Once Upon a Time a DTD Evolved into Another DTD ...

Lina Al-Jadir and Fatmé El-Moukaddem

Department of Mathematics and Computer Science
American University of Beirut
P.O. Box 11-0236, Beirut, Lebanon
{lina.al-jadir,fme05}@aub.edu.lb

Abstract. XML has become an emerging standard for data representation and data exchange over the web. In many applications a schema is associated with an XML document to specify and enforce the structure of the document. The schema may change over time to reflect a change in the real-world, a change in the user's requirements, mistakes or missing information in the initial design. In this paper, we consider DTDs as XML schema mechanism, and present an approach to manage DTD evolution. We build a set of DTD changes. We identify invariants which must be preserved across DTD changes. We define the semantics of each DTD change such that the new DTD is valid, existing documents conform to the new DTD, and data is not lost if possible. We illustrate our approach with a scenario.

1 Introduction

The *eXtensible Markup Language* (XML) has become an emerging standard for data representation and data exchange over the World Wide Web. Although an XML document is self-describing, in many applications [15] [19] [13] a schema is associated with the document to specify and enforce its structure. The schema of an XML document is allowed to be irregular, partial, incomplete, not always known ahead of time, and may consequently change frequently and without notice [11]. Moreover, the schema may change over time to reflect a change in the real-world, a change in the user's requirements, and mistakes in the initial design [18]. Most of the current XML systems do not support schema changes [18]. Modifying the schema of XML documents is not a simple task, since the documents which conform to the old schema must be transformed in order to conform to the new schema. In this paper, we consider the *document type definition* (DTD) as XML schema mechanism, and present an approach to manage DTD evolution. We propose a set of DTD changes, and define their semantics by preconditions and postactions, such that the new DTD is valid, existing XML documents conform to the new DTD, and data is not lost if possible. We illustrate our approach with a scenario requiring the modification of a DTD.

XEM [18] is an approach which handles DTD evolution. It supports 14 DTD changes. The semantics of these DTD changes is given by preconditions and results, to ensure the validity of the new DTD and the conformity of XML documents. Our approach differs from XEM in three points. First, our approach supports more DTD

D. Konstantas et al. (Eds.): OOIS 2003, LNCS 2817, pp. 3–17, 2003.

changes (see §3.1), such as changing the parent or child in a parent-child relationship, changing a parent-child relationship to an attribute and vice-versa, changing the order of a parent-child relationship, renaming an attribute, changing the element of an attribute, and changing the type of an attribute (examples of these DTD changes are given in §4). Moreover, groups are handled differently in the two approaches. XEM supports the DTD changes convertToGroup and FlattenGroup, while our approach supports changing a group to an element and vice-versa. Second, the same DTD change may have different semantics in the two approaches. For example, when changing the cardinality of a subelement S in the definition of element E from repeatable to non-repeatable, XEM removes all occurences of the subelement S except the first, while in our approach this DTD change is rejected if an instance of element E has more than one occurence of subelement S in the document. Third, avoiding data loss in the XML document when modifying its DTD is a major concern in our approach. It motivates the existence of some of our DTD changes (not available in XEM) and our semantics of DTD changes (different from XEM's semantics). Our approach, like XEM, is tightly-coupled with a database system. The XML DTD and documents are mapped into a database schema and a database instance respectively. DTD changes are implemented as database schema changes.

In [17] a loosely-coupled approach (SAXE) is proposed. An XML-Schema change is expressed as an Update-XQuery statement. This statement is rewritten into a safe Update-XQuery statement, by embedding constraint checking operations into the query, to ensure the consistency of XML documents. The safe query can be then executed by any XML system supporting the Update-XQuery language.

In [5] the authors tackle a different problem. They propose an approach to evolve a set of DTDs, representative of the documents already stored in a database, so to adapt it to the structure of new documents entering the database.

The paper is organized as follows. Section 2 introduces a running example. Section 3 presents the framework of our approach to manage DTD evolution. Section 4 describes DTD changes through a scenario. Section 5 concludes the paper.

2 Running Example

Figure 1 gives a DTD example about a musical band and a document which conforms to this DTD. We define a *component* as an element (e.g. `Band`) or a group surrounded by parentheses (e.g. `(History | Awards)`). There are three element kinds. An *element* can be empty (e.g. `Joined`), atomic (e.g. `Name`), or composite (e.g. `Band`). The `Band` element has four children (three elements and one group), i.e. there are four *parent-child relationships* involving `Band` as parent. The `Band-Member` relationship has order 3 (i.e. `Member` is the third child of `Band`) and cardinality '+' (i.e. minimum cardinality 1, maximum cardinality *n*), while the `Name-PCDATA` relationship has order 1 and cardinality '–' by default (i.e. minimum cardinality 1, maximum cardinality 1). The `Member` element has two *attributes*. The `Plays` attribute of `Member` is of type IDREF and is implied (i.e. minimum cardinality 0, maximum cardinality 1). We use the term *Band element* for the element as defined in the DTD, and the term *Band instance* for `<Band>content</Band>` in the document.

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Instrument*)>
<!ELEMENT Member (Name, Role, Joined)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREF #IMPLIED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
<!ELEMENT Instrument (Description)>
<!ATTLIST Instrument Id ID #REQUIRED>
<!ELEMENT Description (#PCDATA)>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981">
            <Name>J. Bond</Name>
            <Role>Singer</Role>
            <Joined Year="2000"/>
      </Member>
      <Member BDate="15-02-1979" Plays="G1">
            <Name>C. Kent</Name>
            <Role>Musician</Role>
            <Joined Year="2000"/>
      </Member>
      <Instrument Id="G1">
            <Description>Guitar</Description>
      </Instrument>
</Band>
```

Fig. 1. DTD example and XML document conforming to the DTD

3 DTD Evolution Framework

In this section, we present the framework of our approach to modify the DTD of XML documents. It is similar to the framework used in database schema evolution [4] [14] [9] [2] [3]. It consists in a set of DTD changes, invariants, and the semantics of DTD changes.

3.1 Set of DTD Changes

We build the set of DTD changes as follows: for each XML feature (component, parent-child relationship, attribute) we apply the create, delete, and update primitives. The set of DTD changes obtained is given in figure 2.

(1) Create a component
  (1.1) Create an empty element
  (1.2) Create a group
(2) Delete a component
(3) Update a component
  (3.1) Change its name, if element
  (3.2) Change its component kind (element ↔ group)
  (3.3) Change its element kind, if element
(4) Create a parent-child relationship
(5) Delete a parent-child relationship
(6) Update a parent-child relationship
  (6.1) Change its parent
  (6.2) Change its child
  (6.3) Change its minimum cardinality
  (6.4) Change its maximum cardinality
  (6.5) Change it to an attribute
  (6.6) Change its order
(7) Create an attribute
(8) Delete an attribute
(9) Update an attribute
  (9.1) Change its name
  (9.2) Change its element
  (9.3) Change its type
  (9.4) Change its minimum cardinality
  (9.5) Change its maximum cardinality
  (9.6) Change it to a parent-child relationship
  (9.7) Change its default value
  (9.8) Change its fixed declaration
  (9.9) Change its enumeration list, if enumerated

Fig. 2. Set of DTD changes in our approach

3.2 Invariants

XML invariants are properties that must be always satisfied, even across DTD changes. We identify the following invariants from [6]:

- An empty element has no children. An atomic element has one PCDATA child. A composite element has children which are elements or groups[1].
- No element may be declared more than once.
- The type of an attribute is CDATA, or ID, or IDREF(s), or an enumeration list.
- No attribute may be declared more than once for the same element.
- The default declaration of an attribute is either a default value, or #IMPLIED, or #REQUIRED, or #FIXED with a default value.
- No element may have more than one ID attribute.
- An ID attribute is defined either with a default value or as required.

[1] An element with mixed content is a composite element with one repeatable child which is a group. This group is a choice between PCDATA and other elements, for example <!ELEMENT Instrument ((#PCDATA | Description)*)>.

- ID values uniquely identify the elements which bear them.
- An IDREF value matches the value of some ID attribute.
- The default value of an attribute is compatible with the attribute type.

3.3 Semantics of DTD Changes

We define the semantics of each DTD change by preconditions and postactions such that the new DTD is valid (i.e. the invariants are preserved), existing documents conform to the new DTD, and data is not lost if possible. *Preconditions* are conditions that must be satisfied to allow the DTD change to occur. Otherwise, the DTD change is rejected by the XML system. *Postactions* are transformations that take place in the DTD and the documents as consequences of the DTD change. We give the semantics of DTD changes in the next section.

4 DTD Evolution Scenario

In this section, we imagine a scenario requiring the modification of the Band DTD. In each scene we, as designers, apply several DTD changes (written in italic and followed by their number in figure 2) and data changes. Data changes are add/delete/update element instances in the XML document. For DTD changes, we give the conditions that are checked by the XML system (preconditions), and the consequences on both the DTD and the XML document that are applied by the XML system (postactions).

4.1 Scene 1

The producer of the band is missing in the document, and should be added after the band members.

- We create an atomic element `Producer`: create empty element (1.1), then create Producer-PCDATA relationship with order 1 and cardinality '–' (4). Our document remains unchanged.
- We add to `Band` a child `Producer` after its child `Member`: create Band-Producer relationship with order 3.4 and cardinality '?' (4). The order parameter is either n or n.n+1. The latter case means that the relationship is added between the relationships with order n and order n+1. Creating a parent-child P-C relationship with order o has the following conditions. (i) P is an empty element, or P is a composite element and C is not PCDATA (true in our example), or P is a group. Note that P can not be atomic, it has to be first updated to composite as we will see in scene 2. (ii) P has no instances, or the cardinality of the new relationship is optional ('?', '*') (true in our example), or the new relationship is added as an alternative (i.e. its order is equal to an existing order), or P is an empty element and C is PCDATA and cardinality is '–'. (iii) The order of the new relationship is between 0 and max+1, where max is the highest order of a parent-child relationship

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Producer, Instrument*)>
<!ELEMENT Producer (#PCDATA)>
<!ELEMENT Member (Name, Role, Joined)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREF #IMPLIED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
<!ELEMENT Instrument (Description)>
<!ATTLIST Instrument Id ID #REQUIRED>
<!ELEMENT Description (#PCDATA)>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981">
            <Name>J. Bond</Name>
            <Role>Singer</Role>
            <Joined Year="2000"/>
      </Member>
      <Member BDate="15-02-1979" Plays="G1">
            <Name>C. Kent</Name>
            <Role>Musician</Role>
            <Joined Year="2000"/>
      </Member>
      <Producer>Sony</Producer>
      <Instrument Id="G1">
            <Description>Guitar</Description>
      </Instrument>
</Band>
```

Fig. 3. Updated DTD and document (scene 1)

for P (true in our example, $0 < 3.4 \leq 5$). The consequences of this change are the following. (i) If P was empty, it becomes atomic or composite according to C. (ii) If o has the form n.n+1, then the order of subsequent relationships for P (i.e. with order $\geq$ n+1) is incremented by 1, so that the new relationship gets the order n+1 (in our example, the order of `Band-Instrument` is updated to 5 and `Band-Producer` gets the order 4). If o has the form n and there is another relationship P-X with the same order, then X and C become alternatives (in our example, if we created `Band-Producer` with order 3 and cardinality '–', the `Band` element would have been `<!ELEMENT Band (Name, (History | Awards)?, (Producer | Member+), Instrument*)>`). Our document remains unchanged.

- We modify the XML document by adding Sony as producer of the band: `<Producer>Sony</Producer>`.
- We can now make the `Band-Producer` relationship mandatory: change the minimum cardinality of Band-Producer to 1 (6.3). As conditions of this change, the child is not PCDATA, and the document satisfies the new minimum cardinality

when increasing it, i.e. from optional ('?', '*') to mandatory ('–', '+') (true in our example). The updated DTD and document are given in figure 3.

4.2 Scene 2

The information about the producer is incomplete, and should include the country, for example Sony in UK.

- We change `Producer` to a composite element: *change the element Producer from atomic to composite (3.3)*. Changing an atomic element E to composite has two consequences. (i) In the DTD, an atomic element Tag1 is created, and the definition of E is replaced by Tag1 (in our example, `<!ELEMENT Producer (Tag1)>`). (ii) In the document, the content of E instances is surrounded by <Tag1> and </Tag1> (in our example, `<Producer><Tag1>Sony</Tag1></Producer>`). Note that the content of E instances is not lost.
- We rename the element `Tag1` to `Company` to make it meaningful: *change the name of element Tag1 to Company (3.1)*. Uniqueness of the new element name is checked as condition (true in our example). Renaming element E to E' has two consequences. (i) E is replaced by E' wherever it is used in the DTD (in our example, `<!ELEMENT Producer (Company)>`). (ii) In the document, the tags <E> and </E> are replaced by <E'> and </E'> (in our example, `<Producer><Company>Sony</Company></Producer>`).
- We create an atomic element `Country`, as before for `Producer`, and add it as child of `Producer`: *create Producer-Country relationship with order 2 and cardinality '?' (4)*.
- We modify the document by adding UK as the producer's country. The updated DTD and document are given in figure 4.

4.3 Scene 3

A member is identified by his/her name, and this should be reflected in the DTD.

- We make Name an attribute of Member instead of being a child of Member: change the Member-Name relationship to an attribute (6.5). Changing a P-C relationship to an attribute has two conditions. (i) P is a composite element, and C is an atomic element, and the cardinality of the relationship is single ('?', '-'), in order to have a CDATA attribute (true in our example). The attribute is implied if the relationship was optional, required otherwise. (ii) The element P does not have an attribute with the same name (true in our example). The consequences of this change are the following. (i) The order of subsequent relationships for P is decremented by 1 (in our example, the order of Member-Role and Member-Joined). (ii) In the document, <C>text</C> is removed from the content of P instances, and the start-tags <P>become <P C="text"> (in our example, see Member instances in figure 5). Note that changing the Member-Name relationship to an attribute is different from deleting this relationship and adding a Name attribute, because in this case the values of Name are lost.

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Producer, Instrument*)>
<!ELEMENT Producer (Company, Country?)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Member (Name, Role, Joined)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREF #IMPLIED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
<!ELEMENT Instrument (Description)>
<!ATTLIST Instrument Id ID #REQUIRED>
<!ELEMENT Description (#PCDATA)>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981">
            <Name>J. Bond</Name>
            <Role>Singer</Role>
            <Joined Year="2000"/>
      </Member>
      <Member BDate="15-02-1979" Plays="G1">
            <Name>C. Kent</Name>
            <Role>Musician</Role>
            <Joined Year="2000"/>
      </Member>
      <Producer>
            <Company>Sony</Company>
            <Country>UK</Country>
      </Producer>
      <Instrument Id="G1">
            <Description>Guitar</Description>
      </Instrument>
</Band>
```

Fig. 4. Updated DTD and document (scene 2)

- We make Name an ID attribute: change the type of attribute Name from CDATA to ID (9.3). Changing a CDATA attribute to an ID attribute has one condition: the values taken on the attribute are unique and do not appear on any other ID attribute (true in our example). Our document remains unchanged.

  Then we find out that there is a mistake in the DTD since a member of the band can play several instruments.
- We update the type of Plays from IDREF to IDREFS: change the maximum cardinality of attribute Plays to n (9.5). The maximum cardinality can be changed only for IDREF(s) attributes. As condition of this change, the values taken on the attribute satisfy the new maximum cardinality when decreasing it, i.e. from IDREFS to IDREF. Our document remains unchanged.

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Producer, Instrument*)>
<!ELEMENT Producer (Company, Country?)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Member (Role, Joined)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREFS #IMPLIED
                  Name ID #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
<!ELEMENT Instrument (Description)>
<!ATTLIST Instrument Id ID #REQUIRED>
<!ELEMENT Description (#PCDATA)>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981" Name="J. Bond">
            <Role>Singer</Role>
            <Joined Year="2000"/>
      </Member>
      <Member BDate="15-02-1979" Plays="G1 P2" Name="C.Kent">
            <Role>Musician</Role>
            <Joined Year="2000"/>
      </Member>
      <Producer>
            <Company>Sony</Company>
            <Country>UK</Country>
      </Producer>
      <Instrument Id="G1">
            <Description>Guitar</Description>
      </Instrument>
      <Instrument Id="P2">
            <Description>Piano</Description>
      </Instrument>
</Band>
```

Fig. 5. Updated DTD and document (scene 3)

- We modify the document by adding the piano instrument and updating the `Plays` value to reflect the fact that Kent plays also piano. The updated DTD and document are given in figure 5.

4.4 Scene 4

All the members joined the musical band at the same time, which means that there is no need to store the date for each member.

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Producer, Instrument*, Joined+)>
<!ELEMENT Producer (Company, Country?)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Member (Role)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREFS #IMPLIED
                  Name ID #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
<!ELEMENT Instrument (Description)>
<!ATTLIST Instrument Id ID #REQUIRED>
<!ELEMENT Description (#PCDATA)>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981" Name="J. Bond">
            <Role>Singer</Role>
      </Member>
      <Member BDate="15-02-1979" Plays="G1 P2" Name="C. Kent">
            <Role>Musician</Role>
      </Member>
      <Producer>
            <Company>Sony</Company>
            <Country>UK</Country>
      </Producer>
      <Instrument Id="G1">
            <Description>Guitar</Description>
      </Instrument>
      <Instrument Id="P2">
            <Description>Piano</Description>
      </Instrument>
      <Joined Year="2000"/>
      <Joined Year="2000"/>
</Band>
```

Fig. 6. Updated DTD and document (scene 4)

- We make `Joined` a child of `Band` instead of `Member`: *change the parent in the Member-Joined relationship to Band (6.1)*. Changing the parent in a P-C relationship to P' has one condition: there is a parent-child P'-P relationship (in our example, `Band-Member`). In other words, we can move a nested element one level up. The consequences of this change are the following. (i) The P'-C relationship takes the order max+1 (in our example, `Band-Joined` takes the order 6), and the order of subsequent relationships for P is decremented by 1. (ii) If P'-P is multi-valued and P-C was single-valued, then P'-C becomes multi-valued (in our

example, `Band-Joined` gets the cardinality '+'). (iii) If P'-P is optional and P-C was mandatory, then P'-C becomes optional. (iv) In the document, all C instances are removed from the content of P instances and added to the content of P' instances at the end (in our example, see `Member` instances and `Band` instance). The updated DTD and document are given in figure 6. Note that changing the parent of the `Member-Joined` relationship is different from deleting this relationship and adding a `Band-Joined` relationship, because in this case the values of `Joined` are lost.

- Since `Joined` instances are repeated in our document, we delete one of the occurences. Then we make the `Band-Joined` relationship single-valued: *change the maximum cardinality of Band-Joined to 1 (6.4)*. As conditions of this change, the child is not PCDATA, and the document satisfies the new maximum cardinality when decreasing it, i.e. from multiple ('*', '+') to single ('?', '–') (true in our example).

4.5 Scene 5

The information on the instruments of the band is found to be useless.

- We get rid of the `Instrument` element: *delete element Instrument (2)*. Deleting an element E has the following consequences. (i) The E-X relationships are deleted, and if X was a child of only E, then the component X is also deleted (in our example, `Instrument-Description` and element `Description`). (ii) the attributes of E are deleted (in our example, attribute `Id`). (iii) The Y-E relationships are deleted (in our example, `Band-Instrument`). (iv) In the document, E instances are deleted (in our example, the two `Instrument` instances).

  Deleting an attribute A of element E has the following consequences. (i) In the document, A="..." is removed from the start-tags <E>. (ii) If A is of type ID, and A values are referenced through an IDREF attribute B, then B="..." is removed from the corresponding start-tags (in our example, when the `Id` attribute is deleted, `Plays="G1 P2"` is removed from `<Member>`).

  Deleting a parent-child P-C relationship has the following consequences. (i) If P was atomic, it becomes empty. If P was composite and C was its only child, then P becomes empty. If P was a group and C was its only child, then the group is deleted. (ii) The order of subsequent relationships for P is decremented by 1 (in our example, when `Band-Instrument` is deleted, the order of `Band-Joined` becomes 5). (iii) In the document, if P was atomic then the content of P instances is removed, otherwise C instances are removed from the content of P instances.

  The updated DTD and document are given in figure 7.

4.6 Scene 6

The address (street and city) of the band members is missing in the document.

- We create two atomic elements `Street` and `City` as before, and add to `Member` a child which is a group composed of those elements: *create a group G (1.2), then create G-Street and G-City relationships with cardinality '–' and order 1 and 2*

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Producer, Joined)>
<!ELEMENT Producer (Company, Country?)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Member (Role)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREFS #IMPLIED
                  Name ID #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981" Name="J. Bond">
            <Role>Singer</Role>
      </Member>
      <Member BDate="15-02-1979" Name="C. Kent">
            <Role>Musician</Role>
      </Member>
      <Producer>
            <Company>Sony</Company>
            <Country>UK</Country>
      </Producer>
      <Joined Year="2000"/>
</Band>
```

Fig. 7. Updated DTD and document (scene 5)

respectively (4), then create Member-G relationship with order 2 and cardinality '?' (4). The element `Member` becomes `<!ELEMENT Member (Role, (Street, City)?)>`.

- We add the address of Bond to our XML document: `<Member BDate="25-05-1981" Name="J. Bond"><Role>Singer</Role><Street>Oxford street</ Street><City>London</City></Member>`.

  Few months later, the address of the producer is needed, and should be also sstored.
- Instead of adding another child `(Street, City)` to the `Producer` element, we change the group `(Street, City)` in `Member` to an element, and use it in both `Member` and `Producer`: *change the group (Street, City) to an element (3.2), then rename this element to Address (3.1), then add Producer-Address relationship with order 3 and cardinality '?' (4)*. Changing a group to an element has two consequences. (i) It adds the element declaration to the DTD (in our example, see element `Address`). (ii) In the document, it adds tags around the group content (in our example, see `Member` instance in fig. 8). This DTD change is useful since it

```
<!ELEMENT Band (Name, (History | Awards)?, Member+,
                Producer, Joined)>
<!ELEMENT Producer (Company, Country?, Address?)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Member (Role, Address?)>
<!ATTLIST Member  BDate CDATA #REQUIRED
                  Plays IDREFS #IMPLIED
                  Name ID #REQUIRED>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Role (#PCDATA)>
<!ELEMENT History (#PCDATA)>
<!ELEMENT Awards (#PCDATA)>
<!ELEMENT Joined (EMPTY)>
<!ATTLIST Joined Year CDATA #REQUIRED>
<!ELEMENT Street (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT Address (Street, City)>
```

```
<Band>
      <Name>Super Band</Name>
      <History>Founded in 1995</History>
      <Member BDate="25-05-1981" Name="J. Bond">
            <Role>Singer</Role>
            <Address>
                  <Street>Oxford street</Street>
                  <City>London</City>
            </Adress>
      </Member>
      <Member BDate="15-02-1979" Name="C. Kent">
            <Role>Musician</Role>
      </Member>
      <Producer>
            <Company>Sony</Company>
            <Country>UK</Country>
            <Address>
                  <Street>Knightsbridge street</Street>
                  <City>London</City>
            </Adress>
      </Producer>
      <Joined Year="2000"/>
</Band>
```

Fig. 8. Updated DTD and document (scene 6)

allows to change a group to an element and to share it as child of other elements. Note that changing the group `(Street, City)` to an `Address` element is different from deleting the `Member-group` relationship and adding another `Member-Address` relationship because in this case the `Street` and `City` values are lost.

- We modify the document by adding the producer's address. The updated DTD and document are given in figure 8.

5 Conclusion

In this paper, we addressed the issue of XML document schema evolution. Schema changes can not be avoided. Consequently, there is a need of XML systems that support them. We proposed an approach to manage DTD evolution. It supports 25 DTD changes, and defines their semantics by preconditions and postactions such that the new DTD is valid, existing documents conform to the new DTD, and data is not lost if possible. Our approach provides a great flexibility, since it allows to change an atomic element to a composite element, an attribute to a parent-child relationship, a group to an element, cardinalities, and order of parent-child relationships, etc., while transforming XML documents accordingly without loss of data.

We implemented our DTD evolution approach using the F2 object database system as the underlying storage system for XML documents [1] [8]. We tested it with sample DTDs and documents. Future work includes testing it in real-life applications. Although we used DTDs as schema specification language, our approach can be easily extended to XML-Schema. In this case, it will support more schema changes, since XML-Schema has a more sophisticated typing mechanism and supports more features. Another extension to our work is to support complex DTD changes, which combine several DTD changes. Future work includes also investigating XML schema integration which can be related to XML schema evolution.

References

1. Al-Jadir L., El-Moukaddem F., "F2/XML: Storing XML Documents in Object Databases", Proc. Int. Conf. on Object-Oriented Information Systems, OOIS, Montpellier, 2002.
2. Al-Jadir L., Estier T., Falquet G., Léonard M., "Evolution Features of the F2 OODBMS", Proc. Int. Conf. on Database Systems for Advanced Applications, DASFAA, Singapore, 1995.
3. Al-Jadir L., Léonard M., "Multiobjects to Ease Schema Evolution in an OODBMS", Proc. Int. Conf. on Conceptual Modeling, ER, Singapore, 1998.
4. Banerjee J., Kim W., Kim H-J., Korth H.F., "Semantics and Implementation of Schema Evolution in Object-Oriented Databases", Proc. ACM Conf. on Management Of Data, ACM SIGMOD, San Francisco, 1987.
5. Bertino E., Guerrini G., Mesiti M., Tosetto L., "Evolving a Set of DTDs according to a Dynamic Set of XML Documents", Proc. EDBT Workshop on XML-Based Data Management, XMLDM, Prague, 2002.
6. Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (eds), "Extensible Markup Language (XML) 1.0 (2nd Edition)", W3C Recommendation, http://www.w3.org/TR/2000/REC-xml-20001006, Oct. 2000.
7. Chung T-S., Park S., Han S-Y., Kim H-J., "Extracting Object-Oriented Database Schemas from XML DTDs Using Inheritance", Proc. Int. Conf. on Electronic Commerce and Web Technologies, EC-Web, Munich, 2001.
8. El-Moukaddem F., "Managing XML Document Schema Evolution", Master's thesis, American University of Beirut, Beirut, 2002.
9. Ferrandina F., Meyer T., Zicari R., Ferran G., Madec J., "Schema and Database Evolution in the O2 Object Database System", Proc. Int. Conf. on Very Large Data Bases, VLDB, Zürich, 1995.

10. Kappel G., Kapsammer E., Rausch-Schott S., Retachitzegger W., "X-Ray - Towards Integrating XML and Relational Database Systems", Proc. Int. Conf. on Conceptual Modeling, ER, Salt Lake City, 2000.
11. Kappel G., Kapsammer E., Retschitzegger W., "XML and Relational Database Systems – A Comparison of Concepts", Proc. Int. Conf. On Internet Computing, IC, Las Vegas, 2001.
12. Klettke M., Meyer H., "XML and Object-Relational Databases - Enhancing Structural Mappings Based on Statistics", Proc. Int. Workshop on the Web and Databases, WebDB, Dallas, 2000.
13. Passi K., Lane L., Madria S., Sakamuri B.C., Mohania M., Bhowmick S., "A Model for XML Schema Integration", Proc. Int. Conf. On Electronic Commerce and Web Technologies, EC-Web, Aix-en-Provence, 2002.
14. Penney D.J., Stein J., "Class Modification in the GemStone Object-Oriented DBMS", Proc. Conf. on Object-Oriented Programming Systems, Languages and Applications, OOPSLA, Orlando, 1987.
15. Pühretmair F., Wöss W., "XML-based Integration of GIS and Heterogeneous Tourism Information", Proc. Int. Conf. On Advanced Information Systems Engineering, CAISE, Interlaken, 2001.
16. Shanmugasundaram J., Tufte K., He G., Zhang C., DeWitt D., Naughton J., "Relational Databases for querying XML Documents: Limitations and Opportunities", Proc. Int. Conf. on Very Large DataBases, VLDB, Edinburgh, 1999.
17. Su H., Kane B., Chen V., Diep C., Guan D.M., Look J., Rundensteiner E., "A Lightweight XML Constraint Check and Update Framework", Proc. ER Workshop on Evolution and Change in Data Management, ECDM, Tampere, 2002.
18. Su H., Kramer D., Chen L., Claypool K., Rundensteiner E., "XEM: Managing the Evolution of XML Documents", Proc. Int. Workshop on Research Issues in Data Engineering, RIDE, Heidelberg, 2001.
19. Wong R.K., Shui W.M., "Utilizing Multiple Bioinformatics Information Sources: An XML Database Approach", Proc. IEEE Int. Symposium on Bioinformatics and Bioengineering, BIBE, Bethesda, 2001.