Semantic Data Modeling using XML Schemas

Murali Mani*, Dongwon Lee, and Richard R. Muntz

Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095, USA
{mani,dongwon,muntz}@cs.ucla.edu

Abstract. Most research on XML has so far largely neglected the data modeling aspects of XML schemas. In this paper, we attempt to make a systematic approach to data modeling capabilities of XML schemas. We first formalize a core set of features among a dozen competing XML schema language proposals and introduce a new notion of *XGrammar*. The benefits of such formal description is that it is both concise and precise. We then compare the features of *XGrammar* with those of the Entity-Relationship (ER) model. We especially focus on three data modeling capabilities of *XGrammar*: (a) the ability to represent *ordered binary relationships*, (b) the ability to represent a set of semantically equivalent but structurally different types as "one" type using the closure properties, and (c) the ability to represent *recursive relationships*.

1 Introduction

With the growing popularity of XML (eXtensible Markup Language) [5] defined by W3C, XML schemas[1] expressed by schema languages (e.g., DTD [5], XML-Schema [14], RELAX [9]) are being widely used to describe data. Even though XML is largely used for transferring data at present, we envision that in the not-so-distant future, XML schemas will be used as the "external schema" for a large portion of the data. This makes the study of modeling capabilities of XML schemas very important. Furthermore, to bridge with other existing data models, it is becoming increasingly important to understand how to map features of XML schema to those of existing models and vice versa. In this paper, we attempt to make a systematic approach to data description using an XML schema and compare it to the widely-used conceptual model, the Entity-Relationship (ER) model [7]. Our contributions in this paper are as follows:

- We formalize a core set of features found in various XML schema languages into *XGrammar* – a grammar notation commonly found in formal language theory. The important building blocks of any XML schema language such

* This author is partially supported by NSF grants 0086116, 0085773, 9817773.

[1] We differentiate two terms – XML schema(s) and XML-Schema. The former refers to a general term for a schema for XML, while the latter [14] refers to one kind of XML schema language proposed by W3C.

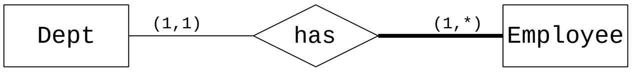

Fig. 1. Cardinality representation in ER model.

as element-subelement relationships are well captured in *XGrammar* in a coherent and precise manner.
- We describe three features of *XGrammar* in detail and compare them with features of ER model: (a) representing *ordered binary relationships*, (b) representing a set of semantically equivalent but structurally different types as "one" type using the property that *XGrammar* is closed under the *union* boolean operation, and (c) representing *recursive relationships* using recursive type definitions. By doing so, we also identify features lacking in ER model to natively support the XML model and extend it to *EER model*.

Besides the new features present in *XGrammar*, they can also represent data modeling features such as n-ary relationships, composite attributes, generalization hierarchy etc. However, they are still in a development phase for schema language proposals such as RELAX, and we do not focus on them in this paper.

1.1 Background: ER Model and Our Extensions

Entity-Relationship model (ER model) and Entity-Relationship diagram (ER diagram) are defined by Chen in 1970s and has been widely used in data modeling ever since [7]. The basic features of the ER model are *entity* types and *relationship* types. An *entity* type is represented by a rectangular labeled box, and a *relationship* type is represented by a diamond labeled box in an ER diagram. For our purposes, we use ER diagram notations in [1]. Here a cardinality of an entity in a relationship is represented as a 2-tuple $(minC, maxC)$. We use this 2-tuple notation mainly to distinguish between the cardinalities: $(0, 1)$ and $(1, 1)$, and also $(0, *)$ and $(1, *)$, which are commonly found in XML schemas. Here $*$ means that there is no upper bound on the cardinality. For instance, the diagram in Fig. 1 illustrates "Dept can have one or many Employees while each Employee can belong to one and only one Dept". Note that our notations of cardinalities are exactly reverse of those used in [1].

In this paper, we are not concerned with the more advanced features of ER model such as the role of an entity in a relationship, n-ary relationships where $n > 2$, attributes of a relationship, and constraints. Instead, we focus only on the basic features of the ER model, and extend it with the following additional features:

- *Order in a binary relationship*: ER model is based on a notion of set, and thus does not have a construct for order. However, in XML model, order is important. For instance, the first and second authors of book are different in XML model. To express such ordering in an ER diagram, we denote

the in-between edge by a thick line. For instance, there is ordering among Employees in Fig. 1.
– *Element-subelement relationship*: One of the main constructs in XML model is the element-subelement (i.e., parent-child) relationship. We represent this relationship using a dummy "`has`" relationship in ER model (i.e., a diamond box with the label `has`). For instance, the relationship that Employee is a subelement of Dept is shown in Fig. 1.

For convenience, we denote *ordered* and *unordered* relationships, say X between two entities A and B, where the cardinality of A is (m_1, M_1), and the cardinality of B is (m_2, M_2) by $A(m_1, M_1) \stackrel{X}{\Longrightarrow} B(m_2, M_2)$ and $A(m_1, M_1) \xrightarrow{X} B(m_2, M_2)$, respectively. ER model with our extensions will be referred to as "EER model" throughout the rest of the paper.

1.2 Related Work

Data modeling is an inherent part of database design, and deals with the structure, organization and effective use of data and the information they represent [15]. Such conceptual modeling of the data has been helped by data models such as ER model [7], which models an enterprise as a set of entities and relationships. However these data models cannot specify ordered relationships (i.e., cannot specify order between objects in a relationship). Ordered relationships exist commonly in practice such as the list of authors of a book. XML schemas, on the other hand, can specify such ordered relationships.

Semantic data modeling using XML schemas has been studied in the recent past. ERX [13] extends ER model so that one can represent a style sheet and a collection of documents conforming to one DTD in ERX model. But order is represented in ERX model by an additional order attribute. Other related work include a mapping from XML schema to an extended UML [4], and a mapping from Object-Role Modeling (ORM) to XML schema [2]. Our approach is different from these approaches; we focus on the new features provided by an XML Schema - element-subelement relationships, new datatypes such as `ID` or `IDREF(S)`, recursive type definitions, and the property that *XGrammar* is closed under union, and how they are useful to data modeling.

1.3 Outline of the Paper

The remainder of this paper is organized as follows. In Sect. 2, we describe *XGrammar* that we propose as a formalization of XML schemas. In Sect. 3, we describe in detail the main features of *XGrammar* for data modeling. In Sect. 4, we show how to convert an *XGrammar* to EER model, and vice versa. In Sect. 5, an application scenario using the proposed *XGrammar* and EER model is given. Finally, some concluding remarks are followed in Sect. 6.

2 Notation for XML Schemas: *XGrammar*

Recently about a dozen XML schema languages have been proposed. Some proposals aim at full-fledged schema languages while others take a minimalistic approach. Therefore, they are not directly comparable with each other. Nevertheless, we believe it is meaningful to compare the main building blocks of each language. In [10] and [12], present authors analyzed various schema language proposals using comparative analysis and formal language theory and categorized them into different classes.

In this section, we propose a new notation called *XGrammar*. Instead of choosing one XML schema language as the basic data modeling language, we extract the most important features from the proposed XML schema languages and formalize them into *XGrammar*. This is an extension of the regular tree grammar definition in [12] (Def. 24), where we used a six tuple notation to precisely describe content models of XML schema languages. In this paper, we extend this 6-tuple notation with attribute definitions and data types.

Informally, *XGrammar* takes the structural specification feature from DTD and RELAX and the data typing feature from XML-Schema. Therefore, unlike DTD, *XGrammar* can specify the exact types of attributes. Furthermore, attributes of `IDREF(S)` type can specify which "target" types the current attributes refer to. *XGrammar* is thus our attempt to formalize some of the core ideas found in the various XML schema languages proposed recently. The benefits of a formal description is that it is both concise and precise. From RELAX, we borrow the notion of *tree* and *hedge* types: the values of a tree type are trees and the values of a hedge type are hedges – a sequence of trees (or an ordered list of trees) [11]. Both tree and hedge types are together called regular expression types in [8].

We use G to denote an *XGrammar* and $L(G)$ to denote the language that G generates. We assume the existence of a set $\widehat{N}$ of non-terminal names, a set $\widehat{T}$ of terminal names and a set $\widehat{\tau}$ of atomic data types defined in [3] (e.g., string, integer, etc), including `ID` and `IDREF(S)`. We use the notations: ϵ denotes the empty string, "$+$" for the union, "," for the concatenation, "$a^?$" for zero or one occurrence, "a^*" for the Kleene closure, and "a^+" for "a, a^*". Now let us define *XGrammar* formally:

Definition 1 (*XGrammar*) An *XGrammar* is denoted by a 7-tuple $\mathbb{G} = (N_T, N_H, T, S, E, H, A)$, where:

- N_T is a set of non-terminal symbols that are tree types, where $N_T \subseteq \widehat{N}$,
- N_H is a set of non-terminal symbols that are hedge types, where $N_H \subseteq \widehat{N}$. We use N to denote $N_T \cup N_H$. Also, we place the constraint $N_T \cap N_H = \phi$.
- T is a set of terminal symbols, where $T \subseteq \widehat{T}$,
- S is a set of start symbols, where $S \subseteq N$.
- E is a set of element production rules of the form "$X \rightarrow a\ RE$", where $X \in N_T$, $a \in T$, and RE is:

$$RE ::= \epsilon \mid \tau \mid n \mid (RE) \mid RE + RE \mid RE, RE \mid RE^? \mid RE^* \mid RE^+$$

Table 1. An example XML-Schema `library.xsd`.

```
<schema xmlns:t='http://www.w3.org/namespace/'>
 <element name='Library'>
  <complexType>
   <sequence>
    <element ref='t:Book' minOccurs='0' maxOccurs='unbounded'/>
    <element ref='t:Magazine' minOccurs='0' maxOccurs='unbounded'/>
    <element ref='t:Person' minOccurs='0' maxOccurs='unbounded'/>
   </sequence>
  </complexType>
 </element>
 <element name='Book'>
  <complexType>
   <sequence>  <element ref='t:EMPTY'/>  </sequence>
   <attribute name='title' type='string' use='required'/>
   <attribute name='authors' type='IDREFS' use='required'/>
   <attribute name='publicationDate' type='date' use='required'/>
  </complexType>
 </element>
 <element name='Magazine'>
  <complexType>
   <sequence>  <element ref='t:EMPTY'/>  </sequence>
   <attribute name='title' type='string' use='required'/>
   <attribute name='editor' type='IDREF' use='optional'/>
   <attribute name='publicationDate' type='date' use='optional'/>
  </complexType>
 </element>
 <element name='Person'>
  <complexType>
   <sequence>
    <element ref='t:Spouse' minOccurs='0' maxOccurs='1/>
    <element ref='t:Person' minOccurs='0' maxOccurs='unbounded'/>
   </sequence>
   <attribute name='personID' type='ID' use='required'/>
   <attribute name='name' type='string' use='optional'/>
  </complexType>
 </element>
 <element name='Spouse'> <complexType mixed='true'/> </element>
</schema>
```

where $\tau \in \hat{\tau}$ and $n \in N$. Note that RE is actually a hedge type, but it might not have a name associated with it. In other words, we can have anonymous hedge types not captured by N_H. Our examples will elaborate this point.

- H is a set of hedge production rules of the form "$X \rightarrow RE$", where $X \in N_H$, and RE is the same as the one for E.
- A is a set of attribute production rules of the form "$X \rightarrow a\ RE$", where $X \in N$, $a \in T$, and $RE ::= \epsilon \mid \alpha \mid (RE) \mid RE, RE$, where α is an attribute definition expression defined as:

$$\alpha ::= \begin{cases} \text{“@”}\ a\ [\ \text{“?”}\ ]\ \text{“::”}\ \tau & \text{if } \tau \notin \{\texttt{IDREF}, \texttt{IDREFS}\} \\ \text{“@”}\ a\ [\ \text{“?”}\ ]\ \text{“::”}\ \tau\ \text{“}\rightsquigarrow\text{”}\ RE_1 & \text{if } \tau = \texttt{IDREF} \\ \text{“@”}\ a\ [\ \text{“?”}\ ]\ \text{“::”}\ \tau\ \text{“}\rightsquigarrow\text{”}\ RE_2 & \text{if } \tau = \texttt{IDREFS} \end{cases}$$

$$RE_1 ::= n \mid (RE_1) \mid RE_1 + RE_1, \quad \text{where } n \in N_T$$

$$RE_2 ::= \epsilon \mid n \mid (RE_2) \mid RE_2 + RE_2 \mid RE_2, RE_2 \mid RE_2^? \mid RE_2^* \mid RE_2^+, \quad \text{where } n \in N$$

For representation, we distinguish attributes from elements in the grammar using "@" as in [6] and specify the type τ of an attribute using "::". A specified attribute is considered mandatory unless qualified by "?". □

Table 2. An example DTD `library.dtd` equivalent to `library.xsd` in Table 1.

```
<!DOCTYPE Library [
  <!ELEMENT Library   (Book*,Magazine*,Person*)>
  <!ELEMENT Book      (EMPTY)>
  <!ATTLIST Book      title  CDATA   #REQUIRED    authors IDREFS  #REQUIRED
                      publicationDate CDATA  #REQUIRED>
  <!ELEMENT Magazine  (EMPTY)>
  <!ATTLIST Magazine  title  CDATA   #REQUIRED    editor  IDREF   #IMPLIED
                      publicationDate CDATA  #IMPLIED>
  <!ELEMENT Person    (Spouse?,Person*)>
  <!ATTLIST Person    personID  ID   #REQUIRED    name    CDATA   #IMPLIED>
  <!ELEMENT Spouse    (#PCDATA)>
]>
```

Table 3. An example XML Document `library.xml` conforming to schemas defined in Tables 1 and 2.

```
<library>
  <book title="Data Structures and Algorithms" authors="aho hopcroft ullman"
        publicationDate="January, 1983"/>
  <book title="Principles of Compiler Design" authors="aho ullman"
        publicationDate="1979"/>
  <book title="Introduction to Automata Theory" authors="hopcroft ullman"
        publicationDate="1979"/>
  <magazine title="Communication of ACM" editor="aho"/>
  <magazine title="IEEE Comp." editor="ullman" publicationDate="Sep,2000"/>
  <person personID="aho" name="Alfred. V. Aho">
    <spouse>WifeOfAho</spouse>
    <person personID="son1" name="Junior_1 Aho"/>
    <person personID="son2" name="Junior_2 Aho"/>
  </person>
  <person personID="ullman" name="Jeffrey. D. Ullman">
    <spouse>WifeOfUllman</spouse>
  </person>
  <person personID="hopcroft" name="John. E. Hopcroft"/>
</library>
```

Example 1. Consider a scenario of "library" in the real world. Tables 1 and 2 show exemplar schema definitions to model the scenario. Note that not all constraints expressed in Table 1 are expressed in Table 2 due to the insufficient expressive power of DTD. Then, the schema definition can be encoded into *XGrammar*: $\mathbb{G}_1 = (N_T, \epsilon, T, S, E, \epsilon, A)$, where

$$N_T = \{Library, Book, Magazine, Person, Spouse\}$$

$$T = \{library, book, magazine, person, spouse, title, authors, editor, publicationDate, personID, name\}$$

$$S = \{Library\}$$

$$E = \{Library \rightarrow library\ (Book^*, Magazine^*, Person^*), Book \rightarrow book\ (\epsilon), Magazine \rightarrow magazine\ (\epsilon), Person \rightarrow person\ (Spouse^?, Person^*), Spouse \rightarrow spouse\ (string)\}$$

$$A = \{Library \rightarrow library(\epsilon), Book \rightarrow book\ (@title :: string, @authors :: \texttt{IDREFS} \rightsquigarrow Person^+, @publicationDate :: date), Magazine \rightarrow magazine\ (@title :: string, @editor^? :: \texttt{IDREF} \rightsquigarrow Person, @publicationDate^? :: date), Person \rightarrow person\ (@personID :: \texttt{ID}, @name^? :: string), Spouse \rightarrow spouse(\epsilon)\}$$

Observe that the `IDREF(S)` attributes identify the target types. For instance, an attribute @*editor* of type `IDREF` identifies the target type as $Person$. This means that in a document conforming to this schema, the value of the @*editor* attribute should be a valid `ID` value of an element of type $Person$. In addition, the target type of an `IDREFS` attribute is specified as a hedge type. For instance, the `IDREFS` attribute @*authors* of *Book* identifies its target type as $Person^+$, which is an anonymous hedge type. This means that the value of @*authors* should be a list of values that are `ID` values of $Person$. Table 3 shows an example XML document that conforms to $\mathbb{G}_1$. □

3 Semantic Data Modeling with *XGrammar*

Three main features of *XGrammar* that help to model XML data are as follows:

1. **Ordered binary relationships:** *XGrammar* can represent ordered binary relationships using element-subelement relationships and `IDREF(S)` attributes. Such relationships occur commonly in real world scenarios. For example, the authors of a book are typically ordered. The EER model can represent ordered binary relationships.
2. **Union types:** The closure properties of the different XML schema language proposals under boolean set operations are studied in [12]. Here, it was shown that (a) proposals such as DTD and XML-Schema are closed under *intersection*, but are not closed under *union* and *difference* operations, and (b) proposals such as RELAX, XDuce and TREX are closed under intersection, union and difference operations. Since *XGrammar* is equivalent to RELAX with respect to its structural expressiveness, *XGrammar* is also closed under all three boolean set operations. Therefore one can define union between any two tree types or hedge types. This is useful for several real-world data integration problems.
   For example, we can define one type representing two semantically equivalent but structurally different types. Consider the types $Book1 \rightarrow book(Title, Author^+)$ and $Book2 \rightarrow book(Title, Author^+, Publisher)$. We can take union of these two types, and define the union type as $Book \rightarrow book(Title, Author^+, Publisher?)$. Similarly consider the types $Book \rightarrow book(Title, Author^+, Publisher^?)$, and $Magazine(Name, Editor^+)$, we can define the union type as $ReadingMaterial \rightarrow (Book\ +\ Magazine)$.
3. **Recursive types:** *XGrammar* can represent recursive relationship types using recursive type definitions. In $\mathbb{G}_1$ of Example 1, we have a recursive type for $Person$ in its element production rule; here the subelements of a $Person$ represent the children of that $Person$. Unlike ER model, in general, XML model has two different notions of recursion: *structural* and *semantic* recursions. *XGrammar* can express both using element or attribute production rules, respectively. For instance, consider the classical "employee-manager-subordinate" relationships of ER model in Fig. 2. This model can be best represented by the two DTDs below. DTD (a) forms a recursion *semantically* if the `subord` attribute is assumed to point to some `employee`'s `name`

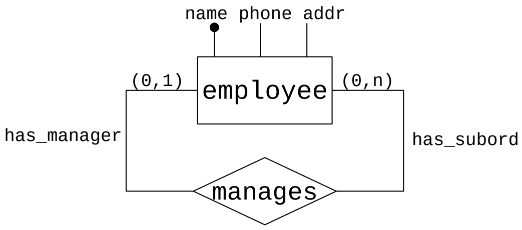

Fig. 2. An example of recursion in ER model.

attributes, while DTD (b) forms a recursion *structurally* since `employee` element can contain `subord` subelement which can in turn contain `employee` element as child again.

```
// DTD (a)                              // DTD (b)
<ELEMENT employee EMPTY>                <ELEMENT employee (subord*)>
<ATTLIST employee                       <ATTLIST employee name   ID #REQUIRED
         name     ID     #REQUIRED                        phone CDATA #IMPLIED
         phone    CDATA  #IMPLIED                         addr  CDATA #IMPLIED>
         addr     CDATA  #IMPLIED       <ELEMENT subord   (employee)>
         subord   IDREFS #IMPLIED>
```

The two DTDs (a) and (b) can be captured in *XGrammar* differently. Attribute production rule will capture the semantic recursion:

$$A = \{Employee \rightarrow employee\ (@subord^{?} :: \texttt{IDREFS} \rightsquigarrow Employee^{+})\}$$

while element production rule will capture the structural recursion:

$$E = \{Employee \rightarrow employee\ (Subord^{*}), Subord \rightarrow subord\ (Employee)\}$$

4 *XGrammar* & EER Model

In this section, we discuss the relationships between *XGrammar* and EER model. Especially, we investigate issues of conversion between the two models.

4.1 Converting *XGrammar* to EER Model

XGrammar $\mathbb{G}_1$ of Example 1 can be, for instance, converted to EER model as shown in Fig. 3. The different types and production rules are converted as follows:

1. *Tree type:* Every tree type in N_T of *XGrammar* is represented as an entity type. For instance, in $\mathbb{G}_1$ of Example 1, there are five tree types and hence five entity types $\{Library, Book, Magazine, Person, Spouse\}$.

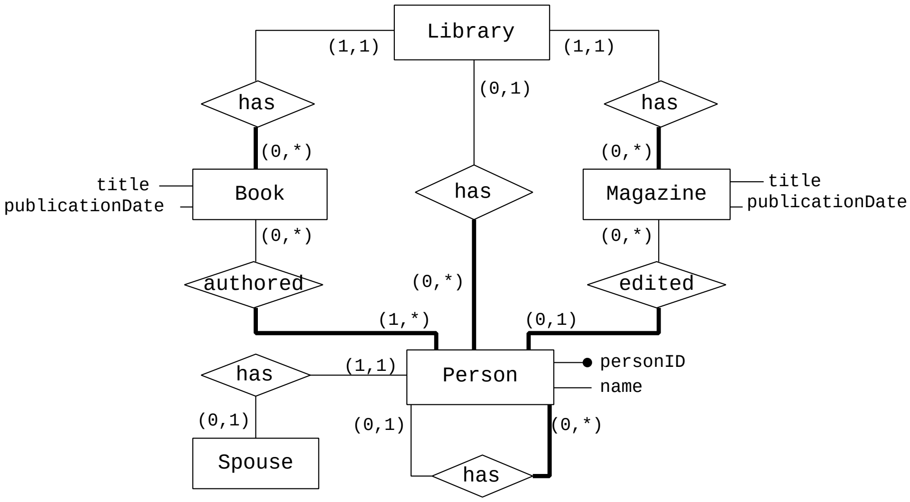

Fig. 3. An EER model representation of *XGrammar* $\mathbb{G}_1$ of Example 1.

2. *Element production rule*: For every "child" tree type in the element production rule, a `has` *ordered* relationship is created. The cardinalities of the parent and child types are decided appropriately based on the regular expression operators such as ",", "+", "*", "?". For example, consider the element production rule $Library \rightarrow library\ (Book^*, Magazine^*, Person^*)$. There are three child tree types. Therefore in EER model, this element production rule will be represented as three ordered relationships, given by $Library(1,1) \overset{has}{\Longrightarrow} Book(0,*), Library(1,1) \overset{has}{\Longrightarrow} Magazine(0,*), Library(0,1) \overset{has}{\Longrightarrow} Person(0,*)$. Note that since $Book$ and $Magazine$ can occur in the document only as child of $Library$, the cardinality of $Library$ in these relationships is $(1,1)$. However, $Person$ can occur as child of either $Library$ or $Person$. Therefore, the cardinality of $Library$ in this relationship is $(0,1)$.
3. `IDREF` *attribute:* `IDREF` attribute is represented as an *unordered* binary relationship. The cardinality of the type that specifies the `IDREF` attribute in the relationship is $(0,*)$. For identifying the cardinality of the target types, we consider three cases with examples:
   – *Case 1. The target type of an* `IDREF` *attribute is a union of at least two tree types*: For example, consider the attribute rule of type X to specify $@A :: \texttt{IDREF} \rightsquigarrow (B \mid C)$. This is represented as two unordered relationships $X(0,*) \xrightarrow{A} B(0,1)$ and $X(0,*) \xrightarrow{A} C(0,1)$. This is irrespective of whether the attribute A is optional or not.
   – *Case 2: The target type of an* `IDREF` *attribute is a single tree type, and this* `IDREF` *attribute is optional*: For example, consider the attribute $@editor^? :: \texttt{IDREF} \rightsquigarrow Person$ for the tree type $Magazine$ in $\mathbb{G}_1$ of Example 1. This is represented as an unordered relationship: $Magazine(0,*)$

Table 4. The relationships specified by $\mathbb{G}_1$ of Example 1.

Type	Order	Relationships
$1:1$	-	$\{Spouse(0,1) \stackrel{has}{\Longrightarrow} Person(1,1)\}$
$1:n$	Ordered	$\{Library(1,1) \stackrel{has}{\Longrightarrow} Book(0,*),\ Library(1,1) \stackrel{has}{\Longrightarrow} Magazine(0,*),$ $Library(0,1) \stackrel{has}{\Longrightarrow} Person(0,*),\ Person(0,1) \stackrel{has}{\Longrightarrow} Person(0,*)\}$
$n:1$	Unordered	$\{Magazine(0,*) \stackrel{edited}{\longrightarrow} Person(0,1)\}$
$n:m$	Ordered RHS	$\{Book(0,*) \stackrel{authored}{\Longrightarrow} Person(1,*)\}$

$\stackrel{edited}{\longrightarrow} Person(0,1)$. Note that the name of the relationship is changed to a verb form (i.e., from "editor" to "edited") for clarity.
 - *Case 3: The target type of an* `IDREF` *attribute is a single tree type, and this* `IDREF` *attribute is mandatory*: For example, consider the `IDREF` attribute for tree type X specified as $@A$ :: `IDREF` $\leadsto (B)$. This is represented as one unordered relationship $X(0,*) \stackrel{A}{\longrightarrow} B(1,1)$.
4. `IDREFS` *attribute:* `IDREFS` attribute is used to specify *ordered* relationships. As for `IDREF` attribute, the cardinality of the type that specifies the `IDREFS` attribute in the relationship is $(0,*)$. For identifying the cardinality of a target type in the relationship, we consider two cases:
   - *Case 1: The* `IDREFS` *attribute is optional*: For example, consider the attribute rule of type X to specify $@A^?$ :: `IDREFS` $\leadsto B^+$. This is represented as $X(0,*) \stackrel{A}{\Longrightarrow} B(0,*)$. In other words, the minimum cardinality of a target type in the relationship is 0.
   - *Case 2: The* `IDREFS` *attribute is mandatory*: This is represented as an ordered binary relationship named with the attribute name just like in the previous case. However, the minimum cardinality of a target type need not be 0. For example, consider the attribute specification for the tree type *Book* in $\mathbb{G}_1$ of Example 1 given by $@Authors$ :: `IDREFS` $\leadsto (Person^+)$. This is represented in the EER diagram as $Book(0,*) \stackrel{authored}{\Longrightarrow} Person(1,*)$.

The different relationships expressed in $\mathbb{G}_1$ of Example 1 are summarized in Table 4. The reader should observe that what we described above are binary relationships expressed in *XGrammar*. This represents only a subset of the relationships expressible in an *XGrammar*. For example, $\mathbb{G}_1$ of Example 1 specifies other relationships such as "the list of $Magazine$s in *Library* occur after the list of *Book*s in the *Library*". Such order specifications are outside the scope of this paper and not discussed further.

4.2 Converting EER Model to *XGrammar*

XML schema supports constraints such as key and foreign key constraints. Based on these constraints, we define *"joinable tree types"* – tree types that have a key-foreign key constraint. Joinable tree types are used to represent relationships, similar to key-foreign key constraints in the relational model. The translation of a given EER model to an *XGrammar* is done using the following steps. These steps are summarized in Table 5.

1. Each entity in EER model is translated to a tree type in *XGrammar*. For example, we will have a tree type corresponding to *Book*, a tree type corresponding to $Magazine$ etc. Every simple attribute can be translated to an attribute for that tree type in *XGrammar*, a composite attribute can be translated to a subelement of the tree type. In our EER example, we have an entity *Book* with two attributes *name* and *publicationDate*. They are translated to a tree type *Book* with an attribute production rule as $Book \rightarrow book(@title, @publicationDate)$.
2. A $1:1$ relationship is translated as follows. Consider a relationship: $A(m_1, M_1) \stackrel{R}{\Longrightarrow} B(m_2, M_2)$, where $M_1 = M_2 = 1$.
   - $m_1 = 0$ and $m_2 = 0$: This is represented using two different tree types with `IDREF` or with foreign key constraints (joinable tree types).
   - $m_1 = 1$ (or similarly $m_2 = 1$): This is represented using an element-subelement relationship, where B is a child of A (or vice versa). If $m_2 = 1$, then the content model of the element production rule for A will specify B, otherwise if $m_2 = 0$, then the content model will specify $B^?$. For example, Fig. 3 has the $1:1$ relationship $Person(1,1) \stackrel{has}{\Longrightarrow} Spouse(0,1)$. This is represented in the *XGrammar* as an element-subelement relationship, where $Spouse^?$ is a subelement of *Person*. If both $m_1 = 1$ and $m_2 = 1$, then we can also represent this relationship using just one tree type as in the ER-relational conversion [1].
3. A $1:n$ relationship is translated as follows.
   - The relationship is *ordered*: We can represent this as an element-subelement relationship, or using `IDREFS` attributes. The EER diagram of Fig. 3 has four ordered $1:n$ relationships. We represent all of them as element-subelement relationships in the *XGrammar*. For example, $Library(1,1) \stackrel{has}{\Longrightarrow} Book(0,*)$ is represented by having $Book^*$ as a subelement of *Library*.
   - The relationship is *unordered*: We can represent this using `IDREFS` attribute or using joinable tree types. The EER diagram of Fig. 3 has one unordered $1:n$ relationship $Magazine(0,*) \stackrel{edited}{\longrightarrow} Person(0,1)$, which is represented by having an `IDREF` attribute for Magazine as $@editor^?$ :: `IDREF` $\rightsquigarrow Person$.
4. A $n:m$ relationship is translated as follows.
   - The relationship is *ordered*: We can represent this using `IDREFS` attribute. For example, consider the ordered $n:m$ relationship in the EER diagram of Fig. 3, $Book(0,*) \stackrel{authored}{\Longrightarrow} Person(0,*)$. This is represented in *XGrammar* using `IDREFS` attribute for *Book* as $@authors?$ :: `IDREFS` $\rightsquigarrow Person^+$.
   - The relationship is *unordered*: This is represented using joinable tree types.

Table 5. Representing the different binary relationships in an *XGrammar* to satisfy the goodness criterion.

Relationship Type	Ordered/Unordered	How to represent them
$1:1$	-	{element-subelement, `IDREF`, joinable tree types}
$1:n$	Ordered	{element-subelement, `IDREFS`}
$1:n$	Unordered	{`IDREF`, joinable tree types}
$n:m$	Ordered	{`IDREFS`}
$n:m$	Unordered	{joinable tree types}

Table 6. The relationships specified by EER diagram of Fig. 4.

Type	Order	Relationships
$1:n$	Ordered	$\{Agency(1,1) \stackrel{funds}{\Longrightarrow} ResearchProject(1,*)$,
		$ResearchProject(1,1) \stackrel{produces}{\Longrightarrow} ResearchReport(1,*)$,
		$ResearchDB(1,1) \stackrel{has}{\Longrightarrow} ResearchTopic(0,*)$,
		$ResearchDB(1,1) \stackrel{has}{\Longrightarrow} Employee(0,*)$,
		$ResearchDB(1,1) \stackrel{has}{\Longrightarrow} Agency(0,*)\}$
$n:1$	Unordered	$\{ResearchReport(0,*) \stackrel{addresses}{\longrightarrow} ResearchTopic(0,1)$,
		$Employee(0,*) \stackrel{isSupervised}{\longrightarrow} Employee(0,1)$,
		$ResearchProject(0,*) \stackrel{IS\_PI}{\longrightarrow} Employee(1,1)\}$
$n:m$	Ordered RHS	$\{ResearchProject(0,*) \stackrel{hasEmployee}{\Longrightarrow} Employee(1,*)\}$

5. A recursive relationship is represented either using semantic recursion or structural recursion. The EER diagram has one recursive relationship, $Person(0,1) \stackrel{has}{\Longrightarrow} Person(0,*)$, which is represented using structural recursion in the *XGrammar* by having $Person^*$ as a subelement of $Person$.

5 Application

In this section, we consider a real world example - that of a *Research Projects database*. This example is modified slightly from the one given in [1] (page 49). We illustrate how this is modeled using *XGrammar*. The EER diagram is shown in Fig. 4. To convert EER diagram to *XGrammar*, first introduce a root tree type for the *XGrammar* – ResearchDB. The child elements of the root are the entities shown in EER diagram through `has` relationships from ResearchDB. There are six entities in this EER model - {ResearchDB, ResearchTopic, Agency, Employee, ResearchProject, ResearchReport}. The entity ResearchDB is mapped to the root tree type for *XGrammar*. The relationships in this EER diagram are shown in Table 6. For this example, we can represent all the ordered $1:n$ relationships using element-subelement relationships, the unordered $n:1$ relationships using `IDREF` attribute, and the ordered $n:m$ relationships using `IDREFS` attributes. The *XGrammar* is given by $\mathbb{G}_2 = (N_T, \epsilon, T, S, E, \epsilon, A)$, where

$$N_T = \{ResearchDB, Agency, ResearchTopic, Employee, ResearchProject, ResearchReport\}$$
$$T = \{researchDB, agency, researchTopic, employee, researchProject, researchReport\}$$

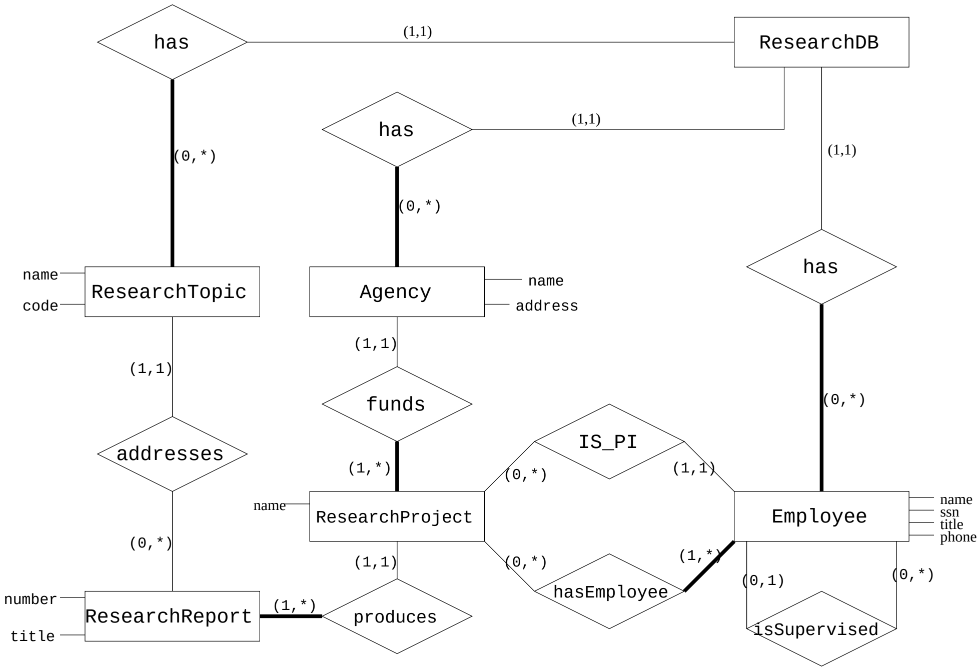

Fig. 4. Application scenario using *XGrammar* and EER model.

$$
\begin{aligned}
S = \{&ResearchDB\}\\
E = \{&ResearchDB \rightarrow researchDB(Agency^*, ResearchTopic^*, Employee^*), Agency \rightarrow\\
&agency(ResearchProject^+), ResearchProject \rightarrow researchProject(ResearchReport^+),\\
&Employee \rightarrow employee(\epsilon), ResearchReport \rightarrow researchReport(\epsilon),\\
&ResearchTopic \rightarrow researchTopic(\epsilon)\}\\
A = \{&ResearchDB \rightarrow researchDB(\epsilon),\\
&Agency \rightarrow agency(@name :: string, @address :: string),\\
&ResearchTopic \rightarrow researchTopic(@name :: string, @code :: integer),\\
&ResearchProject \rightarrow researchProject(@name :: string, @PI :: \mathtt{IDREF} \rightsquigarrow Employee,\\
&\qquad\qquad @members :: \mathtt{IDREFS} \rightsquigarrow Employee^+),\\
&ResearchReport \rightarrow researchReport(@number :: integer, @title :: string,\\
&\qquad\qquad @topic :: \mathtt{IDREF} \rightsquigarrow ResearchTopic)\\
&Employee \rightarrow employee(@name :: string, @ssn :: integer, @title :: string,\\
&\qquad\qquad @phone :: integer, @supervisor^? :: \mathtt{IDREF} \rightsquigarrow Employee)\}
\end{aligned}
$$

6 Conclusions

In this paper, we examined several new features provided by XML schemas for data description. In particular, we examined how ordered binary relationships -

$1 : n$ (through parent-child relationships and `IDREFS` attribute) as well as $n : m$ (through `IDREFS` attribute) can be represented using an XML schema. We also examined the other features provided by XML grammars - representing recursive relationships using recursive type definitions and union types. EER model, conceptualized in the logical design phase, can be mapped on to *XGrammar* (or its equivalent) and, in turn, mapped into other final data models, such as relational data model, or in some cases, the XML data model itself (i.e., data might be stored as XML documents themselves). We believe that work presented in this paper forms a useful contribution to such scenarios.

References

[1] C. Batini, S. Ceri, and S. B. Navathe. *"Conceptual Database Design: An Entity-Relationship Approach"*. The Benjamin/Cummings Pub., 1992.

[2] L. Bird, A. Goodchild, and T. Halpin. "Object Role Modeling and XML-Schema". In *Int'l Conf. on Conceptual Modeling (ER)*, Salt Lake City, UT, Oct. 2000.

[3] P. V. Biron and A. Malhotra (Eds). "XML Schema Part 2: Datatypes". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-2/.

[4] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen. "UML for XML Schema Mapping Specification". http://www.rational.com/media/uml/resources/media/uml_xmlschema33.pdf.

[5] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). "Extensible Markup Language (XML) 1.0". W3C Recommendation, Feb. 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[6] A. Brown, M. Fuchs, J. Robie, and P. Wadler. "MSL: A Model for W3C XML Schema". In *Int'l World Wide Web Conf. (WWW)*, Hong Kong, May 2001.

[7] P. P. Chen. "The Entity-Relationship Model". *ACM Trans. on Database Systems (TODS)*, 1:9–36, 1976.

[8] H. Hosoya and B. C. Pierce. "XDuce: A Typed XML Processing Language". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[9] ISO/IEC. *"Information Technology – Text and Office Systems – Regular Language Description for XML (RELAX) – Part 1: RELAX Core"*, 2000. DTR 22250-1.

[10] D. Lee and W. W. Chu. "Comparative Analysis of Six XML Schema Languages". *ACM SIGMOD Record*, 29(3):76–87, Sep. 2000.

[11] M. Murata. "Hedge Automata: a Formal Model for XML Schemata". Web page, 2000. http://www.xml.gr.jp/relax/hedge_nice.html.

[12] M. Murata, D. Lee, and M. Mani. "Taxonomy of XML Schema Languages using Formal Language Theory". In *Extreme Markup Languages*, Montreal, Canada, Aug. 2001. http://www.cs.ucla.edu/~dongwon/paper/.

[13] G. Psaila. "ERX: A Data Model for Collections of XML Documents". In *ACM Symp. on Applied Computing (SAC)*, Villa Olmo, Italy, Mar. 2000.

[14] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Eds). "XML Schema Part 1: Structures". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-1/.

[15] D. C. Tsichritzis and F. H. Lochovsky. *"Data Models"*. Prentice-Hall, 1982.