[Volume 2 issue 7 July 2014]
Page No.503-511 ISSN :2320-7167

INTERNATIONAL JOURNAL OF MATHEMATICS AND COMPUTER RESEARCH

Conversion Methodology from hierarchical model to Object-relational model with structural and semantic aspects preservation

Mustapha Machkour[a,b], Karim Afdel[a,b], Youness Idrissi Khamlichi[b,c]

[a]*Department of Mathematics and Computer Sciences, Faculty of Sciences, Agadir, Morocco*
[b]*Laboratory of the Computing Systems and Vision, Faculty of Sciences, Agadir, Morocco*
[c]*Department of Computer Sciences, National School of Applied Sciences, Khouribga, Hassan 1 University, Morocco*

machkour@hotmail.com

Abstract

XML has become a standard for data exchange on the Web. These data, after interchange among different sites, are often to exploit, share by applications designed for data stored in databases using the relational model and recently the object-relational model. The latter imposed itself due to its benefits in terms of reuse and sharing that improve productivity for both the developer and end user. In addition, the object-relational model that's an extension of the relational model has benefited greatly from the advantages of relational model in terms of access performance and security. To fully exploit XML data with the benefits of the object-relational model, we propose in this paper a methodology to convert data written in XML format respecting a DTD (Document Type Definition) into a schema of object-relational model.

Keywords: Hierarchical model, Relational model, Object-Relational model, XML, DTD, Semantic constraints, Structural constraints.

1. Introduction

XML (Extensible Markup Language)[6] is a software- and hardware-independent tool for carrying information. XML is widely used to describe and write data and documents to be used in different types of applications that run on heterogeneous software and/or hardware architectures. To exploit well these XML data written with hierarchical schema by applications using relational database systems[8, 10], the schema conversion methods have been proposed [7, 23, 24, 25] and some algorithms for storing and querying these data have been designed and implemented [14, 32, 35, 36, 37, 39, 40].

However, some of these Relational systems, such as Oracle database[30, 33, 34], PostGres[38], Sybase and DB2/IBM, have evolved or are evolving for supporting the characteristic properties of object-relational model[11, 12, 13, 19, 30]. The object-relational model preserves the qualities of the relational model and integrates concepts stemming from the object model such as object type (User-Defined Type: UDT), object, nested objects, collections (Array Type: AT or Multiset Type: MT), inheritance, polymorphism, abstraction…[11, 12, 13, 17, 30].

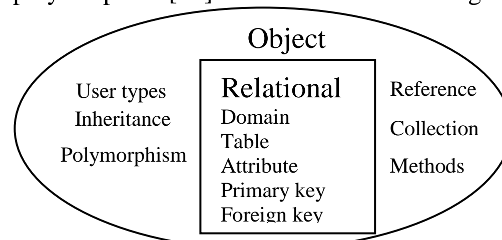
To access the XML data in object-relational database we need a methodology or means for converting an XML schema into an object-relational schema. In this paper, we propose an algorithm to convert an XML schema based on a DTD associated to the XML document into object-relational schema.

2. Related work

Many studies have dealt with the conversion between models. For example, we cite the conversion between Network and Relational models [26, 31], between ER and OO models [5, 21], and between UML and XML models [1, 4, 9, 22].

In our context, many works have been developed for transforming XML documents with a hierarchical schema into equivalent data with relational schema in order to be used by relational database applications. Algorithms have been designed and implemented in this direction. Also, transformation methods from relational data to XML have also been implemented[15, 23]. Furthermore, matching algorithms have been developed for this kind of transformation [27, 29].

However, to overcome the limitations and weaknesses of the relational model, relational schemas are being replaced by the object schemas or extended to the object-relational schemas. Recently, builders of relational database systems add to their systems concepts of object-oriented paradigm to simplify modeling complex structured data and their relationships. These became object-relational database systems (see below figure 1). It is in this context fits the purpose of this article. In this model and in accordance with the standard SQL[1]: 1999 and SQL: 2003 [8, 10, 16, 17, 28] , we use the terms (among others): object type (UDT), object, collection of objects, object reference, collection of references, methods, inheritance, encapsulation and polymorphism[20]. This is illustrated in Figure 1.



[1] Structured Query Language is a standard language for databases.

Figure 1. Concepts of Object-relational model.

3. Transforming XML DTD to Object-Relational schema

In this section and the followings, we present a methodology for transforming an XML DTD schema into object-relational model.

3.1.Definitions and notations

We begin with terminology, definitions and notations for both object-relational model and XML model.
A schema of the object-relational model consists of[16, 17, 28] :

- *Object types(UDT) with attributes or fields (similar to classes in object-oriented programming language),*
- *Reference or Type Reference of an object,*
- *Collections of objects or collections of object references (using varying array or nested table[2]),*
- *Object tables,*
- *Generic type,*
- *Inheritance.*

For the XML schema, we consider the following notation(Figure 2) to represent the definition of an XML element in terms of its attributes and its content model[6].
These notations are similar to those used in context-free grammar or BNF[3] (Backus-Naur Form) [3].
If A is an XML object such an attribute, a list of attributes, an element or a content model of an element ..., A (A underscored) gives the definition of A.
Let E be an XML element, its definition E is given at figure 2.

E :: = <E; Attrs; D>

Figure 2. Definition of XML element.
Items of this definition are explained like as follows:

- The symbols :: = denotes a definition or production;
- E : is the name of the element;
- D: represents the content model of the element E eventually empty;
- Attrs: is a list that contains the attributes of the element. It can also be empty:
  Attrs :: = (Attr1, Attr2…).

The definition of the attribute list: Attrs, that we note: Attrs (underlined Attrs, see above Figure 2), is given by a list containing a definition of each attribute in Attrs. Then we have the following expression:
Attrs :: = (Attr1, Attr2…).
The Definition of each attribute is as follows:

Attri::=<Attri; typeOrValues; Description>.

Figure 3. Definition of attribute of an XML element.
where

- typeOrValues stands for type of the attribute or list of values in the XML model;
- The value of "Description" is given by (using context-free grammar notation [3]):

Description::= #REQUIRED | #IMPLIED | #FIXED value | value.

[2] Terms used in Oracle DBMS.
[3] Backus-Naur Form: Notation used to describe the syntax of languages.

Figure 4. Definition of Description for an attribute.
Obviously, the Meta symbol "|" denotes the alternative.
That's for the representation of XML element and its definition.

3.2.A mapping between XML and Object-relational model

Let us now consider the polymorphic[4] function φ which will allow us converting an XML schema to Object-relational model. This function plays a major role in this article. Its definition is detailed as follows.
For each element E of XML, we associate an object type E in Object-relational model. This object type is given by φ (E). So we have,

φ : E→ φ(E) with
φ(E) ::= E (list_of_attributes_definition)

Figure 5. Definition of the object type φ(E).
where E (right of the symbol :: =) is the object type associated with the XML element E argument of φ.
Attributes of the object type E are given by
(list_of_attributes_definition)
Each element of the list above (figure 5) is a definition of an attribute or field of object type[5] (in object-relational model).
This definition is done by the following expression:

<Attr; Type[;Modifiers] >

Figure 6. Definition of an object-type attribute.
where

- Attr: is the name of attribute (of object type),
- Type: is the type of attribute (of object type),
- Modifiers: represents a list of constraints on values of attribute (of object type). These constraints can contains null, not null, unique, check and foreign key constraint. The brackets surrounding it indicate that list is an option, may be empty as in extended BNF[6] notation[3].

Later on, we show how the attributes of object type "E" are calculated with the function φ.
We have at Figure 2:

E :: = <E; Attrs; D>

The definition of the object type is given by the following formula:

φ(E) ::= E (φ(Attrs) U φ(D))

Figure 7. Definition of an object type.
where the symbol 'U' denotes the union operator.
This can be explained as:
The list of attributes of the object type E is obtained by the union of the image (under φ) of attribute definitions of the XML element and the image (also obtained by φ) of content definition of the element E.
So, to get the structure of the object type E, we have to calculate φ (Attrs) and φ (D).
We start by calculating φ (Attrs).

3.2.1.Calculation of φ(Attrs)

[4] Function having an arbitrary number of different types arguments.
[5] Object type is similar to UDT in SQL: 2003.
[6] Extended BNF notation: BNF extended to use symbols (,),[,],{,}…

φ (Attrs) is a list of attributes definition(of the object type) obtained by the following algorithm:

Algorithm listAttributes;
Input Attrs : list of attributes;
Output φ(Attrs) : list of attribute definitions (of an object type) ;
begin
if Attrs = empty then
*/*There is no attributes for the XML element.*/*
φ (Attrs) ::= empty string;
else
if Attrs = (Attr1, Attr2, ...) then
φ (Attrs) ::=φ (Attr1), φ (Attr2) ...;
end if;
end if;
end;

Figure 8. Calculation of φ (Attrs).

Each Attri, as shown in Figure 3, is given by

Attri::=<Attri; typeOrValues; Description>.

Thus the image of Attri, denoted by φ (Attri), is given by φ(Attri) = φ (<Attri; typeOrValues; Description>).

The value of φ (<Attri; typeOrValues; Description>) is obtained by

φ (<Attri; typeOrValues; Description>) ::= <Attri; φ(typeOrValues) minus Constraints; φ(Description) plus Constraints>.

Figure 9. Definition of the object-type attribute "Attri"

This requires the calculation of φ (typeOrValues), φ(Description) and the constraints that what we do thereafter.

3.2.1.1. Calculation of φ (typeOrValues)

The value of φ (typeOrValues) is a type and constraints on the values of this type. It represents the type and the constraints on the values of object-type attribute associated to the XML element. This value is given using the following table (see Figure 10). The constraints are defined with notations defined in regular expressions[2] that we remind below:

- The character "|" is the alternative;
- The * means 0 or more characters;
- The parentheses are metacharacters for priority and grouping.

<table>
<tr><td rowspan="2">typeOrValues (in XML)</td><td colspan="2">φ (typeOrValues) (in Object relational model)</td></tr>
<tr><td>Type</td><td>Constraints</td></tr>
<tr><td>ID</td><td>Varchar(n)</td><td>(Letter|_)(Letter|_|Digit|:|.|-)*,
UC: Unique Constraint</td></tr>
<tr><td>CDATA</td><td>Varchar(n)</td><td>No constraint</td></tr>
<tr><td>IDREF</td><td>Varchar(n)</td><td>(Letter|_)(Letter|_|Digit|:|.|-)*,
FKC: Foreign Key Constraint</td></tr>
<tr><td>IDREFS</td><td>Varray(p) or Nested table of Varchar(n)</td><td>(Letter|_)(Letter|_|Digit|:|.|-)*,
FKC:Foreign Key Constraint</td></tr>
<tr><td>NMTOKEN</td><td>Varchar(n)</td><td>(Letter|_)(Letter|_|Digit|:|.|-)*</td></tr>
<tr><td>NMTOKENS</td><td>Varray(p) or Nested table of Varchar(n)</td><td>(Letter|_)(Letter|_|Digit|:|.|-)*</td></tr>
<tr><td>Enumerated Attribute list</td><td>Varchar(n)</td><td>(Letter|_)(Letter|_|Digit|:|.|-)*,
ELConstraint : Enumerated List Constraint</td></tr>
</table>

Figure 10. Calculation of φ(typeOrValues).

In what follows, we explain the two right columns in the table above (Figure 10):

In column Type:

- Varchar (n) is a standard type of strings used in database systems. n is the size of type.
- Varray[7] (p) is a data type representing a collection of values in object-relational databases. p is the size of the collection.
- Nested table[8] is a data type used in object-relational databases. It represents a collection of values with unlimited size.

In constraints column (at right in the above table), we have patterns that values of attribute must respect in order to preserve the semantic values of XML elements attribute. Those patterns are similar for all constraints. We can use an applicative constraint to maintain this constraint type (for example check constraint with like operator).

These patterns use

- Letter : regular definition[3] defined by the following expression:
  Letter ::= [A..Za..z].
- Digit : regular definition defined by the following production:
  Digit ::= [0..9].

In column constraints, there are also

- Foreign key Constraint (FKC): constraint represents the usual referential integrity in the database literature.
- Unique Constraint (UC): indicates that attribute values are distinct.
- Enumerated List Constraint (ELConstraint): Constraint with a list of values corresponding to the enumerated value list that specifies the content model of XML elements attributes. We can use *check constraint* with *like operator* to maintain this constraint.

To simplify explication, we call the shared constraint based on regular expression

(Letter|_)(Letter|_|Digit|:|.|-)*

by LexAttrConstraint (Lexical Attribute Constraint).

3.2.1.2. Calculation of φ (Description)

In order to complete the calculation of φ (Attrs), it remains to calculate φ(Description).

The value of φ (Description) is a list of usual constraints in databases system. It is obtained by an algorithm based on the following table:

Description	φ(Description)
#REQUIRED	Not null
#IMPLIED	Null
#FIXED Value	Not null, default Value
Value	default Value

Figure 11. Calculation of φ (Description).

To explain how this function operates, we propose the example below:

<!ELEMENT journal (…)>
<!ATTLIST journal id ID #REQUIRED>
<!ATTLIST journal issn CDATA #IMPLIED>

Figure 12. Element journal.

Calculus of φ (journal)

[7] Type of limited collection used in Oracle DBMS.
[8] Type of unlimited collection used in Oracle DBMS.

The simplified element journal in the example above has two attributes id and issn.

If we apply φ to journal element we obtain:

φ(<u>journal</u>)=journal(φ (<u>id</u>), φ (<u>issn</u>),…).

"journal" (on the right of "=" symbol) is an object type with attributes φ (<u>id</u>), φ (<u>issn</u>)…

In order to have φ(<u>journal</u>) we must calculate

φ (<u>id</u>)= φ (<id;ID;REQUIRED>) and

φ(<u>issn</u>)= φ (<issn;CDATA;#IMPLIED>).

Calculus of φ (<u>id</u>)

According to formula in Figure 9 and table in Figure 10, we have

φ (<u>id</u>) =<id; φ(ID) – (LexAttrConstraint+UC);

φ (#REQUIRED)+ (LexAttrConstraint+UC)>.

Since

φ(ID)=varchar + (LexAttrConstraint+UC)

and

φ (#REQUIRED)=not null.

Then φ (<u>id</u>) becomes

φ (<u>id</u>) = <id; varchar; not null+ LexAttrConstraint+UC)>.

Hence φ (<u>id</u>) is an attribute of journal object with the following specifications :

- *id :name of attribute;*
- *varchar : type of id;*
- *not null, LexAttrConstraint and unique are constraints of id(object attribute).*

Calculus of φ (<u>issn</u>)

Similarly, we get for *φ(<u>issn</u>)*, the following expression

φ (<u>issn</u>)= <issn; varchar; null>

Then the journal object become

journal(<id; varchar; not null+ LexAttrConstraint+UC)>,<issn; varchar; null>,…).

End of example.

We can recapitulate these steps in the following algorithm:

Algorithm Attribute_object_from_attribute_XML_element;
Input Attri: an attribute of an element XML;
Output φ(Attri) : an attribute of un object type;
Begin
Calculate φ(typeOrValues);
Calculate φ(Description);
Return <Attri; φ(typeOrValues) minus Constraints; φ(Description) plus Constraints>;
End;

Figure 13. Algorithm for obtaining an Object attribute from an XML attribute.

3.2.2.Calculating φ (D)

We recall that the expression of φ(<u>E</u>) (see Figure 7) is given by : φ(<u>E</u>)=E (φ(<u>Attrs</u>) U φ(<u>D</u>)).

We have processed φ (<u>Attrs</u>). In order to complete the definition of the list of attributes of the object type, we now proceed to calculate φ (<u>D</u>).

We have in Figure 2

<u>E</u>=<E; <u>Attrs</u>; <u>D</u>> where D is the content model of the XML element E.

D can be:

- *List of symbols between "**<! ELEMENT ElementName** (" and ")>",*
- *EMPTY,*
- *ANY.*

For example, in the following simplified example

<!ELEMENT journal (volume+)>
<!ATTLIST journal id ID #REQUIRED category CDATA #IMPLIED >
<!ELEMENT volume (issue+)>
<!ELEMENT issue (paper+)>
<!ELEMENT paper (title, author)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>

Figure 14. Example of DTD

The *journal* element has the following representation

<u>journal</u>::=<journal; <u>id, category</u> ; <u>volume+</u>>.

The value of D for *journal* element is *volume+*.

The *volume* element definition is

<u>volume</u> :: = <volume; ; <u>issue+</u>>

Similarly, the *paper* element has the following representation

<u>paper</u>::=<paper; ; <u>title,author</u>>.

The value of D for *paper* element is *title, author*.

The *title* element definition is

<u>title</u> :: = <title; ; <u>#PCDATA</u>>,

and author element definition is

<u>author</u> :: = <author; ; <u>#PCDATA</u>>

The value of D for both *title* and *author* is #PCDATA.

Elements of D are connected together by sequence, alternative, Kleene closure, transitive closure and optional value.

In order to simplify the calculation of φ (<u>D</u>), we introduce the following BNF (Backus-Naur Form) grammar representing the content model of elements (E is an element of D). We call this grammar G:

a) E::=ANY
b) E:: =EMPTY,
c) E :: = E, E for the sequence,
d) E :: = E + E for the alternative,
*e) E :: = {E} for the Kleene closure (replace *),*
f) E :: = {E},E for the transitive closure (replace +),
g) E :: = [E] for an optional value(replace ?),
h) E :: = #PCDATA for a simple type.

Figure 15. Elements of the G Grammar.

To this grammar we associate the following grammar that uses the symbol "_" (underscore). We note this grammar <u>G</u> (G underscored). Recall that <u>E</u> as defined above (see Figure 2) gives the definition of element E. The productions order in the two grammars, G and <u>G</u>, is preserved. This grammar is defined as following:

a) <u>E</u>::=<u>ANY</u>
b) <u>E</u>::=<u>EMPTY</u>
c) <u>E</u>::=<u>E</u>, <u>E</u>
d) <u>E</u> ::= <u>E1</u>+<u>E2</u>
e) <u>E</u>::= {<u>E</u>}
f) <u>E</u>::= {<u>E</u>}, <u>E</u>
g) <u>E</u>::= [<u>E</u>]
h) <u>E</u>::=<u>#PCDATA</u>

Figure 16. Elements of the G Grammar.

Let us now define the value of φ for each item of the grammar <u>G</u>. We define a grammar that we call φ (<u>G</u>):

a) φ (<u>E</u>)= φ (<u>ANY</u>)::=AnyData [9] or AnyType [10] . (Generic type in object-relational model).
b) φ (<u>E</u>)= φ (<u>EMPTY</u>)::=Empty string

[9] Type used in Oracle DBMS.
[10] Type used in Oracle DBMS.

c) *φ (E)= φ (E1, E2)::= φ (E1), φ(E2) E1 and E2 are used to distinguish between E at left of '=' and E at the right of the '='.*
d) *φ (E) ::= (+, φ (E1), φ (E2)). Here, we define a generic type that can hold the object types φ(E1) and φ(E2).*
e) *φ (E) ::={φ (E)}, list of φ (E) with null constraint;*
f) *φ (E) ::={φ (E)} list of φ (E) with not null constraint;*
g) *φ(E) ::= [φ (E)], φ (E) with null constraint ;*
h) *φ (E) ::= φ (#PCDATA).*

Figure 17. Elements of the φ(G) Grammar.

The value of φ (# PCDATA) is given by the following expression:

φ (# PCDATA) ::=<value;varchar; PCDATA_constraint>

Figure 18. Value of φ (# PCDATA)

where

- value is an attribute of the object type containing the value of the element XML;
- Varchar representing its type;
- PCDATA_Constraint is a constraint on the values of the attribute. It is defined by the following regular expression[3]:

(Letter | _ | Digit | . | - | :)*.

At this stage, in order to understand how the function φ operates, we propose below some conversion examples from an XML model to an object-relational model.

3.2.2.1. Examples of the transformation

a) Example 1

For element title, we have

title :: = <title; ; #PCDATA>,

If we apply the function φ using its definition, we obtain

φ(title) = φ(<title; ; #PCDATA>)
=title(φ (Attrs)U φ (#PCDATA))
=title(φ (#PCDATA)) since φ (Attrs) is empty(there is no attribute for title).

Then we replace φ (#PCDATA) with its value using the grammar φ (G) and we get the final expression of φ(title):

φ (title)=title(<value;varchar; PCDATA_Constraint >).

Then *title* is an object type (in object-relational model) with an attribute named value. The type of attribute value is varchar and its values verify *PCDATA_constraint* constraint.

b) Example 2

We can do the same with the author element defined as following:

author :: = <author; ; #PCDATA>

and we get

φ (author)=author(φ (#PCDATA)) = author(<value;varchar; PCDATA_constraint >).

c) Example 3

For another complex example that illustrate how *φ* works, we take the paper XML element defined as follows:

paper::=<paper; ; title, author>.

In this case

φ(paper)=paper(φ(title,author))=paper(φ(title), φ(author)),

If we replace φ(title) and φ(author) by their values as computed above, we obtain

φ(paper)=paper(title(<value;varchar; PCDATA_constraint >), author(<value;varchar; PCDATA_constraint >)).

Thus, paper is an object type with two attributes: title and author. Each of these attributes is an object type with an attribute named value.

In general the calculation of φ (D) is given by the following algorithm:

```
Algorithm Calculus_of_Attributes;
Input:  D, a model of content of an XML element E;
Output: φ(D), list of object attributes;
begin
   loop
        select an arbitrary φ (v) in φ (D) with v
        different to E;
        if (φ (v) is not in v (to avoid recursion)) then
          Calculate φ (v) using the φ(G) grammar and
          algorithm at Figure 13;
        End if;
        If (there is no φ(v) in φ (D)) or (each φ(v) in
        φ (D) is in v /* case of recursion*/
        or φ (v)=φ (E)) then
                Exit; /*to leave loop*/
        End if;
   End loop;
End ; /*end of algorithm*/
```

Figure 19. Calculation algorithm of an object Attribute.

d) Example of the calculus of the φ(D) with recursion

To illustrate the calculus of the *φ(D)* with recursion, we consider the following example:

<!ELEMENT paper (title, author, cite?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT cite (paper)>*

Figure 20. Example with recursion.

where the element "cite" represents the cited papers in paper references.

Here we have paper ::= <paper; ; D> where D is *(title, author, cite?).*

Therefore

φ (paper) = paper(*φ(D).*

If we replace *φ(D)* with his value

φ(title, author, [cite]) in last formula ,we obtain

φ (paper) = paper(φ(title, author, [cite]))

and

φ(paper)=paper(φ(title),φ(author),[φ(cite)]).

Figure 21. Intermediate value of φ (paper).

We have already calculated *φ(title)* and *φ(author)* above. Let us find φ(cite).

From the expression *<!ELEMENT cite (paper*)>*, we can write *cite::= <cite;;{paper}>*

If we apply the function *φ* to cite element, we get:

φ(cite)= φ(<cite;;{paper}>)
=cite(φ ({paper}))= cite({φ (paper)}).

Replacing *φ(title), φ(author) and φ(cite),* at figure 21, with their values, we obtain

φ(paper)=paper(φ(D))=
paper(title(<value;varchar; PCDATA_constraint >),
author(<value;varchar; PCDATA_constraint >),
, [cite({φ (paper)})]).

Figure 22. Value of φ (paper).

We notice that we have found *φ(paper)* in *φ(D).* (We remember that D is a definition for paper element).

The process halts here because there is no more *φ(v) in φ(D)without recursion or φ(v) different to φ(paper).*
Next, we present our algorithm of conversion from XML model to object-relational model.

4. Algorithm of conversion

Our algorithm of conversion from XML model to object-relational model, uses CreateObjectType(…) function (Figure 25Figure 25) that creates an object type image of the XML element.
This function needs a recursive function named CreateObjectAttribute(attr(…)) (Figure 23).

4.1.Creation of object attribute

The CreateObjectAttribute function takes an attribute in argument with a list of items and returns the expression: <attr; typeOfAttribute; modifiers>.
This expression, as seen before (Figure 6), represents a definition of an attribute in object-relational model.
This function is based on the values of the function φ defined above. Its definition is given in the following figure.

```
Function CreateObjectAttribute (attr(listOfItems)) return ObjectAttribute ;
y ObjectAttribute ; //y is variable for an object attribute.
i integer initialized by 0;
/*this variable i is for counting the number of attributes which are
        added in the case of the alternative which is not
        surrounded by an element (for example <!ELEMENT a
        (b|c), d>).*/
begin
/*processing of closure*/
1)  for each {x(...)} in attr loop /* x is an element*/
2)      y= CreateObjectAttribute (x(...));
3)      Create a type of nested table named xs (name of x
        concatenated to s) based on object type y;
4)      Replace {x(...)} in attr by <"xs";xs; constraints_on_x>;
        /* therefore "xs" is an attribute of the object-type
        attr. The type of this attribute is xs.*/
5)  end loop;
6)  for each {φ(x)} in attr loop /* x is an element*/
7)      If type x is not yet created then
8)          Create the object type x as incomplete type; /*
            necessary to have recursion*/
9)      End if;
10)     Create a type of nested table named xs (name of type "x"
        concatenated to letter 's' ) based on ref object type "ref x";
        /*(norme sql3)*/
11)     Replace in attr, {φ(x)} by <"xs";xs; ' '>;
12) End loop;
13) for each [x(...)] in attr loop /* x is an element*/
14)     y= CreateObjectAttribute (x(...));
15)     add to y a null constraint;
16)     Replace [x(...)] in attr by y;
17) End loop;
18) Loop
19)     If each item of listOfItems matches "<...>" then

20)       If the attr type is not yet created then
21)         Create an object type named attr where each of
            its attributes corresponds to each item of
            listOfItems;
22)       End if;
23)       Return the object attribute <"attr"; attr;
          list_of_item_constraint>;
24)     Else //case of alternative with named element (for example
        <!ELEMENT a (b|c)>)
25)       If each item of listOfItems matches "<...>"
          except one item that matches "+" then
26)         If the attr type is not yet created then
27)           For each item <x...> in listOfItems loop
28)               Create an object type named "x" if
              it's not created;
29)           End loop;
30)           Create an object type named attr that has
              an attribute named 'value' with a generic
              type (ANYDATA for example);
31)           Add to attr a constraint that limits values
              of the attribute 'value' to objects that are
              instances of types 'x' created by "for each"
              above at lines 28 to 30";
              // we call this constraint: constraint_on_attr;
32)         End if;
33)         Return the object attribute <"attr"; attr;
            list_of_item_constraint+constraint_on_attr>;
34)       Else //case of alternative with unnamed element
            (for example <!ELEMENT a (b|c), d>)
35)         i=0;
36)         For each item (+,...) in (listOfItems) loop
37)           i ← i+1;
38)           Replace, in attr, (+,...) by
              CreateObjectAttribute (_attr_i(+,…));
              //_attr_i is created for alternative.
39)         End loop;
40)         For each item e(...) in (listOfItems) loop
41)           If e(...) doesn't contain directly any φ then
42)             Replace, in attr, e(...) by
            CreateObjectAttribute (e(…));
43)           Else /*Case of recursive element (direct).*/
44)             If e(...) matches e(φ(x)) then
45)               Replace, in attr, e(...) by <"e"; ref x;>;
46)               /*ref type is a type that allows an
                  attribute to contain an object
                  reference. */
47)             Else /*Case of elements mutually
                  recursive.*/
48)               If e(...) matches e(...,φ(x),...) then
49)                 If the x type is not yet created then
50)                   Create the object type x as
                      incomplete type (in order to have
                      recursion);
51)                 End if;
52)                 Replace φ(x) by <"x";ref x;>;
53)               End if;
54)             End if;
55)           End if;
56)         End loop;
57)       End if
58)     End if;
59) End loop;
End; /*End of function : CreateObjectAttribute */
```

Figure 23. CreateObjectAttribute Function.
To illustrate the usage of this function, we consider the example below:

```
<!ELEMENT paper (title, author, cite?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (fn, ln)>
<!ELEMENT fn (#PCDATA)>
<!ELEMENT ln (#PCDATA)>
<!ELEMENT cite (paper*)>
```

Figure 24. Example of DTD.
We have at Figure 21 the expression
φ (paper) = paper(φ(title), φ(author),[φ(cite)]).

In order to apply *CreateObjectAttribute* function to paper, we have to find φ(title) , φ(author) and [φ(cite)].
φ(title) (as seen above) is given by
φ (title)=title(<value;varchar; PCDATA_Constraint >).
The element author is defined by
<!ELEMENT author (fn, ln)>.
So

φ(author)=author(φ(fn), φ(ln)).

The element fn is defined by

<!ELEMENT fn (#PCDATA)>

then
φ(fn) =fn(<value;varchar; PCDATA_Constraint >
We can do the same for ln :
φ(ln) =ln(<value;varchar; PCDATA_Constraint >.
Then φ(author) becomes
φ(author)= author(*fn(<value;varchar;PCDATA_Constraint >)*, *ln(<value;varchar; PCDATA_Constraint >)).*
The value of [φ(cite)] is [cite({φ (paper)})] (see Figure 22).
We replace φ(title), φ(author) and [φ(cite)] in φ(paper) we obtain the expression:
φ(paper)=paper(φ(D))=paper(title(<value;varchar; PCDATA_constraint >),
author(fn(<value;varchar; PCDATA_Constraint >),
ln(<value;varchar; PCDATA_Constraint >)),
[cite({φ (paper)})]).
Applying the CreateObjectAttribute function to paper, we transform recursively:

- *title(<value;varchar; PCDATA_Constraint >);*
- *author(fn(<value;varchar; PCDATA_Constraint >), ln(<value;varchar; PCDATA_Constraint >)) and*
- *[cite({φ (paper)})].*

We begin with *[cite({φ (paper)})].*
To transform *[cite({φ (paper)})]* by function CreateObjectAttribute , we use

i. *lines between 6 and 12 to eliminate symbols "{" , "}" and φ;*
ii. *lines between 13 an 17 to eliminate symbols "[" and "]".*

So, for lines between 6 and 12

- we create an incomplete object type named paper;
- we create a nested table type based on "ref paper" named papers;
- we replace{φ (paper)} by <"papers";papers;' '>.

After that, we get the expression:

[cite(<"papers";papers;' '>)].

Furthermore, with lines between 13 and 17, we apply *CreateObjectAttribute(cite(<"papers";papers; ' '>))* that uses lines between 19 and 23, and returns

<"cite";cite; null_constraint>

We continue the transformation with the element *title*. *title* as seen earlier has the expression:
title(<value; varchar; PCDATA_Constraint >).
Since title contains only items that match " <…>", the call of *CreateObjectAttribute(title(<value…>))* uses lines between 19 and 23, and creates an object type named title with one attribute named value and returns an object-type attribute defined by :
<"title";title; PCDATA_Constraint on "title".value>.
Similarly, for fn and ln element, we obtain the two following object-type attributes:
<"fn";fn; *PCDATA_Constraint on* "fn".value>;
<"ln";ln; *PCDATA_Constraint on* "ln".value>.

Now, search CreateObjectAttribute (author(…)) for the author element.
We have
φ(author)=author(*fn(<value;varchar;*
PCDATA_Constraint >),
ln(<value; varchar; PCDATA_Constraint >)).
In this expression, author has items (fn and ln) that do not match "<…>".
In that case, to have CreateObjectAttribute (author(…)), we use lines 40 and 41 and we get
<"fn";fn; *PCDATA_Constraint on* "fn".value> (obtained by CreateObjectAttribute(fn(…)))
and
<"ln";ln; *PCDATA_Constraint on* "ln".value> (obtained by CreateObjectAttribute(fn(…))).
After this substitution, author become
author(<"fn";fn; *PCDATA_Constraint on* "fn".value>,
<"ln";ln; *PCDATA_Constraint on* "ln".value>)
Then, we can now use statements between 19 and 23 lines:

- create an object type named author with attributes "fn" and "ln";
- return an attribute defined by <"author", author, fn_constraint + ln_constraint>.

Hence, paper has the expression
paper(<"title";title; PCDATA_Constraint on "title".value>,
<"author", author, fn_constraint + ln_constraint>,
<"cite";cite;null_constraint>).
Finally, we obtain the object type : *<"paper"; paper; constraint_on(title,author,cite)>.*
End of example.
Now, let us see how this algorithm works in the case of the alternative. So, we propose the example below:
<!ELEMENT person (name, (email | phone))>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
First, we calculate *φ(person).*
φ(person)=person(φ (name, (email | phone))>
if we use the algorithm at figure 19, we obtain

person=person(φ (name), φ (email | phone))

that becomes
person=person(name(<value;varchar; PCDATA_Constraint >),
(+, email(<value; varchar; PCDATA_Constraint >),phone(<value; varchar; PCDATA_Constraint >))).
Now, we apply "*CreateObjectAttribute*" function to "person (…)":

- *by lines 40, 41and 42, we transform name(<…>) to*
  <"name";name;PCDATA_Constraint on name.value>;
- *with lines 34 to 39, we transform (+, email(…), phone(…)) to*
  _person_1 (+,email(…), phone(…));
- *with lines 40 - 42, we obtain :*
  - for email : <"email"; email; PCDATA_Constraint on email.value>;
  - for phone: <"phone"; phone; PCDATA_Constraint on phone.value>)>;

Consequently "*_person_1*" becomes
_person_1(+,<"email"; email; PCDATA_Constraint on email.value>,
<"phone"; phone; PCDATA_Constraint on phone.value>);
- *with lines 25 to 33 applied to "_person_1" we get*

<"_person_1"; _person_1;
constraints_on_email_phone +
constraint_on_"_person_1">.
and *person* takes the structure
person (<"name";name; PCDATA_Constraint on name.value>;
<"_person_1"; _person_1; constraints_on_email_phone +constraint_on_"_person_1">)
Finally, we apply lines between 19 and 23 to person obtained above and we get:
<"person"; person; constraints_on_name_phone…>
End of example.

4.2.Creation of objet type associated to XML schema

Now we consider the function *CreateObjectType*. It takes an object obtained by applying the function φ to *root* element of XML document and returns an object type (UDT) with constraints. This function applies only to this type of object. The code of this function is as following:

```
Function CreateObjectType (Object(listOfItems)) return ObjectType;
y ObjectAttribute ; /*y is an object attribute variable.*/
Begin
y=CreateObjectAttribute(Object(listOfItems));
/* y has the form <"Object"; Object; Constraints>.*/
return: <Object, Constraints>;
//an object type with its constraints.
End;
```

Figure 25. CreateObjectType function.

If we apply CreateObjectType to paper below (that we suppose the root of document):
<"paper"; paper; constraint_on(title,author,cite)>
we obtain the object type
<paper, constraint_on(title,author,cite)>.
Finally, we present the algorithm of conversion.

```
Algorithm Conversion;
Input :  valid XML document with its DTD; Let be E the root of this document;
Output: an object-relational schema;
Begin
1)  Calculate φ(E) using the rules presented above at
    figure 17.
2)  Let be "E(listOfItems)" this value;
3)  Let be <E, Constraints> the object type obtained by
    CreateObjectType(E(listOfItems));  /*algorithm at
    figure 24 */
4)  Create an object table named "E_Table" with object
    type E and constraints defined by E;
    /*" E_Table" is a an object table where we store the
content of the XML document.*/
End;  /*end of Conversion*/
```

Figure 26. Algorithm of Conversion.

Then, as we have seen in the last above algorithm, we finish the transformation of structure.
The content (values of elements and attributes) of the XML document will be stored in the object table created by the last instruction (at line 4) of conversion algorithm (Figure 25). The object type of this table is the root element of the XML structure or XML document.
Applying this algorithm to the example above "paper"(see Figure 24), we create an object table named "Paper_Table" based on object type paper. The table has constraints defined by constraint_on(title,author,cite). The structure of table can be similar to the following:

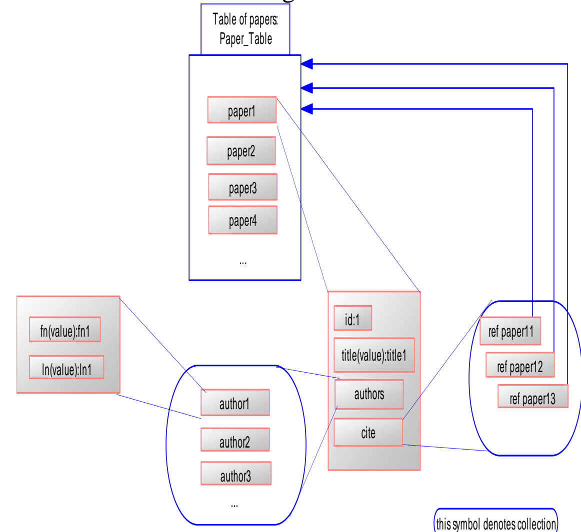


Figure 27. Structure of Paper table.

5. Conclusion

In this paper, we have presented a methodology to transform hierarchical XML DTD schema into object-relational schema. This method offers many advantages for of object-relational database users for storing, manipulating and retrieving XML document with preserving structural and some semantic aspects (using constraints). Also with this method we can retrieve the structure for the initial document from its transformed object-relational model. Finally, comparing with others methods, our method integrates XML elements within few object tables.

References

[1] L. Al-Jadir and F. El-Moukaddem, "F2/XML: Storing XML Documents in Object Databases," International Conference on Object Oriented Infomation Systems, Montpellier, France, 2002.
[2] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and J. D. Ullman, "Compilers Principles, Techniques, & Tools," 2nd ed, 2007, pp. 116-122,159-163.
[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and J. D. Ullman, "Compilers Principles, Techniques, & Tools," 2nd ed, 2007, pp. 42-50, 197-199, 204-205.
[4] V. Bisova and K. Richta, "Transformation of UML Models into XML," ADBIS-DASFAA Symposium on Advances in Databases and Information Systems, Prague, Czech Republic, 2000.
[5] A. Boccalatte, D. Giglio, and M. Paolucci, "An Object-Oriented Modeling Approach Based on Entity-Relationship Diagrams and Petri Nets," IEEE Internal conference on Systems, Man and Cybernetics,San Diego, CA, 1998.
[6] T. Bray, Paoli, J., Sperberg-McQueen, and a. M. C. M., E., "Extensible Markup Language (XML) 1.0 (Second Edition)," W3C Recommendation. http://www.w3.orglTR2OOOlREC- XML-20001006l, 2000/10.

[7] E. Castro, D. Cuadra, and M. Velasco, "From XML to Relational Models," Informatica, vol. 21(4), pp. 505-519, 2010/12.
[8] T. M. Connolly and C. E. Begg, Database Systems: A Practical Approach to Design, Implementation, and Management, 4 ed., 2005.
[9] R. Conrad, D. Scheffner, and J. C. Freytag, "XML Conceptual Modeling using UML," International Conference on Conceptual Modeling, Salt Lake City, UT 2000.
[10]C. Coronel, S. Morris, and P. Rob, Database Systems: Design, Implementation, and Management, 10 ed.: Cengage Learninig, 2012.
[11]H. Darwen and C. J. Date, "The Third Manfesto," SigMOD Record 24(1), pp. 39-49, 1995.
[12]C. J. Date, "Preview of The Third Manifesto," Database Programming & Design Journal (San Francisco, CA: Miller Freeman Publications), vol. 11(8), 1998(8).
[13]C. J. Date and H. Darwen, Databases, Types And the Relational Model: The Third Manifesto, 3 ed.: Addison-Wesley, 2007.
[14]A. Deutsch, M. Fernandez, and D. Suciu, "Storing Semistructured Data with STORED," In Proc. of ACM SIGMOD, Philadelphia, PN, 1999.
[15]A. Duta, B. K., and R. Alhajj, "ConvRel: Relationship conversion to XML nested structures," in Proceedings of ACM SIG Symposium on Applied Computing, pp. 698-702, 2004.
[16]A. Eisenberg and J. Melton, "SQL:1999, formerly known as SQL3," SIGMOD Record, vol. 28(1), March 1999.
[17]A. Eisenberg, J. Melton, K. G. Kulkarni, J.-E. Michels, and F. Zemke, "SQL: 2003," SIGMOD Record, vol. 33(1), pp. 119-126, 2004.
[18]A. A. A. El-Aziz and A. Kannan., "Mapping XML DTDs to Relational Schemas," In Proceedings of the 2nd International Conference on Computer Communication and Informatics (ICCCI), pp. 1–7, 10-12, Jan 2012.
[19]H. García-Molina, J. D. Ullman, and J. Widom, Database Systems :The Complete Book: Prentice Hall, 2002.
[20]G. Gardarin, "Databases," in Databases, Eyrolles, Ed., ed, 2001, pp. 442-443.
[21]M. Gogolla, A. K. Huge, and B. Randt, "Stepwise Re-Engineering and Development of Object-Oriented Database Schemata," International Workshop on Database and Expert Systems Applications, Vienna, Austria, 1998.
[22]J. Hou, Y. Zhang, and Y. Kambayashi, "Object-Oriented Representation for XML Data," International Symposium on Cooperative Database Systems for Advanced Applications, Beijing, China, 2001.
[23]S. Kanagaraj and D. S. Abburu, "Converting Relational Database Into Xml Document " IJCSI International Journal of Computer Science Issues, vol. 9(2), pp. 127-131, 2012/3.
[24]J. Kim, D. Jeong, and D.-K. Baik, "A Translation Algorithm for Effective RDB-to-XML Schema Conversion Considering Referential Integrity Information," Journal of Information Science and Engineering, vol. 25, pp. 137-166, 2009/1.
[25]D. Lee, M. Mani, and W. W. Chu, "Schema Conversion Methods between XML and Relational Models," Knowledge Transformation for the Semantic Web, IOS Press, pp. 245-252, 12 2003.
[26]Y. E. Lien, "On the Equivalence of Database Models," Journal of the ACM 29, pp. 333–362, 1982.
[27]J. Madhavan, P. A. Berstein, and E. Rahm, "Generic Schema Matching with Cupid," International Conference on Very Large Data Bases, Roma, Italy, 2001.
[28]J. Melton, Advanced SQL:1999: Understanding Object-Relational and Other Advanced Features (The Morgan Kaufmann Series in Data Management Systems), 2003.
[29]R. J. Miller, L. Haas, and M. A. Hernandez, "Schema Mapping as Query Discovery," International Conference on Very Large Data Bases, Cairo, Egypt, 2000.
[30]S. Navathe and R. Elmasri, "Fundamentals of Database Systems," ed: Addison-Wesley, 2011, pp. 353-413.
[31]S. B. Navathe, "An Intuitive Approach to Normalize Network Structured Data," International Conference on Very Large Data Bases, Montreal, Quebec, Canada, 1980.
[32]OPENXML, "Retrieving and Writing XML Data.," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsql/ac_openxml_759d.asp, 2004.
[33]J. Price, "Oracle Database 11g SQL," ed: McGraw -Hill, 2008, pp. 379-473.
[34]J. W. Rahayu, D. Taniar, and E. Pardede, Object-Oriented Oracle: IRM Press, 2006.
[35]A. Schmit, M. L. Kersten, M. Windhouwer, and F. Wass, "Efficient Relational Storage and Retrieval of XML documents," In WebDB (Informal Proceedings), pp. 47-52, 2000.
[36]J. Shanmugasundaram, I. Tatarinov, E. Shekita, J. Kiernan, E. Viglas, and J. Naughton, "A General Technique for Querying XML Documents using a Relational Database System," SIGMOD, 2001.
[37]J. Shanmugasundaram, K. Tufte, G. He, X. C., D. D., and N. J., "Relational databases for querying XML documents: limitations and opportunities," in: VLDB, Edinburgh, Scotland, 1999/9.
[38]M. Stonebraker, L. A. Rowe, and M. Hirohama, "The Implementation of PostGres," IEEE on Knowledge and Data Engineering, vol. 2, pp. 125-142, Mars 1990.
[39]I. Tatarinov, S. Iglas, D., and V., "Storing and Quering Ordered XML Using a Relational Database System," ACM SIGMOD, Wisconsin, USA, 2002.
[40]M. YoshiKawa and T. Amagasa, "XRel: A Path -based approach to storage and retrieval of XML documents using relational databases," ACM Trans. on Internet Technology 2001.