Augmenting Traditional Conceptual Models to Accommodate XML Structural Constructs

Reema Al-Kamha, David W. Embley, and Stephen W. Liddle

Brigham Young University, Provo, Utah 84602, U.S.A.
{reema, embley}@cs.byu.edu, {liddle}@byu.edu

Abstract. Although it is possible to present XML Schema graphically, such representations do not raise the level of abstraction for XML schemata in the same way traditional conceptual models raise the level of abstraction for data schemata. Traditional conceptual models, on the other hand, do not accommodate several XML Schema structures. Thus, there is a need to enrich traditional conceptual models with new XML Schema features. After establishing criteria for XML conceptual modeling, we propose an enrichment to represent the XML features missing in traditional models. We argue that our solution can be adapted generally for traditional conceptual models and show how it can be adapted for two popular conceptual models.

1 Introduction

Recently, many organizations have begun to store their data using XML, and XML Schema has become the preeminent mechanism for describing valid XML document structures. Moreover, the number of applications that use XML as their native data model have increased. This increases the need for well-designed XML data models and the need for a conceptual model for designing XML schemas.

Several commercial tools provide support for graphically representing XML Schema structures. Visual Studio .NET [10] from Microsoft, Stylus Studio [15] from DataDirect Technologies, and XML Spy [14] from Altova all have their own proprietary methods for graphically representing XML structures. Each of them includes a graphical XML Schema editor that uses connected rectangular blocks to present the schema. Although these products provide visual XML Schema editing tools, they do not raise the level of abstraction because they only provide a direct view of an XML Schema document. Thus, these graphical representations do not serve the objective of conceptualizing XML Schema to be used in modeling and design.

In systems modeling and design, traditional conceptual models have proven to be quite successful for graphically representing data at a higher level of abstraction. Conceptual models represent components and their relationships to other components in the system under study in a graphical way, at a conceptual level of understanding. Popular conceptual models that achieve these objectives are ER [3], extended ER models [16], and UML [2, 17].

XML Schema, however, introduces a few features that are not explicitly supported in these and similar conceptual models. The most important of these features include the ability to (1) order lists of concepts, (2) choose alternative concepts from among several, (3) declare nested hierarchies of information, (4) specify mixed content, and (5) use content from another data model.

The chapter makes the following contributions. First, it proposes conceptual representation for XML content structures that are not explicitly present in traditional conceptual models. Second, based on the underlying idea of the proposed representation, it suggests ways to represent missing XML content structures in two of the most popular conceptual models, ER and UML.

We present the details of our contributions as follows. Section 2 lists criteria an XML conceptual model should satisfy. Section 3 describes the structural constructs in XML Schema that are missing in traditional conceptual models. Section 4 explains how we model these features of XML Schema in a modeling language we call Conceptual XML (C-XML). Section 5 compares our proposal with other proposals for ways to extend some traditional conceptual models to represent some XML features and shows how to adapt C-XML representations for traditional conceptual models. Section 6 summarizes and draws conclusions.

2 XML Modeling Criteria

Lists of requirements for XML conceptual models have been presented in [18], [13], and [9]. Some of these requirements cover general goals of conceptual modeling, while others are specific to XML. General requirements include the following:

- *Graphical notation.* The notation should be graphical and should be user-friendly [9, 13, 18].
- *Formal foundation.* The conceptual model should have a formal foundation [9, 13, 18].
- *Structure independence.* The notation should ensure that the basics of the conceptual model are not influenced by the underlying structure, but reflect only the conceptual components of the data [9, 13, 18].
- *Reflection of the mental model.* The conceptual model must be consistent with a designer's mental conceptualization of objects and their interrelationships [13]. For example, there should be no distinction between element and attribute on the conceptual level, and hierarchies should not be required.
- *N-ary relationship sets.* The conceptual model should be able to represent n-ary relationship sets at the conceptual level [9].
- *Views.* It should be possible to transform the model to present multiple user views [9].
- *Logical level mapping.* There should be algorithms for mapping the conceptual modeling constructs to XML Schema [9, 18].
- *Constraints.* The conceptual model should support common data constraints such as cardinality and uniqueness constraints [13].
- *Cardinality for all participants.* The hierarchical structure of XML data restricts the specification of cardinality constraints only to nested participants;

however, it should be possible to specify cardinality constraints for all participants at the conceptual level [9].
- *Ordering.* The conceptual model should be able to order a list of concepts [9, 13].
- *Irregular and heterogeneous structure.* The conceptual model should introduce constructs for modeling irregular and heterogeneous structure [9].
- *Document-centric data.* The conceptual model should be able to represent the mixed content and open content that XML Schema provides [9, 13, 18].

3 Missing Modeling Constructs

In this section we give an overview of the structural constructs in XML Schema that are missing in traditional conceptual models. We explain each and give a motivating example, which we also use in later sections to illustrate conceptual model augmentations.

The *sequence* structure specifies that the child concepts declared inside it must appear in an XML document in the order declared. Each ordered child concept can occur zero or more times within the sequence constrained by *minOccurs* and *maxOccurs* attributes. Likewise, the entire *sequence* itself can occur zero or more times. The default value for both *minOccurs* and *maxOccurs* is always *1*. The *sequence* construct may include several types of child constructs: *element*, *group*, *choice*, *sequence*, and *any*. Lines 15–23 in Figure 1 specify that in a complying XML document an element *School* contains a sequence of required *SchoolName*, *SchoolAddress*, and *SchoolID* elements, and an optional *SchoolMascot* element.

The *choice* structure specifies that for each choice only one of the child concepts declared within it can appear in an XML document. Each child concept in the *choice* can occur zero or more times within the choice constrained by *minOccurs* and *maxOccurs* attributes. Likewise, the entire *choice* itself can occur zero or more times. The default value for *minOccurs* and *maxOccurs* for both the entire choice and the component children is *1*. The *choice* construct may include several types of child constructs: *element*, *group*, *choice*, *sequence*, and *any*. In Figure 1, lines 47–55 specify that in a complying XML document an element *ContactInfo* contains one or two choices, and each choice contains either one *PhoneNumber*, one *Email*, or one *Fax*.

By default, structural constructs in XML Schema can contain child elements, but not text. To allow mixed content (child elements and text), XML Schema provides a *mixed* attribute that can be set to true. In Figure 1, lines 43–58 show an example of mixed content for a complex type. Setting *mixed* to true enables character data to appear between the child elements of *RecommendationLetter* in a complying XML document. Thus, the content of *RecommendationLetter* may, for example, be "<RecommendationLetter> <ProfessorName> Dr. Jones </ProfessorName> recommends this student. Email <ContactInfo><Email>jones@ univ.edu </Email> </ContactInfo> with questions. </RecommendationLetter>".

The *any* and *anyAttribute* structures of XML Schema let designers reuse components from foreign schemata or namespaces. The *any* structure allows

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:element name="StudentInfo">
      <xs:complexType>
         <xs:sequence>
            <xs:choice>
               <xs:element name="Name" type="xs:string"/>
               <xs:sequence>
                  <xs:element name="FirstName" type="xs:string"/>
                  <xs:element name="MiddleName" type="xs:string" minOccurs="0" maxOccurs="2"/>
                  <xs:element name="LastName" type="xs:string"/>
               </xs:sequence>
            </xs:choice>
            <xs:sequence maxOccurs="5">
               <xs:element name="School">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="SchoolName" type="xs:string"/>
                        <xs:element name="SchoolAddress" type="xs:string"/>
                        <xs:element name="SchoolID" type="xs:string"/>
                        <xs:element name="SchoolMascot" type="xs:string" minOccurs="0"/>
                     </xs:sequence>
                  </xs:complexType>
                  <xs:key name="schoolKey">
                     <xs:selector xpath=".//School"/>
                     <xs:field xpath="SchoolName"/>
                     <xs:field xpath="SchoolAddress"/>
                  </xs:key>
                  <xs:key name="schoolIDKey">
                     <xs:selector xpath=".//School"/>
                     <xs:field xpath="SchoolID"/>
                  </xs:key>
               </xs:element>
               <xs:element name="GraduationDate" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="Month" type="xs:string"/>
                        <xs:element name="Year" type="xs:string"/>
                     </xs:sequence>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
            <xs:element name="RecommendationLetter" minOccurs="0" maxOccurs="3">
               <xs:complexType mixed="true">
                  <xs:all>
                     <xs:element name="ProfessorName" type="xs:string"/>
                     <xs:element name="ContactInfo">
                        <xs:complexType>
                           <xs:choice maxOccurs="2">
                              <xs:element name="PhoneNumber" type="xs:string"/>
                              <xs:element name="Email" type="xs:string"/>
                              <xs:element name="Fax" type="xs:string"/>
                           </xs:choice>
                        </xs:complexType>
                     </xs:element>
                  </xs:all>
               </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="StudentNumber" type="xs:ID" use="required"/>
         <xs:anyAttribute namespace="##any"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Fig. 1. More Example of Choice/Sequence Structures in XML Schema.

the insertion of any element belonging to a list of namespaces, and it can have *minOccurs* and *maxOccurs* attributes to define the number of occurrences of the *any* construct. The *anyAttribute* structure allows the insertion of any attribute belonging to a list of namespaces. Both *any* and *anyAttribute* can have *namespace* and *processContents* as attributes. The attribute *namespace* specifies the namespaces that an XML validator examines to determine the validity of an element in an XML document. The attribute *processContents* specifies how the XML processor should handle validation against the elements specified by the *any* or *anyAttribute*. In Figure 1, the *any* element in line 59 specifies that zero or more elements from any other namespace can appear after the *RecommendationLetter* element. Further, the *anyAttribute* specification in line 62 indicates that the *StudentInfo* element can have additional attributes from any namespace. When *processContents* is *strict*, the XML processor must obtain the schema for the required namespaces and validate the elements. When *processContents* is set to *lax*, the XML processor attempts the same processing as for *strict*, but ignores errors if validation fails. When *processContents* is *skip*, the XML processor does not attempt to validate any elements from the specified namespaces.

In XML Schema, it is possible to nest structural constructs, thus forming a hierarchy of nested constructs. In Figure 1, for example, *StudentInfo* has the attributes *StudentNumber* and *anyAttribute*, and it also contains the following structures in order: first, either a *Name* or a sequence of one *FirstName*, zero to two *MiddleName*'s, and one *LastName*; second, one to five sequences such that each sequence includes one *SchoolName*, one *SchoolAddress*, and an optional *GraduationDate* (the *GraduationDate* itself contains a *Month* followed by a *Year*); third, an element *RecommendationLetter* that has two elements, *ProfessorName* and *ContactInfo* (*ContactInfo* in turn contains one to two choices such that in each choice either *PhoneNumber* or *Email* or *Fax* is specified); and fourth, an optional *any* element.

4 C-XML

In this section we propose an enrichment to represent XML Schema content structures that are usually missing in traditional conceptual models. Since hypergraphs provide a general representation for conceptual models, we begin with an augmented hypergraph whose vertices and edges are respectively object sets and relationship sets, and whose augmentations consist of decorations that represent constraints. A hypergraph foundation is amenable to the requirements of XML Schema, and thus this choice simplifies the correspondence between conceptual models and XML Schema. We call our representation Conceptual XML (C-XML).

We derive C-XML from OSM [5], a hypergraph-based conceptual model that defines structure in terms of *object sets* (or *concepts*), *relationship sets*, and *constraints* over these object and relationship sets. Figure 2 shows a C-XML model instance that corresponds to the XML schema of Figure 1. An object set with a solid border indicates a nonlexical concept, a dashed border indicates a lexical

concept, and a double solid/dashed border indicates a *mixed* concept.[1] A shaded object set indicates a high-level object set that groups other object and relationship sets into a single object set. Lines connecting object sets are *relationship sets*. A *participation constraint* specifies how many times an object in a connected relationship may participate in a relationship set. For the most common participation constraints (*0:1*, *1:1*, *0:**, and *1:**), C-XML uses graphical notation as a shorthand: (1) an "o" on a connecting relationship-set line designates *optional participation*, while the absence of an "o" designates *mandatory*, and (2) an arrowhead specifies a *functional constraint*, limiting participation of objects on the tail side of the arrow to be at most one.

The *sequence* structure representation must be able to specify concepts in a sequence in a particular order. Also, the representation must be able to specify the minimum and maximum numbers of occurrence of the whole sequence and of each child element within the sequence. For C-XML we let a bounded half circle with a directional arrow represent a sequence. The sequenced child concepts connect to the curved side, and the parent concept that contains the sequenced child concepts connects to the flat side. We place participation constraints for the entire sequence near the connection to the parent. We place participation constraints for each child near the curved side of the sequence symbol. Note that C-XML has participation constraints that represent the minimum and maximum number of occurrences of the sequence in the relationship set between the parent and the sequence. C-XML also allows participation constraints that represent the minimum and maximum allowed occurrences of the sequence in the relationship set between the sequence and each sequenced child concept.

The representation for *choice* is similar in appearance to the representation for *sequence*, but instead of an arrow we use a vertical bar to indicate choice.

For *any* and *anyAttribute* we use a high-level object set to indicate that it contains some content from another schema. XML Schema is not specific enough to designate which concept, and thus we cannot specify which concept. We therefore name these concepts "any". Conceptually, in C-XML whether the concept is an attribute or an element does not matter, and we do not distinguish between these cases.

We now evaluate C-XML with respect to the criteria for XML conceptual models in Section 2.

- *Graphical notation.* We have presented a sufficient graphical notation, but this is just one possibility among many.
- *Formal foundation.* OSM has a solid formal foundation in terms of predicate calculus (see Appendix A of [5]). In OSM, each object set maps to a one-place predicate, and each n-ary relationship set ($n \geq 2$) relationship set maps to

[1] In an XML document, the content string for a mixed concept might be interspersed among a number of child nodes. However, in C-XML the *mixed* concept does not explicitly specify how text and child elements can be interleaved. If the pattern for interspersing chunks of the string among child nodes matters, then the user must model text nodes explicitly (in combination with a sequence structure) rather than use the generic *mixed* construct.

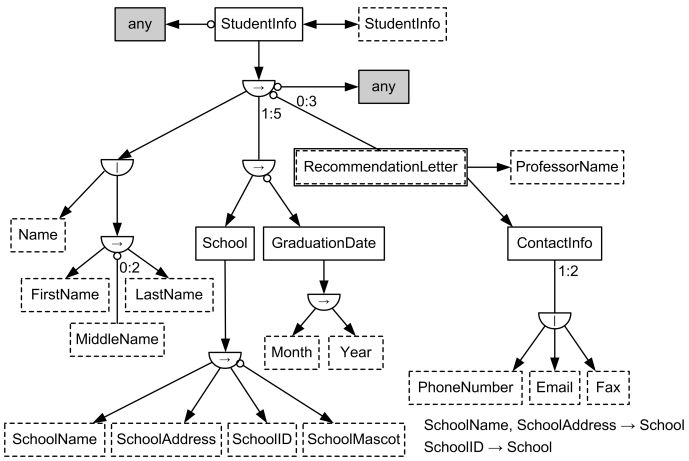

Fig. 2. Sequence/Choice Structures for Figure 1.

an n-place predicate. Each constraint (e.g. a participation constraint) maps to a closed predicate-calculus formula. In the appendix of this chapter we provide formal representations for the added features for C-XML: sequence, choice, mixed content, and general co-occurrence constraints.

– *Structure independence.* XML in general, and XML Schema in particular, are strongly hierarchical in nature. C-XML is capable of representing the hierarchical aspect of XML Schema, but C-XML is more general, flexible, and conceptual. For example, C-XML allows multiple sequence and choice structures to be associated directly with a single concept (XML Schema allows only one sequence or choice structure for the content of an element). Also, C-XML supports the intermixing of ordinary relationship sets with sequence and choice structures. From this conceptual structure, we can derive many possible hierarchical representations. Similarly, C-XML defines generalized versions of the concepts of sequence, choice, and mixed content. C-XML provides a conceptual perspective that is structurally independent of XML Schema.
– *Reflection of the mental model.* Given its structure independence and generality, C-XML is well suited to reflect the mental model (design) of a modeler. C-XML can represent hierarchical and non-hierarchical structure. Conceptually, whether a concept is an attribute or an element does not matter, and C-XML does not distinguish between them. C-XML is also able to represent both sequences among related entities and non-sequences among related entities. Choices among alternative related entities are also possible, and choice is distinct from generalization/specialization so that neither is overloaded. C-XML supports mixed content and open content. Finally, C-XML provides for all XML cardinality constraints; indeed it provides for a very large spectrum of cardinality constraints [8] encompassing and going beyond those provided by XML.

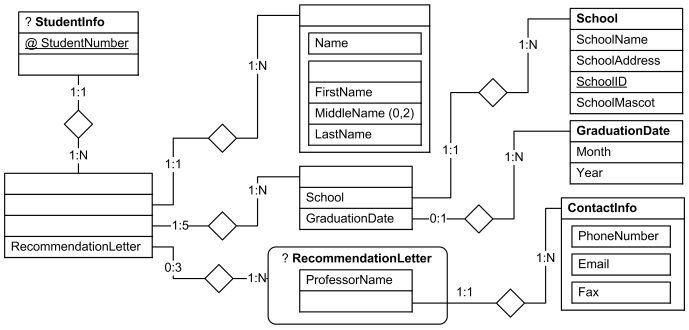

Fig. 3. Best Representation of Figure 1 using XER Notation.

- *N-ary relationship sets.* C-XML supports n-ary relationship sets, ($n \geq 2$).
- *Views.* High-level object sets constitute a formal view mechanism, as do high-level relationship sets [5]. As described above, C-XML also can represent both hierarchical and non-hierarchical views.
- *Logical level mapping.* We have implemented automatic conversions from XML Schema to C-XML and vice versa.
- *Constraints.* C-XML supports several kinds of constraints: set constraints, referential-integrity constraints, cardinality constraints, and general constraints.
- *Cardinality for all participants.* C-XML goes further than XML Schema, even allowing cardinality constraints for children of a sequence or choice.
- *Ordering.* C-XML explicitly supports ordering with its sequence construct.
- *Irregular and heterogeneous structure.* The features that give C-XML its structure independence (described above) provide for the modeling of irregular and heterogeneous structure.
- *Document-centric data.* C-XML is able to represent both mixed content and open content.

5 Augmenting ER and UML

A number of conceptual modeling languages for XML Schema have been described in the literature. Sengupta and Mohan [11] and Necasky [9] present fairly recent surveys. As we explain in this section, however, most of these efforts do not support the full generality of XML Schema.

5.1 ER

Sengupta et al. [12] propose XER as an extension to the ER model for XML. Figure 3 shows an example of XER; in fact, it shows the best that can be done to represent the XML schema in Figure 1. As we will see, it does not capture all the concepts and constraints in the XML schema in Figure 1.

XER represents an entity such as *StudentInfo* or *GraduationDate* using a rectangle with a title area giving the name of the entity and the body giving the

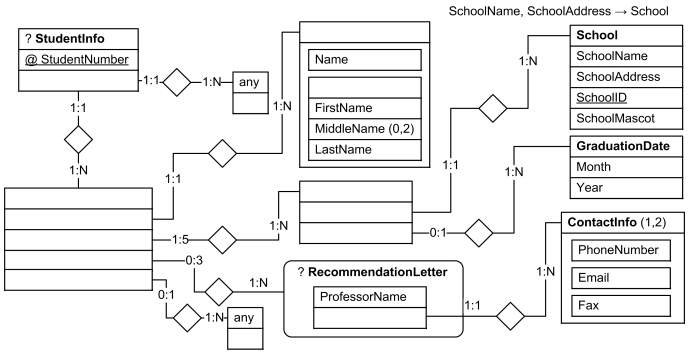

Fig. 4. Possible Way to Represent XML Schema Document in Figure 1 in ER-XML.

attributes. For example, in Figure 1, *Month* and *Year* are sequenced elements nested under the element *GraduationDate*, so in Figure 3 *Month* and *Year* are represented as attributes for the *GraduationDate* entity. Multi-valued attributes are also allowed; their multiplicity constraints are in parentheses. *MiddleName*, for example, is a multi-valued attribute with a multiplicity *(0,2)*. XML attributes in an XER entity are prefixed with @, and key attributes are underlined. The attribute *StudentNumber* is a key in Figure 1, so in Figure 3 it appears as an underlined attribute with a prefix of @.

An XER entity can be ordered or unordered. Additionally, an XER entity can be mixed.

- Ordered Entity. XER entities are ordered by default from top to bottom. The ordered entity *GraduationDate* in Figure 3 indicates that its attributes are ordered first *Month*, then *Year*.
- Unordered Entity. An unordered entity is represented by placing a question mark (?) in front of the entity name. *StudentInfo* in Figure 3 is an unordered entity.
- Mixed Entity. A mixed entity is represented in XER using a rounded rectangle. *RecommendationLetter* in Figure 3 is a mixed entity.

XER relationships denote a connection between two or more entities, but in XER they can also denote that a complex entity contains a complex element as one of its sub-elements. When an entity E in XER has an attribute A and this attribute A by itself is an entity that contains other attributes, then A appears in the XER diagram twice, once as an attribute inside the entity E, and once as an entity A. In addition, there is a connection between the attribute A inside the entity E and the entity A. If *minA:maxA* is the participation constraint on A within E and *minE:maxE* is the participation constraint on E for A, *minA:maxA* appears on the side of the attribute A within E, and *minE:maxE* appears on the side of the entity A. For example, *RecommendationLetter* has two attributes *ProfessorName* and *ContactInfo*, but *ContactInfo* by itself is an entity. Thus, a relationship set appears between the attribute *ContactInfo* inside *Recommenda-*

tionLetter and the entity *ContactInfo*. A participation constraint of *1:1* appears on the side of the attribute *ContactInfo* inside *RecommendationLetter* to denote that *RecommendationLetter* has one *ContactInfo*, and a participation constraint of *1:N* appears on the *ContactInfo* entity side to denote that *ContactInfo* is for one or more *RecommendationLetter*s.[2]

XER represents the choice concept in XML Schema as a generalization/specialization. The generalization term in XER refers to the concept of having an entity that can have different specialization entities in an ISA relationship. XER represents a generalization using a covering rectangle containing the specialized XER entities. This, the authors claim in [12], is equivalent to using the "xs:choice" tag in XML Schema. In Figure 3 the rectangle representing the entity *ContactInfo* contains the rectangles of entities of choice elements *PhoneNumber*, *Email*, and *Fax*.

Comparing the conceptual components for C-XML (e.g. Figure 1) and XER (e.g. Figure 3), we see that several constructs and constraints are missing in XER. First, XER lacks the ability to represent the minimum and the maximum occurrence of the whole sequence or choice within a containing entity when either of their values is more than *1*. For example, XER cannot represent the minimum and maximum occurrence of *1* to *2* for the *choice* within the entity *ContactInfo*. Second, XER has no representation for *any* and *anyAttribute* structures. For example, in Figure 3 the entity *StudentInfo* is missing the *anyAttribute*, and the sequence contained inside the *StudentInfo* entity does not have *any*. Third, XER has no representation for composite keys. For example, in Figure 3 the representation that *SchoolName* and *SchoolAddress* together constitute a key for the entity *School* is missing. Fourth, although XER has a representation for a single key, this representation only applies when the key for an entity is an attribute of that entity. The representation is not able to specify a key constraint for an entity within the context of another entity.

Beyond these omissions, we have several concerns about some representations in XER.

- Representing *choice* by generalization/specialization is problematic; the formal definition of *choice* differs from the formal definition of generalization/specialization. First, *choice* contains different types of alternative concepts, but all the specialized concepts in generalization/specialization hierarchies typically must have the same type. Second, in generalization/specialization hierarchies any specialized concept inherits relationship sets from its generalization concepts, while in *choice*, the alternative concepts do not inherit relationship sets. Third, the participation constraints for *choice* allow alternative concepts to appear more than one time, while in generalization/specialization hierarchies specialized concepts can appear at most once.
- In XER it is not clear from [12] whether it is possible to represent an entity without having a name for the entity. For Figure 3 we assume that we are able to represent an entity in XER with a null name. Also, in XER it is not

[2] Although ER more commonly uses look-across cardinality constraints, the designers of XER have chosen to use participation constraints [12].

clear whether it is possible to have an empty slot in an entity to indicate that an attribute by itself is an entity without a name. We also assume for Figure 3 that we are able to do so in XER. From [12] it is not clear whether it is possible to have hierarchies of *choice* and *sequence* structures, but we assume that this is possible as Figure 3 shows.
- In XER when an entity has an attribute and this attribute is also an entity, the model instance in XER has an attribute and an entity with the same name. This redundancy might cause problems if XER developers are able to write the two names independently.

In light of these omissions and concerns, we extend XER, augmenting it with constructs and constraints that are missing and resolving our concerns. Figure 4 shows our suggested way of representing the schema in Figure 1 in ER-XML, our ER augmentation for XML. We add *any* and *anyAttribute* concepts to XER. We have chosen to add a representation of *any* and *anyAttribute* as entities with the name *any*. We also add minimum and maximum occurrence to *sequence* and *choice*, placing this minimum and maximum in parentheses in the name slot, following the name, if any, of the entity that declares the *sequence* or *choice*. We have chosen to add a representation for key constraints by allowing functional dependencies that must hold within entity sets or along paths of relationship sets. Thus, for example, as Figure 4 shows, we can specify the composite key *SchoolName*, *SchoolAddress* by the functional dependency *SchoolName*, *SchoolAddress* $\longrightarrow$ *School*. Although we use the same notation for *choice*, we do not consider the representation of *choice* in ER-XML to be a generalization concept. Finally, we do not repeat attribute names, writing the name only in the entity that represents the attribute.

5.2 UML

Conrad et al. [4] add features to UML to enable mappings from class diagrams to XML DTDs. Figure 5 shows an example; in fact, it shows the best that can be done to represent the XML schema in Figure 1. Unfortunately, it does not capture all the concepts and constraints in the XML schema in Figure 1.

As described in [4], Conrad et al. augment UML aggregation so that it can be transformed into a *sequence* construct or a *choice* construct. The designation {*sequence*} specifies a left-to-right ordering of elements, and the designation {*choice*} specifies a choice among elements. For a sequence the first constituent element is marked as *1*, the second as *2*, and so forth. A *sequence* or *choice* construct may have cardinality to represent the minimum and maximum occurrence of the entire sequence or choice. For example, the class *ContactInfo* in Figure 5 has one to two choices {*choice* : 1..2} of the classes *PhoneNumber*, *Email*, and *Fax*. For an *any* structure, the notation in [4] uses the «*content*» stereotype.

Comparing the conceptual components for C-XML (e.g. Figure 1) and extended UML presented in [4] (e.g. Figure 5), we see that several constructs are missing. First, the extended UML in [4] lacks the ability to represent an

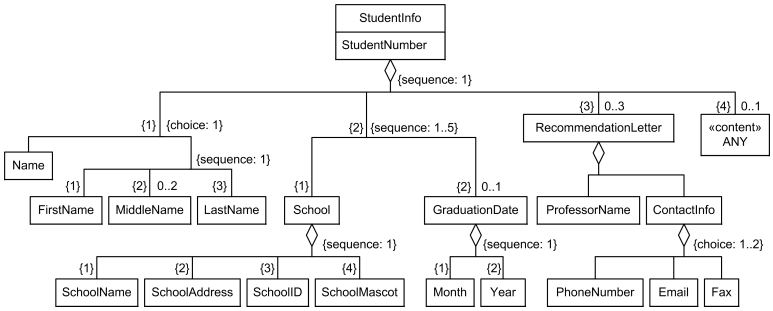

Fig. 5. Best Representation of Figure 1 Using Conrad Notation.

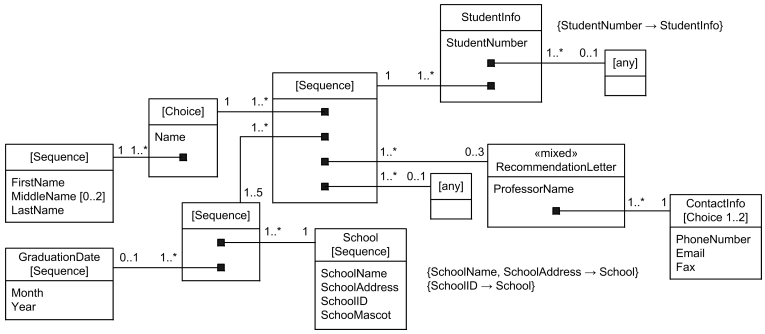

Fig. 6. Possible Way to Represent XML Schema Document in Figure 1 in UML-XML.

anyAttribute. For example, in Figure 5 the class *StudentInfo* is missing the *anyAttribute*. Second, the extended UML in [4] lacks the ability to represent mixed content. In Figure 5 the class *RecommendationLetter* does not appear as having mixed content. Third, the extended UML in [4] lacks key constraints, although, in principle, we could specify key constraints using OCL (the constraint language of UML).

Besides these omissions, we have concerns about the suggested representation of sequence and choice in [4]. The suggested representations can only be applied between classes, not between attributes. This is because Conrad et al. augment UML aggregation for *sequence* and *choice*. Since the aggregation in UML applies to classes, the notation forces attributes to be represented as classes. For example, to represent the *GraduationDate* class as a sequence of *Month* and *Year*, would-be attributes *Month* and *Year* must each become a class first.

To overcome these difficulties, we need to extend and adjust the representations in [4]. Figure 6 shows our suggested extensions and adjustments by rendering Figure 1 in UML-XML, our UML augmentation for XML.

– We have chosen to represent the *anyAttribute* as an associated class with the *any* content type rather than as a stereotype. For mixed content we use

the «*mixed*» stereotype. The *RecommendationLetter* class in Figure 6 is an example.
- We suggest representing *sequence* and *choice* in a different way so that we do not force attributes to be represented as classes. When attributes in a class are ordered, we add the designation [*Sequence*] under the class name to specify a top-to-bottom ordering of the attributes. We also add *minOccurs..maxOccurs*, if needed, to express participation different from the default. For example, in Figure 6, the designation [*Sequence*] is added under *GraduationDate*. Similarly, we allow designating a choice construct by adding [*Choice minOccurs..maxOccurs*], allowing *minOccurs .. maxOccurs* to be omitted when it is *1..1*, the default. For example, in Figure 6, the designation [*Choice 1..2*] is added under *ContactInfo*.
- We add notation to denote that a class contains an attribute and that this attribute is a class that contains other attributes. A connection appears that connects an empty slot indicating the presence of an attribute inside the class with the class containing other attributes. For example, we indicate that *ContactInfo* is an attribute inside the class *RecommendationLetter* by the connection inside *RecommendationLetter* that extends to *ContactInfo*. Note also that *ContactInfo* by itself is a class that contains attributes. A multiplicity of *1* is added to the *ContactInfo* class side and a multiplicity of *1..** is added to the *ContactInfo* attribute side in the *RecommendationLetter* class to denote that *ContactInfo* is for *1* or more *RecommendationLetter*s and each *RecommendationLetter* has one *ContactInfo*.
- For the case when a sequence or choice is a complex attribute inside a class *C*, the *sequence* or *choice* is represented as a class with no name but has the designation [*Sequence*] or [*Choice*], and we connect the empty slot inside the class *C* with the class that represents the *sequence* or *choice*. For example, the class *StudentInfo* has a complex sequence attribute. Further, this sequence by itself is a class that contains other attributes including another complex choice attribute and a complex sequence attribute.
- We can specify key constraints in UML by using OCL. But, since this is a common task, we have an alternative representation that we can add to a diagram. We have chosen to add a representation for key constraints by allowing functional dependencies which must hold within classes or along paths of associations. Thus, for example, as Figure 6 shows, we can specify the composite key *SchoolName*, *SchoolAddress* by the functional dependency $\{SchoolName, SchoolAddress \longrightarrow School\}$.

5.3 ER-XML, UML-XML, and C-XML

Comparing ER-XML, UML-XML, and C-XML, we make the following observations according to the criteria for XML conceptual modeling we described in Section 2. Criteria from Section 2 not listed here have equal validity among the three models (e.g. all three have a *graphical notation*).

- *Formal foundation.* C-XML has a solid formal foundation in terms of predicate calculus. ER-XML and UML-XML are respectively derived from XER

as described in [12] and UML as described in [4]. There is no formal foundation for XER [9], and the underlying formalism of UML is not fully developed [6]. In principle both could have complete formal foundations.

- *Reflection of the mental model.* ER-XML distinguishes attributes from entities and UML-XML distinguishes attributes from classes. C-XML represents all concepts as object-set nodes in hypergraphs. Forcing attributes to be embedded within an entity/class has the disadvantage that a user of UML-XML or ER-XML has to decide before representing any concept whether it should be an attribute or entity/class. Distinguishing between an attribute and an entity/class is not necessary and may even be harmful as a mental-model conceptualization. Goldstein and Storey [7] showed that this can be a major source of errors in conceptual modeling.
- *Views.* Hypergraphs are typically more amenable to translations to various views and even alternate XML schemas such as normalized XML schemas. Further, although not discussed here, C-XML supports both high-level object sets and high-level relationship sets as first class concepts [5]. Neither ER-XML nor UML-XML supports high-level view constructs.
- *Logical level mapping.* We have implemented both a mapping from XML Schema to C-XML and vice versa [1, **?**]. In principle mappings to and from XML Schema and ER-XML as well as UML-XML are possible.
- *Cardinality for all participants.* The nesting representation for ER-XML and UML-XML restricts the specification of cardinality constraints to only the nesting participants. C-XML specifies cardinality constraints for all participants, beyond even those supported by XML Schema.

6 Conclusion

In this chapter we discussed the structural constructs in XML Schema that are missing in traditional conceptual models. Our proposed solution is to enrich conceptual models with the ability to order a list of concepts, choose alternative concepts from among several, specify mixed content, and use content from another data model. We presented our solution using C-XML, and we showed that our solution can be adapted and used for the ER and UML languages.

We also presented requirements for conceptual modeling for XML. We based these requirements on those presented in [9], [**?**], and [18]. We evaluated C-XML against these requirements and showed that C-XML satisfies all of them, which makes C-XML a good candidate for a conceptual modeling language for XML. We also argued that ER-XML and UML-XML, our adaptations for ER and UML, also largely satisfy these requirements, but do not satisfy them as well as does C-XML.

We have implemented C-XML, and we have implemented conversions from XML Schema to C-XML and vice versa. Currently, we are working on a formal proof that our conversions to and from C-XML and XML Schema preserve information and constraints.

7 Acknowledgments

This work is supported in part by the National Science Foundation under grant number IIS-0083127 and by the Kevin and Debra Rollins Center for eBusiness under grant number EB-05046.

References

1. R. AL-Kamha. *Translating XML Schema to Conceptual XML.* Technical Report, Computer Science Department, Brigham Young University, November 2006.
2. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, Reading, Massachusetts, 1999.
3. P.P. Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.
4. R. Conrad, D. Scheffner, and J.C. Freytag. XML conceptual modeling using UML. In *Proceedings of the Ninteenth International Conference on Conceptual Modeling (ER2000)*, LNCS 1920:558–571, 2000.
5. 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.
6. A Formal Semantics for UML Workshop, October 2006. URL: `http://www.cs.queensu.ca/~stl/internal/uml2/MoDELS2006/`.
7. R.C. Goldstein and V.C. Storey. Some findings on the intuitiveness of entity-relationship constructs. In *Proceedings of the Eighth International Conference on Entity-Relationship Approach (ER'89)*, pages 9–23, Toronto, Canada, October 1989. North-Holland.
8. S.W. Liddle, D.W. Embley, and S.N. Woodfield. Cardinality constraints in semantic data models. *Data & Knowledge Engineering*, 11(3):235–270, 1993.
9. M. Necasky. Conceptual modeling for XML: A survey. In *Proceedings of the DATESO 2006 Annual International Workshop on Databases, Texts, Specifications and Objects (DATESO 2006)*, pages 40–53, Desna, Czech Republic, April 2006.
10. Visual Studio.NET, Microsoft. `http://www.msdn.microsoft.com/vstudio`.
11. A. Sengupta and S. Mohan. *Formal and Conceptual Models for XML Structures—The Past, Present, and Future.* Technical Report 137–1, Indiana University, Information Systems Department, Bloomington, Indiana, April 2003.
12. A. Sengupta, S. Mohan, and R. Doshi. XER — extensible entity relationship modeling. In *Proceedings of XML 2003*, Philadelphia, Pennsylvania, December 2003.
13. A. Sengupta and E. Wilde. *The Case for Conceptual Modeling for XML.* Technical Report No. 242, Computer Engineering and Networks Laboratory, ETH Zurich, February 2006.
14. XMLSpy, Altova. `http://www.xmlspy.com`.
15. Stylus Studio. `http://www.stylusstudio.com/xml_schema_editor.html`.
16. T.J. Teorey, D. Yang, and J.P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, June 1986.
17. UML 2.0 superstructure specification, August 2005.
18. E. Wilde. Towards conceptual modeling for XML. In *Proceedings of the Berliner XML Tage 2005 (BXML2005)*, pages 213–224, Berlin, Germany, September 2005.

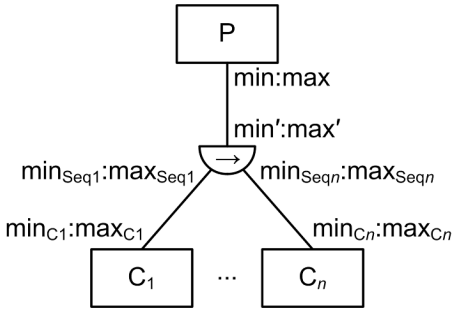

Fig. 7. Sequence Structure in C-XML.

Sequence

Figure 7 shows the schematic structure of a sequence. Exactly one parent object set connects to a sequence of n children, $n \geq 0$, with participation constraints on the several connections as Figure 7 shows. A sequenced child may be either an object set or a nested sequence or choice structure. In general, there may be many sequences in a model instance, and since we do not explicitly name sequence structures, we denote a particular sequence, the kth sequence, by $Sequence_k$. Let P be the name of the parent object set for $Sequence_k$, and let C_1, ..., C_n be the names of the n child object sets or nested sequences or choices that are sequenced within $Sequence_k$. To impose order, we introduce the unary predicate *Order*, which we can think of as an object set containing as many ordinal numbers as we need 1, 2, ... We denote the minimum and maximum cardinalities of $Sequence_k$ according to Figure 7. Let min and max be, respectively, the minimum and maximum number of occurrences of $Sequence_k$ allowed for an object in P. Let min_{C_i} and max_{C_i}, $1 \leq i \leq n$, be, respectively, the minimum and maximum number of allowed occurrences of C_i objects within $Sequence_k$. Let min' and max' be, respectively, the minimum and maximum number of occurrences of $Sequence_k$ sequences in the relationship set between P and $Sequence_k$. Finally, let min_{Seq_i} and max_{Seq_i}, $1 \leq i \leq n$, be, respectively, the minimum and maximum allowed occurrences of $Sequence_k$ in the relationship set between $Sequence_k$ sequences and C_i (i.e. the number of C_i objects that can be associated with $Sequence_k$ for a given order position). For $Sequence_k$, we have the following object sets, relationship sets, and constraints.

Object Sets:

- $P(x)$
- $Sequence_k(x)$
- $Order(x)$
- $C_1(x)$, ..., $C_n(x)$

Relationship Sets:

- $P(x)$ *contains* $Sequence_k(y)$
- $C_1(x)$ *has* $Order(1)$ *in* $Sequence_k(y)$

– ...
– $C_n(x)$ *has* $Order(n)$ *in* $Sequence_k(y)$

Referential Integrity:

– $\forall x \forall y(P(x)$ *contains* $Sequence_k(y) \Rightarrow P(x) \wedge Sequence_k(y))$
– $\forall x \forall y(C_1(x)$ *has* $Order(1)$ *in* $Sequence_k(\mathrm{y}) \Rightarrow C_1(x) \wedge Order(1) \wedge Sequence_k(y))$
– ...
– $\forall x \forall y(C_n(x)$ *has* $Order(n)$ *in* $Sequence_k(y) \Rightarrow C_n(x) \wedge Order(n) \wedge Sequence_k(y))$

Participation Constraints:

– $\forall x(P(x) \Rightarrow \exists^{\geq min} y(P(x)$ *contains* $Sequence_k(y))) \wedge$
$\forall x(P(x) \Rightarrow \exists^{\leq max} y(P(x)$ *contains* $Sequence_k(y)))$
– $\forall x(Sequence_k(x) \Rightarrow \exists^{\geq min'} y(P(y)$ *contains* $Sequence_k(x))) \wedge$
$\forall x(Sequence_k(x) \Rightarrow \exists^{\leq max'} y(P(y)$ *contains* $Sequence_k(x)))$
– $\forall x(Sequence_k(x) \Rightarrow \exists^{\geq min_{Seq_1}} y_1 \ ... \ \exists^{\geq min_{Seq_n}} y_n($
$C_1(y_1)$ *has* $Order(1)$ *in* $Sequence_k(x)$
$\wedge \ ... \ \wedge$
$C_n(y_n)$ *has* $Order(n)$ *in* $Sequence_k(x))) \wedge$
$\forall x(Sequence_k(x) \Rightarrow \exists^{\leq max_{Seq_1}} y_1 \ ... \ \exists^{\leq max_{Seq_n}} y_n($
$C_1(y_1)$ *has* $Order(1)$ *in* $Sequence_k(x)$
$\wedge \ ... \ \wedge$
$C_n(y_n)$ *has* $Order(n)$ *in* $Sequence_k(x)))$
– $\forall x(C_1(x) \Rightarrow \exists^{\geq min_{C_1}} y(C_1(x)$ *has* $Order(1)$ *in* $Sequence_k(y)))$
$\wedge$
$\forall x(C_1(x) \Rightarrow \exists^{\leq max_{C_1}} y(C_1(x)$ *has* $Order(1)$ *in* $Sequence_k(y)))$
$\wedge \ ... \ \wedge$
$\forall x(C_n(x) \Rightarrow \exists^{\geq min_{C_n}} y(C_n(x)$ *has* $Order(n)$ *in* $Sequence_k(y)))$
$\wedge$
$\forall x(C_n(x) \Rightarrow \exists^{\leq max_{C_n}} y(C_n(x)$ *has* $Order(n)$ *in* $Sequence_k(y)))$

Choice

The schematic structure of a choice is similar to sequence (see Figure 7). Exactly one parent object set connects to a group of n children, $n \geq 0$, and the participation constraints for choice are similar to those for sequence. As with sequence, children of a choice may be either object sets or nested sequence or choice structures. In general, there may be many choices in a model instance, and since we do not explicitly name choice structures, we denote a particular choice the kth choice, by $Choice_k$. Let P be the name of the parent object set for $Choice_k$, and let C_1, ..., C_n be the names of the n child object sets or nested sequences or choices that are alternatives for $Choice_k$. Let min and max be, respectively, the minimum and maximum number of occurrences of $Choice_k$ allowed for an object in P. Let min_{C_i} and max_{C_i}, $1 \leq i \leq n$, be, respectively, the minimum and maximum number of allowed occurrences of C_i objects within $Choice_k$. Let min' and max' be, respectively, the minimum and maximum number of occurrences of $Choice_k$ choices in the relationship set between P and $Choice_k$. Finally, let min_{Cho_i} and max_{Cho_i}, $1 \leq i \leq n$, be, respectively, the minimum and maximum

allowed occurrences of $Choice_k$ choices in the relationship set between $Choice_k$ and C_i. For $Choice_k$, we have the following object sets, relationship sets, and constraints.

Object Sets:

- $P(x)$
- $Choice_k(x)$
- $C_1(x)$, ..., $C_n(x)$

Relationship Sets:

- $P(x)$ *contains* $Choice_k(y)$
- $C_1(x)$ *is alternative for* $Choice_k(y)$
- ...
- $C_n(x)$ *is alternative for* $Choice_k(y)$

Referential Integrity:

- $\forall x \forall y(P(x)$ *contains* $Choice_k(y) \Rightarrow P(x) \wedge Choice_k(y))$
- $\forall x \forall y(C_1(x)$ *is alternative for* $Choice_k(\mathrm{y}) \Rightarrow C_1(x) \wedge Choice_k(y))$
- ...
- $\forall x \forall y(C_n(x)$ *is alternative for* $Choice_k(y) \Rightarrow C_n(x) \wedge Choice_k(y))$

Participation Constraints:

- $\forall x(P(x) \Rightarrow \exists^{\geq min} y(P(x)$ *contains* $Choice_k(y))) \wedge$
  $\forall x(P(x) \Rightarrow \exists^{\leq max} y(P(x)$ *contains* $Choice_k(y)))$
- $\forall x(Choice_k(x) \Rightarrow \exists^{\geq min'} y(P(y)$ *contains* $Choice_k(x))) \wedge$
  $\forall x(Choice_k(x) \Rightarrow \exists^{\leq max'} y(P(y)$ *contains* $Choice_k(x)))$
- $\forall x(Choice_k(x) \Rightarrow$
  $(\exists^{\geq min_{Cho_1}} y_1\ (C_1(y_1)$ *is alternative for* $Choice_k(x)) \wedge$
  $\exists^{\leq max_{Cho_1}} z_1\ (C_1(z_1)$ *is alternative for* $Choice_k(x))) \vee$
  $\neg \exists w_1\ (C_1(w_1)$ *is alternative for* $Choice_k(x)))$
  $\wedge$ ... $\wedge$
  $\forall x(Choice_k(x) \Rightarrow$
  $(\exists^{\geq min_{Cho_n}} y_n(C_n(y_n)$ *is alternative for* $Choice_k(x)) \wedge$
  $\exists^{\leq max_{Cho_n}} z_n(C_n(z_n)$ *is alternative for* $Choice_k(x))) \vee$
  $\neg \exists w_n\ (C_n(w_n)$ *is alternative for* $Choice_k(x)))$
- $\forall x(C_1(x) \Rightarrow \exists^{\geq min_{C_1}} y(C_1(x)$ *is alternative for* $Choice_k(y))) \wedge$
  $\forall x(C_1(x) \Rightarrow \exists^{\leq max_{C_1}} y(C_1(x)$ *is alternative for* $Choice_k(y)))$
  $\wedge$ ... $\wedge$
  $\forall x(C_n(x) \Rightarrow \exists^{\geq min_{C_n}} y(C_n(x)$ *is alternative for* $Choice_k(y))) \wedge$
  $\forall x(C_n(x) \Rightarrow \exists^{\leq max_{C_n}} y(C_n(x)$ *is alternative for* $Choice_k(y)))$

Mutual-Exclusion Constraints:

- $\forall x(Choice_k(x) \Rightarrow$
  $(\exists y_1(C_1(y_1)$ *is alternative for* $Choice_k(x)) \wedge$
  $\neg \exists y_2(C_2(y_2)$ *is alternative for* $Choice_k(x))\ \wedge \ldots \wedge$

$\neg\exists y_n(C_n(y_n)$ *is alternative for* $Choice_k(x)))$
$\vee \ldots \vee$
$(\neg\exists y_1(C_1(y_1)$ *is alternative for* $Choice_k(x))\ \wedge \ldots \wedge$
$\neg\exists y_{i-1}(C_{i-1}(y_{i-1})$ *is alternative for* $Choice_k(x)) \wedge$
$\exists y_i(C_i(y_i)$ *is alternative for* $Choice_k(x)) \wedge$
$\neg\exists y_{i+1}(C_{i+1}(y_{i+1})$ *is alternative for* $Choice_k(x))\ \wedge \ldots \wedge$
$\neg\exists y_n(C_n(y_n)$ *is alternative for* $Choice_k(x)))$
$\vee \ldots \vee$
$(\neg\exists y_1(C_1(y_1)$ *is alternative for* $Choice_k(x))\ \wedge \ldots \wedge$
$\neg\exists y_{n-1}(C_{n-1}(y_{n-1})$ *is alternative for* $Choice_k(x)) \wedge$
$\exists y_n(C_n(y_n)$ *is alternative for* $Choice_k(x))))$

Mixed Content

Formally, marking an object set P as *mixed* is a template for creating a relationship set to a lexical object set $Text$ of type string: $P[1]$ *contains* $[1{:}*]Text$. The string associated with an object in a mixed object set may be interspersed among direct child elements.

Generalized Co-Occurrence:

A generalized co-occurrence constraint $A_1, ..., A_n \rightarrow B_1, ..., B_m$ is shorthand for an ordinary co-occurrence constraint written over a high-level relationship set connecting the object sets $A_1, ..., A_n$, $B_1, ..., B_m$. If the subgraph that connects these object sets is unique, we can derive the corresponding high-level relationship set automatically. Otherwise, in addition to specifying the co-occurrence constraint, the user must also specify the derived relationship set using Prolog-like syntax (e.g., $r(A, B)$:- $r_1(A, X), r_2(X, B)$). The formal definition of co-occurrence constraints appears in Appendix A of [5].