A Mechanism for Data Conversion between XML and Network Data Model

Jeang-Kuo Chen*
Department of Information Management
Chaoyang University of Technology
jkchen@mail.cyut.edu.tw

Ching-Jen Liu
Department of Information Management
Chaoyang University of Technology
s9214623@mail.cyut.edu.tw

Abstract-*Recently, data exchange is essential in business communication. XML is suitable for carrying data with one-to-many relationship because of its hierarchical structure. However, there are many data with many-to-many relationship in the real world. How to use XML to include such data efficiently is very important. Many researches focus on data conversion between XML and relational data model. None studies this topic for network data model. However, some enterprises and organizations are still using databases of network data model to save data with many-to-many relationship. In this paper, we propose a mechanism to convert the data of network data model to an XML document and vice versa. A data exchange model is illustrated to demonstrate how to implement the proposed mechanism.*

Keywords: XML, Network Data Model, Data Conversion, Many-to-many Relationship

1. Introduction

Undoubtedly, data exchange takes an important role in business transaction. Some mediums such as SGML, HTML, XML, and PDF [7], [12], [13] are used to perform data exchange. The famous and popular medium is XML [1], [9], [11], [14], [15] developed by W3C in 1998. An XML document is self-contained in both structure and data. Therefore, XML is quite good for saving data with one-to-many relationship. However, data with many-to-many relationship are widespread in the real world. For example, in a university database, a course can be enrolled by many students while a student can enroll many courses. The relationship of data in a hierarchical data model is usually one-to-many. Some special data with many-to-many relationship may need to be kept in an improved hierarchical structure. This improved structure is so-called network data model [10]. Enterprises or organizations that use databases of network data model cannot exchange data via XML documents because XML is difficult to hold many-to-many relationship. Although [2] proposed a model for data exchange between hierarchical database (HDB) and XML, this model cannot be applied directly to network data model and XML. Many researches focus on data conversion between XML document and relational data model [3], [6], [8], [9]. None studies this topic for network data model.

IBM adds *the logical relationship mechanism* [6] to its famous HDB: IMS. It means that a child segment can have a logical parent segment beside the physical parent segment. A network data model can be performed by a hierarchical data model associated with the logical relationship mechanism [10]. Therefore, a hierarchical data model can be used as a network data model to hold data with many-to-many relationship. Now IBM's IMS is still being used in some organizations and enterprises. They may use IMS as network database (NDB) to save data with many-to-many relationship. It is necessary to develop techniques of data exchange via XML for these enterprises or organizations. In this paper, we propose a mechanism to convert data with many-to-many relationship between an XML document and an NDB. A data exchange model is illustrated to show how the proposed method can be applied to data exchange, via XML, between NDBs of two enterprises or organizations. In this model, two modules are implemented to transform data in an NDB into an XML document and vice versa.

2. Previous Work

2.1. XML

XML is the abbreviation of eXtensible Markup Language [16] proposed by W3C in 1998. Derived from SGML [4], XML rapidly becomes a popular and standard data exchange media. Not only a markup language, XML is also a description language for presenting both data and structure. This characteristic makes XML suitable for data exchange via network. The two advantages of XML are public and self-describing [2]. An XML document is independent of any platform; it can be shared with any user or system.

The basic structure of an XML document is defined in [16], [17], [18], and [19]. A well-formed XML document must obey the following rules. There is only one root element. Every start tag must have a corresponding end tag. The attribute value must be quoted. Elements must be nested properly. The element and attribute names are case sensitive. An example of an XML is shown in Figure 1. There are four elements <College>, <Department>, <Teacher>, and <Student> in this document. The root element is <College>. Element <Department> is the parent element of element <Teacher>. Element <Student> is the child element of element <Teacher>. The string "tid" is an attribute of element <Teacher>. To attribute "tid," the string "tid" is the attribute name while "T001" is the attribute value. In this XML document, the elements constitute a hierarchical architecture to establish a one-to-many relationship among these elements.

2.2. Logical Relationship in IMS

IMS is a database of hierarchical data model. The one (parent segment)-to-many (child segment) relationship of data is implemented in the earlier versions of IMS. In 1970, IBM enhanced the purely hierarchical approach in IMS by adding the concept of the logical relationship mechanism and induced IMS/2. With this mechanism, many-to-many relationship can be established in the data of IMS. In the hierarchical data model, each individual entity type is implemented as a *segment* [5]. The logical relationship mechanism facilitates segments interrelated from the same or different databases. If two different databases are involved, they are called *physical* and *logical databases*, respectively [6]. Three major segment types must be defined for the logical relationship mechanism. The *logical child* segment [6] is a child segment, in the logical database, of a parent segment in the physical database. The *logical parent* segment [6] is a parent segment, in the logical database, of a child segment in the physical database. The *physical parent* segment [6] is a parent segment of a child segment and both are in the same physical database.

IMS provides five pointers to implement the logical relationship mechanism. The five pointers are called *hierarchical forward* (HF), *physical parent* (PP), *logical parent* (LP), *logical child first* (LCF), and *logical twin forward* (LTF), respectively [6]. The HF pointer is used to point to the next segment in hierarchical sequence retrieval. The PP pointer is used to point to a physical parent segment. The LP pointer is used to point to a logical parent segment. The LCF pointer is used to point to the first occurrence of a logical child segment of a parent segment. The LTF pointer is used to point form a specific logical twin to the logical twin stored after it. An example of an NDB with data of many-to-many relationship is shown in Figure 2. Figure 2(a) shows the logical structure of the NDB. The physical structure of the NDB is shown in Figure 2(b). The root segment "Department" containing two instances has two child segments "Teacher" and "Project." As a bridge segment, "Participation" has "Teacher" and "Project" as its physical and logical parents, respectively. By the segment "Participation," there is a many-to-many relationship between segment "Teacher" and "Project." A teacher can participate in many projects such that Eric has two projects "internet" and "database." Likewise, a project can be performed by many teachers such that project "database" has two participators Eric and Tom.

3. Data Exchange Model

A data exchange model is illustrated to demonstrate how an enterprise (or organization) can exchange data with another enterprise (or organization). A many-to-many relationship exits in the exchanged data. The architecture of the data exchange model is shown in Figure 3. In the model, the extracted data of an NDB in the source unit is converted into an XML document by the *NtoX* module. Then, the XML document is transmitted to the destination unit via network. The *XtoN* module next converts the XML document into the original data and saves the recovered data to the NDB in the destination unit. The modules *NtoX* and *XtoN* are described as follows.

3.1. The NtoX module

The *NtoX* module is used to convert the data in an NDB to an XML document. The segment structure of NDB is shown in Figure 4. There are two portions, prefix portion and data portion, in the segment structure. The prefix portion contains *level* and *pointer* fields. The level field is used to present the level of a particular segment. The pointer fields are composed of HF, PP, LP, LCF, and LTF pointers. These pointers are used to implement the logical relationship mechanism. The data portion saves the real data.

A segment called *n* will be transformed into an element named *n*. The dependent segments of segment n will be transformed into the child elements of element *n*. The name and value of each field in a segment are transformed into the name and value of the corresponding attribute in an element, respectively. Duplicate elements may occur in an XML document in order to keep the information of many-to-many relationship of data. When the LP pointer is not null in a segment instance *i* which is a logical child in a logical relationship, the absolute path (i.e., the string from the root to the related segment instance) of the logical parent segment instance pointed by the LP pointer in the segment instance *i* must be saved as an additional attribute into the element instance corresponding to the segment instance *i*. When the LCF pointer is not null in a segment instance *i* which is a logical parent in a logical relationship, the absolute path of the logical child segment instance pointed by the LCF pointer in the segment instance *i* must be saved as an additional attribute into the element instance corresponding to the segment instance *i*. The algorithm for implementing the *NtoX* module is listed as follows.

```
Algorithm NtoX(NDB_TREE N, XML_DOC X)
// Transform the data of a network database into an
XML document. //
input: N // a pointer to a network database //
output: X // an XML document //
begin
  STACK S; // a stack for saving temporary data //
  POINTER lt_ptr; //a pointer to a logical twin
  segment instance//
  if X is empty, then write the database name as the
  start tag of the root element to X;
  else
    get the name of N as the name of an element start
    tag and write to X;
    get the names and values of fields in N as the
    names and values of attributes of the element
    corresponding to N and write to X;
  endif;
//check whether N is a logical child node or not //
  if LP pointer in N is not nil, then
    add PP_path as an attribute name to the element
    corresponding to N and get the absolute path of
    N.PP as the value of attribute PP_path, then
    write to X;
    add LP_path as an attribute name to the element
    corresponding to N and get the absolute path of
    N.LP as the value of attribute LP_path, then
    write to X;
      if LTF pointer in N is not nil, then
        add LTF_path as an attribute name to the
        element corresponding to N and get the
        absolute path of N.LTF as the value of
        attribute LTF_path, then write to X;
      endif;
  endif;
 //check whether N is a logical parent node or not //
  if LCF pointer in N is not nil, then
    add LCF_path as an attribute name to the
    element corresponding to N and get the absolute
    path of N.LCF as the value of attribute
    LCF_path, then write to X;
    assign N.LCF to lt_ptr;
      while lt_ptr is not nil, do
         get the name of lt_ptr as the name of an
         element start tag and write to X;
         get the names and values of fields in lt_ptr
         as the names and values of attributes of the
         element corresponding to lt_ptr and write to
         X;
         add PP_path as an attribute name to the
         element corresponding to lt_ptr and get the
         absolute path of lt_ptr.PP as the value of
         attribute PP_path, then write to X;
         add LP_path as an attribute name to the
         element corresponding to lt_ptr and get the
         absolute path of lt_ptr.LP as the value of
         attribute LP_path, then write to X;
         if LTF pointer in lt_ptr is not nil, then
            add LTF_path as an attribute name to the
            element corresponding to lt_ptr and get the
            absolute path of lt_ptr.LTF as the value of
            attribute LTF_path, then write to X;
         endif;
        assign lt_ptr.LTF to lt_ptr;
      end while;
  endif;
  if N.HF is not nil, then
     if N.level < N.HF.level, then push(N, S);
     else
        write the close tag of N to X;
        if N.level > N.HF.level, then
           for the number (N.level -N.HF.level) of
           top items in the stack S, do
              pop up an item i from S;
              write close tag of i to X;
           end for;
        endif;
     endif;
  else
     write the close tag of N to X;
     for all items in the stack S, do
        pop up an item i from S;
        write the close tag of i to X;
     end for;
     write the close tag of the root element to X;
     return X;
  endif;
 call NtoX(N.HF, X); // recursive call NtoX //
end NtoX.
```

3.2. The XtoN module

The *XtoN* module is used to transform an XML document into an NDB. The root element of an XML document is transformed into the database name of an NDB. Every non-root element is transformed into a segment, except for the duplicate elements. All segments derived from the XML elements are stored orderly and their names are the same as those of these XML elements. An element and its child elements are transformed into a parent segment and its child segments, respectively. All child elements of the same parent element in an XML document are stored at the same level. Except for the special four attributes PP_path, LP_path, LCF_path, and LTF_path, the attributes of an XML element are transformed into the fields of the segment derived from the same XML element. The values of the attributes PP_path, LP_path, and LTF_path are used to set the pointers PP, LP, and LTF, respectively, in a logical child segment (i.e.,“bridge” segment) of a logical relationship. The value of attribute LCF_path is used to set the pointer LCF in a logical parent segment of a logical relationship. If an element instance is a bridge instance, it occurs twice in the XML document. One of the two bridge instances should be ignored to avoid creating duplicate segment instances in the NDB. The details algorithm of the *XtoN* module is listed as follows.

```
Algorithm XtoN(XML_DOC X , NDB_TREE N)
// Transform an XML document to a network
database. //
input: X // an XML document //
output: N // a pointer to a network database //
begin
  STRING data_string; //a string variable for saving
  a tag//
  STACK S; //a stack for saving a tag as a parent
  element//
  INTEGER level_count; //initial value is -1//
  read total data of the root element of X and use the
  start tag as the database name of N;
  while X is not empty, do
    read a tag from X to data_string;
    if data_string is a start tag, then
    push(data_string, S);
      if the current element instance is a sub-element
      of the element on the top of S, then
        push(the current element, S);
        increase the value of level_count by 1;
      endif;
      if the value of attribute LP_path in the current
      element instance is not nil and the value of
      attribute PP_path in the current element is the
      same with the absolute path of the parent of
      the current element, then
        create a current segment instance, pointed
        by the HF pointer of the previous created
        segment instance, for the corresponding
        current element instance;
        copy all the attributes in current element
        instance as corresponding fields to current
        segment instance excluding the four special
        attributes PP_path, LP_path, LTF_path, and
        LTF_path;
        assign the level number to the level field of
        current segment instance by counting the
        levels of the path in PP_path of current
        element instance;
        save the values of attributes PP_path and
        LP_path to the pointers of PP and LP,
        respectively, in current segment instance;
          if the value of attribute LTF_path in
          current element instance is not nil, then
            save the value of attribute LTF_path to
            the pointer LTF in current segment
            instance;
          endif;
      else
        create a current segment instance, pointed
        by N or the HF pointer of the previous
        created segment instance, for the
        corresponding current element instance;
        copy all the attributes in current element
        instance as corresponding fields to current
        segment instance excluding the four special
        attributes PP_path, LP_path, LTF_path, and
        LTF_path;
        assign the value of level_count to the level
        field of current segment instance excluding
        the four special attributes PP_path, LP_path,
        LTF_path, and LTF_path;
          if the value of attribute LCF_path of
          current element instance is not nil, then
            save the value of LCF_path to the
            LCF field of current segment instance;
          endif;
        endif;
    else //data_string is a close tag//
        pop(S); decrease the value of level_count by
        1;
    endif;
  end while;
  return N;
end XtoN.
```

4. A Data Exchange Scenario

A scenario is given to introduce the application of the proposed mechanism between two schools. Assume one school A will share its department information to another school B. Both A and B use the NDB to manage data and agree to exchange data by XML. The NDB and the XML document are shown in Figure 2(b) and Figure 5, respectively. School A uses the *NtoX* module to create the XML document of its NDB automatically. When the XML document is transmitted to school B, the *XtoN* module is used to transform the XML document into the original NDB and the exchange work is done. Likewise, school B can transmit its NDB data to school A via XML.

5. Conclusion

XML is good for recording data with one-to-many relationship. However, it is not easy for XML to hold the information of many-to-many relationship of data. In this paper, we propose a mechanism to solve this problem by adding several attributes to each special element that is derived from a logical child/parent segment. To implement our method, we use a data exchange model for showing how data with many-to-many relationship can be transformed between an NDB and an XML document. By this model, organizations or enterprises which use NDB can share data to each other. Besides, they also can share data to enterprises or organizations that use relation databases because the XML document is a standard data medium. The proposed method not only can convert data of NDB, but also can convert data of object-oriented database or native XML database if the module *NtoX* or *XtoN* is modified for converting the data of the other data models.

References

[1] E. Bertino and E. Ferrari, "XML and data

integration," *IEEE Internet Computing,* Vol. 5, Issue: 6, pp. 75-76, 2001.

[2] J. K. Chen and M. J. Liu, "A Model for Data Exchange between XML document and Hierarchical Databases," *proceedings of the 2002 International Computer Symposium,* Dec. 18-21, Hualien, Taiwan, ROC, Vol. 5, Session 8, E8-1, 2002

[3] J. Fong, F. Pang, and C. Bloor, "Converting relational database into XML document," *Proceedings of 12th International Database and Expert Systems Applications,* 2001.

[4] E. R. Harold, *XML Bible*, John Wiley & Sons, 2001.

[5] IBM, *IMS Primer, http://www.redbooks.ibm.com*

[6] IBM, *IMS/ESA V5 Admin Guide: DB, http://www-306.ibm.com/software/data/ims/v5pdf/DFSA10C6.PDF*

[7] G. Kappel, S. Rausch-Schott, S. Reich, and W. Retschitzegger, "Hypermedia document and workflow management based on active object-oriented databases," *Proceedings of the Thirtieth Hwaii International Conference, System Sciences,* Vol. 4, pp. 377-386, 1997.

[8] J. S. Kim, W. Y. Lee, and K. H. Lee, "The cost model for XML documents in relational database systems," *ACS/IEEE International Conference, Computer Systems and Applications,* pp. 185-187, 2001.

[9] T. Kudrass, "Management of XML documents without schema in relational database systems," *Information and Software Technology,* Vol. 44, Issue: 4, pp. 269-275, 2002.

[10] T. William Olle, *The Codasyl Approach to Data Base Management*, John Wiley & Sons, 1978.

[11] J. Singh, "XML for power market data exchange," *IEEE Power Engineering Society Winter Meeting,* Vol. 2, pp. 755-756, 2001.

[12] M. B. Spring, "Reference model for data interchange standards," *Computer,* Vol. 29, Issue: 8, pp. 87-88, 1996.

[13] R. Summers, J. J. L. Chelsom, D. R. Nurse, and J. D. S. Kay, "Document management: an Intranet approach," *IEEE the 18th Annual International Conference, Engineering in Medicine and Biology Society, Bridging Disciplines for Biomedicine,* Vol. 3, pp. 1236-1237, 1997.

[14] M. Sundaram and S. S. Y. Shim, "Infrastructure for B2B exchanges with RosettaNet," *The third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems,* pp. 110-119, 2001.

[15] S. Wegener and D. Davis, "XML TPS data exchange," *IEEE AUTOTESTCON proceedings of Systems Readiness Technology Conference,* pp. 605-615, 2001.

[16] W3C, Extensible Markup Language (XML) 1.0 (second Edition) W3C Recommendation, 6 October 2000, *http://www.w3.org/TR/2000/REC-xml-20001006*, 2000.

[17] W3C, XML Schema Part 0: Primer W3C Recommendation, 2 May 2001, *http://www.w3.org/TR/2001/REC-xmlschema-0-20010502*, 2001.

[18] W3C, XML Schema Part 1: Structures W3C Recommendation, 2 May 2001,*http://www.w3.org/TR/2001/REC-xmlschema-1 -20010502*, 2001.

[19] W3C, XML Schema Part 2: Datatypes W3C Recommendation, 2 May 2001, *http://www.w3.org/TR/2001/REC-xmlschema-2-20010502*, 2001.

```
<College>
  <Department name="IM">
    <Teacher tid="T001" name="Eric">
      <Student sid="S001" name="John"/>
    </Teacher>
  </Department>
  <Department name="IE">
    <Teacher tid="T101" name="Tom">
      <Student sid="S101" name="Peter"/>
      <Student sid="S102" name="Tim"/>
    </Teacher>
  </Department>
</College>
```

Figure 1. An example of an XML.

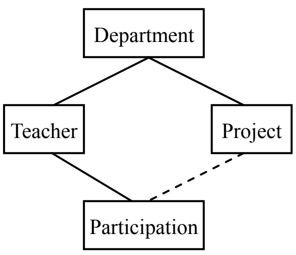

Figure 2(a) Logical structure

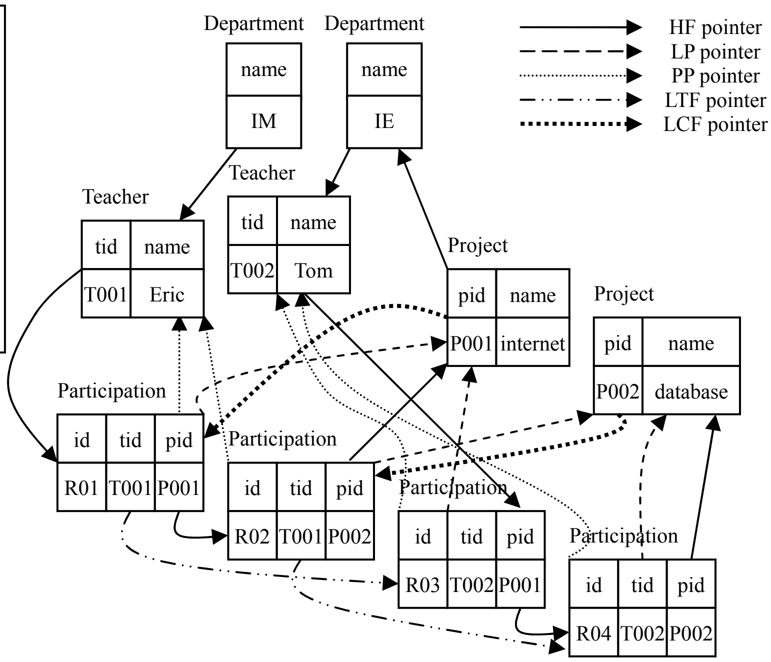

Figure 2(b) Physical structure

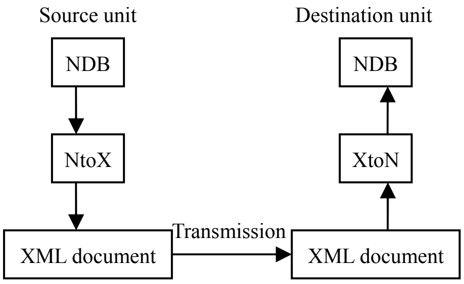

Figure 3. The architecture of the data exchange model.

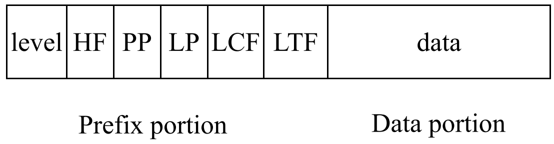

Figure 4. A segment structure of NDB.

```
<Department name=”IM” >
  <Teacher name=”Eric” tid=”T001”>
    <Participation id=”R01” tid=”T001” pid=”P001”
        PP_path=”Department/Teacher/T001”
        LP_path=”Department/Project/P001”
        LTF_path=”Department/Teacher/Participation/R03”>
    </Participation>
    <Participation id=”R02” tid=”T001” pid=”P002”
        PP_path=”Department/Teacher/T001”
        LP_path=”Department/Project/P002”
        LTF_path=”Department/Teacher/Participation/R04”>
    </Participation>
  </Teacher>
  <Project name=”internet” pid=”P001”
    LCF_path=” Department/Teacher/Participte/R01” >
    <Participation id=”R01” tid=”T001” pid=”P001”
        PP_path=”Department/Teacher/T001”
        LP_path=”Department/Project/P001”
        LTF_path=” Department/Teacher/Participation/R03”>
    </Participation>
    <Participation id=”R03” tid=”T002” pid=”P001”
        PP_path=”Department/Teacher/T002”
        LP_path=”Department/Project/P001”>
    </Participation>
  </Project>
</Department>
<Department name=” IE” >
  <Teacher name=”Tom” tid=”T002”>
    <Participation id=”R03” tid=”T002” pid=”P001”
        PP_path=”Department/Teacher/T002”
        LP_path=”Department/Project/P001”>
    </Participation>
    <Participation id=”R04” tid=”T002” pid=”P002”
        PP_path=”Department/Teacher/T002”
        LP_path=”Department/Project/P002”>
    </Participation>
  </Teacher>
  <Project name=”database” pid=”P002”
    LCF_path=” Department/Teacher/Participation/R02” >
    <Participation id=”R02” tid=”T001” pid=”P002”
        PP_path=”Department/Teacher/T001”
        LP_path=”Department/Project/P002”
        LTF_path=”Department/Teacher/Participation/R04”>
    </Participation>
    <Participation id=”R04” tid=”T002” pid=”P002”
        PP_path=”Department/Teacher/T002”
        LP_path=”Department/Project/P002”>
    </Participation>
  </Project>
</Department>
```

Figure 5. An XML document of the scenario.