however, was more challenging that originally expected. This paper therefore answers this challenge by showing how to produce XML documents that are not only in XNF—compact and free of data redundancy—but are elegant as well.

Acknowledgements: This material is based upon work supported by the National Science Foundation under grant No. IIS-0083127.

REFERENCES

References

[1] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach.* Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[2] D.W. Embley and W.Y. Mok. Developing XML documents with guaranteed ‘good’ properties. In

Proceedings of the 20th International Conference on Conceptual Modeling (ER2001), pages 426–441, Yokohama, Japan, November 2001.

[3] W.Y. Mok. A comparative study of various nested normal forms. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):369–385, 2002.

[4] W.Y. Mok, D.W. Embley, and Y-K. Ng. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems*, 21(1):77–106, March 1996.

[5] XML schema part 0: Primer: W3c working draft, 2000. URL: http://www.w3.org/TR/2000/WD-xmlschema-0-20000407/.

[6] Home Page for a listing of organizations producing industry-specific XML DTDs. URL: http://xml.org/xmlorg_registry/index.html.

```
<!DOCTYPE University[
<!ELEMENT University
        (Grad_Students, Faculty_Members)>
    <!ELEMENT Grad_Students (Grad_Student)*>
        <!ELEMENT Grad_Student
                (Program, Advisor, Hobbies)>
            <!ATTLIST Grad_Student
                    value CDATA #REQUIRED>
            <!ELEMENT Program (#PCDATA)>
            <!ELEMENT Advisor (#PCDATA)>
            <!ELEMENT Hobbies (Hobby)*>
                <!ELEMENT Hobby (#PCDATA)>
    <!ELEMENT Faculty_Members (Faculty_Member)*>
        <!ELEMENT Faculty_Member
                (Department, Hobbies)>
            <!ATTLIST Faculty_Member
                    value CDATA #REQUIRED>
            <!ELEMENT Department (#PCDATA)>
]>
```

(a) Sample DTD Specification

```
<University>
    <Grad_Students>
        <Grad_Student value=”Pat”>
            <Program>PhD</Program>
            <Advisor>Kelly</Advisor>
            <Hobbies>
                <Hobby>Hiking</Hobby>
                <Hobby>Skiing</Hobby>
            </Hobbies>
        </Grad_Student>
        <Grad_Student value=”Tracy”>
            <Program>MS</Program>
            <Advisor>Kelly</Advisor>
            <Hobbies>
                <Hobby>Hiking</Hobby>
                <Hobby>Sailing</Hobby>
            </Hobbies>
        </Grad_Student>
        <Grad_Student value=”Chris”>
            <Program>MS</Program>
            <Advisor>Kelly</Advisor>
            <Hobbies>
            </Hobbies>
        </Grad_Student>
    <Grad_Students>
    <Faculty_Members>
        <Faculty_Member value=”Kelly”>
            <Department>CS</Department>
            <Hobbies>
                <Hobby>Hiking</Hobby>
                <Hobby>Skiing</Hobby>
            </Hobbies>
        </Faculty_Member>
        <Faculty_Member value=”Noel”>
            <Department>Math</Department>
            <Hobbies>
                <Hobby>Sailing</Hobby>
            </Hobbies>
        </Faculty_Member>
    </Faculty_Members>
</University>
```

(b) Sample XML Document

Figure 4: A More Sophisticated DTD with a Complying XML Document