International Journal of Multidisciplinary and Current Research

Research Article

ISSN: 2321-3124
Available at: http://ijmcr.com

Converting XML Schema into Object-Relational Model with Data Constraints Preservation

Mustapha Machkour[†‡], Said Aminzou[†], Karim Afdel[†‡], Youness Idrissi Khamlichi[‡]

[†]Department of Computer Sciences, Faculty of Sciences, Agadir, Morocco
‡Laboratory of the Computing Systems and Vision, Faculty of Sciences, Agadir, Morocco
[†] Laboratory of Industrial and Computer engineering, ENSA Agadir, Morocco
[‡]Department of Computer Engineering, ENSA of Khouribga, Morocco

Accepted 05 June 2015, Available online 11 June 2015, **Vol.3 (May/June 2015 issue)**

Abstract

XML is widely used for data exchange between systems and applications. XML data are described in hierarchical form using XML schema languages such as XML DTD (Document Type Definition), W3C XML Schema or XDR (XML Data Reduced). To make these data accessible to relational database systems, which represent a dominant and efficient tool for storing and retrieving structured data, schema conversion methods must be developed. The early conversion methods have focused on mappings between DTDs and relational model. But, due of the needs to hold complex data, certain relational database builders have evolved their systems into object-relational model. Meanwhile, XML Schema language developed to describe data in more detail than DTD is a mature technology. So, in this paper we propose a methodology for translating data described in XML Schema language into object-relational model. This translation preserves integrity constraints defined in XML Schema and uses composition of conversions.

Keywords: *XML Schema, DTD, Data Constraint, Object-Relational Model, Conversion Composition*

1. Introduction

XML (Extensible Markup Language) [6] is the de facto standard for the data exchange between different types of applications. It would be beneficial to have a method to store and retrieve XML data as relational data in relational databases [10] which currently are the most used type of databases. For this purpose, methods and algorithms of schema conversion have been developed [8,18,19]. The first algorithms dealt with XML DTD[1][7] schema. However, some relational systems like Oracle database [12,22,23] PostGres [24] Informix and DB2/IBM, have added object characteristics and become object-relational systems.

Recently, W3C[2] XML Schema[3] language [4,16,25], among other (e.g., XDR[4][20] and SOX[5][13]), is increasingly used to describe XML data. This is due, compared to the DTD, to its richness in terms of structures, type and constraints, and the backing of W3 Consortium.

To follow such changes, the current issue is to translate XML data described by XML Schema language into object-relational schema. Thus, our contribution in this paper is to propose a new conversion methodology as solution.

This methodology is done by using two steps shown in Figure 1. The XML Schema is first converted into a DTD schema as an intermediate representation and extended with the constraints of type, value and structure. We name this extended DTD schema. This extended DTD, designated eDTD, is then translated into Object-relational schema (ORS).

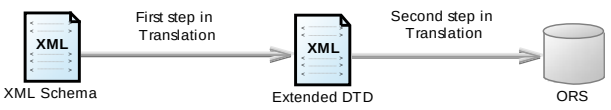


Figure 1 Conversion from XML Schema to Object-relational schema using DTD

Our rationale for using model composition is as follows: 1) there are software (e.g., Altova XMLSpy, XMLBlueprint and Netbeans) that enable the mapping from XML Schema to XML DTD even though they do not consider type, value and structure constraints, 2) we can learn from algorithms already developed for mapping XML DTD into database schema.

The remainder of this paper is organized as follows. Section 2 presents an overview of model conversions. In

[1] Document Type Definition.
[2] World Wide Web Consortium.
[3] The initial 's' in Schema should be capitalized.
[4] XML-Data Reduced.
[5] Schema for Object-Oriented XML.

Section 3, notions and definitions for describing our transformations and conversions are given. Steps in our mapping methodology are described in Sections 4, 5 and 6. The Section 7 concludes the paper.

2. Related work

Many studies have dealt with the conversion between data models at different levels e.g., conceptual, logical and physical. Among those, in this paper, we cite the conversion between: Entity-Relationship and Object-Oriented models[5], Relational model and XML schema[8,17], XML DTD and Object Model [1], and, UML and XML models [9]. There are also tools for converting XML schema into different XML schema languages (e.g., XML DTD and XML Schema).

In our context, we have closely reviewed algorithms that have been developed for mapping between XML (often based on DTD) and relational schemas, and, between XML DTD and Object-Relational schemas.

Our work focuses on conversion from XML Schema into Object-Relational model using DTD schema as intermediate schema.

3. Definitions and notations

In this section we describe the definitions and notations from object-relational, XML DTD and XML Schema used in this paper.

3.1 Object-relational model

According to the standard SQL[6]: 2003 [11,12,14,15,21] a schema of the object-relational model includes:

- 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[7]),
- Object tables : tables that store objects,
- Generic type,
- Inheritance: for creating type hierarchies.

Below, we use notations similar to those used in context-free grammar or BNF[8] (Backus-Naur Form) [3].

3.2 XML DTD

Let E be an XML element, its definition $\underline{E}$ (E underscored) is given as follows:

$$\underline{E} \rightarrow \langle E; \underline{Attrs}; \underline{D} \rangle$$

Figure 2 Definition of XML element using to its DTD.

where

- E : is the name of the element;
- The symbols → denotes a definition or production;
- D: represents the content model of the element E eventually empty;
- Attrs: is a list containing the attributes of the element E, i.e.

Attrs → (Attr1, Attr2…).

The definition of "Attrs" that we note $\underline{Attrs}$ (underlined Attrs) is given by a list containing the definition of each attribute of "Attrs". This can be expressed as following:

$\underline{Attrs}$ →($\underline{Attr1}$, $\underline{Attr2}$…).

The Definition of each attribute is as follows:

$$\underline{Attri} \rightarrow \langle Attri; typeOrValues; Description \rangle$$

Figure 3 Definition of an XML attribute in terms of its DTD

where

- typeOrValues stands for type of the attribute or list of values in the XML model;
- The value of Description is given by :

Description→#REQUIRED|#IMPLIED|#FIXED value|value/

Figure 4 Definition of Description for an attribute

Obviously, the Meta symbol "|" denotes the alternative.

3.3 XML Schema

For XML Schema, we give notations for element and type that represent its fundamental components.

If E is an element in XML Schema, its definition is given by $\underline{E}$ (E double underlined):

$$\underline{E} \rightarrow \langle \underline{E}; Type \rangle$$

Figure 5 Definition of XML element in XML Schema.

The "Type" as shown in the following BNF expression, represents a type of an XML element.

Type→simpleType | complexType | simpleContent | complexContent

Figure 6 Simplified type in XML Schema

The next sections describe the transformations from XML Schema to ORM. In this article, we consider XML Schema in Russian Doll pattern with only one global element, the root element.

4. Converting XML Schema into Extended DTD

There is several software that map XML Schema into XML DTD but don't preserve constraints. To remedy this

[6] Structured Query Language is a standard language for databases.
[7] Terms used in Oracle DBMS.
[8] Backus-Naur Form: notation used to describe language grammars.

deficiency we extend the DTD schema to a new XML schema able to support the constraints. We call this schema extended DTD (in shortcut eDTD). The purpose of this section is to show how to map XML Schema into eDTD.

4.1 Converting XML Schema into Extended DTD

Let E be an element in XML Schema. Its definition is given by (see Figure 5):
$\underline{\underline{E}} \rightarrow <\underline{E}; \text{Type}>$.
Let Ψ be a function defined from XML Schema to eDTD.
If E is an element and Attr is an attribute, their images by Ψ are given by:
Ψ : {XML Schema language} → {eDTD language}

$$\underline{\underline{E}} \rightarrow \Psi(\underline{\underline{E}}) = \underline{E}_{extended}$$

$$\underline{\underline{Attr}} \rightarrow \Psi(\underline{\underline{Attr}}) = \underline{Attr}_{extended}$$

Figure 7 Function Ψ from XML Schema to DTD

The expression $\underline{E}_{extended}$ is defined as follows:

$$\underline{E}_{extended} \rightarrow <\underline{E}; \text{Constraints}>.$$

Figure 8 Representation of an element in eDTD

Where
- "E" (E underscored) represents the definition of E in DTD, see Figure 2;
- "Constraints" are constraints of element E in XML Schema language.

Similarly, for attribute Attr, we have the following representation:

$$\underline{Attr}_{extended} \rightarrow <\underline{Attr}; \text{Constraints}>.$$

Figure 9 Representation of an attribute in Extended DTD

Where
- Attr is the definition of Attr in DTD as shown in Figure 3;
- "Constraints" are constraints of the attribute "Attr" in XML Schema language that are not taking account in XML DTD.

In this paper, we consider three types of constraints, value constraints, type constraints and structure constraints, explained as follows:
- Value constraints: These constraints check the value of the element, especially element with simpleType. They are based on the values of minInclusive, maxExclusive attributes…;
- Type constraints: These constraints deal with nature, length of the element (integer, float, date, time) and check the value of type, base, length, minLength, maxLength attributes;
- Structure constraints: These constraints treat the values of minOccurs and maxOccurs attributes which are present in "sequence", "choice", "all" and simple element.

To obtain the value of these constraints, we consider the three followings functions:
- valueConstraint(element) that returns a list of value constraints of the element given in its argument;
- typeConstraint(element) that returns a list of type constraints; and
- structureConstraint(element) that returns structure constraint of the element given in its argument.

Before continue, we note that valueConstraint and typeConstraint functions can have an attribute as argument (i.e., we can call valueConstraint(Attr) where Attr is an attribute).

We now present the bodies of these functions.
The body of valueConstraint function is given below:

```
Function valueConstraint (element E) return constraints;
/*function called for element E that has a simple type*/
Cs: string; /* variable to concatenate all constraints*/
begin
Cs=""; // initialization
for each constraint c in (Bounds, Pattern, Enumerated values, default, fixed) of E loop
create a logic constraint using c;
/*let LC be the name of this constraint*/
set Cs=Cs +","+ LC; /* "+" denotes concatenation*/
end loop;
return Cs;
end;
```

Figure 10 valueConstraint function

The body of typeConstraint function is shown below:

```
Function typeConstraint (element E) return constraints;
/*function called for element E that has a simple type*/
Cs : string; /* variable for grouping all constraints*/
begin
Cs="";
for each constraint c in (type, base, Length, Precision) of E loop
create a logic constraint using c; /*let LC be the name of this constraint*/
set Cs=Cs + ","+ LC;
end loop;
return Cs;
end;
```

Figure 11 TypeConstraint function

The body of structureConstraint function is as follows:

```
Function structureConstraint (element E) return string;
/*function called for element E that has different default value for minOccurs and/or maxOccurs*/
v_minOccurs, v_maxOccurs : string;
begin
```

Let v_minOccurs be the value of minOccurs associated to element E;
Let v_maxOccurs be the value of maxOccurs associated to element E;
return v_minOccurs+ "," + v_maxOccurs;
end;

Figure 12 structureConstraint function

Using these functions, Constraints can be expressed as follows:

Constraints → (valueConstraint | typeConstraint | structureConstraint)*.

Figure 13 XML Schema Constraints

The best way to understand this conversion from XML Schema into eDTD is through an example, shown in the figure below.

```
<xs:element name="nb_pages">
<xs:simpleType>
<xs:restriction base="xs:integer">
<xs: minInclusive value="4"/>
<xs:maxInclusive value="10"/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

Figure 14 XML Schema of element "nb_pages"

In this schema, the name of the element is "nb_pages". To have the expression of "$\underline{nb\_pages}_{extended}$"(i.e. the expression of nb_pages and its constraints (see Figure 8), we need to calculate the image by Ψ of nb_pages and add to it nb_pages constraints defined in XML Schema.
The value of nb_pages, using formula in Figure 5, is
nb_pages=<nb_pages; Type>
where
Type is a string delimited by "<xs:simpleType>"
and "</xs:simpleType>".

The image by Ψ of nb_pages is given by
Ψ(nb_pages) = nb_pages
=<nb_pages; ;#PCDATA>,
since: 1) the correspondent type to integer in DTD schema is #PCDATA and 2) the structure of the element nb_pages in DTD schema is <!ELEMENT nb_pages (#PCDATA)>.

The constraints of nb_pages (see Figure 14) are:

- Type constraint: the type of nb_pages is integer (can be expressed by regular expression [0..9]+). The value of this constraint is returned by typeConstraint function defined in Figure 11.
- Value constraint: value of nb_pages must be between 4 and 10. The value of this constraint is returned by valueConstraint function defined in Figure 10.

Hence, the representation of nb_pages in eDTD is given by:
$\underline{nb\_pages}_{extended}$ = <nb_pages;
typeConstraint(nb_pages),
valueConstraint(nb_pages)>.

To illustrate the conversion that deal with structure constraints, we consider the example in the figure below:

```
<xs:element name="authors">
<xs:complexType>
    <xs:sequence  minOccurs="1" maxOccurs="5">
        <xs:element name="author">
        <xs:simpleType>
        <xs:restriction base="xs:string">
    <xs:minLength value="4"/>
    <xs:maxLength value="50" />
    </xs:restriction>
    </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:element>.
```

Figure 15 XML Schema of element "authors"

In this example, the definition of "authors" element is
authors=<authors;Type>.
The value of its "Type" is the string delimited by "<xs:complexType>" and "</xs:complexType>".
The reformulation of this XML Schema in DTD is given by

```
<!ELEMENT authors (author)+>
<!ELEMENT author (#PCDATA)>
```

Figure 16 DTD of element "authors"

If we apply to the element "authors", the notations defined in Figure 2 and Figure 7 we get
Ψ(authors) = authors
=<authors; ; (author)+>.
Similarly for the element "author", we get
Ψ(author) = author
=<author; ; #PCDATA).
To have the structure constraints for the element "authors" we call structureConstraint function described in Figure 12. Its result is "1,5" where "1" (resp. 5) is the value of minOccurs (resp. maxOccurs).

Then, the value of the element "authors" with constraints in extended DTD is

$\underline{authors}_{extended}$ =<authors;structureConstraint (authors)>

Figure 17 Representation of authors in extended DTD

and the value of author in extended DTD is given by

author$_{extended}$ =<author;valueConstraint(author)>

Figure 18 Representation of author in extended DTD.

Likewise we calculate the representation of the attributes of an XML Schema in extended DTD.
For instance, in the figure below, we have an XML Schema representing a journal element with two attributes id and issn.

```
<xs:element name="journal">
<xs:complexType>
    <xs:sequence>
    <xs:element name="titre" type="xs:string"/>
    </xs:sequence>
        <xs:attribute name="id" type="xs:ID"
    use="required"/>
    <xs:attribute name="issn">
    <xs:simpleType >
    <xs:restriction base="xs:string">
    <xs:length value="9" />
    <xs:pattern value="\d{4}\-\d{3}[/dX]"/>
    </xs:restriction>
    </xs:simpleType>
        </xs:attribute>
</xs:complexType>
</xs:element>
```

Figure 19 XML Schema of journal element

Let us look for the images by Ψ of id and issn attributes and their representation in extended DTD.
The DTD schema associated to the journal element is:

```
<!ELEMENT journal (title)>
<!ATTLIST journal id ID #REQUIRED
            issn CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>.
```

Figure 20 journal element in XML DTD

Using expressions in Figure 7, Figure 3 and Figure 5 the image by Ψ of "issn" attribute is specified by
Ψ(issn)=issn=<issn; CDATA; #IMPLIED>.
We have for issn$_{extended}$ (see Figure 9) the expression below:
issn$_{extended}$=<issn; Constraints>
where
"Constraints" is calculated by valueConstraint(issn) and typeConstraint(issn) as follows:

- valueConstraint(issn) is specified by the value of pattern of issn, we name this constraint valueCissn;
- typeConstraint(issn) is specified by the value of length of issn, we name this constraint typeCissn.

Then the expression of "issn" attribute in eDTD is

issn$_{extended}$=<issn; valueCissn, typeCissn >.

Figure 21 Value of issn with schema constraints.

With the same procedure, we obtain the expression of "id" attribute. In effect, we have
Ψ (id)=id =<id; ID; #REQUIRED> (see Figure 20),
and then
id$_{extended}$=<id; Constraints>.
In this last expression, the value of "Constraints" is empty, because "id" attribute have not value constraint and type constraint.

Hence, the expression of id in extended DTD becomes:

id$_{extended}$=<id>.

Figure 22 Value of id with schema constraints

So far, we have shown how an element E described in XML Schema (i.e. E: E double underlined) can be translated into eDTD (i.e. E$_{extended}$) which represents an intermediate schema. In the next sections, we describe the conversion from eDTD to Object Relational Schema (ORS) in order to complete our conversion methodology.

5. Transforming XML DTD into Object-Relational schema

In this section, we present algorithms to translate an eDTD schema into object-relational schema.

In order to take into account the constraints defined by XML Schema, this conversion uses the notations indicated in Figure 8 and Figure 9. These constraints will be added, for both element and attribute, to DTD constraints that we will calculate in this section.

Let us now consider the polymorphic[9] function ϕ which allows us converting an eDTD schema into Object-relational schema.
This function ϕ maps each element E of eDTD schema onto an object type[10] "E". In other words, the value of ϕ for a given argument E$_{extended}$ is an object type "E". So, we have the mapping

ϕ: eDTD → ORS
ϕ(E$_{extended}$) = E(list_of_attributes_definition)

Figure 23 Definition of the object type ϕ(E)

where "list_of_attributes_definition" is a list of the attribute definitions of the object type E.
To describe an attribute "Attr" of an object type we use the following notation:

<Attr; Type[;Modifiers] >,

Figure 24 Definition of an object type attribute

[9] Function having an arbitrary number of different types arguments.
[10] Object type is similar to UDT in the standard SQL: 2003.

where

- "Type": is the type of the attribute "Attr" in object-relational model;
- Modifiers: represents constraints of the attribute "Attr". These include schema constraints obtained at the previous section and those obtained from DTD of the element E, e.g., null, not null, unique, check and foreign key constraint. The brackets indicate that "Modifiers" is an option and may be empty as in extended BNF[11] notation [3].

Now, we show how to calculate the attributes of the object type "E" using the function ϕ.
We have in Figure 2, the production:
E → <E; Attrs; D>.
The definition of the object type "E" is given by the following formula:

$$\varphi(E) = E\,(\varphi(Attrs) \cup \varphi(D))$$

Figure 25 Definition of an object type.

where the symbol 'υ' denotes the union operator.
With this formula, we mean that the list of attributes of the object type E is obtained by the union of the image (by ϕ) of Attrs which represents the attributes list of the XML element E, and the image (also obtained by ϕ) of its content (i.e., D).
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).

5.1 Calculation of φ(Attrs)

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 DTD element.*/*
φ (Attrs)=""; //empty string
else
if Attrs=(Attr1, Attr2, ...) then
φ(Attrs) = (φ (Attr1), φ (Attr2) ...);
end if;
end if;
return φ(Attrs);
end;

Figure 26 Calculation of ϕ (Attrs)

The expression of Attri, as shown in Figure 3, is
Attri → <Attri; typeOrValues; Description>.

If we apply ϕ to Attri, we obtain
ϕ(Attri) = ϕ(<Attri; typeOrValues; Description>).
The value of ϕ (<Attri; typeOrValues; Description>) is specified by the formula:
ϕ (<Attri; typeOrValues; Description>) =
<Attri; ϕ(typeOrValues) minus Constraints;
ϕ(Description) + Constraints + Constraint_on_Attri>.

Figure 27 Definition of the attribute "Attri"

- This requires the calculation of:
- ϕ(typeOrValues),
- ϕ(Description),
- Constraints and
- Constraint_on_Attri. This is obtained using valueConstraint, typeConstraint and structureConstraint functions that are explained in subsection 4.1.

(a)*Calculation of φ(typeOrValues)*

The value of ϕ(typeOrValues) is a type and constraints on the attribute "Attri". This value is based on the following table. Constraints are defined using regular expressions[2] in which:

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

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

Figure 28 Calculation of ϕ(typeOrValues)

The two most-right columns of this table are explained as follows:

i) In the "Type" column, we have:
   - Varchar(n): a basic database type for strings. n is the maximum number of characters.
   - Varray[12](p): a data type representing a variable-length array in object-relational databases. p is the size of the collection.
   - Nested table[13]: a data type representing unlimited collection in object-relational databases.

ii) In the "Constraints" column, we have:

[11] Extended BNF notation: BNF notation including the symbols: (,),[,],{,}…

[12] Type of variable-length collection used in Oracle DBMS.
[13] Type of unlimited collection used in Oracle DBMS.

- patterns that attribute in object-relational model must respect in order to preserve the semantic of the XML element attributes. These patterns are similar for all constraints and expressed using the regular definitions [2]: Letter→[A..Za..z] and Digit→[0..9]. For simplification, we call this shared constraint: LexAttrConstraint (as contraction of Lexical Attribute Constraint);
- Foreign key Constraint (FKC) represents the usual referential integrity in the database literature;
- Unique Constraint (UC) indicats that attribute values are distinct;
- Enumerated List Constraint (ELConstraint): represents a constraint that limits the values of attribute in object-relational model to those enumerated in the content model of its corresponding in XML DTD

(b) Calculation of φ(Description)

In order to complete the calculation of ϕ(Attrs), we must calculate ϕ(Description). The value of ϕ(Description) is a list of usual constraints in database system. It is obtained using the following table:

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

Figure 29 Calculation of ϕ(Description)

To show how the function ϕ works, we consider in the example below, the element journal obtained from XML Schema, shown in Figure 19.
<!ELEMENT journal (…)>
<!ATTLIST journal id ID #REQUIRED>
<!ATTLIST journal issn CDATA #IMPLIED>

Calculus of ϕ(journal)
The element journal in this example has two attributes: id and issn.

If we apply ϕ to "journal" element we obtain:
ϕ(journal)=journal(ϕ(id), ϕ(issn),…).
"journal" (at the right of "=" symbol) is an object type with attributes ϕ(id), ϕ(issn)…
In order to have ϕ(journal) we must calculate
ϕ(id)= ϕ (<id;ID;REQUIRED>)
and
ϕ(issn)= ϕ (<issn;CDATA;#IMPLIED>).

i) Calculus of φ(id)

According to formula in Figure 27 and table in Figure 28, we have
ϕ (id) =<id; ϕ(ID) – (LexAttrConstraint+UC);
ϕ(#REQUIRED)+(LexAttrConstraint+UC)+
Constraints_on_id[14]>.
Moreover,
ϕ(ID)=varchar+(LexAttrConstraint+UC),
ϕ(#REQUIRED)=not null
and
Constraints_on_id="" (see Figure 22).
Hence, ϕ(id) becomes
ϕ(id)=<id; varchar; not null+LexAttrConstraint+UC>.
This signifies that ϕ(id) is an attribute of the object type "journal" with the following specifications:
- id: name of object-type attribute;
- varchar: type of attribute id;
- not null, LexAttrConstraint, and unique are constraints on id.

ii) Calculus of φ(issn)
If we apply the same procedure for ϕ(issn), we obtain the following expression
ϕ(issn)=<issn; varchar; null + Constraint_on_issn>
So, the expression of object type "journal" with its attributes is:
journal(
<id;varchar;not null+LexAttrConstraint+UC)>
<issn; varchar; null + Constraint_on_issn>,…
).
That's all for the image of the element journal (i.e. ϕ(journal)).

We can recapitulate these steps in the following algorithm:

Algorithm XML_Attribute_to_Object_Attribute;
Input Attri: an attribute of a DTD element;
Output φ(Attri) : an object- type attribute;
Begin
Calculate φ(typeOrValues);
Calculate φ(Description);
Return <Attri; φ(typeOrValues) minus Constraints;
φ(Description) plus Constraints>;
End;

Figure 30 Algorithm that map an XML attribute onto an Object-type attribute

5.2 Calculating φ(D)

We recall that the expression of ϕ(E) (see Figure 25) is given by :
ϕ(E)=E (ϕ(Attrs) U ϕ(D)).
We have already shown how to obtain ϕ(Attrs). We now calculate ϕ(D).
We have in Figure 2:
E=<E; Attrs; D>
where D is the content model of the element E in XML DTD.

[14] Constraints from XML Schema on id attribute.

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

To see what it can be the value of D, we consider the example below:

<!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 31 Example of DTD

Here, the element "journal" has the following representation:
journal=<journal; id, category ; volume+>.
The value of D for element "journal" is "volume+".
The definition of the element "volume" is
volume=<volume; ; issue+>.
Similarly, the element paper has the following representation:
paper=<paper; ; title,author>.
The value of D for the element "paper" is
"title, author".
The representation for the element "title" is given by
title = <title;;#PCDATA>

Figure 32 Definition of the element title.

and likewise the element "author" has the expression :
author =<author;; #PCDATA>.
Hence, the value of D for both "title" and "author" is #PCDATA.

By definition, elements of D are connected together by sequence, alternative, Kleene closure, transitive closure and optional value [7].

In order to simplify the calculation of ϕ(D), we use the following BNF (Backus-Naur Form) grammar for representing the content model. We call this grammar G (E is an element of DTD).

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 33 BNF (Backus-Naur Form) grammar G.

To have the value of E which represents the definition of element E (see Figure 2), we associate to the grammar G the following grammar which we note G.

a) E → ANY
b) E → EMPTY
c) E → E, E
d) E → E1+E2
e) E → {E}
f) E → {E}, E
g) E → [E]
h) E →#PCDATA

Figure 34 BNF (Backus-Naur Form) grammar G.

Now, if we apply ϕ to each item of the grammar G, we obtain a grammar shown in figure below. We call this grammar ϕ(G).

a) φ(E)→ φ(ANY)=AnyData[15] or AnyType[16]. (Generic type in object-relational model).
b) φ(E)→ φ(EMPTY)=Empty string
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 35 BNF (Backus-Naur Form) grammar ϕ(G).

The value of ϕ(#PCDATA), that represents a simple type of an element E, is given by the following expression:

φ(#PCDATA)=<value;varchar; PCDATA_Constraint plus Constraints_on_E>

Figure 36 Value of ϕ(#PCDATA)

where
- value is an attribute of the object type containing the value of the XML element;
- Varchar representing the type of the attribute value;
- PCDATA_Constraint is a constraint on "value" attribute. It is defined by the following regular expression[2]: (Letter | _ | Digit | . | - | :)*.
- Constraints_on_E: represents constraints defined by XML Schema of E. we recall that
- $E_{extended}$=<E; Constraints_on_E) (see Figure 8).

To see how the function ϕ converts an XML DTD into an object-relational model, we propose below some conversion examples.

[15] Type used in Oracle DBMS.
[16] Type used in Oracle DBMS.

(a) Example 1

For element "title", we have (see Figure 32)
title=<title;;#PCDATA> and *$title_{extended}$=<title;Constraints>* (see Figure 8)
If we apply the function ϕ on "title", 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).
Afterwards we replace ϕ(#PCDATA) with its value using the grammar ϕ(G) and get the final expression of ϕ(title):
ϕ(title)=title(<value;varchar;PCDATA_Constraint+ Constraints>).
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 both "*PCDATA_Constraint*" and "Constraint" constraints.

(b) Example 2

We can do the same for the element "author" defined by (see Figure 18)
author = <author; ; #PCDATA> and
$author_{extended}$ =<author; authorCValue>
and we get
φ(author)=author(φ(#PCDATA))
=author(<value;varchar;
PCDATA_Constraint + authorCValue>).

(c) Example 3

For a complex example that illustrates 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 + Constraints>),
author(<value; varchar;
PCDATA_Constraint+ Constraints>)).
Hence, "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, 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 30;
    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 37 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 38 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 its value *φ(title, author, [cite])*
in last formula, we get
φ (paper) = paper(ϕ(title, author, [cite])).
Then
φ(paper)=paper(φ(title),φ(author),[φ(cite)]).

Figure 39 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 obtain:
φ(cite) = φ(<cite;;{paper}>)
=cite(φ ({paper}))
= cite({φ (paper)}).
Replacing *φ(title), φ(author) and φ(cite)* in Figure 39 with their values, we obtain
φ(paper)=paper(φ(D))
=paper(title(<value;varchar; PCDATA_Constraint+ Constraints>),
author(<value; varchar;
PCDATA_Constraint+ Constraints>),
[cite({φ (paper)})]]

Figure 40 Value of ϕ(paper)

We notice that we have found *φ(paper)* in *φ(D)* which is a definition for the "paper" element. Hence, the process

halts here since there is no more *φ(v) in* ϕ(D) without recursion nor ϕ(v) different to ϕ(paper).

6. Algorithm of conversion

In this section, we present the conversion algorithm from XML DTD to object-relational model. This algorithm uses "*CreateObjectType(…)*" function that creates an object type representing the image by ϕ of an XML element. Its body, presented in subsection 6.2, calls the function "C*reateObjectAttribute(attr(…))*" detailed in the next subsection 6.1.

6.1 Creation of object type attribute

To obtain the attributes of an object type, we use the function "*CreateObjectAttribute*" which takes as argument an attribute with a list of items and returns the expression:
"*<attr; typeOfAttribute; modifiers>*" .
This expression, as seen in Figure 24, represents a definition of the attribute "attr" in object-relational model.
The body of this function is given in the figure below:

```
Function CreateObjectAttribute (attr(listOfItems)) return ObjectAttribute ;
y ObjectAttribute ; //y is variable for an object attribute.
i integer initialized by 0;
/*the variable i is counts the number of attributes which are added in the case of the alternative that is not surrounded by an element This case is treated at line 34)*/
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 having its attributes corresponding to 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 (e.g., ANYDATA);
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
     //(e.g., <!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;>;
     /*ref type is a type that allows an attribute to contain an object reference.*/
46)  Else /*Case of elements mutually recursive.*/
47)  If e(…) matches e(…,φ(x),…) then
48)  If the x type is not yet created then
49)  Create the object type x as incomplete type (in
```

```
                order to have recursion);
50)              End if;
51)              Replace φ(x) by <"x";ref x;>;
52)            End if;
53)         End if;
54)       End if;
55)      End loop;
56)     End if
57)    End if;
58)  End loop;
End; /*End of function : CreateObjectAttribute */
```

Figure 41 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 42 Example of DTD

We have in Figure 39 the expression:
φ(paper)=paper(φ(title), φ(author),[φ(cite)]).
Let us apply *CreateObjectAttribute* function to *paper(φ(title), φ(author),[φ(cite)]).*
In this case :

- attr *is "paper"* and
- listOfItems is "ϕ(title), ϕ(author),[ϕ(cite)]".

To obtain the image of paper by *CreateObjectAttribute* function, we have to find ϕ(title), ϕ(author) and [ϕ(cite)].
The value of ϕ(title) (as seen above) is given by
φ (title)=title(<value; varchar; PCDATA_Constraint + Constraints >).
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 + Constraints>
We can do the same for ln:
ϕ(ln)=ln(<value;varchar;
PCDATA_Constraint + Constraints >.
Then ϕ(author) becomes
ϕ(author)= author(*fn(<value; varchar;*
PCDATA_Constraint + Constraints>),
ln(<value; varchar;
PCDATA_Constraint + Constraints>)).
The value of [ϕ(cite)] is
[cite({ϕ (paper)})] (see Figure 40).

If 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 + Constraints>),
author(fn(<value; varchar;
PCDATA_Constraint + Constraints >),
ln(<value; varchar;
PCDATA_Constraint + Constraints >)),
[cite({φ (paper)})]).
Applying the CreateObjectAttribute function to paper, we transform recursively:

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

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

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

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*. The element *"title",* as seen earlier, has the expression:
title(<value;varchar;PCDATA_Constraint+ 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 +Constraints >.
Similarly, for fn and ln element, we obtain the two following object-type attributes:
<"fn";fn;*PCDATA_Constraint on* "fn".value + *Constraints* >
and
<"ln";ln; *PCDATA_Constraint on* "ln".value+ *Constraints* >.
For convenience, we suppose that we have XML Schema constraints "Constraints" for both fn and ln elements (for instance the value of length facet).
Now, search CreateObjectAttribute (author(…)) for the author element.

We have ϕ(author)=author(*fn(<value; varchar; PCDATA_Constraint + Constraints >)*,
ln(<value; varchar; PCDATA_Constraint + Constraints >)).
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 + *Constraints*> (obtained by CreateObjectAttribute(fn(…)))
and
<"ln";ln; *PCDATA_Constraint on* "ln".value + *Constraints*> (obtained by CreateObjectAttribute(fn(…))).
After this substitution, author becomes
author(<"fn";fn; *PCDATA_Constraint on* "fn".value + *Constraints* >,
<"ln";ln;*PCDATA_Constraint on* "ln".value+ *Constraints* >)
Then, we can 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 + Constraints >,
<"author"; author; fn_constraint +ln_constraint>,
<"cite";cite;null_constraint>).
Finally, we obtain the object type:
<"paper";paper;constraint_on(title,author,cite)>.
Now, let us see how this function works in the case of the alternative. For this, we propose the example below:
<!ELEMENT person (name, (email | phone))>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>

First, we calculate *φ(person).*
We have
ϕ(person)=person(ϕ(name, (email | phone))>.
If we use the algorithm at Figure 37, we obtain
person=person(ϕ (name), ϕ (email | phone)),
which becomes
person=person(
name(<value;varchar; PCDATA_Constraint + Constraints >),
(+,email(<value;varchar;PCDATA_Constraint + Constraints>),
phone(<value; varchar; PCDATA_Constraint+ Constraints >))).

Now, we apply "*CreateObjectAttribute*" function to "person (…)":

- with lines 40, 41 and 42, then lines between 19 and 23, we transform "name(<…>)" to

<"name";name;PCDATA_Constraint on name.value + schemaConstraints>;

- 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 + Constraints >;
and for phone: <"phone";phone;PCDATA_Constraint on phone.value+Constraints *>)*>;
Thus "*_person_1"* becomes
_person_1(+,<"email"; email;
PCDATA_Constraint on email.value+ Constraints>,
<"phone"; phone; PCDATA_Constraint on phone.value + Constraints >);

- 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 + Constraints >;
<"_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…>

6.2 Creation of the objet type associated to XML DTD

Now we consider the function *CreateObjectType*. It takes an object obtained by applying the function ϕ to *root* element of XML DTD document and returns an object type (UDT) with constraints. This function is called only with this type of object. Its body is presented below:

```
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 43 CreateObjectType function

If we call *CreateObjectType* with object paper (that we suppose the root of document) defined by
<"paper"; paper; constraint_on(title,author,cite)>,
we obtain the object type
<paper, constraint_on(title,author,cite)>.
We present in the following subsection the algorithm of conversion.

6.3 Algorithm of conversion

The algorithm of conversion takes a valid XML document with its XML Schema and returns an object-relational schema through XML DTD.

Algorithm Conversion;
Input: a valid XML document with its XML Schema; Let be E the root of this document;
Output: an object-relational schema;
Begin

1) *Calculate Ψ(E)= E /* see Figure 7*/*
2) *Calculate φ(E) using the rules presented above in Figure 35;*
3) *Let be "E(listOfItems)" this value;*
4) *Let be <E, Constraints> the object type obtained by*

CreateObjectType(E(listOfItems));
/*algorithm at Figure 43*/

5) *Create an object table named "E_Table" with object type E and constraints defined by E;*

/"E_Table" is a an object table in which we store the content of the XML document.*/*
*End; /*end of Conversion*/*

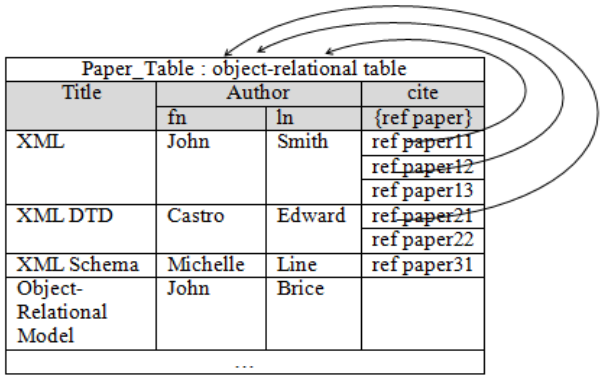
Figure 44 Algorithm of Conversion

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 5) of conversion algorithm. The object type of this table is the root element of the XML document.
Let us apply this algorithm to the following example that represents an XML Schema for the element named paper.

```
<xs:element name="paper">
<xs:complexType>
<xs:sequence>
<xs:element name="title">
    <xs:simpleType><xs:restriction base="xs:string">
    <xs:length value="50"/></xs:restriction>
    </xs:simpleType></xs:element>
    <xs:element name="author">
    <xs:complexType><xs:sequence>
    <xs:element name="fn">
        <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:length value="15"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="ln">
        <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:length value="15"/>
        </xs:restriction>
      </xs:simpleType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="cite"  minOccurs="0">
<xs:complexType><xs:sequence>
<xs:element ref="paper"  minOccurs="0"
maxOccurs="unbounded"/>
</xs:sequence></xs:complexType></xs:element>
</xs:sequence></xs:complexType>
</xs:element>
```

Figure 45 XML Schema for paper

First, we create the image by Ψ of the element paper (i.e. Ψ(paper)=$paper_{extended}$, see Figure 7). The structure of paper is shown in Figure 42.
Next we create the image of paper by ϕ (i.e. ϕ(paper)=paper, see Figure 23).
Finally, 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 is given below:



Paper_Table : object-relational table			
Title	Author		cite
	fn	ln	{ref paper}
XML	John	Smith	ref paper11
			ref paper12
			ref paper13
XML DTD	Castro	Edward	ref paper21
			ref paper22
XML Schema	Michelle	Line	ref paper31
Object-Relational Model	John	Brice	
...			

Figure 46 Structure of paper_table

Conclusion

In this paper, we have presented a methodology to translate XML Schema into object-relational schema using mapping composition. The XML Schema is converted into extended XML DTD which in turn is converted into object-relational schema.

This methodology conserves integrity constraints and, comparing to others methods, integrates XML elements in a few object tables.

In perspectives, we envisage use this technique of composition to convert other XML schemas into database models through DTD.

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 Edition ed., 2007.
[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] P. V. Biron and A. Malhotra, "Xml Schema Part 2: Datatypes.," W3C, 28 October 2004, Second Edition [http://www.w3.org/TR/xmlschema-2/].

[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, J. Paoli, and C. M. Sperberg-Mcqueen, "Extensible Markup Language (Xml) 1.0," *W3C, http://www.w3.org/TR/REC-xml,* February 1998.

[7] T. Bray, J. Paoli, C. M. Sperberg-Mcqueen, and E. Maler, "Extensible Markup Language (Xml) 1.0 (Second Edition)," *W3C Recommendation. http://www.w3.orgITR2OOOlREC- XML-20001006l,* 2000/10.

[8] E. Castro, D. Cuadra, and M. Velasco, "From Xml to Relational Models," *Informatica,* vol. 21(4), pp. 505-519, 2010/12.

[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] C. J. Date, "Preview of the Third Manifesto," Database Programming & Design Journal (San Francisco, CA: Miller Freeman Publications), vol. 11(8), 1998(8).

[12] C. J. Date and H. Darwen, Databases, Types and the Relational Model: The Third Manifesto, 3 ed.: Addison-Wesley, 2007.

[13] A. Davidson, M. Fuchs, and M. H. e. Al., "Schema for Object-Oriented Xml 2.0," *W3C, July 1999.(http://www.w3.org/TR/Note-SOX)*.

[14] A. Eisenberg and J. Melton, "Sql:1999, Formerly Known as Sql3," *SIGMOD Record,* vol. 28(1), March 1999.

[15] A. Eisenberg, J. Melton, K. G. Kulkarni, J.-E. Michels, and F. Zemke, "Sql: 2003," *SIGMOD Record,* vol. 33(1), pp. 119-126, 2004.

[16] D. C. Fallside and P. Walmsley, "Xml Schema Part 0: Primer," *W3C, 28. October 2004, Second Edition. [http://www.w3.org/TR/xmlschema-0/]*.

[17] 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.

[18] 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.

[19] D. Lee, M. Mani, and W. W. Chu, "Solving Schema Conversion Problem between Xml and Relational Models: Semantic Approach," *ResearchGate,* 2003/7.

[20] Microsoft, "Xml Schema Developer's Guide," *Internet Document, May 2000. (http://msdn.microsoft.com/xml/ XMLGuide/schemaoverview.asp)*.

[21] S. Navathe and R. Elmasri, "Fundamentals of Database Systems," ed: Addison-Wesley, 2011, pp. 353-413.

[22] J. Price, "Oracle Database 11g Sql," ed: McGraw -Hill, 2008, pp. 379-473.

[23] J. W. Rahayu, D. Taniar, and E. Pardede, *Object-Oriented Oracle*: IRM Press, 2006.

[24] 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.

[25] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn:, "Xml Schema Part 1: Structures," *W3C, 28. October 2004, Second Edition [http:// www.w3.org /TR/xmlschema-1/]*.