F2/XML: MANAGING XML DOCUMENT SCHEMA EVOLUTION

Lina Al-Jadir, Fatmé El-Moukaddem
Department of Computer Science, American University of Beirut, P.O. Box 11-0236, Beirut, Lebanon

Keywords: XML, schema evolution, object-oriented database system

Abstract: XML has become an emerging standard for data representation and data exchange on the Web. Although XML data is self-describing, most application domains tend to use document schemas. Over a period of time, these schemas need to be modified to reflect a change in the real-world, a change in the user's requirements, mistakes or missing information in the initial design. Most of the current XML management systems do not support schema changes. In this paper, we propose the F2/XML method to manage XML document schema evolution. We consider XML documents associated with DTDs. Our method consists in three steps. First, the DTD and XML documents are stored as a database schema and a database instance respectively. Second, DTD changes are applied as schema changes on the database. Third, the updated DTD and XML documents are retrieved from the database. Our method supports a complete set of DTD changes. The semantics of each DTD change is defined 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 implemented our method in the F2 object-oriented database system.

1 INTRODUCTION

As *eXtensible Markup Language* (XML) has become an emerging standard for data representation and data exchange on the Web, it has gained attention in the database (DB) community. Recently, researchers have addressed the problem of storing XML data in databases and processing XML queries using the mature technology of database systems (Florescu and Kossmann 1999, Shanmugasundaram et al. 1999, Shimura, Yoshikawa, and Uemara 1999, Kappel et al. 2000, Klettke and Meyer 2000, Schmidt et al. 2000, Chung et al. 2001). In this paper, we go further, and use schema evolution capabilities of a database system (DBMS) in order to manage XML document schema evolution.

In many applications a schema is associated with an XML document to specify and enforce the structure of the document. 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 (Kappel, Kapsammer, and Retschitzegger 2001). 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 (Su et al. 2001). Most of the current XML management systems do not support schema changes. 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. This problem is similar to schema evolution in databases (Banerjee et al. 1987, Penney and Stein 1987, Tresch 1991, Ferrandina et al. 1995, Al-Jadir et al. 1995, Al-Jadir and Léonard 1998). Our approach is to use what has been done in database schema evolution and apply it to XML documents.

In this paper, we consider the *Document Type Definition* (DTD) as XML schema mechanism. To modify the DTD of XML documents we propose the F2/XML method which consists in three steps (see Figure 1). Step 1: the DTD is stored as an object database schema and the XML documents are stored as an object database instance. Step 2: DTD changes are applied as schema changes on the database. Step 3: the updated DTD and XML documents are retrieved from the database. Our method supports a complete set of DTD changes. The semantics of each DTD change is defined 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 implemented our method in the F2 general-purpose DBMS, but it can be

Al-Jadir L. and El-Moukaddem F. (2004).
F2/XML: MANAGING XML DOCUMENT SCHEMA EVOLUTION.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 251-258
DOI: 10.5220/0002640602510258

applied using any object or object-relational DBMS which supports schema evolution.

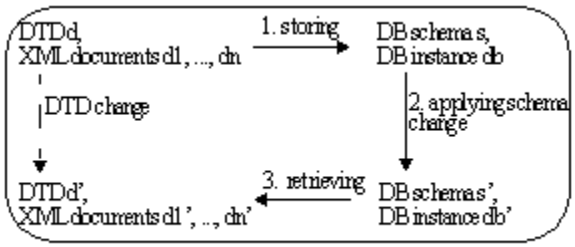

Figure 1: F2/XML method to manage DTD evolution

The paper is organized as follows. Section 2 reviews related work. Section 3 describes the storage and retrieval of XML documents in/from object databases in the F2/XML method. Section 4 presents the DTD changes supported in F2/XML with their semantics and implementation. Section 5 concludes the paper.

2 RELATED WORK

Several approaches have been proposed to store XML documents in databases. Some of them are independent of DTDs (Florescu and Kossmann 1999, Shimura, Yoshikawa, and Uemara 1999, Schmidt et al. 2000). Others map the DTD into a relational database schema (Shanmugasundaram et al. 1999, Kappel et al. 2000), an object-relational database schema (Klettke and Meyer 2000), or an object database schema (Chung et al. 2001). In (Shanmugasundaram et al. 1999) the authors propose three inlining techniques to generate a relational schema from a DTD after simplifying the DTD and building a DTD graph. In (Kappel et al. 2000) the authors propose mappings between a DTD and a relational schema according to the characteristics of XML elements and XML attributes. The mappings are not hard-coded within an application, but stored within the meta-schema. In (Klettke and Meyer 2000) the authors present some straightforward mappings to transform a DTD into an object-relational schema, and propose to use hybrid databases, i.e. databases with a data type XML, using statistics. In (Chung et al. 2001) the authors store XML data in an object-oriented database using an inlining technique, and propose to use inheritance in case of alternative and optional elements.

Our F2/XML method (Steps 1 and 3) has the following advantages. It uses the object model which allows us to represent parent-child relationships by direct references (no need to create manually join database attributes and foreign keys as in relational DB), and repetition of children by multi-valued database attributes (no need to create separate relations as in relational DB). Note that we use the term *database attributes* for attributes of a class/relation, and *XML attributes* for attributes of an element. Our method stores alternatives among children. The other approaches (except Kappel et al. 2000) either do not store the DTD, or remove alternatives by simplifying the DTD, or use inheritance (which may lead to an explosion of the number of subclasses due to all combinations). Our method stores the order among children, which is missing in some approaches. It keeps track of groups, while the other approaches (except Kappel et al. 2000) do not. Moreover, it allows us to go backward, i.e. to retrieve back the DTD from the database without loss of information, which is not possible in the other approaches (except Kappel et al. 2000).

XEM (Su et al. 2001) is an approach which handles DTD evolution. It supports 14 DTD changes, the semantics of which is given by preconditions and results to ensure the validity of the new DTD and the conformity of XML documents. Our F2/XML method differs from XEM. It supports more DTD changes (see §4.1), such as changing the parent or child in a parent-child relationship, changing a parent-child relationship to an XML 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. Moreover, our semantics of the same DTD changes is different. For example, when changing the cardinality of a child C in the definition of element E from repeatable to non-repeatable, XEM removes all occurences of the child C except the first, while F2/XML rejects this DTD change if an instance of element E has more than one occurence of child C in the document. Avoiding data loss in the XML document when changing its DTD is a major concern in our method. 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). Finally, F2/XML (unlike XEM) implements DTD changes as database schema changes performed by primitive and triggered methods.

Our approach, like XEM, is tightly-coupled with a database system. SAXE (Su et al. 2002) is a loosely-coupled approach for XML-Schema evolution. 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 (Bertino et al. 2002) 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.

3 STORAGE AND RETRIEVAL OF XML DOCUMENTS

In this section we present Steps 1 and 3 of the F2/XML method. The former stores an XML document in an object database, while the latter retrieves it from the database (Al-Jadir and El-Moukaddem 2002).

3.1 Running Example

Figure 2 gives a DTD example about a musical band, and Figure 3 gives a document which conforms to this DTD.

```
<!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 ((#PCDATA |
                       Description)*)>
<!ATTLIST Instrument Id ID #REQUIRED>
<!ELEMENT Description (#PCDATA)>
```

Figure 2: Band DTD

3.2 Extending the Meta-Schema

Since the F2 DBMS supports uniformity of objects (i.e. database objects, schema objects, and meta-schema objects are stored, accessed, and manipulated in the same way) (Al-Jadir et al. 1995), its meta-schema is accessible and can be easily extended. To store DTDs and XML documents in F2 databases, we add the following classes to the F2 meta-schema (see Figure 4).

We add the class XMLComponent as a subclass of TupleClass. It inherits the attribute className of CLASS, and has the compKind attribute (component

```
<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">Guitar
  </Instrument>
</Band>
```

Figure 3: Band XML document that conforms to the Band DTD

kind is “element” or “group”). It has 2 subclasses: XMLElement (to store elements) and XMLGroup (to store groups). XMLElement has the elemKind attribute (element kind is “empty”, “atomic”, or “composite” as in (Kappel, Kapsammer, and Retschitzegger 2001)) and is specialized into XMLEmpty (to store empty elements, e.g. Joined), XMLAtomic (to store atomic elements, e.g. Name), and XMLComposite (to store composite elements, e.g. Band). XMLGroup has a boolean attribute isMixed (e.g. (#PCDATA | Description) is a mixed group).

We also add the classes XMLRelationship (to store parent-child relationships, e.g. Band-Name, Name-PCDATA) and XMLAttribute (to store element attributes, e.g. BDate) as subclasses of ATTRIBUTE. Thus they inherit the attributes: attributeName, originClass, domainClass, minCard, maxCard and attKind (attribute kind is “DBAttribute”, “XMLRelationship”, or “XMLAttribute”). XMLRelationship has an additional integer attribute order. XMLAttribute has three additional attributes: attType (CDATA, enum, ID, or IDREF), defaultValue, and isFixed (boolean).

3.3 Storing XML Documents in Object Databases

The F2/XML method, in Step 1, stores a DTD as a database schema and XML documents as a database instance.

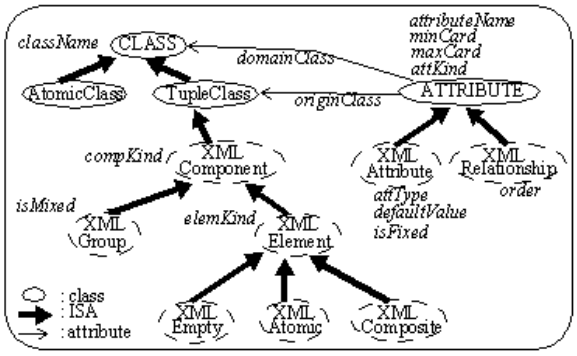

Figure 4: Extending the F2 meta-schema (added classes are dashed)

3.3.1 Storing an XML DTD as a Database Schema

First, our method builds a directed DTD graph as follows. Each element E in the DTD is represented as a node labeled E. If element E has a child C, this is represented by an edge from node E to node C. This edge is labeled with an integer indicating the order of the child in element E, and a cardinality (?, –, ∗, +). The simplest case is when the child is an element. If the child is a group (i.e. between parentheses), an edge links node E to an intermediate unlabeled node. To this node are added edges for the components of the group. If element E is atomic, an edge links node E to node PCDATA. Two alternate children of element E take the same order on the corresponding edges. An attribute of element E is represented by an edge from node E to node CDATA or to node XMLID (in case of ID or IDREF(S). In the last case, the edge is dashed) or to a new node (in case of enumerated attribute). This edge is labeled with the attribute name and the cardinality '?' if #IMPLIED or '–' if #REQUIRED (if the attribute is of type IDREFS, the cardinality becomes '∗' or '+' respectively). The DTD graph corresponding to the Band DTD is shown in Figure 5.

Second, our method maps the DTD graph into a database schema in a straightforward way. Each node labeled N is mapped into a class (object in XMLEmpty or XMLAtomic or XMLComposite) named N. Unlabeled nodes are mapped into classes (objects in XMLGroup) named group1, group2, etc. Dashed nodes are mapped into predefined classes. Each edge labeled with order o, from node A to node B, is mapped into an attribute (object in XMLRelationship) of class A, having domain class B, named B and taking order o. Similarly, each edge labeled with name l, from node A to node B, is mapped into an attribute (object in XMLAttribute) of class A, having domain class B, and named l. The edge cardinality labels are mapped into minimal and maximal cardinalities of the DB attribute, i.e. '?' into (0,1), '–' into (1,1), '∗' into (0,m) and '+' into (1,m). The value of m is set by default to 10, and can be changed later by a schema change (Al-Jadir et al. 1995). Note that a multi-valued DB attribute is implemented as list-of to maintain the order among values.

3.3.2 Storing an XML Document as a Database Instance

At this stage, the document's DTD is stored in a database. Our method parses the XML document to get its tree representation. Starting from the root node R, it retrieves the children of node R and stores their names in a string S1. It queries the meta-schema to get the attributes of the class corresponding to node R, and forms a regular expression S2 (S2 corresponds to the definition of the root element in the DTD). It applies then regular expression match between S1 and S2, and creates objects in the corresponding classes in the database. This process is applied recursively. The database

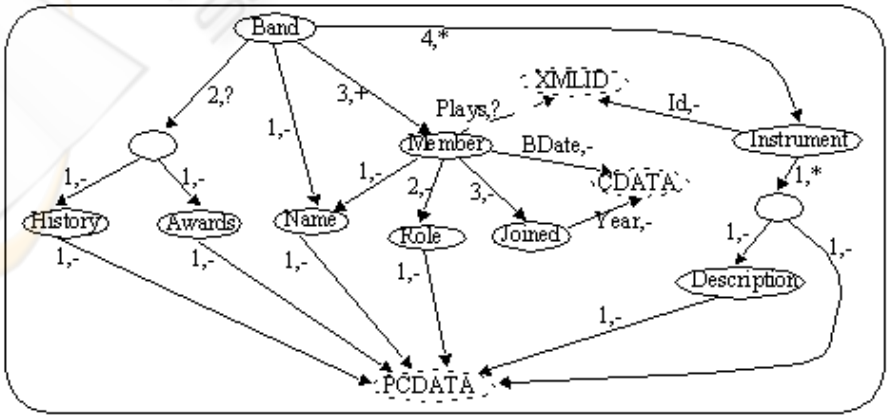

Figure 5: DTD graph for the Band DTD

instance corresponding to the Band XML document is shown in Figure 6. Note that classes PCDATA and CDATA are atomic classes and contain string atomic objects.

3.4 Retrieving XML Documents from Object Databases

The F2/XML method, in Step 3, retrieves the DTD and the documents stored in the database. By querying the meta-schema, it retrieves the documents' DTD easily. To retrieve an entire document, it finds first the object representing its root instance (this information is recorded when the document is stored). Then it navigates through the database by querying the meta-schema and following this object's attribute values. Although the document is fragmented, the object model allows easy navigation (instead of joins as in relational DB) to reconstruct the document.

We tested the storage and retrieval of XML documents in/from F2 databases with Shakespeare's plays and DBLP bibliography (Al-Jadir and El-Moukaddem 2002).

4 DTD EVOLUTION

In this section we present Step 2 of the F2/XML method. We build a complete set of DTD changes. We list the invariants that must be preserved across DTD changes. We define then the semantics of DTD changes. This framework is similar to the one used in DB schema evolution (Banerjee et al. 1987, Penney and Stein 1987, Tresch 1991, Ferrandina et al. 1995, Al-Jadir et al. 1995). Then we implement DTD changes as database schema changes.

4.1 Set of DTD Changes

Which DTD changes to support? To answer this question, we look at the XML part of the F2 meta-schema (Figure 4) since it reflects the XML model. For each of its classes, we apply the primitive methods create, delete, and update on its objects. We define the set of DTD changes thus built as *complete*, since it includes all the possible "atomic" DTD changes. The set of DTD changes supported in the F2/XML method is shown in Figure 7. Note that we omit two changes because they emerge from the F2 implementation level and not from the XML context: change the isMixed value of a group, and update the name of a parent-child relationship. Our definition of completeness of a set of DTD changes is different from XEM's definition. The latter (set of operations that allows to transform any DTD d into any DTD d') in (Su et al. 2001) does not take into account the data. Indeed, reducing a DTD d to an empty DTD and then building the new DTD d' will have as consequence the loss of data in the XML documents (deleting all elements in the DTD deletes all element instances in the documents).

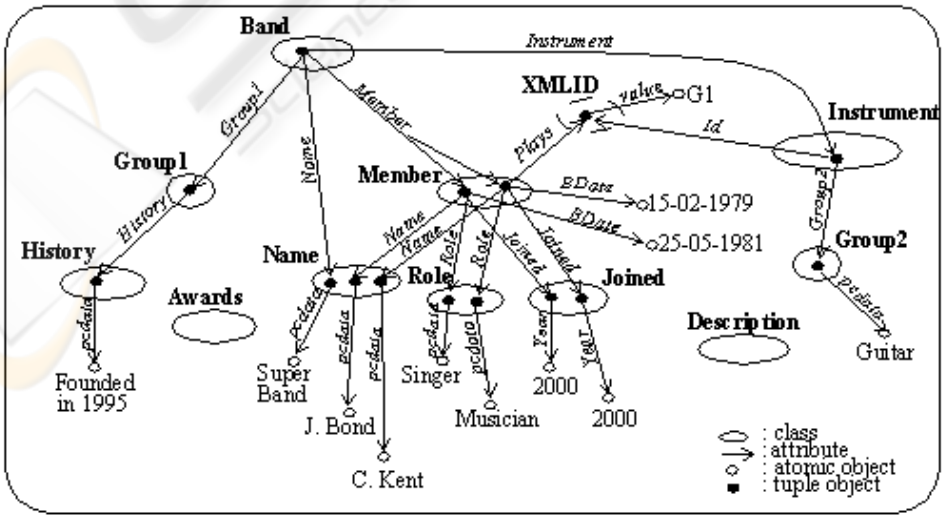

Figure 6: Database instance corresponding to the Band XML document

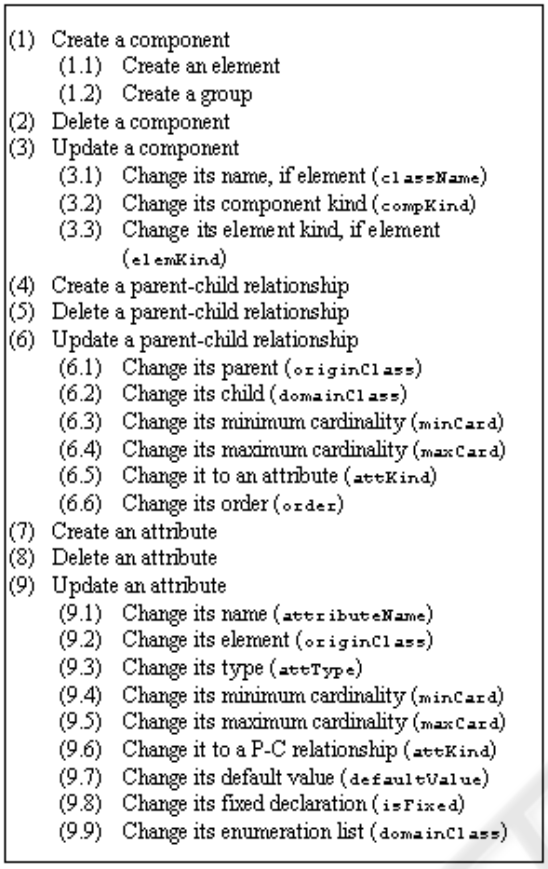

Figure 7: Set of DTD changes in F2/XML

4.2 XML Invariants

XML invariants are properties that must always be satisfied, even across DTD changes. We identify the following invariants from (Bray et al. 2000):

- An empty element has no children. An atomic element has one PCDATA child. A composite element has children which are elements or groups. Note that an element with mixed content (e.g. Instrument) is a composite element with one repeatable child which is a group. This group is a choice between PCDATA and other elements.
- No element may be declared more than once.
- The type of an attribute is either 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.
- 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.

4.3 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. XML invariants are preserved), existing XML 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 system. Postactions are actions that take place as consequences of the DTD change. They are applied by the system on the DTD and on the documents.

As an example, we give the semantics of changing the parent in a parent-child relationship (DTD change 6.1 in Figure 7). The parameters of this DTD change are the P-C relationship and the new parent P'.

Preconditions:

- there exists a parent-child P'-P relationship. In other words, we can move a nested element only one level up.

Postactions on the DTD:

- the new P'-C relationship takes the order max+1 (max is the highest order of a relationship for P'), and the order of subsequent relationships for P is decremented by 1.
- if the P'-P relationship is multi-valued ('+' or '*') and P-C was single-valued ('–' or '?'), then P'-C becomes multi-valued.
- if the P'-P relationship is optional ('?' or '*') and P-C was mandatory ('–' or '+'), then P'-C becomes optional.

Postactions on the document:

- all C instances are removed from the content of P instances and added to the content of P' instances at the end.

We illustrate this DTD change with our Band example. We find out that all members joined the musical band in the same year. Thus there is no need to store the year for each member. Consequently, we make Joined a child of Band instead of Member, i.e. we change the parent in the Member-Joined relationship to Band. The precondition of this DTD change is satisfied since there is a Band-Member relationship. As postactions on the DTD, the Band-Joined relationship takes the order 5, and gets the

cardinality '+' (see Figure 8). As postactions on the document, the Joined instances are removed from the Member instances and added to the Band instance (see Figure 9). It is up to the user to keep the duplication (two Joined instances), or remove it and then change the cardinality of the Band-Joined relationship to '–' (DTD change 6.4 in Figure 7). Note that changing the parent in the Member-Joined relationship is different from deleting this relationship and adding a Band-Joined relationship, because in this case the content and attribute values of Joined would be lost.

```
<!ELEMENT Band (Name, (History |
       Awards)?, Member+, Instrument*,
       Joined+)>
<!ELEMENT Member (Name, Role)>
...
```

Figure 8: Band DTD after changing the parent in the Member-Joined relationship to Band

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

Figure 9: Band XML document after changing the parent in the Member-Joined relationship to Band

Other examples of DTD changes can be found in (Al-Jadir and El-Moukaddem 2003).

4.4 Implementation of DTD Changes

Let us recall that modifying the DTD of XML documents is done in F2/XML by storing the DTD and documents in a database, modifying the database schema, and retrieving the updated DTD and documents from the database. To each DTD change corresponds a DB schema change (by construction of the set of DTD changes, in §4.1). A DB schema change is implemented in the F2 DBMS by a primitive method (create, delete, update) and triggered methods (Al-Jadir et al. 1995). The triggered methods implement the semantics of DTD changes (defined in §4.3). We wrote the triggered methods for all our DTD changes.

As an example, we give the implementation of changing the parent in a parent-child relationship. A parent-child relationship is stored as a database attribute (object in the meta-class XMLRelationship which is a subclass of ATTRIBUTE, in §3.2). Modifying the origin class of this attribute corresponds to modifying the parent in the parent-child relationship. This schema change is implemented by the primitive method update (which is the same for all objects), and a triggered method (for the event before-update of the attribute originClass of the class XMLRelationship) to check the precondition, and four triggered methods (for the event after-update of the attribute originClass of the class XMLRelationship) to apply the postactions on the DB schema and DB instance (see semantics in §4.3). In our Band example, modifying the origin class of the attribute Joined, from class Member to class Band, results in the database shown in Figure 10. From this updated database are retrieved the updated DTD and document shown in Figures 8 and 9.

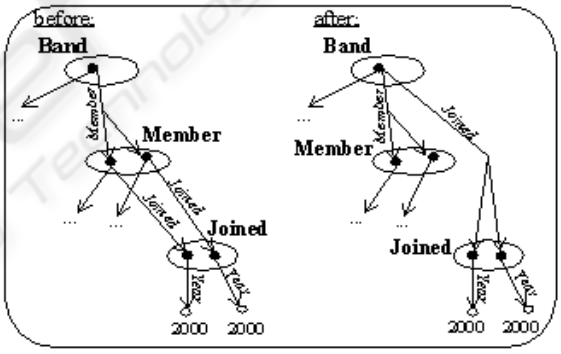

Figure 10: Updated database after changing the parent in the Member-Joined relationship to Band

5 CONCLUSION

In this paper, we addressed the issue of XML document schema evolution. We proposed and implemented the F2/XML method to handle it. Our method is based on the similarity with database schema evolution. It stores XML documents with their DTD in an object database, applies DTD changes as DB schema changes, and then retrieves the updated DTD and documents from the database. Our method supports 25 DTD changes, which form a complete set of DTD changes according to our definition of completeness. The semantics of each

DTD change is defined 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. To each DTD change corresponds a DB schema change which is implemented by a primitive method and triggered methods in the F2 DBMS.

Although we used DTDs as schema specification language, our method can be easily extended to XML-Schema. In this case, it will support more changes, since XML-Schema has a more sophisticated typing mechanism and supports more features. Future work includes testing our method in real-life applications. Querying and manipulating XML documents stored in databases are other important issues that we need to address. Also performance issues deserve to be studied. Research efforts have been put recently to benchmark XML databases (Schmidt et al. 2001).

REFERENCES

Al-Jadir L., El-Moukaddem F., 2002. F2/XML: Storing XML Documents in Object Databases. *Proc. Int. Conf. on Object-Oriented Information Systems*, OOIS'02, Montpellier, France.

Al-Jadir L., El-Moukaddem F., 2003. Once Upon a Time a DTD Evolved into Another DTD. *Proc. Int. Conf. on Object-Oriented Information Systems*, OOIS'03, Geneva, Switzerland.

Al-Jadir L., Estier T., Falquet G., Léonard M., 1995. Evolution Features of the F2 OODBMS. *Proc. Int. Conf. on Database Systems for Advanced Applications*, DASFAA'95, Singapore.

Al-Jadir L., Léonard M., 1998. Multiobjects to Ease Schema Evolution in an OODBMS. *Proc. Int. Conf. on Conceptual Modeling*, ER'98, Singapore.

Banerjee J., Kim W., Kim H-J., Korth H.F., 1987. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *Proc. ACM Conf. on Management Of Data*, SIGMOD'87, San Francisco, USA.

Bertino E., Guerrini G., Mesiti M., Tosetto L., 2002. Evolving a Set of DTDs according to a Dynamic Set of XML Documents. *Proc. EDBT Workshop on XML-Based Data Management*, XMLDM'02, Prague, Czech Republic.

Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (eds), 2000. Extensible Markup Language (XML) 1.0 (2nd Edition). *W3C Recommendation*, http://www.w3.org/TR/2000/REC-xml-20001006, Oct. 2000.

Chung T-S., Park S., Han S-Y., Kim H-J., 2001. Extracting Object-Oriented Database Schemas from XML DTDs Using Inheritance. *Proc. Int. Conf. on Electronic Commerce and Web Technologies*, EC-Web'01, Munich, Germany.

Ferrandina F., Meyer T., Zicari R., Ferran G., Madec J., 1995. Schema and Database Evolution in the O2 Object Database System. *Proc. Int. Conf. on Very Large Data Bases*, VLDB'95, Zürich, Switzerland.

Florescu D., Kossmann D., 1999. Storing and Querying XML Data Using an RDBMS. *IEEE Data Eng. Bulletin*, vol. 22, no 3, pp. 27-34.

Kappel G., Kapsammer E., Rausch-Schott S., Retchitzegger W., 2000. X-Ray - Towards Integrating XML and Relational Database Systems. *Proc. Int. Conf. on Conceptual Modeling*, ER'00, Salt Lake City, USA.

Kappel G., Kapsammer E., Retschitzegger W., 2001. XML and Relational Database Systems – A Comparison of Concepts. *Proc. Int. Conf. On Internet Computing*, IC'01, Las Vegas, USA.

Klettke M., Meyer H., 2000. XML and Object-Relational Databases - Enhancing Structural Mappings Based on Statistics. *Proc. Int. Workshop on the Web and Databases*, WebDB'00, Dallas, USA.

Penney D.J., Stein J., 1987. Class Modification in the GemStone Object-Oriented DBMS. *Proc. Conf. on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA'87, Orlando, USA.

Schmidt A., Kersten M., Windhouwer M., Waas F., 2000. Efficient Relational Storage and Retrieval of XML Documents. *Proc. Int. Workshop on the Web and Databases*, WebDB'00, Dallas, USA.

Schmidt A., Waas F., Kersten M., Florescu D., Carey M.J., Manolescu I., Busse R., 2001. Why and How to Benchmark XML Databases. *SIGMOD Record*, vol. 30, no 3.

Shanmugasundaram J., Tufte K., He G., Zhang C., DeWitt D., Naughton J., 1999. Relational Databases for querying XML Documents: Limitations and Opportunities. *Proc. Int. Conf. on Very Large DataBases*, VLDB'99, Edinburgh, UK.

Shimura T., Yoshikawa M., Uemura S., 1999. Storage and Retrieval of XML Documents using Object-Relational Databases. *Proc. Int. Conf. on Database and Expert Systems Applications*, DEXA'99, Florence, Italy.

Su H., Kane B., Chen V., Diep C., Guan D.M., Look J., Rundensteiner E., 2002. A Lightweight XML Constraint Check and Update Framework. *Proc. ER Workshop on Evolution and Change in Data Management*, ECDM'02, Tampere, Finland.

Su H., Kramer D., Chen L., Claypool K., Rundensteiner E.A., 2001. XEM: Managing the Evolution of XML Documents. *Proc. Int. Workshop on Research Issues in Data Engineering*, RIDE'01, Heidelberg, Germany.

Tresch M., 1991. A Framework for Schema Evolution by Meta Object Manipulation. *Proc. Workshop on Foundations of Models and Languages for Data and Objects*, Aigen, Austria.