Extending XML Schema with Nonmonotonic Inheritance*

Guoren Wang and Mengchi Liu

School of Computer Science, Carleton University, Canada
{wanggr,mengchi}@scs.carleton.ca

Abstract. Nonmonotonic inheritance is a fundamental feature of object-oriented data models. In this paper, we extend XML Schema with nonmonotonic inheritance due to its powerful modeling ability to support multiple inheritance, overriding of elements or attributes inherited from super-elements, blocking of the inheritance of elements or attributes from super-elements, and conflict handling. Another key feature of object-oriented data models is polymorphism. We introduce it into XML to support polymorphic elements and polymorphic references.

1 Introduction

Several XML schema languages have been proposed, such as DTD [2], *SOX* [3], *XML Schema* [5] to constrain and define a class of XML documents. However, they do not support inheritance at all except for *XML Schema* and *SOX* [6].

Nonmonotonic multiple inheritance is a fundamental feature of object-oriented data models [4,7]. In object-oriented languages with multiple inheritance, a class may inherit attributes and methods from more than one superclass. For example, class *TA* might inherit attributes and methods directly from classes *teacher* and *student*. In a multiple inheritance hierarchy, users can explicitly override the inherited attributes or methods and block the inheritance of attributes or methods from superclasses [7]. One of the problems with multiple inheritance is that ambiguity may arise when the same attribute or method is defined in more than one superclass. Therefore, conflict resolution is important in object-oriented database systems with multiple inheritance and most systems use the superclass ordering to solve the conflicts [4,7].

In this paper, we extend XML Schema with nonmonotonic inheritance due to its powerful modeling ability to support multiple inheritance, overriding of elements or attributes inherited from super-elements, blocking of the inheritance of elements or attributes from super-elements, and conflict handling. Another key feature of object-oriented data models is polymorphism. We introduce it into XML to support polymorphic elements and polymorphic references.

* Guoren Wang's research is partially supported by NSFC of China(60273079) and Mengchi Liu's research is partially supported by NSERC of Canada.

M.A. Jeusfeld and Ó. Pastor (Eds.): ER 2003 Workshops, LNCS 2814, pp. 402–407, 2003.

```
(01) <xsd:element name="univ" type="univType"/>
(02) <xsd:complexType name="univType">
(03)   <xsd:sequence>
(04)     <xsd:element name="person" type="personType"
(05)                  minOccurs="0" maxOccurs="'unbounded'/ >
(06)     <xsd:element name="course" type="courseType"
(07)                  minOccurs="0" maxOccurs="'unbounded'/ >
(08)   </xsd:sequence>
(09) </xsd:complexType>
(10) <xsd:complexType name="personType">
(11)   <xsd:sequence>
(12)     <xsd:element name="name" type="xsd:string"/>
(13)     <xsd:element name="birthdate" type="xsd:date"/>
(14)     <xsd:element name="addr" type="addrType"/>
(15)     <xsd:element name="homephone" type="xsd:string"/>
(16)   </xsd:sequence>
(17)   <xsd:attribute name="pid" type="xsd:ID" use="required"/>
(18) </xsd:complexType>
(19) <xsd:complexType name="addrType">
(20)   <xsd:sequence>
(21)     <xsd:element name="street" type="xsd:string"/>
(22)     <xsd:element name="city" type="xsd:string"/>
(23)     <xsd:element name="state" type="xsd:string"/>
(24)     <xsd:element name="zip" type="xsd:string"/>
(25)   </xsd:sequence>
(26) </xsd:complexType>
```

Fig. 1. Type definitions for elements univ, person and addr

2 Extensions to XML Schema

Figure 1 shows the type definitions for elements *univ*, *person* and *addr*. Although they have the same syntax as the original XML Schema, some of them (for example lines (04)-(07) of Figure 1) have different semantic constraints on XML instance documents due to the introduction of the polymorphic feature.

Figure 2 shows the type definition for element *student* that inherits *personType*. Because the inheritance mechanism provided by XML Schema is not flexible and powerful, we extend it as follows:

(1) In a type hierarchy, a subtype may have more than one supertype to support nonmonotic multiple inheritance. Therefore, the attribute *base* of the extension machanism is modified as *bases*, e.g. in line (03) of Figure 2.

(2) In the original XML Schema, a subtype inherits all elements but not attributes from its supertype. Although attributes are different from elements, they are a special kind of information from users' point of view. Therefore, in the Extended XML Schema, a subtype inherits not only elements but also attributes from its supertypes. Note that no other ID attributes are allowed to be declared in the subtype since *pid* is an ID attribute inherited by the subtype.

(3) In the Extended XML Schema, a specific component element or attribute in the subtype may override the element or attribute defined in the supertype. For example, the component element *addr* in *student* inherited from *personType* is overridden with a new simple type, as shows in line (05) of Figure 2. Note that there is no special syntax extension for overriding of element and attribute.

Sometimes, it is necessary to allow a subtype to block the inheritance of attributes and elements from its supertypes. For example, *teachers* usually prefer

```
(01) <xsd:complexType name="studentType">
(02)   <xsd:complexContent>
(03)     <xsd:extension bases="personType">
(04)       <xsd:sequence>
(05)         <xsd:element name="addr" type="xsd:string"/>
(06)         <xsd:element name="dept" type="xsd:string"/>
(07)         <xsd:element name="takes">
(08)           <xsd:attribute name="courses" type="IDREFS"
(09)                          target="courseType" use="implied"/>
(10)         </xsd:element>
(11)       </xsd:sequence>
(12)       <xsd:attribute name="sno" type="xsd:string" use="required"/>
(13)     </xsd:extension>
(14)   </xsd:complexContent>
(15) </xsd:complexType>
```

Fig. 2. Type definition for element Student

```
(01) <xsd:complexType name="teacherType">
(02)   <xsd:complexContent>
(03)     <xsd:extension bases="personType">
(04)       <xsd:sequence>
(05)         <xsd:element name="workphone" type="xsd:integer"/>
(06)         <xsd:element name="salary" type="xsd:float"/>
(07)         <xsd:element name="dept" type="xsd:string"/>
(08)         <xsd:element name="teaches">
(09)           <xsd:attribute name="courses" type="IDREFS"
(10)                          target="courseType" use="implied"/>
(11)         </xsd:element>
(12)       </xsd:sequence>
(13)       <xsd:block from="personType">
(14)         <xsd:element name="homephone"/>
(15)       </xsd:block>
(16)       <xsd:attribute name="tno" type="xsd:string" use="required"/>
(17)     </xsd:extension>
(18)   </xsd:complexContent>
(19) </xsd:complexType>
```

Fig. 3. Type definition for element Teacher

to use *workphone* rather than *homephone* as their contact phone. It is reasonable that in the definition of the subtype *teacherType* the inheritance of *homepone* is blocked from its supertype *personType*. Therefore, the *blocking* mechanism is introduced as shown in lines (13)-(15) of Figure 3. The *blocking* mechanism has an attribute *from* specifying from which type the inheritance is blocked and some components specifying attributes and elements to be blocked.

Another extension to XML Schema is typing of IDREF and IDREFS. The literature [1] pointed out that neither XML Schema nor DTD support typing of IDREF and IDREFS. In this case, a reference may point to any kind of element instance. One cannot require a reference to point to only an expected kind of element instances. For example, it is possible that attribute @courses of the element *takes* in *student* references a person rather than a course. So, we extend attribute declaration for specifying the type of IDREF or IDREFS, for example, in lines (08)-(09) of Figure 2 and in lines (09)-(10) of Figure 3.

In Figure 4, type *TAType* inherits elements and attributes from both supertypes *studentType* and *teacherType*. There are two conflicts to be resolved, since elements *addr* and *dept* are declared on both supertypes *studentType* and

```
(01) <xsd:complexType name="TAType">
(02)   <xsd:complexContent>
(03)     <xsd:extension bases="studentType teacherType">
(04)       <xsd:rename>
(05)         <xsd:element name="dept" from="studentType" as="student-dept"/>
(06)         <xsd:element name="dept" from="teacherType" as="teacher-dept"/>
(07)       </xsd:rename>
(08)       <xsd:block from="studentType">
(09)         <xsd:element name="addr"/>
(10)       <xsd:block>
(11)     </xsd:extension>
(12)   </xsd:complexContent>
(13) </xsd:complexType>
```

Fig. 4. Type definition for element TA

```
(01) <xsd:complexType name="courseType">
(02)   <xsd:sequence>
(03)     <xsd:element name="name" type="xsd:string"/>
(04)     <xsd:element name="desc" type="xsd:string"/>
(05)     <xsd:element name="takenBy">
(06)       <xsd:attribute name="students" type="xsd:IDREFS"
(07)                      target="studentType" use="implied"/>
(08)       <xsd:attribute name="teachers" type="xsd:IDREFS"
(09)                      target="teacherType" use="implied"/>
(10)     </xsd:element>
(11)   </xsd:sequence>
(12)   <xsd:attribute name="cid" type="xsd:ID" use="required"/>
(13) </xsd:complexType>
(14) <xsd:complexType name="underCourseType">
(15)   <xsd:complexContent>
(16)     <xsd:extension bases="courseType"/>
(17)   </xsd:complexContent>
(18) </xsd:complexType>
(19) <xsd:complexType name="gradCourseType">
(20)   <xsd:complexContent>
(21)     <xsd:extension bases="courseType"/>
(22)   </xsd:complexContent>
(23) </xsd:complexType>
(24) <xsd:element name="student" type="studentType">
(25) <xsd:element name="teacher" type="teacherType">
(26) <xsd:element name="TA" type="TAType">
(27) <xsd:element name="underCourse" type="underCourseType">
(28) <xsd:element name="gradCourse" type="gradCourseType">
```

Fig. 5. Type definitions for elements course, underCourse and gradCourse

teacherType. In our *Extended XML Schema*, three ways can be used to handle conflicts. In the first way, a conflict resolution declaration is specified explicitly to indicate from which supertype an element or attribute is inherited, for example, the *block* construct in lines (08)-(10) of Figure 4 indicates that the declaration of *addr* is inherited from the supertype *teacherType* rather than from *studentType*. In the second way, the names of elements or attributes causing conflicts are explicitly renamed in the inheriting element declaration, for example, in the subtype *TAType* declaration, the *rename* construct in line (05) of Figure 4 renames element *dept* inherited from supertype *studentType* to *student-dept* while the *rename* construct in line (06) of Figure 4 from *teacherType* to *teacher-dept*. Finally, if there is a conflict and there is no conflict resolution declaration, then the element or attribute is inherited from the supertype in the order the supertypes are listed in the *extension* construct of the type definition.

```
<univ>
<– person instance –>
  <person pid="100">
    <name> Jaonne Barbosa </name>
    <birthdate> 1965-04-07 </birthdate>
    <addr>
      <street> 310 University </street>
      <city> Ottawa </city>
      <state> Ontario </state>
      <zip> K1S 5B6 </zip>
    </addr>
    <homephone> 5073322 </homephone>
  </person>
<– student instance –>
  <student pid="200">
    <name> Jones Gillmann </name>
    <birthdate> 1976-02-25 </birthdate>
    <addr> 708D Somerset St </addr>
    <homephone> 6185708 </homephone>
    <dept> Computer Science </dept>
    < takes courses="CS200 CS300" />
  </student>
<– teacher instance –>
  <teacher pid="300">
    <name> Alley Srivastava </name>
    <birthdate> 1957-06-26 </birthdate>
    <addr>
      <street> 56 Broson </street>
      <city> Ottawa </city>
      <state> Ontario </state>
      <zip> K2B 6M8 </zip>
    </addr>
    <workphone> 2314343 </workphone>
    <salary> 1200.00 <salary>
    <dept> Computer Science </dept>
    <teaches courses="CS200 CS400"/>
  </teacher>
<– TA instance –>
  <TA pid="400">
    <name> Alice Bumbulis </name>
    <birthdate> 1976-08-29 </birthdate>
    <addr>
      <street> 440 Albert </street>
      <city> Ottawa </city>
      <state> Ontario </state>
      <zip> K1R 6P6 </zip>
    </addr>
    <homephone> 2915318 </homephone>
    <workphone> 2502600 </workphone>
    <student-dept> CS </student-dept>
    <teacher-dept> SE </teacher-dept>
    <takes courses="CS400" />
    <teaches teaches="CS300" />
  </TA>
<– course instance –>
  <course cid="CS100" >
    <name> Introduction to CS </name>
    <desc> Continuing Education </desc>
  </course>
<– underCourse instances –>
  <underCourse cid="CS200" >
    <name> Introduction to DBS </name>
    <desc> Basic concepts </desc>
    <takenBy students="200" />
    <taughtBy teachers="300" />
  </underCourse>
  <underCourse cid="CS300" >
    <name> Introduction to SE </name>
    <desc> Basic concepts </desc>
    <takenBy students="200" />
    <taughtBy teachers="400" />
  </underCourse>
<– gradCourse instance –>
  <gradCourse cid="CS400" >
    <name> DBMS </name>
    <desc> Impl. Techniques </desc>
    <takenBy students="400" />
    <taughtBy teachers="300" />
  </gradCourse>
</univ>
```

Fig. 6. An XML instance document

Figure 5 shows the type definitions for elements *course*(lines (01)-(13)), *underCourse*(lines (14)-(18)) and *gradCourse*(lines (19)-(23)), and other element declarations(lines (24)-(28)).

3 Extensions to XML Instance Document

Consider the examples described before, type *personType* has three direct or indirect subtypes, *studentType*, *teacherType* and *TAType*, and type *courseType* two direct subtypes, *underCourseType* and *gradCourseType*. When polymorphism is introduced into XML, an element instance of *personType* in a valid instance document can be substituted with an instance of elements of its subtypes, and the instance document should still be valid. If the type of an element has at least one subtype, then the element is polymorphic. For example, element *person* is polymorphic since type *personType* has three direct or indirect subtypes. the *person* element instance can be substituted by instances of *student*, *teacher*,

or *TA* since their types all are subtypes of *personType*. Similarly, the instance of *course* can be substituted by instance of *underCourse* and *gradCourse*. The substituting element instances are referred to as polymorphic instances. From lines (04)-(07) of Figure 1, we can see that element *univ* can contain a number of *person* and *course* element instances; that is *univ*→*person*∗,*course*∗. Therefore, element *univ* can contain seven component element instances due to polymorphism: *person*, *student*, *teacher*, *TA*, *course*, *underCourse* and *gradCourse* instances.

Now we extend *XML Schema* with polymorphic reference, which is similar to polymorphic element. A little bit complicated example for polymorphic reference is that a teacher may teach several courses including *underCourses* and *gradCourses* as well, see the definition of element *teacher* in Figure 3 and its instance in Figure 6. In the definition, *teaches* is an IDREFS to *course*. If polymorphic references are supported by the system(that is, *teaches* can also be used to reference to either *underCourse* or *gradCourse* elements as their types all are subtypes of the type of element *course*), the following six combinations are valid in the instance document: (1) a teacher teaches courses; (2) a teacher teaches underCourses; (3) a teacher teaches gradCourses; (4) a TA teaches courses; (5) a TA teaches underCourses; and (6) a TA teaches gradCourses.

Polymorphic reference is introduced to meet the above requirements. An IDREF or IDREFS attribute of a given element can point to instance(s) of the substituting elements of the element. It is referred to as *polymorphic references*.

4 Conclusions

In this paper, we extend XML Schema to support the key object-oriented features such as nonmonotonic inheritance, overriding, blocking, conflict handling. Moreover, we extend XML instance document with polymorphism, including typing of references, polymorphic elements and polymorphic references.

References

1. Lewis, P.M., Bernstein, A., Kifer, M.: Databases and transaction processing: an application-oriented approach. Addison Wesley (2002)
2. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: Extensible Markup Language (XML) 1.0. 2nd Edn. Available at `http://www.w3.org/TR/REC-xml` (2000)
3. Davidson, A., Fuchs, M., Hedin, M.: Schema for Object-Oriented XML 2.0. W3C Note. Available at `http://www.w3.org/TR/NOTE-SOX` (1999)
4. Dobbie, G., Topor, R.W.: Resolving Ambiguities caused by Multiple Inheritance. In: Proceedings of the 4th DOOD International Conf. Singapore (1995) 265–280
5. Fallside, D.C.: XML Schema Part 0: Primer. W3C Recommendation. Available at `http://www.w3.org/TR/xmlschema-0/` (2001)
6. Lee, D., Chu, W.W.: Comparative analysis of six XML schema languages. ACM SIGMOD Record. **29** (2002) 117–151
7. Liu, M., Dobbie, G., Ling, T.W.: A logical foundation for deductive object-oriented databases. ACM Transaction on Database Systems. **27** (2002) 117–151