

# Constraint Specification for XML: A Closer Look

Murali Mani

Department of Computer Science  
Worcester Polytechnic Institute, USA  
{mmani}@cs.wpi.edu

**Abstract.** The XML database community felt the necessity for incorporating explicit constraint specification schemes for XML, so that XML may be used for database applications. This was one of the reasons for a new schema language standard called W3C XML-Schema, in place of DTD, the original standard. However, several fundamental problems exist in XML constraint specification today, which are yet to be solved. This paper is an attempt to investigate answers to some of these questions. We study explicit constraint specification requirements for XML using the following assumption: An XML schema language should provide the ability to specify a normalized XML schema to model any database application scenario. Using this approach, we are able to systematically come up with a constraint specification scheme for XML.

## 1 Introduction

Over the last few years, XML (eXtensible Markup Language) [BPS00] published by W3C has been widely used for modeling applications of the web, including text applications, and database applications. In this work, we focus on XML and database applications. Database applications use XML primarily in one of the following two ways: (a) A database application is modeled in XML, and the resulting data is stored in native XML or in relational/object-relational databases (b) An XML view is published over a relational/object-relational data source; queries and updates are performed against this XML view and translated into the underlying data sources. Both these application scenarios require explicit constraint specification for XML: in (a), we need to express application semantics, and in (b), we need to expose the application semantics to the user application who sees only the XML view.

A first requirement in this study is to formalize what database applications are. Our formalism is based on the following assumption: *a database application is one that can be modeled by Entity Relationship (ER) model [Che76].* ER schemas when translated to relational schema, make use of the structures (relations and attributes) and explicit constraints (key and foreign key constraints). Translation of ER schemas to XML schemas has been studied in [Man03]; the lessons learnt there has greatly influenced this work.

A second formalism that is required is what is a good schema language. We base this definition on normalization as defined in [AL03], where a normalized

schema is defined as one with no redundancy as: Consider all possible instances of the schema. If for any instance, any value can be determined by the other values in the instance, then that implies that there exists redundancy in that instance, and hence the schema is not normalized. Examples of normalization of relational schema are shown in Table 1. Note that the final schema used by the application could be unnormalized. However, *we require that a good schema language allow us to specify a normalized schema for any application scenario.*

Student			S1		S2	
sName	pName	pRes	sName	pName	pName	pRes
A	X	DB	A	X	X	DB
B	X	DB	B	X		

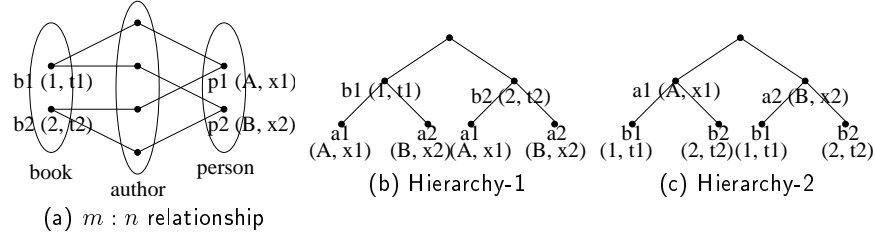
(a) Unnormalized relational schema      (b) Normalized relational schema

There exists FD:  $pName \rightarrow pRes$       corresponding to (a)

**Table 1.** (a) shows an unnormalized schema, we can remove one value “DB” and because of the FD, we can still determine the value. (b) shows the result of classical relational normalization

Let us first try to see why we need explicit constraints in XML. In XML, structures are specified as hierarchies. Hierarchies capture  $1 : n$  relationships implicitly, for example, if a professor has students as children, that implies a  $1 : n$  relationship. However, pure hierarchies are insufficient to come up with a normalized schema for any scenario [TL82]. This can be illustrated with the following example: Consider two entity types *book* and *person*, and a relationship type, *author*, which represents the persons who authored the books as shown in Figure 1. Also suppose *author* is a  $m : n$  relationship type; in other words, a person could have authored many books, and a book could have been authored by many persons. Figure 1 illustrates that there could be redundancy if we represent this relationship using hierarchies. To avoid such redundancy, XML needs to support explicit constraint specification. In this paper, we systematically discuss and develop an explicit constraint specification scheme for XML.

The paper is outlined as follows. In Section 2, we describe the background required. First we discuss XGrammar, based on regular tree grammars [MLM01], which is used to specify structures in XML. We further describe database concepts required such as functional dependencies, examine how constraints have been traditionally studied in relational and ER context, and how the constraint specification in XML differs from them. In Section 3, we study the various requirements and issues in XML constraint specification. In Section 4, we come up with an explicit constraint specification scheme for XML. We finally conclude with future work in Section 5.



**Fig. 1.** (a) shows example instance of  $m : n$  relationship *author* between *book* and *person*, each with two attributes, and with the first attribute as the key. Book *b1* is written by *p1* and *p2*, so is *b2*. (b) shows *author* relationship represented as *person* being child of *book*; there is redundancy, we can remove one of the  $x1$  values, and still determine it from other values. (c) shows *author* represented as *book* is child of *person*; there is redundancy, we can remove one of the  $t1$  values, and still determine it from other values.

## 2 Background

In this section, we introduce background material required. First we describe how structures are specified in an XML schema language. We will call our XML Schema language as XGrammar. XGrammar is based on regular tree grammar theory, and is an abstraction of three standards: DTD, W3C XML-Schema and RELAX-NG [MLM01]. We then define functional dependencies for the sake of completeness, and then study differences between constraint specification in ER and relational models and constraint specification requirements in XML.

### 2.1 XGrammar

We use  $\mathbb{G}$  to denote an XGrammar and  $L(\mathbb{G})$  to denote the language that  $\mathbb{G}$  generates. We assume the existence of a set  $\widehat{N}$  of non-terminal symbols, a set  $\widehat{E}$  of element names, a set  $\widehat{A}$  of attribute names, and a set  $\widehat{\tau}$  of atomic data types as defined in [BM01], such as string, integer, ID, IDREF(S) etc. We use the following notations:  $\epsilon$  denotes the empty string,  $+$  denotes union, “ $\cdot$ ” denotes concatenation, “ $a^?$ ” denotes zero or one occurrence, “ $a^*$ ” denotes Kleene star, and “ $a^+$ ” denotes “ $a, a^*$ ”. Determining whether a document  $D \in L(\mathbb{G})$  can be done in time linear with respect to the size of the document [MLM01].

**Definition 1** (*XGrammar*) An XGrammar is denoted by a 6-tuple  $\mathbb{G} = (N, E, A, S, P, \Sigma)$ , where

- $N$  is a finite set of non-terminal symbols, where  $N \subseteq \widehat{N}$ .
- $E$  is a finite set of element names, where  $E \subseteq \widehat{E}$ .
- $A$  is a finite set of attribute names, where  $A \subseteq \widehat{A}$ .
- $S$  is the set of start symbols, where  $S \subseteq N$ .
- $P$  is the set of production rules of the form  $X \rightarrow x (RE)$ , where  $X \in N$ ,  $x \in E$ , and  $RE$  is:  
 $RE ::= \epsilon \mid \tau \mid @a \mid Y \mid (RE + RE) \mid (RE, RE) \mid (RE)^? \mid (RE)^* \mid (RE)^+$ ,  
where  $\tau \in \widehat{\tau}$ ,  $a \in A$ ,  $Y \in N$ .

- $\Sigma$  is the set of constraints, which include:
  - Attribute type. For any attribute  $a \in A$ , we define the type  $\tau \in \hat{\tau}$  for it. (other explicit constraint specification will be added in Section 4).

**Determining type for an element.** An important problem that impacts the constraint specification problem is that of determining the type for an element (also called interpretation in [MLM01]). Given a document  $D \in L(\mathbb{G})$ , we say that the type for an element (subtree) say  $t$  is  $X \in N$ , if  $D$  can be generated from  $\mathbb{G}$  such that  $t$  is generated from  $X$ . A significant difference in XML from traditional data models and programming languages is that the type for an element  $t$  may be ambiguous; in other words, it is possible for  $t$  to be generated from multiple non-terminals. We will see in Section 3, how this potential ambiguity influences the constraint specification scheme. It is possible to impose different restrictions as to what subset of regular tree grammars are valid schemas, as done in DTD and W3C XML-Schema, to ensure that there is no ambiguity. However RELAX-NG, a widely used XML schema language standardized under ISO, imposes no such restrictions. Therefore a document valid with respect to a RELAX-NG schema may have elements whose type may be ambiguous.

**Example 1** An example XGrammar, and a document valid with respect to this XGrammar is given in Table 2. We show the attribute type constraints only for the interesting cases: attributes of type ID or IDREF (S). Note that for this example, given the document and the XGrammar, we can identify the types for each element in the document unambiguously. For example, the type for both the elements with name *person* is *Person*.  $\square$

$  \begin{aligned}  N &= \{Library, Person, Book, Paper, Review\} \\  E &= \{library, person, book, paper, review\} \\  S &= \{Library\} \\  P &= \{Library \rightarrow library (Person^*), \\  &\quad Person \rightarrow person (@name, \\  &\quad\quad ((@city, @state) + @zip), \\  &\quad\quad (Book^* + Paper^*), Review^*), \\  &\quad Book \rightarrow book (@btitle, @ISBN, \\  &\quad\quad @year?, @BID), \\  &\quad Paper \rightarrow paper(@ptitle, @year?, \\  &\quad\quad @journal?, @PID), \\  &\quad Review \rightarrow review(@art, @rating)\} \\  \Sigma &= \text{Attribute type constraint: } \{@BID::ID, \\  &\quad @PID::ID, @art::IDREF\}  \end{aligned}  $	<pre> &lt;library&gt;   &lt;person name='A'     city='LA' state='CA'&gt;     &lt;book btitle='T1' ISBN='I1'       BID='B1'&gt;     &lt;book btitle='T2' ISBN='I2'       BID='B2'&gt;     &lt;review art='P1' rating='9'&gt;     &lt;review art='B2' rating='9'&gt;   &lt;/person&gt;   &lt;person name='B' zip='90095'&gt;     &lt;paper ptitle='T3' PID='P1'&gt;     &lt;review art='B1' rating='9'&gt;     &lt;review art='B2' rating='10'&gt;   &lt;/person&gt; &lt;/library&gt; </pre>
--	--

**Table 2.** Example XGrammar and a valid XML document

**XPath Expressions** The operations we will consider on the XML model are XPath expressions [W3Cb] (subset of full XPath syntax). The subset we consider include the following three axes: child, attribute and parent. Some example path expressions and the results for the document in Table 2 are shown below.<sup>1</sup>

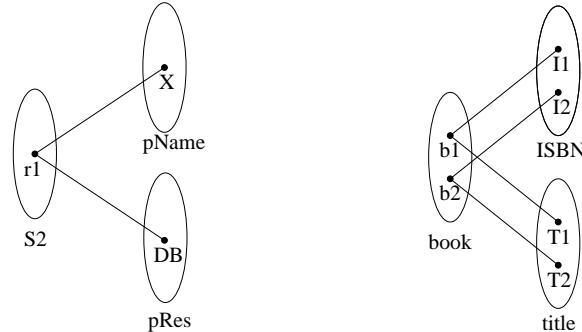
$$\begin{aligned} person/@name &= \{A, B\} \\ person/review/@art &= \{P1, B1, B2\} \\ review/parent :: person/@name &= \{A, B\} \end{aligned}$$

## 2.2 Functional Dependencies

Functional dependencies (FDs) for the relational model are well known [GMUW02]; let us look at an example. Consider the relational schema shown in Table 1. We say that the schema satisfies the FD:  $pName \rightarrow pRes$ , if in any instance of the schema, if there are two rows with the same value for  $pName$ , they will have the same value for  $pRes$  (in other words,  $pName$  determines  $pRes$ ).

Keys are defined in the relational model using FDs. Consider a table  $R$  with attributes  $A_R$ . We say that the key for  $R$  is  $K$  if  $K \rightarrow A_R$ .<sup>2</sup> For example, in Table 1 (b), the key for table  $S2$  is  $pName$ , as shown in Figure 2 (a).

FDs have been defined for the XML model as well in [AL02]. For example, for books in the XGrammar in Table 2, we have the FD  $ISBN \rightarrow btitle$ . Keys for XML are also defined based on FDs. For example, we say that the key for the set of books in the XGrammar in Table 2 is  $ISBN$ , if we have the FD:  $ISBN \rightarrow book$ . Note the difference between the relational and the XML models: we consider each book “object” (element) to have an id, and we want the key to determine this ID (the key may or may not determine the other values).



(a) Example instance of relational Keys (b) Example instance of XML Keys

**Fig. 2.** Illustrating Relational and XML Keys

<sup>1</sup> In XPath syntax, they would have a // appended as a prefix to each of the expressions, we omit them for convenience. Further the order of the results of a path expression as well as whether a path expression returns duplicate values are insignificant and do not impact our results; hence they will be ignored.

<sup>2</sup> We also require that there exists no  $K' \subset K$ , such that  $K' \rightarrow A_R$ .

### 2.3 Key Specification for Relational/ER models Vs. XML

The key specification for relational/ER models are much simpler than that of XML because in relational/ER models, any attribute is required to be simple (and not a collection of values). However, as shown in Figure 3 (a), in XML, we can have an attribute having multiple values (an IDREFS attribute or a repeating subelement). Note here that we are defining the keys for  $O$  elements. Each  $O$  element has two attributes:  $A$  and  $B$ ; and an  $O$  element can have a collection of values for  $B$ . Therefore the FD  $A \rightarrow B$  is false. However, we have the FD  $A \rightarrow O$ , and hence  $A$  is a key for the  $O$  elements. This definition is used in all the XML constraint specification schemes [W3Ca,BDF<sup>+</sup>02,FKS02,AL02].

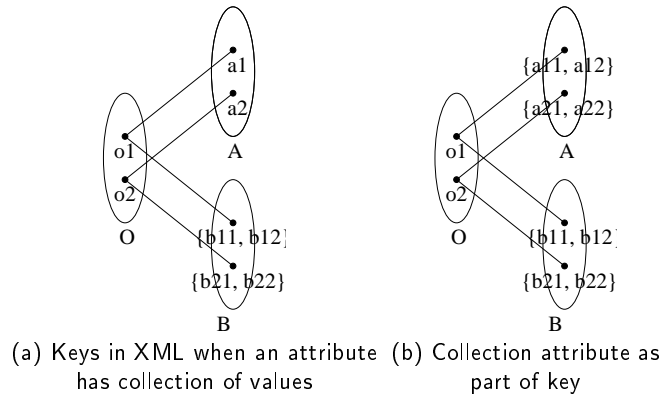


Fig. 3. Some Fundamental Differences between Relational and XML Keys

Another significant difference when we consider keys for XML is whether an attribute that forms part of a key can have a collection of values, and if so, what is the semantics. Figure 3 (b) shows this scenario, which does not occur in ER/relational models. The question is whether  $(A, B)$  is a key for  $O$ . We answer this question by considering the tuples formed by  $O \times A \times B$ , and considering whether in the above set of tuples, the FD:  $(A, B) \rightarrow O$  is true.

Let us look at an example, consider the XML schema and instance in Table 2. Consider the *person* elements; the question is whether  $\langle review/@art, review/@rating \rangle$  is a key for *person*. There are two semantics that have been used in current literature as shown in Figure 4. According to [BDF<sup>+</sup>02,FKS02], the FD:  $\langle review/@art, review/@rating \rangle \rightarrow person$  does not hold over the instance; whereas according to [AL02], the FD:  $\langle review/@art, review/@rating \rangle \rightarrow person$  holds. W3C XML-Schema prohibits any key attribute to have a collection of values. We will use the semantics as defined in [BDF<sup>+</sup>02,FKS02].

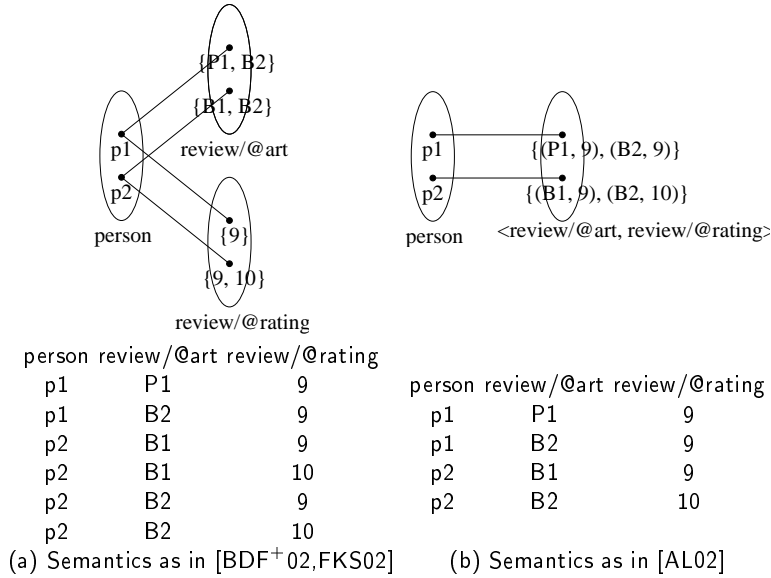


Fig. 4. Semantics for XML Keys when a key attribute has a collection of values

### 3 Issues in XML Constraint Specification

In the previous section, we defined keys for a set of elements in an XML document using FDs. It is well accepted in XML circles that a “key attribute” (also called “field”) is specified as a path expression. For example, we could specify that the key for a set of elements is  $\langle a/b/@c, x/@y \rangle$ . We also discussed the semantics of keys when key attributes could have a collection of values. In this section, we further discuss various other issues in XML constraint specification.

**Path-Based vs Type-Based Constraint Specification** One important question that needs to be answered is how to specify the set of elements for which the key is specified. There are two options: (a) *using a path expression*: If we use this semantics, the path expression returns a set of elements in the document, and the key is defined for this set of elements (b) *using a type*: If we use this semantics, the key is defined for the set of elements that belong to a type. Path expressions are used in [W3Ca,BDF<sup>+</sup>02], and types are used in [FKS02].

Let us consider some examples as to when this difference could be significant. Consider a recursive schema that describe papers, their authors, and their citations, given by the following production rules:  $Paper \rightarrow paper(@ptitle, Person^*, Paper^*)$ . We can specify the key for all the papers in the document is given by  $ptitle$  if we specify the key using types as: key for the *Paper* type is  $\langle @ptitle \rangle$ . To specify a similar constraint using path expressions, we need to expand our path expression syntax to have a recursive axis, such as the descendant axis.

Let us consider another example, that describes books, papers and their authors, given by the production rules:  $Library \rightarrow library(Book^*, Paper^*)$ ,  $Book \rightarrow book(@ISBN, Person^*)$ ,  $Paper \rightarrow paper(@ptitle, Person^*)$ ,  $Person \rightarrow person(@name)$ . Suppose we want to specify a constraint such as: key for the path expression  $library/book/person$  is  $\langle @name \rangle$ . This cannot be specified using types, unless we introduce subtyping, and define new types.

The two examples show that in terms of expressive power, both types and path expressions are not strictly more expressive than each other. However, there is another significant observation. *We can specify key constraints on types only when the types for the elements can be determined unambiguously, there is no such requirement if we specify keys on path expression.* This implies that we can have type-based constraint specification for schema languages like DTD, and W3C XML-Schema, and not for schema languages like RELAX-NG. However, path-expression-based constraint specification can be used with any of the XML schema languages, DTD, W3C XML-Schema or RELAX-NG.

**Relative keys** The concept of relative keys is defined in [W3Ca,BDF<sup>+</sup>02], whereas relative keys are not used in [FKS02]. Let us first try to define relative keys. A relative key is specified as: the key for  $(r, s)$  is given by  $\langle f \rangle$ . Here  $f$  is the set of “key attributes” (fields) specified as path expressions;  $r$  is the relative axis, and  $s$  is the selector axis. The semantics for such a key is as follows: consider a set of elements  $P$  that “result” from  $r$ . For each element  $p$  in  $P$ ,  $s$  will again produce a set of elements  $S_p$ . Key for this set of elements is given by  $\langle f \rangle$ .

Let us look at an example. Consider the XML schema given by the production rules:  $Library \rightarrow library(Book^*)$ ,  $Book \rightarrow book(Person^*)$ ,  $Person \rightarrow person(@name)$ . Suppose the key for persons within a book is given by the name. We can specify this as key for  $(Book, Person)$  is given by  $\langle @name \rangle$ . Note that  $r$  and  $s$  can be specified using path expressions or types. The relative axis is specified using types in [W3Ca], and using path expressions in [BDF<sup>+</sup>02].

It is possible to define relative keys “without using a relative axis”. For the above example, we can say the key for  $Person$  is given by  $\langle parent :: book, @name \rangle$ . In other words, the main idea behind relative keys is that a key attribute (field) returns a set of objects (rather than values). For example,  $parent :: book$  returns a set of  $book$  objects, whereas other key attributes like  $@name$  returns a set of values. Let us examine this more closely.

One of the reasons we define keys is for specifying integrity constraints on the data. Another equally important usage is that keys are used to refer to objects (typically by foreign keys). For example, consider the XML schema given by the production rules:  $Library \rightarrow library(Book^*, Person^*)$ ,  $Book \rightarrow book(@ISBN, @author)$ ,  $Person \rightarrow person(@name)$ . Suppose we have specified the key for  $Person$  is  $\langle @name \rangle$ . Now we want to specify that the authors of books must refer to persons. This is typically done in relational model using foreign key, such as  $Book(@author) \text{ REFERENCES } Person(@name)$ . In XML also we will use foreign keys, however relative keys impose several restrictions on usage of foreign keys.

Suppose we have a relative key specified as: key for  $(r, s)$  is  $\langle f \rangle$ . Now in order to identify an  $s$  object, we need to specify both the value of the key and the  $r$  object. Specifying the  $r$  object is not easy. In W3C XML-Schema,  $r$  is indirectly specified by requiring that the foreign key is specified for elements within  $r$ . Let us look at an example XML Schema defined by the following production rules:  $Root \rightarrow root(R^*), R \rightarrow r(T^*, S^*), T \rightarrow t(@fk), S \rightarrow s(@k)$ . An instance is shown in Figure 5. Suppose the key is defined as key for  $(R, S)$  is  $\langle @k \rangle$ . The instance shown satisfies the key constraint. Now we define the foreign key for  $T$  within  $R$  as:  $T(@fk) \text{ REFERENCES } S(@k)$ . We see that  $t1$  references  $s1$  and  $t2$  references  $s3$ . Another approach for solving this issue, which is currently being discussed by the W3C XML-Schema WG, is that of “chaining foreign keys”; however the discussion of this concept is not significant, and hence omitted.

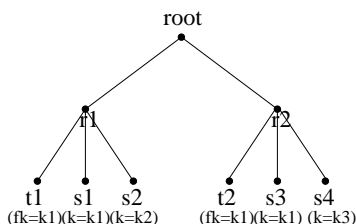


Fig. 5. Illustrative relative keys and foreign keys.

**IDREF and IDREFS vs Foreign Keys** It is shown in [Man03] that any ER schema can be modeled as an XML schema, where the XML schema language is XGrammar extended with key and foreign key constraints. An outline of the reasoning is: any ER schema can be modeled as a relational schema, and a relational schema can be translated naively into XML using XGrammar extended with key and foreign key constraints as shown in several works including [LMCC02]. However, XML traditionally has supported other kinds of constraints called IDREF and IDREFS constraints. We know that these constraint specification schemes are not really needed; however let us still study them closely.

An IDREF constraint is similar to foreign keys. For specifying an IDREF constraint, we first define an ID attribute (similar to keys), and then we can define IDREF attributes. Let us look at an example scenario of books, libraries and authors, that we saw in the previous section:  $Library \rightarrow library(Book^*, Person^*), Book \rightarrow book(@ISBN, @PersonRef), Person \rightarrow person(@name, @PersonID)$ . We define the attribute type constraints as:  $@PersonID :: ID, @PersonRef :: IDREF$ , and a key constraint: key for  $Person$  is  $(@name)$ . Note that the same scenario can be modeled using foreign keys in stead of IDREF attribute, as shown before.

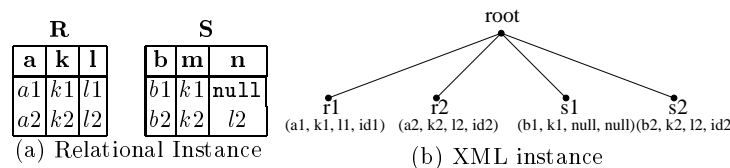
Let us study some differences between IDREF attributes and foreign keys. Similar to joins used for “traversing” foreign keys, we can traverse ID and IDREF attributes using functions:  $fn:idref$  and  $fn:id$  respectively provided

by XQuery [W3Cb]. We may also use joins also, if needed. For example, to find the names of authors of a book with ISBN= $i1$ , we can write a path expression as:  $library/book[@ISBN = i1]/@PersonRef/fn : id()/@name$  (or)  $library/person[@PersonID/fn : idref()/@ISBN = i1]/@name$ . In other words, IDREF assists in specifying queries using “navigation” rather than joins.

Another difference between IDREF and foreign keys is that IDREF attributes can be used as a “union type of foreign keys”. Let us illustrate this with an example. Consider an XML schema with the following production rules:  $Library \rightarrow library(Book^*, Paper^*, Person^*)$ ,  $Book \rightarrow book(@ISBN, @BookID)$ ,  $Paper \rightarrow paper(@title, @PaperID)$   $Person \rightarrow person(@name, @article)$ . Let the attribute type constraints be:  $BookID :: ID$ ,  $PaperID :: ID$ ,  $article :: IDREF$ . Now the article attribute in  $Person$  can refer to either a book or a paper. To represent this using foreign keys, we need two foreign key attributes for  $Person$ , one referencing books, and the other referencing papers.

Another question that one might want to ask is whether we can do away with foreign keys. It was shown in [LMCC02], that in order to translate relational schemas to XML schemas, there are cases when IDREF is insufficient, and we need foreign keys. This can be illustrated with the following example: Consider a relational schema with 2 tables as  $R(a, k, l)$ , and  $S(b, m, n)$ ; with key constraint: key for  $R$  is  $(k, l)$  and foreign key constraint:  $S(m, n)$  REFERENCES  $R(k, l)$ . A relational instance of this is shown in Table 3 (a).

Let us assume that the foreign key attributes  $m$  or  $n$  can take null values. Now, an XML schema that represents the foreign key constraint using hierarchies, or using IDREF attributes will have redundancy<sup>3</sup>. For example, let us consider the XML schema using IDREF attributes as:  $Root \rightarrow root(R^*, S^*)$ ,  $R \rightarrow r(a, k, l, RID)$ ,  $S \rightarrow s(b, m, n, RRef)$ . If we consider the XML instance shown in Table 3 (b), we can see that for  $s_2$ , we can remove the value of the  $m$  attribute, and because of the value of the  $RRef$  attribute, we can still determine that the value of the  $m$  attribute must be  $k_2$ .



**Table 3.** This figure shows IDREF attribute resulting in redundancy. We can remove  $k_2$  from  $s_2$  and still determine this value.

In order to make use of IDREF attribute similar to that of foreign keys, schema languages like W3C XML-Schema allow IDREF attributes to specify the types that the IDREF attribute should refer to. IDREFS attribute is similar to

<sup>3</sup> There is an assumption here that any non-terminal symbol will appear on the RHS of at most one production rule.

that of specifying content of an element, and hence we can specify the target types that an IDREFS attribute may also refer to.

## 4 Constraint Specification Scheme for XGrammar

In the previous section, we discussed several issues to consider when coming up with a constraint specification scheme for XML. Let us first revisit them and examine the choices we will make for our scheme. The first choice is whether we should use type-based or path-expression-based constraint specification scheme; we will use type-based scheme, for the reasons as described in [FKS02]; ER schemas can be translated into XML schemas with type-based constraint specification scheme. This imposes a restriction on our schema language as mentioned in Section 3. We cannot use a full-fledged regular tree grammar based language such as RELAX-NG, in stead we need to impose restrictions. We require that our XGrammar be a *single-type tree grammar* [MLM01]. Roughly speaking, a single-type tree grammar requires for any element name, say  $e \in E$  of possible children of a non-terminal symbol, say  $X \in N$ , there should be only one possible non-terminal symbol “corresponding to  $e$ ” that appears in the production rule for  $X$ . This grammar class corresponds to W3C XML-Schema.

Secondly, we will not allow relative keys. Similarly we will not allow a key attribute (field) to specify a path expression that returns a set of elements (rather than a set of values). The reason is that if we allow relative keys, the foreign key specification requires several restrictions. Further, we do not need relative keys when we map an ER schema to an XML schema. However, remember that our path expression syntax includes the parent axis. Thirdly we will use IDREF and IDREFS attributes, and we will specify their target types.

The constraints for an XGrammar  $\mathbb{G}$  is denoted by  $\Sigma$ .  $\Sigma$  includes:

- Attribute type. For any attribute  $a \in A$ , we define the type  $\tau \in \hat{\tau}$  for it. Further, we specify the target types for the following attribute types.
  - IDREF as:  $a :: IDREF \rightsquigarrow RE_1$ , where  $RE_1$  is the target type defined as:  $RE_1 ::= X \mid (RE_1) \mid (RE_1 + RE_1)$ , Here  $X \in N$ .
  - IDREFS as:  $a :: IDREFS \rightsquigarrow RE_2$ , where  $RE_2$  is the target type defined as:  $RE_2 ::= \epsilon \mid X \mid (RE_2) \mid (RE_2 + RE_2) \mid (RE_2, RE_2) \mid (RE_2)^? \mid (RE_2)^* \mid (RE_2)^+$ . Here  $X \in N$ .
- Key constraints as:  $key(X) = F$ , where  $X \in N$ , and  $F$  is a set of path expressions, that are the key attributes (fields). This specifies that  $F$  is the key for all elements of type  $Y$ .
- Unique Constraints as:  $unique(X) = F$ , where  $X \in N$ , and  $F$  is a set of path expressions. This specifies that  $F$  is unique for all elements of type  $Y$ .
- Foreign-key constraints as:  $X(F_1) REFERENCES Y(F_2)$ , here  $X, Y \in N$ , and  $F_1, F_2$  are sets of path expressions. We require that there be a key constraint  $key(Y) = F_2$ , or a unique constraint  $unique(Y) = F_2$ .

**Example 2** Table 4 shows key and IDREF constraints added to the XGrammar in Table 2. □

IDREF constraint: $\{\text{@art::IDREF} \rightsquigarrow (\text{Book} + \text{Paper})\}$ Key constraints: $\text{key}(\text{Person}) = \langle \text{@name} \rangle,$ $\text{key}(\text{Book}) = \langle \text{@ISBN} \rangle,$ $\text{key}(\text{Paper}) = \langle \text{@title} \rangle,$ $\text{key}(\text{Review}) = \langle \text{parent} :: \text{person}/\text{@name}, \text{@art} \rangle$
---

**Table 4.** Constraints added to XGrammar in Table 2

More examples of using our constraint specification scheme to model different application scenarios can be obtained from [Man03].

## 5 Conclusions and Future Work

In this paper, we explored constraint specification for XML schemas. We came up with a constraint specification scheme starting from a general description of XML schema requirements. Different approaches have been used in the past for coming up with constraint specification schemes: (a) W3C XML-Schema started by collecting different use cases from industry, and then came up with a constraint specification scheme; several features in W3C XML-Schema were voted by industry representatives as to whether they believe that feature will be useful or not (b) Research proposals like [BDF<sup>+</sup>02,FKS02] tried to study constraint specification primarily by looking at whether the problem of “reasoning about constraints” is decidable; one of the problems they try to study is whether given a schema, we can determine whether there exists a non-empty document that is valid with respect to the schema. We however used a different approach to study this problem. We first generalized the requirements for an XML schema language as: it should be capable of modeling any database application scenario (ER schema) without redundancy. Further, we discussed some of the main issues that make XML schema languages novel: ambiguous types and how they affect constraint specification, the difficulties associated with relative keys, and the differences between IDREF attributes and foreign keys.

This work is directly related to the community that develops XML standards. However, we believe it is important for the database researchers and practitioners to gain understanding of some of the key issues in XML schemas. Also several database researchers do not seem to be aware of the existence of multiple XML schema language standards, and their usage in practice. Outside of the major database vendors like Microsoft, Oracle, IBM, we have observed that DTDs are the most commonly used XML schema language. Also, several DTD users are migrating to RELAX-NG rather than W3C XML-Schema; we believe this is because RELAX-NG uses a syntax that is similar to that of DTD, and also because of the simplicity. Also country-wise, Japan announced RELAX as the standard XML schema language. Further RELAX-NG is based directly on well-known theoretical concepts of regular tree language theory. XQuery [W3Cb] operates on type definitions of W3C XML-Schema, and rely on named typing;

however, as part of their type inference, they allow specification of structures that can be any arbitrary regular tree grammar (they do not impose restriction of single-type tree grammar), and hence cannot be captured by W3C XML-Schema, but can be captured by RELAX-NG.

Even though our work studies only constraint specification scheme for XML schemas, it is our conviction, based on our previous experience, that the most appropriate XML schema language for database applications is RELAX-NG, with type-based constraint specification. However, we will disallow schemas that are not single-type tree grammars to model application semantics; however the result of operations will be regular tree grammars with no restriction. These are some of the main observations made in [Man03]; also this appears to be the approach being currently pursued by the XQuery WG.

Our future work with respect to constraint specification include, taking the ideas to the different standards groups. We will further attempt to model different real world application scenarios (such as the test cases from industry) using these concepts.

**Acknowledgments** We acknowledge the inputs and stimulating discussions from several members of the xml-dev mailing list.

## References

- [AL02] M. Arenas and L. Libkin. “A normal form for XML documents”. In *ACM PODS*, Madison, Wisconsin, June. 2002.
- [AL03] M. Arenas and L. Libkin. “An information-theoretic approach to normal forms for relational and XML data”. In *ACM PODS*, San Diego, CA, June. 2003.
- [BDF<sup>+</sup>02] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. “Keys for XML”. *Computer Networks*, 39(5):473–487, Aug. 2002.
- [BM01] P. V. Biron and A. Malhotra (Eds). “XML Schema Part 2: Datatypes”. W3C Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-2/>.
- [BPS00] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). “Extensible Markup Language (XML) 1.0 (2nd Edition)”. W3C Recommendation, Oct. 2000. <http://www.w3.org/TR/2000/REC-xml-20001006>.
- [Che76] P. P. Chen. “The Entity-Relationship Model”. *ACM Trans. on Database Systems (TODS)*, 1:9–36, 1976.
- [DSD] DSD. Dsd: Document structure description. <http://www.brics.dk/DSD/>.
- [FKS02] W. Fan, G. M. Kuper, and J. Simeon. “A Unified Constraint Model for XML”. *Computer Networks*, 39(5):489–505, Aug. 2002.
- [GMUW02] H. Garcia-Molina, J. D. Ullman, and J. Widom. “*Database Systems - The Complete Book*”. Prentice Hall, 2002.
- [LMCC02] D. Lee, M. Mani, F. Chiu, and W. W. Chu. “NeT & CoT: Translating Relational Schemas to XML Schemas”. In *ACM CIKM*, McLean, Virginia, Nov. 2002.
- [Man03] M. Mani. “*Data Modeling using XML Schemas*”. PhD Dissertation, UCLA CS, July. 2003. <http://www.cs.wpi.edu/~mmani/publications.html>.

- [MLM01] M. Murata, D. Lee, and M. Mani. “Taxonomy of XML Schema Languages using Formal Language Theory”. In *Extereme Markup Languages*, Montreal, Canada, Aug. 2001.
- [TL82] D. C. Tsichritzis and F. H. Lochovsky. “*Data Models*”. Prentice-Hall, 1982.
- [W3Ca] W3C. Xml-schema working group. <http://www.w3c.org/XML/Schema.html>.
- [W3Cb] W3C. Xquery working group. <http://www.w3c.org/XML/Query.html>.