Identity Constraints for XML

Anne Brüggemann-Klein <brueggem@in.tum.de>
Mustapha Maalej <maalej@in.tum.de>
Marouane Sayih <sayih@in.tum.de>

Abstract

In this paper, we attempt to explain clearly our reading of XML Schema's identity constraint concepts. We illustrate our reading extensively with examples, in the style of a tutorial. We also illustrate usage styles and limitations of identity constraints in XML Schema. Finally, we demonstrate how a more general notion of identity constraints that is adapted to the hierarchical nature of XML documents can be expressed with XPath 2.0. Hence, the limitations that we have identified can be by-passed with assertions as introduced by XML Schema 1.1.

This paper is the full version of “XML Schema Identity Constraints Revisited” as presented at XML Prague 2014.

Table of Contents

Introduction

XML Schema transfers one of the fundamental database concepts into the realm of XML documents, namely identity constraints. Using XML Schema's identity constraint mechanisms, schema developers may impose that specific elements in a document are uniquely identified by some of the data they contain, so that these data serve as key sequences that identify or stand for the elements; they may also demand that other elements refer to such key sequences with their own data, guaranteeing referential integrity.

The contributions of this paper are as follows: Primarily, we attempt to explain clearly our reading of XML Schema's identity constraint concepts. We illustrate our reading extensively with examples, in the style of a tutorial. All examples validate as expected with Saxon. We also illustrate usage styles and limitations of identity constraints in XML Schema. Finally, we demonstrate how the limitations that we have identified can be by-passed with assertions as introduced by XPath 2.0 and XML Schema 1.1.

This investigation was motivated by and is applicable to the second author's PhD work that concerns XForms documents that are generated from an XML Schema and serve as editors for instances of the

schema. Since the XForms editors should support identity constraints, it was necessary to understand the concepts and implementation options with XPath. It is also relevant to the third author's PhD work on formal methods such as Abstract State Machines or XML Statecharts for XML technology when using XML Schema as as case study.

There has been some critical discussion in the XML community on the wording and semantics of the XML Schema section on identity constraints. In this paper, we hope to clarify some issues and to illustrate benefits and limitations. We also discuss extensions of XML Schema identity constraints and their implementations with the core XML technology XPath. The general discussion is in terms of an abstract data model for identity constraint constructs; syntax is used only in examples.

Why identity constraints?

Identity constraints are an integral concept of relational databases, which consist of a set of tables, each of which is regularly structured via a set of attributes (columns) and filled with a set of records (rows). Identity constraints with relational databases come in two flavors: Key and foreign key. A key for a table is a group of attributes whose values, called a key sequence, identify a record. Keys express intent and enhance performance through index tables. A foreign key ties tables together. It is a group of attributes whose values (key sequences) reference a record in the same or another table, thus expressing relationships such as part-of, is-a, extends, or user-defined relationships.

XML has some built-in methods to express relationships:

- Hierarchy or containment, which can be used to represent part-of relationships or composition.
- Repetition or ordering among siblings, which can be used to represent sequential order.
- Inheritance and substitution groups, which can be used to represent is-a relationships.
- ID and IDREF, which are of (limited) use to express user-defined relationships.

Furthermore, XML documents have always used “soft”, less formalized methods such as hypertext linking to express relationships.

However, beyond these concepts, obviously the authors of XML Schema felt the need to have a more powerfull and formal mechanism to express identity constraints for unique identification and references in XML, presumably for the cases where XML is used not so much to represent natural-language structured text but more formally defined semi-structured data.

XML Schema's concepts for identity constraints: key

What is a key? Generally speaking, a key defines how to identify, within some scope, objects by values of some of the objects' components, partial data in the object that serve to differentiate it from others. The values of an objects components, called a key sequence, act as an identifier for that object, the keyed object for or target object of the key sequence. We speak of the key condition of the key constraint. For example, a real estate agency identifies property by ZIP code, street name and house number within the scope of a country.

In XML Schema, an element declaration may contain a key constraint[1] that determines which sub-parts of a conformant element information item should be uniquely determined by which combination of values that each of the sub-parts contains. Each key constraint has a name that is unique within the schema, a selector function `s` and a fields function `f` that is made up of a sequence ($f_1,\ldots,f_n$) of n field functions $f_1,\ldots,f_n$ for some natural number n. Each element information item `E` that instantiates

[1] Actually, XML Schema introduces two similar concepts called `key` and `unique`. For the sake of brevity, we only discuss `key` in this article.

the element declaration establishes a scope for the key constraint; it satisfies the key constraint if and only if the following conditions hold:

- The expression `E.s()`[2] evaluates to a sequence of element information items on the `descendent-or-self` axis of `E`. The set of those element information items is called the target set of `E`.
- For each element information item N in the target set of E, the expression $N.f_i()$ evaluates to an element information item or an attribute information item of an element information item on the descendant-or-self-axis of `N`; in the former case, the element information item $N.f_i()$ must be declared to be of a simple type, with no subelements allowed. We call the sequence of string[3]values of $N.f()=(N.f_1(),\ldots,N.f_n())$ the key sequence of `N`.
- The key sequences of the element information items in the target set of `E` are all different sequences.[4] This condition gives rise to an index table at element information item `E` for the key constraint that collects pairs of key sequences and corresponding element information items in the target set of element `E`. Such an index table has no multiple entries for any one key sequence.

Once more, the key condition must be satisfied for each element scope `E` whose declaration carries the key.

XML Schema's key concept is obviously transfered from relational databases, where a schema may require of a table that each of its rows is uniquely determined by the combination of values of a specific selection of table attributes. The main difference is that elements in XML are more freely structured than tables. Hence, we need to (or may) customize explicitly which sub-parts of the element should be uniquely identified by the key: In databases, automatically all rows in a table must be uniquely identified; in XML Schema, a general selector function defines the set of target elements. Also, in XML Schema we have more flexibility in selecting the fields that make up a unique key sequence: They can be any textual values in (sub-)elements or attributes of (sub-)elements of the "keyed" element, as defined by a general fields function, not just attributes, as in relational databases.

The schema collections.xsd in the appendix illustrates key constraints with progressing complexity. The scenario is a repository of collections of items. There are a number of ways how collections are structured and how items are assigned to collections, as illustrated in Figure 1, “Alternatives how to structure collections.”.

Figure 1. Alternatives how to structure collections.

```
<xs:element name="collections">
 <xs:complexType>
  <xs:choice>
   <xs:element ref="collection" minOccurs="1" maxOccurs="unbounded"/>
   <xs:element ref="collectionsDBStyle"/>
   <xs:element ref="collectionXMLStyle"/>
   <xs:element ref="collectionsWithLocalOwners"/>
   <xs:element ref="collectionsWithGlobalOwners"/>
   <xs:element ref="collectionsWithOwnersDBStyle"/>
   <xs:element ref="collectionsWithOwnersHybridStyle"/>
   <xs:element ref="collectionsWithOwnersSensible"/>
  </xs:choice>
 </xs:complexType>
</xs:element>
```

On the most elementary level, see Figure 2, “The collection structure.”, we have a sequence of `collection` elements, each containing a sequence of `item` elements that are uniquely identified

[2]We are using an object-oriented notational style, so `E.s()` stands for function `s` applied to argument `E`.
[3]XML Schema deals with typed values. We are simplifying to strings in this paper, without any relevant loss of generality. XML Schema also has some condition on nillable elements that we are ignoring here, blending out some subtle points of type definitions.
[4]This is the main uniqueness condition that makes a key a key. The notion of equality of sequences needs to be adapted when items in the sequence are not just strings but typed values.

by their attribute `idItem` within that collection container. This constraint is expressed by the key `collectionKey` which is defined within the element declaration of `collection`. Consequently, at the instance level, each element information item `collection` has its own index table for the key constraint `collectionKey`, so that key sequences can be re-used in different element scopes. This is illustrated in the XML document sampleCollection.xml, see Figure 3, “The collection instance.”.

Figure 2. The collection structure.

```
<xs:element name="collection">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="collectionKey">
  <xs:selector xpath="item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
</xs:element>
<xs:element name="item" type="itemType"/>
<xs:complexType name="itemType">
 <xs:attribute name="idItem" type="xs:string" use="required"/>
</xs:complexType>
```

Figure 3. The collection instance.

```
<collections xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="collections.xsd">
    <collection>
        <item idItem="a"/>
        <item idItem="b"/>
        <!-- No duplicate idItem values allowed within collection.
        <item idItem="a"/>
        -->
    </collection>
    <collection>
        <!-- Duplicate idItem values allowed in different collections.-->
        <item idItem="a"/>
        <item idItem="c"/>
        <!-- No duplicate idItem values allowed within collection.
        <item idItem="a"/>
        -->
    </collection>
</collections>
```

This example uses hierarchical structure that is typical for XML. A database-like approach would use a flat repository of collections and another repository of items that indicate to which collection they belong. The schema collections.xsd, see Figure 4, “The collectionsDBStyle outer structure.” and Figure 5, “The collectionsDBStyle inner structure.”, illustrates that way of structuring the data with an element `collectionsDBStyle` that may contain sequences of subelements `collectionDBStyle` and `itemInCollection`. Elements `itemInCollection` express to which collection they belong by their attribute `idCollection`, not by the hierarchy within a collection container.

Figure 4. The collectionsDBStyle outer structure.

```
<xs:element name="collectionsDBStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="collectionDBStyle" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="itemInCollection" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="collectionDBStyleKey">
  <xs:selector xpath="collectionDBStyle"/>
  <xs:field xpath="@idCollection"/>
 </xs:key>
 <xs:key name="itemInCollectionKey">
  <xs:selector xpath="itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="itemRefersToCollection" refer="collectionDBStyleKey"> [3 lines]
</xs:element>
```

Figure 5. The collectionsDBStyle inner structure.

```
<xs:element name="itemInCollection">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="itemType">
    <xs:attribute name="idCollection" type="xs:string" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
<xs:element name="collectionDBStyle">
 <xs:complexType>
  <xs:attribute name="idCollection" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
```

The key constraint `collectionDBStyleKey` expresses that elements `collecionDBStyle` must be uniquely indentified by their attributes `idCollection`. The key constraint `itemInCollection` expresses that elements `itemInCollection` are uniquely identified by the combination of their `idCollection` and `idItem` attributes.

In addition, the schema expresses referential integrity with the key reference `itemRefersToCollection`, a feature we explain in the next section.

The XML document sampleCollectionDBStyle.xml, see Figure 6, “The collectionsDBStyle instance.” illustrates the database style of structuring the data on the instance level.

Figure 6. The collectionsDBStyle instance.

```
<collectionsDBStyle xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="collections.xsd">
    <collectionDBStyle idCollection="x"/>
    <collectionDBStyle idCollection="y"/>
    <!-- No duplicate idCollection values allowed.
    <collectionDBStyle idCollection="y"/>
    -->
    <itemInCollection idCollection="x" idItem="a"/>
    <itemInCollection idCollection="x" idItem="b"/>
    <!-- Duplicate idItem values allowed,
         as long as idCollection values are different. -->
    <itemInCollection idCollection="y" idItem="a"/>
    <itemInCollection idCollection="y" idItem="c"/>
    <!-- No duplicate idCollection/idItem values allowed.
    <itemInCollection idCollection="y" idItem="a"/>
    -->
    <!-- Referenced collection must exist (referential integrity).
    <itemInCollection idCollection="z" idItem="a"/>
    -->
</collectionsDBStyle>
```

Finally, with the element `collectionXMLStyle`, we further explore the use of key constraints for hierarchical structures by adding `collectionXMLStyle` recursively, see Figure 7, “The collectionXMLStyle structure.”.

Figure 7. The collectionXMLStyle structure.

```
<xs:element name="collectionXMLStyle">
 <xs:complexType>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element ref="collectionXMLStyle"/>
   <xs:element ref="item"/>
  </xs:choice>
  <xs:attribute name="idCollection" type="xs:string" use="required"/>
 </xs:complexType>
 <xs:key name="collectionXMLStyleKey">
  <xs:selector xpath="collectionXMLStyle"/>
  <xs:field xpath="@idCollection"/>
 </xs:key>
 <!-- [13 lines]
</xs:element>
```

Each element `collectionXMLStyle` establishes its own scope for key constraints on unique collection and item IDs. The relationships between containing and contained elements is expressed implicitly, through hierarchy, not through explicit references. Alternatively, we can demand globally unique IDs for collections and items in this scenario by activating the keys `globalCollectionXMLStyle` and `globalItemKey`.

Instantiation of recursive structures is illustrated by sampleCollectionXMLStyle.xml, see Figure 8, “The collectionXMLStyle instance.”.

Figure 8. The collectionXMLStyle instance.

```
<collectionXMLStyle xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="collections.xsd" idCollection="x">
    <collectionXMLStyle idCollection="z">
        <collectionXMLStyle idCollection="x">
            <item idItem="a"/>
            <item idItem="b"/>
            <collectionXMLStyle idCollection="z">
                <collectionXMLStyle idCollection="x">
                    <item idItem="a"/>
                    <item idItem="b"/>
                </collectionXMLStyle>
                <collectionXMLStyle idCollection="y">
                    <item idItem="a"/>
                    <item idItem="c"/>
                </collectionXMLStyle>
            </collectionXMLStyle>
        </collectionXMLStyle>
        <collectionXMLStyle idCollection="y"> [3 lines]
    </collectionXMLStyle>
</collectionXMLStyle>
```

Propagating index tables upwards

Element information items that carry a key constraint make their index tables available for referencing, just as keys in one table of a relational database can be used as foreign keys in another table to refer to table rows.

But, there are problems with this idea: Different index tables for different element instances that conform to the same key constraint might have conflicting entries; that is, entries for the same key sequences but different target elements. How can we deal with local index tables, across which conflicts arise naturally? From where can we reference them? Obviously, we must be able to decouple referencing and referenced elements structurally; but how do we reconcile this requirement with local index tables?

How to design a reference concept that utilizes XML Schema key constraints? The case is less clear-cut than the definition of the key-foreign key concept in relational databases, for reasons of hierarchy: In XML documents, element information items that carry identity constraints may be buried in the document hierarchy and they may be recursively nested.

The XML Schema concept is that index tables propagate upwards in the element hierarchy and that each element that gets an index table that way may use it for referencing, as explained in the next section. There is only one problem with that approach: Entries in the index tables of two sibling element informations might be in conflict, as might be entries in the index tables of a parent and child element information item. in conflict. Two entries consisting of a key sequence and an element information item are in conflict if the two key sequences are equal but the two element information items are not. XML Schema defines two conflict resolution rules for index table propagation:

- If two or more sibling element information items have conflicting entries for some key sequence in their index table, none of the entries is included in the parent element information item's index table: Competing children cancel each other out.
- If a parent and a child element information item have conflicting entries for some key sequence in their index table, the parent's index table keeps its entry and the child's entry is discarded at the parent level: Parents dominate children.

There are a number of implications:

First, a key can only be used for referencing at an element information item at which it is defined or above in the element information item hierarchy.

Second, entries in index tables that propagate upwards in the element information item hierarchy may become unavailable for referencing, due to conflict resolution rules. Hence, structural decoupling of referencing and referenced elements does not work for local index tables in hierarchical structures.

Finally, at an element information item `E`, all entries that arise directly from a key at `E`, are available for referencing. Since parents dominate children, they are never removed from that element information item's index table for the key, although they might be cancelled out further up in the hierarchy.

This is our best effort at reading the conflict resolution rules. Implementations disagree with our reading in an inconsistent manner.

We illustrate these effects after discussing XML Schema's key reference constructs.

XML Schema's concepts for identity constraints: Referencing keys

What is a key reference? Generally speaking, a key reference defines how to refer from one object to another object via a key by values of some of the first object's components. The values of the first object's components act as a key sequence that keys the second object. The key sequence must be present in the key; that is, no dangling references or referential integrity. For example, a real-estate agency references a property in a contract via its key.

In XML Schema, an element declaration may also contain a key reference constraint. Just like a key constraint, a key reference constraint has a name that is unique within the schema, a selector function `s` and a field function $f=(f_1,\ldots,f_n)$ of width `n` for some natural number `n`. In addition, it has the name of the key constraint that it refers to. The fields functions of the key reference constraint and of the referenced key constraint must have identical width. An element information item `E` satisfies the key reference constraint if and only if there is an index table for the referenced key constraint at E that has, for each element information item `N` in the target set `E.s()`, an entry in the index table for the key sequence `N.f()`. The referenced index table includes any entries that originate directly at `E`, plus any entries that propagate upwards from below `E` and survive conflict resolution.

An element in scope `E` can reference another element via a sequence of values from the first element's textual components if there exists an index table at `E` that has an entry for the sequence of values, acting as a key sequence. The second element must be the target element for that key sequence. This is the key reference condition or referential integrity constraint.

The semantics of this condition is that the element information item `N` refers to the node that appears in the index table for the key sequence `N.f()`. Hence, reference key constraints guarantee referential integrity.

Let us now look at examples that add owner information to our collections scenario in a number of ways.

On the most elementary level, see Figure 9, “The collectionWithLocalOwners structure.”, generalizing the `collection` elements, we have elements `collectionWithLocalOwners` that contain `item` elements and `owner` elements, the latter listing the `item` elements that this owner owns. As before, `item` elements within the collection are uniquely identified by their attribute `idItem`, as expressed by key constraint `uniqueItems`. In addition, each `item` element can be owned by at most one `owner`, as expressed by key constraint `uniqueOwnership`. Finally, each `item` that has an `owner` must be listed within the collection, as expressed by key reference constraint `ownedItem2ItemInCollection`. Conversely, we could also postulate that each item in a collection must be owned by someone, via a second key reference constraint. We position key and key reference constraints at the element declaration for `collectionWithLocalOwners`. Hence, items listed in a collection and ownership information for these items live completely within each collection, without interfering with items and owner in parallel collections. This works but is of limited value semantically. We illustrate with the XML instance sampleCollectionsWithLocalOwners.xml, see Figure 10, “The collectionsWithLocalOwners instance.”.

Figure 9. The collectionWithLocalOwners structure.

```
<xs:element name="collectionWithLocalOwners">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="owner" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueItems">
  <xs:selector xpath="item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:key name="uniqueOwnership">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="ownedItem2itemInCollection" refer="uniqueItems">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:keyref>
</xs:element>
```

Figure 10. The collectionsWithLocalOwners instance.

```
<collections xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="collections.xsd">
    <collectionsWithLocalOwners>
        <collectionWithLocalOwners>
            <item idItem="a"/>
            <item idItem="b"/>
            <owner>
                <item idItem="a"/>
            </owner>
            <owner>
                <!-- Each item within a collection can only be owned once. [2 lir
            </owner>
        </collectionWithLocalOwners>
        <collectionWithLocalOwners>
            <item idItem="a"/>
            <item idItem="c"/>
            <owner>
                <item idItem="a"/>
                <item idItem="c"/>
            </owner>
        </collectionWithLocalOwners>
    </collectionsWithLocalOwners>
</collections>
```

Now we reorganize the data slightly, pulling ownership information from the collection and moving it up one level, into a new container element `collectionsWithGlobalOwners`, see Figure 11, “The collectionsWithGlobalOwners structure.”. In tandem, we also move the unique ownership constraint and the key reference constraint that expresses integrity of references from owned items to items in a collection up. The problem with this structure is that `item` attributes `idItem` that are locally unique within collections are not unique at the level of the container element `collectionsWithGlobalOwners`. The conflict resolutions rules for the upwards propagation of index tables imply, that on the instance level items with conflicting `idItem` attributes cannot be referenced by non-local owners. This phenomenon is illustrated in the XML instance sampleCollectionsWithGlobalOwners.xml see Figure 12, “The collectionsWithGlobalOwners instance.”.

Figure 11. The collectionsWithGlobalOwners structure.

```
<xs:element name="collectionsWithGlobalOwners">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="owner" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueGlobalOwnership">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="globalOwnedItem2itemInCollection" refer="collectionKey">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:keyref>
</xs:element>
```

Figure 12. The collectionsWithGlobalOwners instance.

```
<collections xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="collections.xsd">
    <collectionsWithGlobalOwners>
        <collection>
            <!-- This element cannot be referenced by owner,
                 due to conflict resolution rules. -->
            <item idItem="a"/>
            <item idItem="b"/>
        </collection>
        <collection>
            <!-- Duplicate idItem values allowed in different collections.
                 But: This element can not be referenced by owner. -->
            <item idItem="a"/>
            <item idItem="c"/>
        </collection>
        <owner>
            <!-- Reference cannot be resolved. <item idItem="a"/> -->
        </owner>
        <owner>
            <item idItem="c"/>
        </owner>
    </collectionsWithGlobalOwners>
</collections>
```

The previous example demonstrates that key reference constraints are not as fully adapted to the hierarchical nature of XML documents as are key constraints. Before delving further into causes and potential solutions, we show one straight-forward use of key reference constraints in a database-like scenario with flat, non-hierarchical structures in XML instance sampleCollectionsWithOwnersDBStyle.xml, see Figure 13, “The collectionsWithOwnersDBStyle structure.” and Figure 14, “The collectionsWithOwnersDBStyle instance.”.

Figure 13. The collectionsWithOwnersDBStyle structure.

```
<xs:element name="collectionsWithOwnersDBStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="collectionsDBStyle"/>
   <xs:element ref="ownerDBStyle" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueOwnershipDBStyle">
  <xs:selector xpath="ownerDBStyle/itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="ownedItem2ItemInCollectionDBStyle"
  refer="itemInCollectionKey">
  <xs:selector xpath="ownerDBStyle/itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:keyref>
</xs:element>
```

Figure 14. The collectionsWithOwnersDBStyle instance.

```
<collectionsWithOwnersDBStyle xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
    xsi:noNamespaceSchemaLocation="collections.xsd">
    <collectionsDBStyle>
        <collectionDBStyle idCollection="x"/>
        <collectionDBStyle idCollection="y"/>
        <itemInCollection idCollection="x" idItem="a"/>
        <itemInCollection idCollection="x" idItem="b"/>
        <!-- Duplicate idItem values allowed in different collections -->
        <itemInCollection idCollection="y" idItem="a"/>
        <itemInCollection idCollection="y" idItem="c"/>
        <!-- Referenced collection must exist (referential integrity)
        <itemInCollection idCollection="z" idItem="a"/>
        -->
    </collectionsDBStyle>
    <ownerDBStyle>
        <itemInCollection idCollection="x" idItem="a"/>
    </ownerDBStyle>
    <ownerDBStyle>
        <itemInCollection idCollection="y" idItem="a"/>
        <!-- Violation of referential integrity: owned item  must exist
        <itemInCollection idCollection="z" idItem="a"/> -->
    </ownerDBStyle>
</collectionsWithOwnersDBStyle>
```

Let us now explore how we can reconcile key reference constraints with true hierarchies. How can we fix the container element `collectionsWithGlobalOwners`, referencing into locally defined keys? First of all, methodically, it is necessary to give unique IDs to collections. Then we need a method to refer to an item within a specific collection. Theoretically, this could be achieved in to possible ways:

- First, by having a global key constraint that states that items are uniquely defined by their own attribute `idItem` and by the attribute `idCollection` of the collection to which the item belongs. Unfortunately, such a key constraint cannot be defined in XML Schema, because field functions may only point to information within the element that is to be "keyed".

- Second, by keeping the local key constraint that guarantees locally unique attributes `idItem` within each collection and combining it with a two-part, "chaining" key reference constraint that references

the collection with one component and the item within that collection with the second component. Evidently, XML Schema provides no such mechanism that locates a local index table with one component and an entry in that local index table with another component. We feel that such a chaining key reference mechanism would be the perfect complement to local key constraints, fully adapting not only key constraints but also key reference constraints to the hierarchical nature of XML documents.

The XML Schema recommendation provides a solution to our problem that works but, unfortunately, introduces redundancy, namely to repeat the collection id on each item within the collection. This is a cross between the database-style solution and the hierarchical XML approach that accommodates the shortcomings of key and key reference constraints with respect to hierarchical data. We demonstrate this solution with the container element `collectionsWithOwnersHybridStyle` see Figure 15, “The collectionsWithOwnersHybridStyle structure.”. We use the assertion mechanism of XML Schema 1.1 to guarantee that redundant information is consistent. An XML instance for the hybrid solution can be found in sampleCollectionsWithOwnersHybridStyle.xml, see Figure 16, “The collectionsWithOwnersHybridStyle instance.”.

Figure 15. The collectionsWithOwnersHybridStyle structure.

```
<xs:element name="collectionsWithOwnersHybridStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="collectionHybridStyle" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="itemInCollection" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="idCollection" type="xs:string" use="required"/>
     <xs:assert
      test="every $id in itemInCollection/@idCollection satisfies
        $id=@idCollection"
     />
    </xs:complexType>
   </xs:element>
   <xs:element ref="ownerDBStyle" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueItemsHybridStyle"> [4 lines]
 <xs:key name="uniqueCollectionsHybridStyle"> [3 lines]
 <xs:key name="uniqueOwnershipHybridStyle"> [4 lines]
 <xs:keyref name="ownedItems2ItemsInCollectionHybridStyle" [5 lines]
</xs:element>
```

Figure 16. The collectionsWithOwnersHybridStyle instance.

```
<collectionsWithOwnersHybridStyle xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
    xsi:noNamespaceSchemaLocation="collections.xsd">
    <collectionHybridStyle idCollection="x">
        <itemInCollection idCollection="x" idItem="a"/>
        <itemInCollection idCollection="x" idItem="b"/>
    </collectionHybridStyle>
    <collectionHybridStyle idCollection="y">
        <itemInCollection idCollection="y" idItem="a"/>
        <itemInCollection idCollection="y" idItem="c"/>
        <!-- Attributes idCollection in itemInCollection and in the parent
             collectionHybridStyle must be consistent.
        <itemInCollection idCollection="x" idItem="c"/> -->
    </collectionHybridStyle>
    <ownerDBStyle>
        <itemInCollection idCollection="x" idItem="a"/>
        <itemInCollection idCollection="y" idItem="a"/>
        <!-- Referenced collection must exist (referential integrity).
        <itemInCollection idCollection="z" idItem="a"/> -->
        <!-- Referenced collection must exist (referential integrity).
        <itemInCollection idCollection="z" idItem="a"/>
        -->
    </ownerDBStyle>
</collectionsWithOwnersHybridStyle>
```

Unfortunately, the hybrid solution works for one level of collections only. It breaks down for recursive structures.

Implementations

We have tested the implementation of XML Schema identity constraints with Xerces and Saxon. Both parsers seem to implement the key concept correctly. With keyref, both parsers report any references that don't appear in any index table. In the absence of conflicts, Xerces and Saxon accept any valid references (with one exception for Xerces, which appears to be a genuine bug). Xerces does never report conflicting entries in index tables even if entries should not be available at the point of reference due to conflict resolution and upwards propagation (at least according to our understanding of conflict resolution). Saxon always reports conflicting entries in index tables even if conflicts should have been resolved (at least according to our understanding of conflict resolution).

Hence, Xerces and Saxon implement conflict resolution rules of XML Schema differently from each other and differently from our reading of the standard.

Identity constraints with XPath 2.0

We have seen that XML Schema's key reference mechanism, as it is currently defined, is too weak to express referential integrity with locally defined key constraints. We can solve this type of problems, however, with another mechanism that was introduced in XML Schema 1.1, namely assertions. This solution is based on XPath 2.0 and has benefits beyond XML Schema:

- First, we can use the XPath expressions for identity constraints in binding expressions of XForms. Hence, XFGen can generate editors that are aware of identity constraints.
- Second, we can use the XPath expressions for identity constraints in XQuery and XSLT to implement referencing functions such as key() in XSLT, putting constraints to work beyond data integrity.

In the appendix, we demonstrate our solution with the schema recursionWithAssertionsAbsoluteRefs.xsd and the instance

sampleRecursionWithAssertionsAbsoluteRefs.xml. We have defined three XPath expressions to implement the three reference semantics' of Xerces, Saxon and our own reading. With recursionRelativeRefs.xsd and sampleRecursionRelativeRefs.xml, also in the appendix, we illustrate some form of reference chaining on the basis of XPath.

Conclusions and further work

XML Schema transfers one of the fundamental database concepts into the realm of XML documents, namely identity constraints, introducing the concepts of key constraint and key reference constraint. XML Schema allows index tables for key constraints that have local element scope, thus adapting the key-constraint concept successfully to the hierarchical nature of XML documents. However, adaptation is incomplete because locally defined key constraints are not fully available for referencing. The missing piece is referencing into a local index table by chaining with reference to some local scope. Hence, XML Schema identiy constraints work best for “normalized”, flat, database-like documents.

We have demonstrated how assertions, that have been introduced with XML Schema 1.1, can be used as a substitute for key references in scenarios that involve hierarchy. We have implemented the three alternative semantics that come from implementations of identity constraints and our own reading of XML Schema. We have also extended the referencing concept to handle chaining of references into local index tables. With assertions, the key reference constraints are essentially expressed with XPath expressions. This has benefits beyond use in XML Schema assertions: The XPath expressions can be used on limited implementation platforms such as XForms; and they can put constraints to work beyond guaranteering data integrity, for example to implementent navigation functions such as key() in XSLT.

Although XML Schema identity constraints are much more powerful than the ID/IDREF concepts that orginate from XML DTD, they don't seem to be used much in practice. Nevertheless, we see value in having clarified some issues in connection with XML Schema's identity constraints.

This investigation was motivated by the second author's PhD thesis that targets XForms documents that are generated from an XML Schema and serve as fully functional editors for instances of the schema. Naturally, these editors should support identity constraints, guaranteeing uniqueness and referential integrity dynamically, while the document is edited. This can be achieved within XForms, when identity constraints are expressed as XPath expressions. We have demonstrated for the case of key reference constraints, how this can be accomplished with XPath 2.0. A more extensive discussion is included in Mustapha Maalej's PhD thesis.

Further work needs to be done to formalize and fully implement reference chaining.

There are a number of aspects that we have left out of the discussion above:

- Simulating multipart key / key reference with assertions
- Data types when comparing key sequences
- Different points of attachment for key / key reference (element declarations) and assertions (type definitions) and their ramifications
- Error messages when using assertions (less specific than error messages when using key / keyref); Schematron being a viable alternative
- Scope issues: Reference constraints cannot always put with the referencing element or its type
- Document boundaries for identity constraints; Schematron being a viable alternative
- Constraints on XPath expressions for selector and field functions: check when simulating key / key reference with assertions
- Declaration of intent: weaker with assertions than with key / key reference, with implications for modeling and performance

Acknowledgement

Eric van der Vlist in personal communication pointed out to us that XML Schema's concepts of identity constraints are better adapted to database-like flat structures than to truely hierarchical structures, providing a valuable entry point into this investigation. Thank you!

Bibliography

[M2014] Mustapha Maalej: *Generieren von XML-Editoren in XForms aus XML Schema.* Ph.D. Thesis, TU München, 2014. In preparation.

[TBMM2004] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn: *XML Schema Part 1: Structures Second Edition.* W3C Recommendation, W3C, October 2004, *http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.*

[vdV2011] Eric van der Vlist: *XML Schema.* O'Reilly, Kindle Edition, 2011.

[W2012] Priscilla Walmsley: *Definite XML Schema.* Prentice Hall, 2nd edition, 2012.

A. Appendix: Listing collections.xsd

This paper comes with an XSD file collections.xsd, as listed below completely. Screenshots of the schema and instances are embedded into the paper.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Items in a collection have unique IDs.
There are a number of ways how collections may be structured.
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified">
 <xs:element name="collections">
  <xs:complexType>
   <xs:choice>
    <xs:element ref="collection" minOccurs="1" maxOccurs="unbounded"/>
    <xs:element ref="collectionsDBStyle"/>
    <xs:element ref="collectionXMLStyle"/>
    <xs:element ref="collectionsWithLocalOwners"/>
    <xs:element ref="collectionsWithGlobalOwners"/>
    <xs:element ref="collectionsWithOwnersDBStyle"/>
    <xs:element ref="collectionsWithOwnersHybridStyle"/>
    <xs:element ref="collectionsWithOwnersSensible"/>
   </xs:choice>
  </xs:complexType>
 </xs:element>
 <xs:element name="collection">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
  <xs:key name="collectionKey">
   <xs:selector xpath="item"/>
   <xs:field xpath="@idItem"/>
```

```
 </xs:key>
</xs:element>
<xs:element name="item" type="itemType"/>
<xs:complexType name="itemType">
 <xs:attribute name="idItem" type="xs:string" use="required"/>
</xs:complexType>
<xs:element name="collectionsDBStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="collectionDBStyle" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="itemInCollection" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="collectionDBStyleKey">
  <xs:selector xpath="collectionDBStyle"/>
  <xs:field xpath="@idCollection"/>
 </xs:key>
 <xs:key name="itemInCollectionKey">
  <xs:selector xpath="itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="itemRefersToCollection" refer="collectionDBStyleKey">
  <xs:selector xpath="itemInCollection"/>
  <xs:field xpath="@idCollection"/>
 </xs:keyref>
</xs:element>
<xs:element name="itemInCollection">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="itemType">
    <xs:attribute name="idCollection" type="xs:string" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
<xs:element name="collectionDBStyle">
 <xs:complexType>
  <xs:attribute name="idCollection" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
<xs:element name="collectionXMLStyle">
 <xs:complexType>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element ref="collectionXMLStyle"/>
   <xs:element ref="item"/>
  </xs:choice>
  <xs:attribute name="idCollection" type="xs:string" use="required"/>
 </xs:complexType>
 <xs:key name="collectionXMLStyleKey">
  <xs:selector xpath="collectionXMLStyle"/>
  <xs:field xpath="@idCollection"/>
 </xs:key>
 <!--
       <xs:key name="itemGroupedByCollectionKey">
           <xs:selector xpath="item"/>
           <xs:field xpath="@idItem"/>
       </xs:key>
```

```
        <xs:key name="globalCollectionXMLStyle">
            <xs:selector xpath=".|.//collectionXMLStyle"/>
            <xs:field xpath="@idCollection"/>
        </xs:key>
        <xs:key name="globalItemKey">
            <xs:selector xpath=".//item"/>
            <xs:field xpath="@idItem"/>
        </xs:key>
        -->
</xs:element>
<xs:element name="collectionsWithLocalOwners">
 <xs:complexType>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
   <xs:element ref="collectionWithLocalOwners"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="collectionWithLocalOwners">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="owner" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueItems">
  <xs:selector xpath="item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:key name="uniqueOwnership">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="ownedItem2itemInCollection" refer="uniqueItems">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:keyref>
</xs:element>
<xs:element name="owner">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="collectionsWithGlobalOwners">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="owner" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueGlobalOwnership">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="globalOwnedItem2itemInCollection" refer="collectionKey">
  <xs:selector xpath="owner/item"/>
  <xs:field xpath="@idItem"/>
```

```
 </xs:keyref>
</xs:element>
<xs:element name="collectionsWithOwnersDBStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="collectionsDBStyle"/>
   <xs:element ref="ownerDBStyle" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueOwnershipDBStyle">
  <xs:selector xpath="ownerDBStyle/itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:keyref name="ownedItem2ItemInCollectionDBStyle"
  refer="itemInCollectionKey">
  <xs:selector xpath="ownerDBStyle/itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:keyref>
</xs:element>
<xs:element name="ownerDBStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="itemInCollection" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="collectionsWithOwnersHybridStyle">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="collectionHybridStyle"
     minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="itemInCollection"
        minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="idCollection" type="xs:string" use="required"/>
     <xs:assert
      test="every $id in itemInCollection/@idCollection satisfies
              $id=@idCollection"
     />
    </xs:complexType>
   </xs:element>
   <xs:element ref="ownerDBStyle" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:key name="uniqueItemsHybridStyle">
  <xs:selector xpath="collectionHybridStyle/itemInCollection"/>
  <xs:field xpath="@idCollection"/>
  <xs:field xpath="@idItem"/>
 </xs:key>
 <xs:key name="uniqueCollectionsHybridStyle">
  <xs:selector xpath="collectionHybridStyle"/>
  <xs:field xpath="@idCollection"/>
 </xs:key>
 <xs:key name="uniqueOwnershipHybridStyle">
```

```
   <xs:selector xpath="ownerDBStyle/itemInCollection"/>
   <xs:field xpath="@idCollection"/>
   <xs:field xpath="@idItem"/>
  </xs:key>
  <xs:keyref name="ownedItems2ItemsInCollectionHybridStyle"
   refer="uniqueItemsHybridStyle">
   <xs:selector xpath="ownerDBStyle/itemInCollection"/>
   <xs:field xpath="@idCollection"/>
   <xs:field xpath="@idItem"/>
  </xs:keyref>
 </xs:element>
 <xs:element name="collectionsWithOwnersSensible">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="collectionWithID" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="idCollection" type="xs:string" use="required"/>
     </xs:complexType>
     <xs:key name="uniqueItemsInCollection">
      <xs:selector xpath="item"/>
      <xs:field xpath="@idItem"/>
     </xs:key>
    </xs:element>
    <xs:element ref="ownerDBStyle" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:assert
  test="every $ownerItem in ownerDBStyle/itemInCollection satisfies
  (some $collection in collectionWithID satisfies
  ($ownerItem/@idCollection=$collection/@idCollection and
  (some $item in $collection/item satisfies $ownerItem/@idItem=$item/@idItem)))"
    />
  </xs:complexType>
  <xs:key name="uniqueCollections">
   <xs:selector xpath="collectionWithID"/>
   <xs:field xpath="@idCollection"/>
  </xs:key>
  <xs:key name="uniqueOwnershipSensible">
   <xs:selector xpath="ownerDBStyle/itemInCollection"/>
   <xs:field xpath="@idCollection"/>
   <xs:field xpath="@idItem"/>
  </xs:key>
 </xs:element>
</xs:schema>
```

B. Appendix: Listing recursionWithAssertionsAbsoluteRefs.xsd

```
<?xml version="1.1" encoding="UTF-8"?>
```

```
<!DOCTYPE xs:schema [
<!ENTITY keyrefXercesStyle
    '<!-- No dangling references from collectionOwned to collection (Xerces mod
   <xs:assert id="ModelXerces" test="every $collectionOwned in owners//collecti
   (count(collections//collection/@id[.=$collectionOwned/collectionRef/@id]) ge
<!ENTITY keyrefSaxonStyle
    '<!-- No dangling and no ambiguous references from collectionOwned to colle
   <xs:assert id="ModelSaxon" test="every $collectionOwned in owners//collectio
    (count(collections//collection/@id[.=$collectionOwned/collectionRef/@id]) =
<!ENTITY keyrefTUMStyle
    '<!-- The TUM reading of the standard:
      two conflict resolution rules: siblings block each other, parents dominat
      <xs:assert id="ModelTUM" test="every $collectionOwned in owners//collecti
      (some $collection in collections//collection satisfies (
     $collection/@id = $collectionOwned/collectionRef/@id and
     (count(collections//collection/@id[.=$collectionOwned/collectionRef/@id])
     = count($collection/descendant-or-self::collection/@id[.=$collectionOwned/
]>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified"
 xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning" vc:minVersion="1.1">
 <xs:element name="collectionsWithOwners">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="collections">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <!-- In lieu of key -->
      <xs:assert
       test="count(collection/@id)=count(distinct-values(collection/@id))"
      />
     </xs:complexType>
    </xs:element>
    <xs:element name="owners">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="owner" type="ownerType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <!-- Use one of three &keyref(Xerces|Saxon|TUM)Style; references
   to emplement one of three proposed semantics for key reference. -->
   &keyrefTUMStyle;
  </xs:complexType>
 </xs:element>
 <xs:element name="collection" type="collectionType"/>
 <xs:complexType name="collectionType">
  <xs:sequence>
   <xs:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <!-- In lieu of key, as above -->
  <xs:assert
   test="count(collection/@id)=count(distinct-values(collection/@id))"
```

```
 />
 </xs:complexType>
 <xs:complexType name="ownerType">
  <xs:sequence>
   <xs:element name="collectionOwned" minOccurs="1" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="collectionRef">
       <xs:complexType>
        <xs:attribute name="id" type="xs:string" use="required"/>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

C. Appendix: Listing sampleRecursionWithAssertionsAbsoluteRefs.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<collectionsWithOwners xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="recursionWithAssertionsAbsoluteRefs.xsd">
 <!--
 Conflict type "siblings": collection "y"
 Conflict type "parent":   collection "w"
 -->
 <collections>
  <collection id="x">
   <collection id="z">
    <collection id="y"/>
   </collection>
   <collection id="w">
    <collection id="y">
     <collection id="v">
      <collection id="w"/>
     </collection>
    </collection>
   </collection>
  </collection>
 </collections>
 <owners>
  <owner>
   <collectionOwned>
    <collectionRef id="v"/>
    <!--
    In the Xerces model (no dangling references), all references to collections
    are accepted.
```

```
    In the Saxon model (all references must be unique, across hierarchy), refer
    to "w" and "y" are rejected.
    In the conflict resolution model (parents dominate children, siblings block
    references to "w" are accepted and to "y" are rejected.
    -->
   </collectionOwned>
  </owner>
  </owners>
</collectionsWithOwners>
```

D. Appendix: Listing recursionRelativeRefs.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 elementFormDefault="qualified"
 xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning" vc:minVersion="1.1">
 <xs:element name="collectionsWithOwners">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="collections">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="description" type="xs:string" use="optional"/>
      <xs:assert
       test="count(collection/@id)=count(distinct-values(collection/@id))"
      />
     </xs:complexType>
    </xs:element>
    <xs:element name="owners">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="owner" type="ownerType" minOccurs="0"
        maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="optional"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:assert
    test="every $collectionOwned in owners//collectionOwned satisfies
    (some $collection in collections//collection satisfies (
    deep-equal($collection/ancestor-or-self::collection/xs:string(@id),
      $collectionOwned/collectionRef/xs:string(@id))
    ))"
   />
  </xs:complexType>
 </xs:element>
 <xs:element name="collection" type="collectionType"/>
```

```
 <xs:complexType name="collectionType">
  <xs:sequence>
   <xs:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:assert
   test="count(collection/@id)=count(distinct-values(collection/@id))"/>
 </xs:complexType>
 <xs:complexType name="ownerType">
  <xs:sequence>
   <xs:element name="collectionOwned" minOccurs="1" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="collectionRef" minOccurs="1" maxOccurs="unbounded">
       <xs:complexType>
        <xs:attribute name="id" type="xs:string" use="required"/>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

E. Appendix: Listing sampleRecursionRelativeRefs.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<collectionsWithOwners xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="recursionRelativeRefs.xsd">
 <collections>
  <collection id="x">
   <collection id="z">
    <collection id="y"/>
   </collection>
   <collection id="w">
    <collection id="y">
     <collection id="v">
      <collection id="w"/>
     </collection>
    </collection>
   </collection>
  </collection>
 </collections>
 <owners>
  <owner>
   <collectionOwned>
    <collectionRef id="x"/>
    <collectionRef id="w"/>
```

```
      <collectionRef id="y"/>
      <collectionRef id="v"/>
      <collectionRef id="w"/>
    </collectionOwned>
  </owner>
 </owners>
</collectionsWithOwners>
```