Representing Generalization/Specialization in XML Schema

Reema Al-Kamha[1] David W. Embley[1] Stephen W. Liddle[2]

[1]Computer Science Department
[2]Information Systems Department
Brigham Young University, Provo, Utah 84602, U.S.A.
reema@cs.byu.edu, embley@cs.byu.edu, liddle@byu.edu

Abstract: XML is an effective universal data-interchange format, and XML Schema has become the preeminent mechanism for describing valid XML document structures. Generalization/specialization and its constraints are fundamental concepts in system modeling and design, but are difficult to express and enforce with XML Schema. This mismatch leads to unnecessary complexity and uncertainty in XML-based models. In this paper we describe how to translate various aspects of generalization/specialization from a conceptual model into XML Schema. We also explore what needs to be added to XML Schema to handle the other aspects of this fundamental modeling construct. If XML Schema were to include our proposed constructs, it would be fully capable of faithfully representing generalization/specialization, thus reducing the complexity of the XML models that rely on generalization/specialization.

1 Introduction

The scientific community has long recognized the importance of *generalization*—and its inverse, *specialization*—as a fundamental and highly useful modeling construct (see, for example, [SS77]). Generalization/specialization is used broadly in conceptual models such as UML [BRJ99], and EER [TYF86], and in description logics [BN03]. The main idea in generalization/specialization, also called the *is-a* relationship, is that one set, class, or concept is a subset of another. If *A* is a generalization of *B* (or equivalently, if *B* is a specialization of *A*), we say that *B* is a subset of *A* (*B is-a A*). In general, concepts form a hierarchy wherein a generalization may have many specializations, and a specialization may have many generalizations. It is often useful, however, to define constraints over generalization/specialization hierarchies. For example, we can declare two specializations of a common generalization to be *mutually exclusive*. We can also declare the specializations of a concept to be complete in the sense that their *union* contains all members of the generalization. If both of these constraints are present (a common occurrence), the specializations *partition* the generalization space. A less common constraint is the situation where a specialization constitutes the *intersection* of its multiple generalizations.

In this paper we illustrate our examples using *Conceptual XML* (C-XML) [ELAK04]

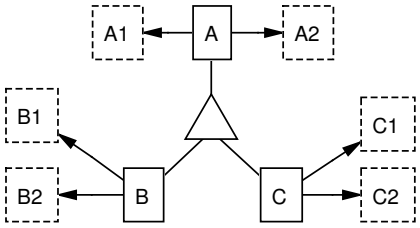

Figure 1: Generalization/Specialization in C-XML.

which is a conceptual model consisting of object sets, relationship sets, and constraints over these object and relationship sets.[1] In C-XML, we represent object sets or concepts by writing names inside rectangles, with a solid border indicating a nonlexical concept and a dashed border indicating a lexical concept. In Figure 1, each nonlexical concept has two related lexical concepts whose relationship sets are functional, indicated by arrows (e.g., for each *A*, there is exactly one *A1*, but multiple *A*'s may have the same *A1* value). In C-XML, a triangle denotes generalization/specialization. For example, in Figure 1 the set of objects in *B* is a subset of the set of objects in *A*. C-XML allows modelers to constrain generalizations by writing a constraint symbol in a triangle. A plus symbol ($+$) in indicates that the specialization sets are mutually exclusive. A union symbol ($\cup$), specifies that the set of objects in the generalization is the union of the specialization object sets. A plus and union together ($\uplus$) specify that the specializations partition the generalization since there is both a union and a mutual-exclusion constraint. An intersection symbol ($\cap$) indicates that the members of the specialization object set constitute the intersection of the generalization object sets.

These simple definitions find many different, intricate, and complicating expressions in conceptual models and schema description languages. For example, a typical object-oriented "class" is a *type* rather than a mathematical set, and it uses the *inheritance* relationship and the notion of substitutability in place of the more general concept of generalization/specialization and simple *is-a* semantics. This leads to a potential mismatch between how we model the real world and how we implement information systems.[2]

XML Schema has rapidly become the method of choice for describing XML document structures. Since XML is the de facto standard for modern data interchange, it is important that we understand how to properly capture and enforce constraints on XML document structures. Thus, a number of researchers have studied how to transform conceptual models into XML Schema. A chapter in [Car01] describes how to translate a UML model instance into XML Schema. A chapter in [Dau03] presents Relax NG and introduces how to translate from the Asset Oriented Modeling conceptual model into XML Schema. Yet another study shows how to translate Object Role Modeling into XML Schema [BGH00].

[1] The particular choice of conceptual model is not critical to this paper, since the various conceptual models and description logics typically have very similar underlying generalization/specialization constructs.

[2] Thus some developers adopt the rule of thumb that class derivation (inheritance) should only be used when *is-a* also holds for the derivation relationship. But this rule is not applied universally.

In each case, the discussion is about translation in general, and does not focus specifically on the problem of fully capturing all the semantics of generalization/specialization. In this paper we deal with the full details of translating generalization/specialization and its constraints into XML Schema.

The remainder of the paper proceeds as follows. In Section 2 we describe the mechanisms available in XML Schema to represent generalization/specialization. In Section 3 we show how to use those mechanisms to capture the semantics of certain forms of generalization/-specialization and its constraints. Since XML Schema is incapable of fully representing all of the necessary semantics, in Section 4 we describe a relatively small but important set of augmentations that would allow XML Schema to do a complete job. We conclude in Section 5.

2 Generalization/Specialization Mechanisms in XML Schema

There are several mechanisms in XML Schema that support generalization/specialization. The foundational information construct in XML, of course, is the *element*, which together with the *attribute* construct and element nesting is sufficient to represent all data structures. So the starting point for any translation from a conceptual model to XML is to map "concepts" to "elements." Relationships typically map either to attributes or to nested elements. There are significant complications when we consider finer points like object identity, but the overall process of structure mapping is fairly clear-cut and generally intuitive.

However, once we have a basic structure encoded in XML, how can we capture generalization/specialization relationships and their constraints? We find three constructs in XML Schema that support various aspects of generalization/specialization: (1) derived types, (2) substitution groups, and (3) abstract elements. We consider each construct in turn.

2.1 Derived Types

In XML Schema, each element has a *type* that describes valid element content. Types come in two broad categories: *simple* and *complex*. One simple type can be derived from another by restriction. For example, *string* is a simple type, and we can specify a customized type, *GenericTLD*, as the set of strings that correspond to the generic top-level internet domains by restricting the *string* type as follows:

```
<xs:simpleType name="GenericTLD">
    <xs:restriction base="xs:string">
        <xs:enumeration value="com" />
        <xs:enumeration value="edu" />
        <xs:enumeration value="gov" />
        <xs:enumeration value="net" />
        <xs:enumeration value="org" />
```

```
        . . .
    </xs:restriction>
</xs:simpleType>
```

Similarly, complex types may be derived by restriction from a base type. Valid restrictions include those that increase the constraints on attributes or elements in the complex type in a way that is compatible with the base type. For example, an optional element in the base type may be required in the derived type. Thus, the derivation of both simple and complex types by restriction results in a set of allowed values for the derived type that is a subset of the allowed values for the base type. This notion is similar to the conceptual *is-a* relationship.

Extension of complex types involves creating a derived complex type whose content model is a superset of its base type's content model. When we extend a complex type, we can add to the derived type extra attributes or elements in addition to those found in the content model of the base type as follows:

```
<xs:complexType name="A">
    <xs:sequence>
        <xs:element name="A1" type="xs:string" />
        <xs:element name="A2" type="xs:string" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="B">
    <xs:complexContent>
        <xs:extension base="A">
            <xs:sequence>
                <xs:element name="B1" type="xs:string" />
                <xs:element name="B2" type="xs:string" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

In this case, type *B* is derived by extension from type *A*. In addition to including *A1* and *A2* elements, *B*-type elements also include *B1* and *B2* elements.

The concept of complex type extension is a "reuse" mechanism similar in some aspects to typical object-oriented inheritance mechanisms. It is distinct from the conceptual *is-a* relationship, though extension does indeed form a type hierarchy that supports substitution of a more specialized type for a more general type. Some may argue that derivation necessarily implies a form of generalization/specialization, but we see it as a convenience mechanism for writing efficient code, not as a conceptual mechanism for defining generalization/specialization hierarchies.

2.2 Substitution Groups

In XML Schema, global elements can be organized into a *substitution group*, wherein a particular set of elements can be substituted for a named element, called the *head element*. For example, if elements *B* and *C* were each declared to be substitutable for *A* by including the attribute *substitutionGroup="A"* in the declarations of elements *B* and *C*, then the meaning is that *B* or *C* may appear anywhere that *A* is required. The presence of a substitution group does not require use of the substitutable elements, nor does it preclude the use of the head element. It simply establishes a way for a set of elements to be used interchangeably.

The concept of a substitution group constitutes a form of generalization/specialization, though it is not identical to the natural subset notion of generalization/specialization that corresponds to the *is-a* relationship described in the introduction. Instead, a substitution group defines an equivalence class of elements that can be used interchangeably. However, substitution groups can form hierarchies similar to *is-a* hierarchies, and we can construe them to denote a relationship much like *is-a*. Indeed, we argue that the use of a substitution group implies conceptual generalization/specialization in the sense that one concept (a substitutable element) is a special kind of another concept (the head element).

2.3 Abstract Elements and Types

It is possible to require the use of substitution for a particular element or type by declaring it to be *abstract*. An element declared to be abstract cannot be used in an instance document—a non-abstract substitutable element must be used instead. Thus, declaring an element as abstract requires the specification of a substitution group. Similarly, declaring a type to be abstract requires the use of concrete types that extend the abstract type. In both cases, abstract elements are associated with concept hierarchies that are related to the conceptual *is-a* relationship.

3 Representing Generalization/Specialization in XML Schema

Given the foundational XML Schema mechanisms described in Section 2, we now turn our attention to how we can actually represent conceptual generalization/specialization in XML Schema. There are two cases of conceptual generalization/specialization that we are able to represent faithfully in XML Schema, two cases that are problematic, and two other cases that are not possible (directly). When a generalization/specialization hierarchy does not include multiple generalizations for any specialization (i.e., no concept has more than one parent concept), we are able to represent generalization/specialization relationships with partition and mutual-exclusion constraints in a straightforward manner as we show in Sections 3.1.1 and 3.1.2. We are also able to represent union constraints and unconstrained generalization/specialization relationships, but as we describe in Section 3.2, these cases

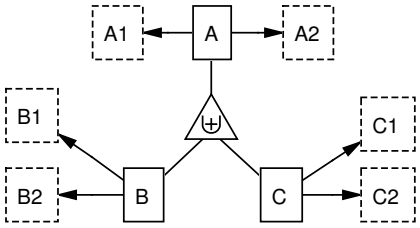

Figure 2: Generalization/Specialization Partition Constraint in C-XML.

are more problematic. In Section 3.3 we discuss the cases that we cannot model reasonably in XML Schema, namely generalization/specialization relationships involving multiple generalizations.

3.1 Straightforward Cases

The two straightforward cases of generalization/specialization constraints are partition and mutual-exclusion. In both cases, we start by translating concepts into elements and attributes, and relationship sets into attributes and nested elements. The primary means for representing generalization/specialization in XML Schema is captured by the notion of substitution groups, so we have chosen to represent each generalization/specialization relationship with an XML Schema substitution group. We now describe how to represent partition and mutual-exclusion constraints in XML Schema.

3.1.1 Partition Constraints

Figure 2 shows a C-XML model instance where specialized concepts *B* and *C* form a partition of the general *A* concept. In set terminology, we say that $B \cup C = A$ and $B \cap C = \{\}$. Figure 3 shows our XML Schema translation of this model instance.

The translation from C-XML in Figure 2 to XML Schema in Figure 3 proceeds as follows. We begin by introducing *Document* as a root-level node that contains a sequence of *A* elements. We declare *A* as an abstract type whose content is defined by the complex type *Atype* (line 11). Since *A* is abstract, it cannot appear independently in an instance document—either *B* or *C* must be substituted. This serves the purpose of covering the union constraint (recall that partition is the combination of union and mutual exclusion), since *A* must necessarily be defined as the union of the set of *B*'s and *C*'s that actually appear in the instance document.

Atype declares that the content model of *A* includes exactly one *A1* element and exactly one *A2* element (lines 12-16). Furthermore, we define an object identifier attribute *OID* of type *ID* (line 17) that serves as a unique identifier for each *A*. XML Schema defines the special *ID* type to be unique across an entire document instance. Since *B* and *C* must be

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 3:         elementFormDefault="qualified" attributeFormDefault="unqualified">
 4: <xs:element name="Document">
 5:   <xs:complexType>
 6:     <xs:sequence minOccurs="0" maxOccurs="unbounded">
 7:       <xs:element ref="A" />
 8:     </xs:sequence>
 9:   </xs:complexType>
10: </xs:element>
11: <xs:element name="A" type="Atype" abstract="true" />
12: <xs:complexType name="Atype">
13:   <xs:sequence>
14:     <xs:element name="A1" minOccurs="1" maxOccurs="1" />
15:     <xs:element name="A2" minOccurs="1" maxOccurs="1" />
16:   </xs:sequence>
17:   <xs:attribute name="OID" type="xs:ID" use="required" />
18: </xs:complexType>
19: <xs:element name="B" type="Btype" substitutionGroup="A" />
20: <xs:complexType name="Btype">
21:   <xs:complexContent>
22:     <xs:extension base="Atype">
23:       <xs:sequence>
24:         <xs:element name="B1" type="xs:string" minOccurs="1" maxOccurs="1" />
25:         <xs:element name="B2" type="xs:string"  minOccurs="1" maxOccurs="1" />
26:       </xs:sequence>
27:     </xs:extension>
28:   </xs:complexContent>
29: </xs:complexType>
30: <xs:element name="C" type="Ctype" substitutionGroup="A" />
31: <xs:complexType name="Ctype">
32:   <xs:complexContent>
33:     <xs:extension base="Atype">
34:       <xs:sequence>
35:         <xs:element name="C1" type="xs:string" minOccurs="1" maxOccurs="1" />
36:         <xs:element name="C2" type="xs:string" minOccurs="1" maxOccurs="1" />
37:       </xs:sequence>
38:     </xs:extension>
39:   </xs:complexContent>
40: </xs:complexType>
41: </xs:schema>
```

Figure 3: XML Schema Translation of C-XML in Figure 2.

mutually exclusive, we can ensure that the sets are disjoint simply by providing a unique surrogate identifier for each element. A key point here is how we deal with the issue of object identity. How do we know whether a *B* element and a *C* element represent the same *A* object? Since *A*, *B*, and *C* are all nonlexical, we need to associate object identifiers with them. Attribute *OID* serves this purpose, and because *OID* must be unique across all elements, we are guaranteed that no *B* element will have the same *OID* value as some *C* element. Hence we know that *B* and *C* are mutually exclusive (even if a *B* object and a *C* object share the same *A1* and *A2* values).

The remainder of Figure 3 accounts for the specialized structure of *B* and *C*, each with its own pair of related concepts. Elements *B* and *C* are members of the substitution group whose head element is *A* (lines 19 and 30). Both elements *B* and *C* derive their content models by extension from the base *Atype* (lines 20-29 and 31-40 respectively).

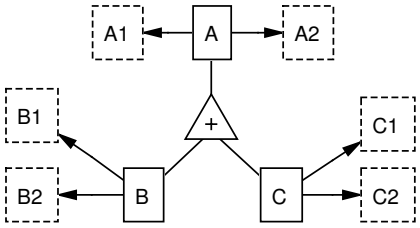

Figure 4: Generalization/Specialization Mutual-Exclusion Constraint in C-XML.

3.1.2 Mutual-Exclusion Constraints

Figure 4 shows a C-XML model instance that is similar to the model instance in Figure 2, except that the partition constraint is replaced with the weaker mutual-exclusion constraint. The translation of the model instance to XML Schema is identical to the partition-constraint case with one exception. Since the C-XML model instance in Figure 4 does not force A to be the union of B and C, there may be A's present that are in neither B nor C. That is, we still have $B \cap C = \{\}$, but we no longer have $B \cup C = A$. Instead, we merely know that $B \cup C \subseteq A$. Thus we must allow instances of the A element to be directly present in the XML document instance. We accomplish this by repeating the same translation as before except we declare element A not to be abstract. The only thing that changes from Figure 3 is that on line 11 we write *abstract="false"* instead of *abstract="true"*.

Partition and mutual-exclusion constraints on generalization/specialization relationships are fairly straightforward to represent in XML Schema without introducing many artifacts. The two additional information-carrying elements in the XML Schema translation are the *Document* element, since XML requires a single root-level container element, and the *OID* attribute, which is necessary to capture object identity semantics. We now proceed to the more difficult union constraint and unconstrained generalization/specialization.

3.2 Problematic Cases in XML Schema

In contrast with the straightforward translation of partition and mutual-exclusion constraints from C-XML to XML Schema, unconstrained generalization/specialization and generalization/specialization with only a union constraint are more difficult to handle, and the mapping approach is not entirely satisfactory.

3.2.1 Generalization/Specialization without any Constraint

Figure 1 shows an unconstrained generalization/specialization relationship, where A is the general concept and B and C are specializations of A. In set notation we write $B \subseteq A$

```
 1: <?xml version="1.0" encoding="UTF-8"?>
 2: <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 3:         elementFormDefault="qualified" attributeFormDefault="unqualified">
 4: <xs:element name="Document">
 5:   <xs:complexType>
 6:     <xs:sequence minOccurs="0" maxOccurs="unbounded">
 7:       <xs:element ref="A" />
 8:     </xs:sequence>
 9:   </xs:complexType>
10: </xs:element>
11: <xs:element name="A" type="Atype" abstract="false" />
12: <xs:complexType name="Atype">
13:   <xs:sequence>
14:     <xs:element name="A1" minOccurs="1" maxOccurs="1" />
15:     <xs:element name="A2" minOccurs="1" maxOccurs="1" />
16:   </xs:sequence>
17:   <xs:attribute name="OID" type="xs:string" use="required" />
18: </xs:complexType>
19: <xs:element name="B" type="Btype" substitutionGroup="A" />
20: <xs:complexType name="Btype">
21:   <xs:complexContent>
22:     <xs:extension base="Atype">
23:       <xs:sequence>
24:         <xs:element name="B1" type="xs:string" minOccurs="1" maxOccurs="1" />
25:         <xs:element name="B2" type="xs:string"  minOccurs="1" maxOccurs="1" />
26:       </xs:sequence>
27:     </xs:extension>
28:   </xs:complexContent>
29: </xs:complexType>
30: <xs:element name="C" type="Ctype" substitutionGroup="A" />
31: <xs:complexType name="Ctype">
32:   <xs:complexContent>
33:     <xs:extension base="Atype">
34:       <xs:sequence>
35:         <xs:element name="C1" type="xs:string" minOccurs="1" maxOccurs="1" />
36:         <xs:element name="C2" type="xs:string" minOccurs="1" maxOccurs="1" />
37:       </xs:sequence>
38:     </xs:extension>
39:   </xs:complexContent>
40: </xs:complexType>
41: </xs:schema>
```

Figure 5: XML Schema Translation of C-XML in Figure 1.

and $C \subseteq A$. In particular, this allows for the possibility that the intersection of B and C could be non-empty. And that is where the chief difficulty arises—how do we enforce object identity when $B \cap C \neq \{\}$? Figure 5 shows the best we can do with the available mechanisms in XML Schema to represent this case.

The differences between Figure 5 and Figure 3 are (1) element A is not abstract, thus relaxing the union constraint, and (2) the object identifier attribute *OID* is not of type *ID*, and so we do not enforce uniqueness, thus relaxing the mutual-exclusion constraint.

A, B, and C are still nonlexical concepts, and so they should have an identity in the corresponding XML Schema translation. We can argue that two elements in an XML document instance with the exact same values still have distinct identities because they are written separately in the XML document. Thus we can distinguish between the element instance written first in the document and the element instance written second. However, consider the case where an object is a member of both B and C. Since we have no combined type to represent a *B/C* element, we must write the element first as a B, and then using the same

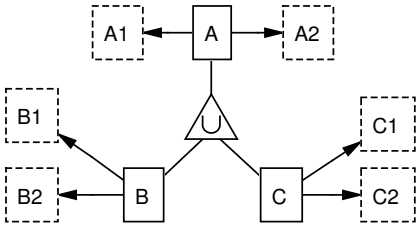

Figure 6: Generalization/Specialization Union Constraint in C-XML.

values for *OID*, *A1*, and *A2*, we must write the element as a *C*. Now the conceptual object that is a member of *B* and *C* is represented as two separate XML elements tied together by a common *OID* value. Besides introducing an update anomaly over *A1* and *A2*, we are in the unsatisfying position of not being able to enforce uniqueness of *OID*.

The alternative is even less satisfying. We could include a combined *B/C* type, but there would still be two major problems. First, there would be an exponential explosion of potential combinations (imagine having just 10 or 20 specializations of one general concept—the XML Schema would be unwieldy to say the least). Second, we would break the nice correspondence between substitution groups and generalization/specialization. So, for example, a combined *B/C* element would let us create *B* elements that do not directly relate to the *B* element which represents the *B* concept in our conceptual model. Iterating over the set of *B* elements would become needlessly difficult.

The advantages of our chosen approach are that it aligns more closely with the conceptual model structure, and we can enforce the appropriate constraints by post-processing outside of the ordinary XML Schema constraint enforcement mechanisms. Nonetheless, as we explore in Section 4, a fully satisfactory approach requires extensions to XML Schema.

3.2.2 Union Constraint

Figure 6 shows a C-XML model instance similar to the previous case except with a union constraint on the generalization/specialization relationship. For this case, our translation approach is similar to the unconstrained case in Figure 5 except we declare the element *A* to be abstract so that it cannot be instantiated directly in a document instance. With a union constraint, we know that $B \cup C = A$, and so we must prevent the situation where an *A* exists that is neither in *B* nor *C*. Specifying *abstract*=*"true"* for *A* accomplishes this.

Unfortunately, our solution in this case suffers from the same problems we describe in Section 3.2.1, and so we are not fully satisfied with the outcome. Nonetheless, it is possible to faithfully represent the conceptual structures of our C-XML model instances in all these cases, even though we sometimes cannot enforce the constraints fully in XML Schema. The unifying mechanism of our C-XML-to-XML Schema translation of generalization/-specialization is that we use substitution groups to represent the generalization/specialization hierarchy.

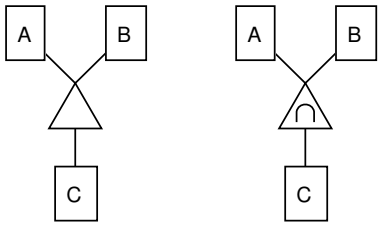

Figure 7: Multiple Generalizations in C-XML.

3.3 The Problem of Multiple Generalizations

The correspondence between generalization/specialization and substitution groups breaks down when we consider multiple generalizations. Figure 7 shows a simple C-XML model instance where concept C is a specialization of both A and B, so that $C \subseteq A$ and $C \subseteq B$. When we specify an intersection constraint on the generalization/specialization, we further require that $A \cap B = C$. As with the unconstrained generalization/specialization case, we must be able to handle the situation where an object is a member of more than one concept. Thus we will rely on the same *OID* mechanism as before.

However, for these cases, we simply have no way of specifying in XML Schema that an element is a member of two substitution groups. The philosophy of XML Schema 1.0 was to implement single inheritance only. Five years ago, one of the editors of the XML Schema standard acknowledged that this is a problem and that the working group might consider adding support for multiple inheritance in the future [DL00]. However, since inheritance combines the *is-a* construct with a code-reuse mechanism, it is not clear that simply adding multiple inheritance will resolve the problem of supporting conceptual generalization/specialization appropriately. We agree that extension from multiple types would be useful, but conceptually what we need even more is the ability for an element to participate in multiple substitution groups. (See [Das01] for ideas on this topic generated in a different but related context.)

4 Resolving the Conceptual Modeling Issues

There are two general approaches we can take to resolve the issues we have raised with respect to capturing conceptual generalization/specialization constructs in XML Schema. First, we could implement constraint-checking external to XML Schema to enforce the meaning of the conceptual model within corresponding XML documents. Alternatively, we could augment XML Schema with a few modest extensions that will support conceptual generalization/specialization directly.

4.1 Post-Processing to Enforce Constraints

Section 3.2 describes a somewhat unsatisfactory approach to mapping unconstrained and union-constrained generalization/specialization to XML Schema. What is missing in these cases is appropriate enforcement of object identity uniqueness. Consider the C-XML model instance in Figure 1 and its translation to XML Schema in Figure 5. If we were to add pragmas to the XML Schema instance to indicate that $B \subseteq A$ and $C \subseteq A$, a post-processor could examine the corresponding document instance and determine whether the object identities are all appropriate. The post-processor would need to verify the following: (1) *OID* is vertically unique across the generalization/specialization hierarchy (so, for example, there is no *B* element whose *OID* value is identical to some *A* element), and (2) when two *OID* values are the same in two sibling classes, they also share the same *A1* and *A2* values.

To implement multiple generalizations, we would need to take a somewhat different approach to laying out the C-XML concepts as XML elements. Instead of relying on substitution groups to map one-to-one with generalization/specialization, we would need to write pragmas to indicate the structure of the conceptual generalization/specialization relationships. So we might write in a specially-formatted comment, for example, that *C* is a specialization of both *A* and *B*, and if an intersection constraint were present we would also note that. A post-processor could readily parse the pragmas and check whether the specified constraints hold. However, since XML Schema would have no way of tying *C* directly to *A* and *B*, we would need to rewrite the schema so that any reference to *A* or *B* could be replaced by a *C* element instead. In general, we could use this strategy to handle all generalization/specialization relationships and constraints.

Certainly the post-processor methodology has significant drawbacks; we now explore a cleaner approach.

4.2 Proposed Extensions to XML Schema

Perhaps the best way to implement conceptual generalization/specialization in XML Schema is to augment XML Schema with a few extensions. For multiple generalizations, it would be straightforward to extend the *substitutionGroup* attribute on a substitutable element so that it admits a list of multiple head elements. For example, to capture the generalization/specialization hierarchy of Figure 7, we could write the following:

<xs:element name="C" substitutionGroup="A,B" />

And if there were an intersection constraint present, we could note it with a distinguished keyword, for example:

<xs:element name="C" intersectionGroup="A,B" />

To capture the concept of union-constrained generalization/specialization, we need to mark head elements with the appropriate constraints. For example, we could mark the union constraint of Figure 6 in this manner:

<xs:element name="A" union="B,C" />

Similarly, mutual-exclusion and partition constraints could replace the word *union* with *mutex* and *partition*, respectively.

Finally, to cover the aforementioned cases and to handle unconstrained generalization/-specialization, we would need to attach unique object identifiers uniformly to all elements in all substitution groups. We could do this by modifying XML Schema to automatically assert the existence of an *OID* attribute for all elements in a substitution group, including the head element(s) and all substitutable elements.

5 Conclusion

Generalization/specialization is an important structure in conceptual modeling, but it is often difficult to implement faithfully in XML Schema. This leads to XML Schema instances that are unnecessarily complex or that misrepresent the original semantics of a conceptual model. To compound the problem, it is often the case that inheritance combines the properties of the conceptual *is-a* relationship with the notion of code reuse. This can sometimes cause awkward structures that are implemented efficiently but do not correspond to a natural conceptual model.

The contributions of this paper include the following:

- We have characterized the nature of conceptual generalization/specialization and have shown how it corresponds to structures in XML Schema.
- We have identified a small set of constructs that could augment XML Schema so that it would fully support conceptual generalization/specialization.

If our proposal were adopted, it would result in better alignment between conceptual models and corresponding XML Schema instances, and the resulting schemas would have the added benefit of being substantially less complex than the alternatives. Given the inherent complexity of enterprise application modeling and development, these advantages could be significant.

Acknowledgements

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 at Brigham Young University under grant EB-05046.

References

[BGH00] L. Bird, A. Goodchild, and T. Halpin. Object Role Modelling and XML-Schema. In *Proceedings of the Ninteenth International Conference on Conceptual Modeling (ER2000)*, pages 309–322, Salt Lake City, Utah, October 2000.

[BN03] F. Baader and W. Nutt. Basic Description Logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 2, pages 43–95. Cambridge University Press, Cambridge, UK, 2003.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, 1999.

[Car01] David Carlson. *Modeling XML Applications with UML: Practical e-Business Applications*. Addison Wesley, Menlo Park, California, 2001.

[Das01] E. M. Dashofy. Issues in Generating Data Bindings for an XML Schema-Based Language. In *Proceedings of the of the Workshop on XML Technologies and Software Engineering (XSE2001)*, Toronto, ONT, Canada, May 2001.

[Dau03] Berthold Daum. *Modeling Business Objects with XML Schema*. Morgan Kaufmann, San Francisco, California, 2003.

[DL00] Dodds, L. Reconstructing DTD Best Practice, June 2000. http://www.xml.com/pub/a/-2000/06/xmleurope/schemas.html.

[ELAK04] David W. Embley, Stephen W. Liddle, and Reema Al-Kamha. Enterprise Modeling with Conceptual XML. In *Proceedings of the 23rd International Conference on Conceptual Modeling(ER2004)*, pages 150–165, Shanghai, China, November 2004.

[SS77] John Miles Smith and Diane C. P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Trans. Database Syst.*, 2(2):105–133, 1977.

[TYF86] 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.