On Using Collection for Aggregation and Association Relationships in XML Object-Relational Storage

Eric Pardede, J.Wenny Rahayu
Department of Computer Science and Computer Eng.
La Trobe University Bundoora VIC 3083 Australia
{ekparded, wenny}@cs.latrobe.edu.au

David Taniar
School of Business Systems
Monash University Clayton VIC 3800 Australia
David.Taniar@infotech.monash.edu.au

ABSTRACT

XML data can be stored in different databases including Object-Relational Database (ORDB). Using ORDB, we get the benefit of the relational maturity and the richness of OO modeling. One modeling concept that can be captured is the collection. Collection structures frequently occur in XML documents especially in two relationship types: aggregation and association. However, very often when the data is stored in a database repository, the collection is flattened. We believe that preserving the collection semantics in the logical and the implementation level will create a better solution.

In this paper we propose methods to preserve the collection in XML data into ORDB using the concept of collection types. We use the Semantic Network diagram to represent the collection of the aggregation and the association in XML data. Each of these relationship types will then be transformed into storage in an ORDB environment. For aggregation, we propose different methods based on the hierarchy constraint. For association, our method is differentiated based on the cardinality.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design – *data models*

General Terms

Management, Design, Theory

Keywords

XML, XML schema, ORDB, collection

1. INTRODUCTION

XML (eXtensible Markup Language) is a document description metalanguage that is used to represent large-scale data and documents in the World Wide Web [11]. For that reason alone, XML requires efficient database storage to keep its data.

Object-Relational Database (ORDB) is increasingly popular as the database storage for XML Data [6]. Its popularity relates to its ability to capture the object-oriented modeling semantic and the maturity of the relational implementation.

Many works have been proposed to map the Data Definition Language (DTD) and the XML Schema into the Object-Relational (OR) Schema [6, 7, 12, 13]. These works have tried to capture different data structures and relationships that are exist in XML Documents.

One of the structures frequently found in XML Data is the collection structure. The existing works however, have mapped the collections into flat implementation model. These practices have diminished the conceptual level semantic. In addition, the transformations have not utilized collection type in ORDB [8, 9].

These reasons have motivated us to propose different methods of preserving collection in XML Data into the XML Schema and the ORDB. We believe that preserving the conceptual semantic in the logical and the implementation level will create a better solution.

Collection in XML Data can appear in two different relationships: *association* and *aggregation*. While association is defined as a "reference" relationship between one to another object in a system, aggregation is a tightly coupled form of association [10]. Aggregation can be defined as a relationship in which a composite object ("whole") consists of other component objects ("parts") [10].

It is the aim of this paper to propose models that can preserve collection for aggregation and association relationships in XML into ORDB. We perform two mapping steps. First is the mapping from the conceptual model using Semantic Network Diagram to the logical model using XML Schema. We extend the algorithm in [3] by proposing different mapping methods for association and aggregation hierarchy. Second, the logical model is mapped into the physical implementation using SQL in ORDB. For this purpose we use the collection data types [8, 9].

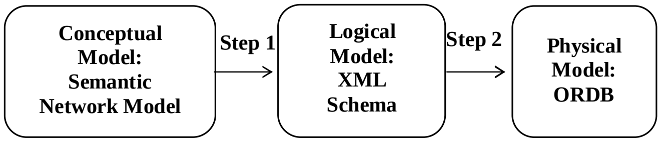

Conceptual Model: Semantic Network Model
Step 1
Logical Model: XML Schema
Step 2
Physical Model: ORDB

Figure. 1. Mapping Steps

SAC'04 March 14-17, 2004, Nicosia, Cyprus.

2. BACKGROUND

In this section we briefly show some existing works on the implementation of collection for XML Data. We also provide a brief knowledge foundation on the semantic network model before we use it in our proposed methods.

2.1 Existing Implementation in XML

Some works have tried to preserve the design modeling and to store the XML data into databases that are based on relational model including the traditional relational database and the ORDB. However, there is no work that utilizes collection data types that was introduced by SQL 1999 [5, 9] and is enriched in SQL4 [8, 9]. Very often when the XML data is stored in a database repository, the collection is not preserved. The collection in aggregation and association relationship is usually flattened or split into an entirely separate table (see Fig.2).

[4] presents a simple mapping of XML data into the relational tables. In this work, they treated XML documents as graphs with edges and leaves. The edges represent the relationships while the leaves represent the values. In this work, the collection appeared in aggregation relationship is mapped into separate flat tables by using composite keys. It has then diminished the collection semantic.

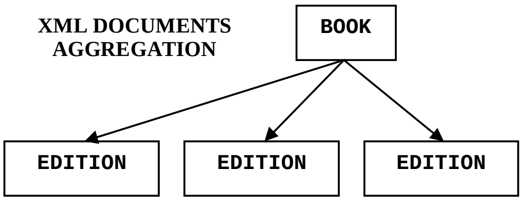
[1] proposes comprehensive mapping from the DTD into the OO Schema and the implementation of the results into tables. In the implementation stage again the aggregation type is flattened. The association relationship is mapped using IDREF. This is usually done when it is not possible to form collection or nesting.

[11] also works in the mapping of the DTD into the relational tables. They develop an algorithm and a prototype that convert the XML documents to tuples, translate the semi structured XML queries to SQL queries, and then convert the results to XML data. This work enlists the limitations of the relational database usage for the XML documents. One of those highlights the limitations of implementing collection semantic in the relational table since this database is unable to have set-valued attributes.

[6] proposes mapping from the XML Schema into the OO/OR Schema. The work compares how the mapping into relational schema can be changed into the OR schema. Thus, it does not cover the unique properties of an OR model such as different types of relationships including aggregation. Regarding association relationship, this work has mentioned the usage of collection to store the reference. Nevertheless, it does not show the step-by step mapping from conceptual level down to the implementation.

[7] proposes mapping from the DTD into the OR Schema. The main contribution of the work is the usage of a hybrid database where the users can select certain attributes to be stored in ORDB and others to be stored as they are (as XML data). It does not show the mapping for different kinds of relationship and data structures.

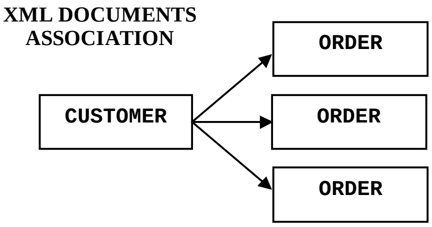
[15] proposes the mapping of the OO Conceptual Model into the XML Schema. This work has included collection for aggregation relationship. However, the usage of UML for XML Data is not complete [3]. In addition, this work does not discuss the usage of collection in association relationship.



```
<xsd:element name = “BOOK” type = “BookType”/>
<xsd:complexType name = “BookType”>...
    <xsd:element name = “EDITION” minOccurs =
    ?0? maxOccurs= ?unbounded?/>...
</xsd:complexType>
</xsd:element>
```

Book	
PK	
book_id	book_title
EN1	Fundamentals of Database System

Book-Edition		
PK, FK	PK	
book_id	ed-no	ed-year
EN1	1	1989
EN1	2	1997
EN1	3	2000



```
<xs:element name = “CUSTOMER” type = “CustomerType” / >
<xs:complexType name = “CustomerType”>
    <xs:element name = “ORDER” type= “OrderType” max
Occurs = “unbounded”/>...
</xs:complexType>

<xs:complexType name = “OrderType”>...
</xs:complexType>
```

Customer	
PK	
CustomerID	CustomerName
MR2	Mark Roberts

Order		
PK	PK	FK
OrderNo	OrderDate	CustomerID
1	25/08/2003	MR2
2	26/08/2003	MR2
3	27/08/2003	MR2

Figure. 2. Collection Flattened in Existing Method

Finally [12, 13] propose the mapping of association and aggregation relationship of XML Schema to ORDB. In [12], the collection in XML Schema is disappeared in the tables since they store the reference in the "many" side or in the separate table. In [13], the collection is preserved in the XML Schema, but again the data is stored separately in cluster tables or nested tables.

We find that the existing works either have not preserved the collection or have preserved one mapping step only. Therefore, in this work we are going to propose the complete mapping to preserve the collection in aggregation and association relationships, from the conceptual level to the implementation.

2.2 Semantic Network: An Overview

In this paper we use the Semantic Network Method [3] to model the conceptual level of the XML Documents structures. The diagram can be viewed as a richer alternative of W3C XML Data Model [14]. The Semantic Network Method is divided into the semantic level and the schema level. The former develops a specific diagram from the XML document structures while the latter maps the diagram into syntax and structure formalism such as DTD and XML Schema.

The semantic network diagram is divided into four major components: nodes, directed edges, labels, and constraints. In Figure 3, there are 5 *nodes*: A, B, X, Y, and Z. The first two are complex nodes while the rest are basic nodes. There are four *directed edges* representing the semantic relationships between the objects. We have different *labels* corresponding to each edge. For example "*p*" indicates in-property relationship and "*a*" represents aggregation relationship. Finally, there are *constraints* added in nodes or edges such as uniqueness, cardinality, adhesion, ordering, etc. This diagram can show the conceptual design of the XML documents more completely than XML data model or UML [2].

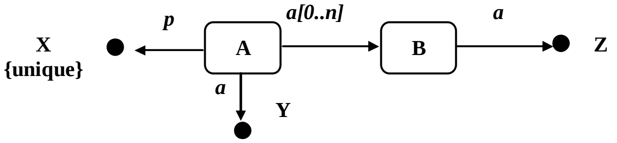


Figure. 3. Semantic Network Diagram

In the schema level the Semantic Network Method maps the four components of the diagram into XML Schema, which is mainly concerned with element/attribute declarations and simple/complex type definitions [4].

In the next section we show how we extend the schema level mapping to capture the collection semantic then we follow through to the implementation of the XML Schema in ORDB.

3. PROPOSED METHODS FOR AGGREGATION RELATIONSHIP

In this section we propose the mapping methods for aggregation relationships in the XML Data into the ORDB. The implementation will be different based on the aggregation constraints.

In the first step we map both the "whole" and the "part" components as complex types. The location of the complex types will be determined by the aggregation constraints. For the second step, we directly map the "whole" complex type as the table and the "part" complex types as the collection type attributes.

Until the time of writing, SQL4 [8, 9] have standardized two collection types, array and multiset. An *array* is a dynamically sized ordered collection that allows duplicates. A *multiset* or a *bag* is an unordered collection that allows duplicates. The collection itself can be of simple data type (such as INTEGER), constructed data type (such as ROW), or user-defined type.

3.1 Implementation of Shareable and Existence-Independent Aggregation

In the first type of aggregation, the "part" components are shareable and their existence is independent from the "whole" component. Therefore, we enable access to the "part" component without firstly accessing the "whole" component. The mapping rules are shown below.

Rule 1:

Step 1: For two types namely T_1 and T_2 with elements/attributes (A,B) and (M,N) respectively, if T_1 can be composed by collection of shareable and existence-independent T_2, implement T_1 as a complex type and T_2 as another complex type accessed as an element in T_1 with maxoccurs constraint = unbound.

Step 2: For two complex types namely T_1 (A,B) and T_2 (M,N), if T_1 can be composed by more than one shareable and existence-independent T_2, implement T_2 as a collection of UDT attribute of table T_1. Transformation result is *Table*

$$T_1\ (A,\ B,\ \overset{n}{\underset{0}{UDT}}\ \ T_2\ (M,\ N))$$

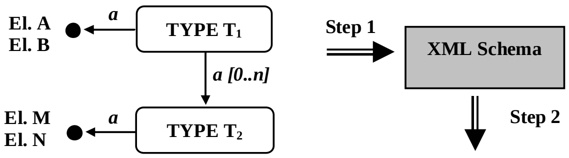


TABLE T_1			
Att A	Att B	Coll.T_2	
		Att M	Att N
Val A	Val B	Val M_1	Val N_1
		Val M_2	Val N_2

```
General Syntax
CREATE TYPE <T2 type schema>
(atti   data_type,...,
atti+m  data_type);

CREATE TABLE <T1 table schema>
(attj   data_type, ....,
 attj+n  MULTISET|ARRAY (<T2 type schema>));
```

Figure 4. Aggregation Type 1 Transformation

Example 1:

Type `STAFF` is the aggregation of type `DEPENDENT` (see Fig.5). The latter type can still exist outside type `STAFF`, probably in type `STUDENT`, etc. The aggregation type will be mapped into collections of UDT in ORDB table. Note that the horizontal line determines the ordering semantic.

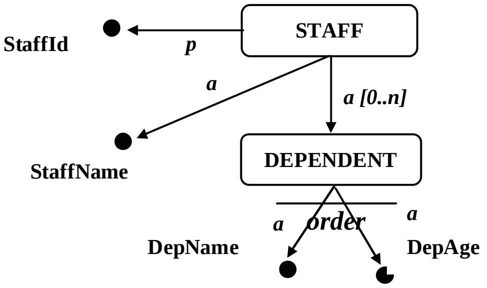


Figure 5. STAFF Aggregation Example

```
Transformation into XML Schema (Step 1):

<xsd:complexType name = ″DEPENDENT_Type″>
  <xsd:sequence>
    <xsd:element name = “DepName” type =
      “xsd:string/>
    <xsd:element name = “DepAge” type =
      “xsd:integer”/ >
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name = ″STAFF_Type″>
  <xsd:attribute name = “StaffID” type = “xsd:ID”
    use = “required”/>
  <xsd:element name = “StaffName” type =
    “xsd:string/>
  <xsd:element name = “DEPENDENT” type =
    “xsd:DEPENDENT_Type” maxOccurs= ″unbounded″/>
</xsd:complexType>
```

```
Transformation into ORDB (Step 2):

CREATE TYPE DependentType
(DepName CHARACTER VARYING(40),
 DepAge INTEGER)/

CREATE TABLE Staff
(StaffID CHARACTER VARYING(5)
 CONSTRAINT StaffID_pk PRIMARY KEY,
 StaffName CHARACTER VARYING(40),
 Dependent MULTISET (DependentType));
```

The first contribution of our method is the transformation of semantic network to XML Schema. We come up with XML Schema where both the “whole” and the “part” components are defined as complex types. Inside the “whole” complex type, we will have an element of the “part” complex type. Having done this, the “part” type can actually be used inside another “whole” complex type (shareable). To preserve the collection, we use the XML Schema syntax **`maxOccurs= ″unbounded″`**.

```
<xsd:complexType name = ″PART_Type″>
    ...
</xsd:complexType>

<xsd:complexType name = ″WHOLE_Type″>
    <xsd:element name = “PART_Name” type =
    “xsd:PART_Type” maxOccurs= ″unbounded″/>...
</xsd:complexType>
```

The second contribution is the transformation of the XML Schema to the ORDB in the form of UDT collection attribute. The mapping is straightforward. The “part” complex type is mapped as a UDT and the “whole” complex type is mapped as a table with one collection attribute formed by the “part” type. We use the SQL syntax **`TABLE(…MULTISET<ARRAY[]> (UDT_TYPE))`**.

3.2 Implementation of Non-Shareable and Existence-Dependent Aggregation

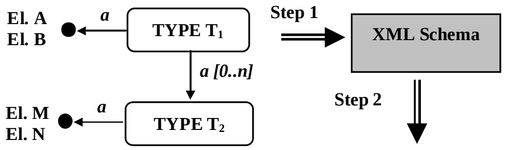
For the next aggregation type, the “part” components are non-shareable and their existence depends on the “whole” component. This type is usually called composition. We need to exclusively define the “part” component inside the “whole” component.

Rule 2:

Step 1: For two types namely T_1 and T_2 with elements/attributes (A,B) and (M,N) respectively, if T_1 can be composed by collection of non-shareable and existence-dependent T_2, implement T_1 as a complex type and T_2 as an inner complex type with maxoccurs constraint = unbound.

Step 2: For two complex types namely T_1 (A,B) and T_2 (M, N), if T_1 can be composed by collection of shareable and existence-dependent T_2, implement T_2 as a collection of ROW attribute of table T_1. Transformation result is *Table*

$$T_1\ (A, B,\ \overset{n}{\underset{0}{Row}}\ T_2\ (M, N))$$



<table>
<tr><th colspan="4">TABLE T_1</th></tr>
<tr><th>Att A</th><th>Att B</th><th colspan="2">Coll.T_2</th></tr>
<tr><td rowspan="3">Val A</td><td rowspan="3">Val B</td><td><b>Att M</b></td><td><b>Att N</b></td></tr>
<tr><td>Val M_1</td><td>Val N_1</td></tr>
<tr><td>Val M_2</td><td>Val N_2</td></tr>
<tr><td></td><td></td><td colspan="2"></td></tr>
</table>

```
General Syntax
CREATE TABLE <T1 table schema>
(atti   data_type, ....,
 atti+m  MULTISET|ARRAY ROW (att(i+m)(j) data_type, ....));
```

Figure 6. Aggregation Type 2 Transformation

Example 2:

`COURSE` is the composition of two multiple types `ASSIGNMENT` and `EXAM` (see Fig.7). The composition type will be mapped into collections of row attribute in ORDB table.

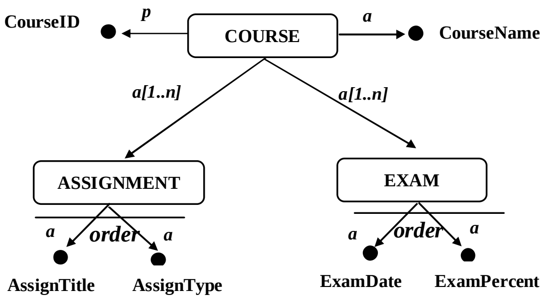


Figure 7. COURSE Composition Example

Transformation into XML Schema (Step 1):

```
<xsd:complexType name = ″COURSE_Type″>
  <xsd:attribute name = “CourseID” type = “xsd:ID”
    use = “required”/>
  <xsd:element name = “CourseName” type =
    “xsd:string/>
  <xsd:element name = “ASSIGNMENT” type =
    maxOccurs= ″unbounded″/>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name = “AssignTitle” type =
          “xsd:string/>
        <xsd:element name = “AssignType” type =
          “xsd:string”/>
      </xsd:sequence>
    </xsd:complexType>
  <xsd:element name = “EXAM” maxOccurs=
    ″unbounded″/>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name = “ExamDate” type =
          “xsd:date/>
        <xsd:element name = “ExamPercent” type =
          “xsd:integer”/ >
      </xsd:sequence>
    </xsd:complexType>
</xsd:complexType>
```

Transformation into ORDB (Step 2):

```
CREATE TABLE Course
(CourseID CHARACTER VARYING(5)
 CONSTRAINT CourseID_pk PRIMARY KEY,
 CourseName CHARACTER VARYING(40),
 Assignment MULTISET
(ROW (AssignTitle CHARACTER VARYING(40),
      AssignType CHARACTER VARYING (20))),
 Exam MULTISET (ROW (ExamDate DATE,
                     ExamPercent INTEGER)));
```

The proposed method from the conceptual to the implementation level has captured the real semantic of the composition hierarchy. In the first transformation, we come up with the XML Schema where the “part” component is defined as a complex type inside the “whole” type element. By doing this we avoid other element type to share the particular “part” component (non-shareable constraint). We also ensure that on removal of the “whole” type, we remove all “part” components that are defined inside it (existence-dependent constraint). To preserve the collection, we use the XML Schema syntax **`maxOccurs= ″unbounded″`**.

```
<xsd:complexType name = ″WHOLE_Type″>...
  <xsd:element name = ″PART_Name″
    maxOccurs=″unbounded″/>>
    <xsd:complexType>...
    </xsd:complexType>
</xsd:complexType>
```

In the second step, we map the outer complex type as the table in the ORDB and the inner complex type as the ROW attribute. To preserve the collection we implement the ROW as a collection with this syntax **`TABLE(…MULTISET<ARRAY[]>(ROW())`**.

4. PROPOSED METHODS FOR ASSOCIATION RELATIONSHIPS

In this section we propose the mapping methods of the association relationship in the XML structures into the ORDB using the collection types. As we want to accommodate the collection, we only cover the association with “many” cardinality: 1:N and N:N.

Like the previous section, the proposed method is divided into two steps. In the first step we map the conceptual model in the semantic network into the XML Schema. The associating types will become the complex types. In the second step we map the complex types as the tables and the “referential” object as the collection type attribute.

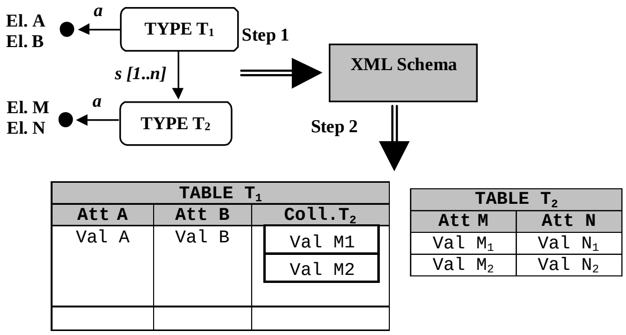
4.1 Implementation of 1:N Association

For the 1:N association relationship, the reference can be stored as a collection inside the type that has “one” side. In usual practice, the XML Schema mapping to ORDB is not straightforward because the location of reference key in both schemas is different. Our method proposes a more straightforward mapping since the reference type is always located in the type that has “one” side.

Rule 3:

Step 1: For two types namely T_1 (A,B) and T_2 (M,N), if T_1 and T_2 has 1:N association relationship, implement both as complex types with T_1 has a collection of reference to T_2

Step 2: For two complex types namely T_1 (A,B) and T_2 (M,N), if T_1 holds collection of reference to T_2, implement both as tables with T_1 has a collection attribute refer to T_2. Transformation result is *Table* T_1 $(A, B, T_2\overset{n}{\underset{1}{\_}}Key)$ and *Table* T_2 (M, N)



General Syntax

```
CREATE TABLE <T_2 table schema>
(att_i   data_type, ....,
 att_i+m data_type);

CREATE TABLE <T_1 table schema>
(att_j   data_type, ....,
 att_j+n MULTISET|ARRAY (T_2 Key data_type));
```

Figure 8. 1:N Association Transformation

Example 3:
Type `FACULTY` has an association relationship with type `BUILDING` (see Fig.9). The reference element/attribute from “one” to “many” side type will be mapped as collections attribute in ORDB table.

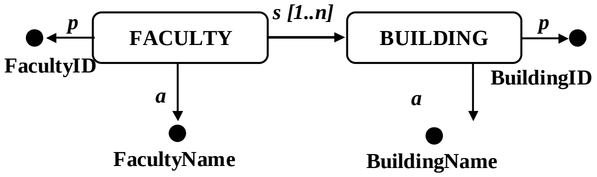


Figure 9. FACULTY-BUILDING Association Example

```
Transformation into XML Schema (Step 1):

<xsd:complexType name = ″BUILDING_Type″>
  <xsd:attribute name = “BuildingID” type =
    “xsd:ID” use = “required”/>
  <xsd:element name = “BuildingName” type =
    “xsd:string/>
</xsd:complexType>

<xsd:complexType name = ″FACULTY_Type″>
  <xsd:attribute name = “FacultyID” type = “xsd:ID
    use = “required”/>
  <xsd:element name = “FacultyName” type =
    “xsd:string/>
  <xsd:attribute name = “BuildingID” type =
    “xsd:string” maxOccurs= ″unbounded″/>
</xsd:complexType>

<keyref name=’BUILDING_BuildingID_Ref”
    refer=”BUILDING_BuildingID”>
    <selector xpath = “FACULTY”>
    <field xpath=”@BuildingID”/></keyref>

<key name=’FACULTY_FacultyID”>
    <selector xpath = “FACULTY”>
    <field xpath=”@FacultyID”/></key>
```

```
Transformation into ORDB (Step 2):

CREATE TABLE Building
(BuildingID CHARACTER VARYING(5)
 CONSTRAINT BuildingID_pk PRIMARY KEY,
 BuildingName CHARACTER VARYING(40));

CREATE TABLE Faculty
(FacultyID CHARACTER VARYING(5)
 CONSTRAINT FacultyID_pk PRIMARY KEY,
 FacultyName CHARACTER VARYING(50),
 BuildingID MULTISET(CHARACTER VARYING(5)));
```

In our method, we utilize the collection to store the relationship between two types. For the first transformation we come up with two complex types. In the “one” complex type we will have collection of element of the “many” complex type. To preserve the collection, we use XML Schema syntax **`maxOccurs=″unbounded″`**. And to maintain the relationship, we use **`key`** and **`keyref`** instead of **`ID`** and **`IDREF`**. Using the formers we enable one to specify scope within which uniqueness applies.

```
<xsd:complexType name = ″ONE_Type″>
    ...
</xsd:complexType>

<xsd:complexType name = ″MANY_Type″>
  <xsd:attribute name = “ONE_Key” ... maxOccurs=
    ″unbounded″/>...
</xsd:complexType>

<key name=”ONE_Key”>
    <selector xpath = “ONE_Type”>...</key>

<keyref name=”ONE_Key_Ref” refer=”ONE_Key”>
    <selector xpath = “MANY_Type”>...</keyref>
```

In the second step, we map the complex types as the tables in the ORDB. In the “many” type we have collection attribute consist of attribute key from the “one” type. If we have a single key we will use a collection of simple data type. Otherwise, we will have a collection of ROW type. To preserve the collection the implementation syntax is **`TABLE(…MULTISET<ARRAY[]>(SIMPLE_TYPE|ROW())`**.

We have one shortcoming of using collection in the second step of association relationship. At present, we cannot use current SQL to embed integrity constraint checking in ORDB. As we know, in traditional methods we can include the foreign key or REF and then define the integrity constraint checking such as ON DELETE CASCADE, ON UPDATE NULLIFY, etc. We still cannot apply this for collection attribute. Nevertheless, it does not mean we cannot have integrity constraint checking for our methods. Triggers and embedded routines are available in ORDB to enforce this task.

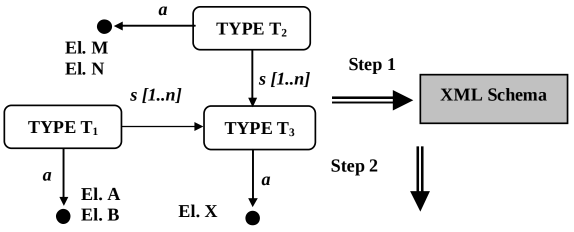
4.2 Implementation of N:N Association

In the N:N association, the reference can be stored as collection inside one of the associated type (see Fig.10). Our method proposes different way of implementing N:N association because we do not require to store the relationship in a separate table.

Rule 4:
Step 1: For two types T_1 (A,B) and T_2 (M,N) have N:N association relationship in T_3 (X), implement both as

complex types with T_1 has a collection of T_3 and reference to T_2.

Step 2: For two complex types namely T_1 (A,B) and T_2 (M,N) have N:N relationship, if T_1 has the collection of the relationship and the reference to T_2, implement both as tables with T_1 has collection ROW attribute. Transformation results are *Table* T_1 *(A, B,* $\overset{n}{\underset{1}{Row}}$ *(*T_2 *Key,* T_3*(X)))* and *Table* T_2 *(M, N)*



<table>
<tr><th colspan="4">TABLE T₁</th></tr>
<tr><th>Att A</th><th>Att B</th><th colspan="2">Coll. T₂ Key and T₃</th></tr>
<tr><td>Val A</td><td>Val B</td><td>T2 Key</td><td>T3 Att X</td></tr>
<tr><td></td><td></td><td>Val M₁</td><td>Val X₁</td></tr>
<tr><td></td><td></td><td>Val M₂</td><td>Val X₂</td></tr>
<tr><td></td><td></td><td colspan="2"></td></tr>
</table>

TABLE T_2	
Att M	**Att N**
Val M_1	Val N_1
Val M_2	Val N_2

```
General Syntax
CREATE TABLE <T2 table schema>
(atti   data_type, ....,
 atti+m  data_type);

CREATE TABLE <T1 table schema>
(attj   data_type, ....,
 attj+n  MULTISET|ARRAY ROW
        (atti   data type, ....,
         atth   data type, ....,
         atth+1 data type);
```

Figure 10. N:N Association Transformation

Example 4:
`STUDENT` and `SUBJECT` have N:N association relationship (see Fig.11). The reference of one type together with the relationships attributes will be mapped as collection in another type table.

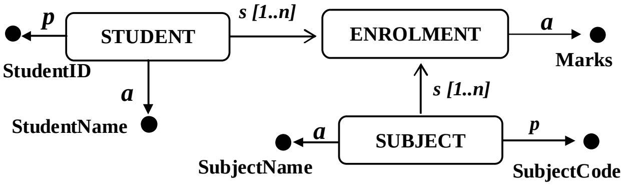


Figure 11. STUDENT-SUBJECT Association Example

`Transformation into XML Schema (Step 1):`

```
<xsd:complexType name = ″SUBJECT_Type″>
  <xsd:attribute name = “SubjectCode” type =
    “xsd:ID” use = “required”/>
  <xsd:element name = “SubjectName” type =
    “xsd:string/>
</xsd:complexType>

<xsd:complexType name = ″STUDENT_Type″>
  <xsd:attribute name = “StudentID” type = “xsd:ID
    use = “required”/>
  <xsd:element name = “StudentName” type =
    “xsd:string/>
  <xsd:element name = “ENROLMENT_Type” maxOccurs=
    ″unbounded″/>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:attribute name = “SubjectCode” type =
          “xsd:string/>
        <xsd:element name = “Marks” type =
          “xsd:integer/>
      </xsd:sequence>
    </xsd:complexType>
</xsd:complexType>

<keyref name=’SUBJECT_SubjectCode_Ref”
  refer=”SUBJECT_SubjectCode”>
  <selector xpath = “STUDENT/ENROLMENT”>
    <field xpath=”@SubjectCode”/></keyref>

<key name=’STUDENT_StudentID”>
  <selector xpath = “STUDENT”>
  <field xpath=”@StudentID”/></key>
```

`Transformation into ORDB (Step 2):`

```
CREATE TABLE Subject
(SubjectCode CHARACTER VARYING(5)
 CONSTRAINT SubjectCode_pk PRIMARY KEY,
 SubjectName CHARACTER VARYING(20));

CREATE TABLE Student
(StudentID CHARACTER VARYING(5)
 CONSTRAINT StudentID_pk PRIMARY KEY,
 StudentName CHARACTER VARYING(20),
 SubjectTaken MULTISET (ROW
        (SubjectCode CHARACTER VARYING(5),
         Marks NUMBER(3)));
```

In the first step we map the associated types into two separate complex types. In one of the complex type we include the key to the other complex type as well as the relationship elements/ attributes. Same as 1:N association, we use **`maxOccurs= ″unbounded″`** with **`key`** and **`keyref`** to preserve the collection and the referential constraint. The difference is now we have additional elements/attributes that come up with the relationship.

In the second step we will likely to have a collection of ROW attribute inside one of the “many” side table. It is because we need to include the key to the other “many” side table and the additional relationship attributes. To preserve the collection we implement the collection with this syntax **`TABLE(…MULTISET <ARRAY[]> (ROW())`**.

5. CONCLUSION

By nature, the XML documents contain a lot of collection structures. Many works have shared the same solution in implementing the collection in database storage by mapping them into flat tables. This practice has diminished the conceptual model semantic. With the existence of collection types in ORDB, we aim to propose different implementation of collection structure in XML.

In this paper we show how the collection in the aggregation and the association relationship of XML data using semantic-network based conceptual model can be preserved in the implementation using the Object Relational Database (ORDB). We propose the usage of collection type both for aggregation and association relationship. In the aggregation we differentiate the method based on the hierarchy constraints, while in the association we differentiate based on the cardinality.

Unlike other works in transformation, our proposed methods cover two straightforward mapping steps, spanned from the conceptual model to the implementation into tables. By doing this, the results maintain the semantic stated in the conceptual level. In addition, using collection type, we have utilized the rich facility in ORDB.

6. REFERENCES

[1] Bourret,R. “Mapping DTDs to Databases”, in http://www.xml.com/pub/a/2001/05/09/dtdtodbs.html, 2001

[2] Conrad, S., Scheffner, D. and Freytag, J. “XML Conceptual Modeling using UML”, *ER 2000*, Springer-Verlag, 2000, 558-571

[3] Feng, L., Chang, E. and T. Dillon, “A Semantic Network-Based Design Methodology for XML Documents”, *ACM TOIS* 20(4), 2002, 390-421

[4] Florescu, D. and Kossmann, D. “Storing and Querying XML Data using an RDMBS”, *IEEE Data Engineering Bulletin* 22(3), 1999, 27-34

[5] Fortier, P. *SQL3 Implementing the SQL Foundation Standard*, McGraw Hill, 1999

[6] Han, W-S., Lee, K-H. and Lee, B.S.“An XML Storage System for Object-Oriented/Object-Relational DBMSs”, *Journal of Object Technology* 2(1), 2003, 113-126

[7] Klettke, M. and Meyer, H. “XML and Object-Relational Database Systems - Enhancing Structural Mappings Based on Statistics”, *WebDB 2000*, Springer-Verlag, 2000, 151-170

[8] Melton, J. (ed.), Database Language SQL – Part 2 Foundation. ISO-ANSI WD 9072-2, International Organization for Standardization, Working Group WG3 (August 2002)

[9] Pardede, E., Rahayu, J.W. and Taniar, D. “New SQL Standard for Object-Relational Database Applications”, *SIIT 2003*, Delft TU, 2003, 191-203

[10] Rumbaugh, J. et al, *Object-Oriented Modelling and Design*, Prentice Hall, 1991

[11] Shanmugasundaram, J., Tufte, K., Zhang, C. He, Ge. DeWitt, D.J. and Naughton, J.F. “Relational Databases for Querying XML Documents: Limitations and Opportunities”, *VLDB 1999*, Morgan-Kauffman, 1999, 302-314

[12] Widjaya, N.D., Taniar, D., Rahayu, J.W. and Pardede, E. “Association Relationship Transformation of XML Schemas to Object-Relational Databases”, *iiWAS 2002*, SCS Publishing House, 2002, 135-142

[13] Widjaya, N.D., Taniar, D. and Rahayu, J.W. "Aggregation Transformation of XML Schemas to Object-Relational Databases", *IICS 2003*, Springer-Verlag, 2003

[14] W.W.W, “The XML data model”, available at http://www.w3.org/XML/Data-model.html/, 2000

[15] Xiou, R., Dillon, T.S., Chang, E., and, Feng, L. “Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema”, *DEXA 2001*, Springer-Verlag, 2001, 795-804