Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 47 (2005) 437–462
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Translating relational schema into XML schema definition with data semantic preservation and XSD graph

Joseph Fong*, San Kuen Cheung

Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong, China

Received 10 May 2004; revised 21 September 2004; accepted 28 September 2004

Abstract

Many legacy systems have been created by using relational database operating not for the Internet expression. Since the relational database is not an efficient way for data explosion, electronic transfer of data, and electronic business on the Web, we introduce a methodology in which a relational schema will be translated to an Extensible Markup Language (XML) schema definition for creating an XML database that is a simple and efficient format on the Web. We apply the Indirect Schema Translation Method that is a semantic-based methodology in this project. The mechanism is that the Relational Schema will be translated into the conceptual model, an Extended Entity Relationship (EER) Model using Reverse Engineering. Afterward, the EER model will be mapped to an XML Schema Definition Language (XSD) Graph as an XML conceptual schema using Semantic Transformation. Finally, the XSD Graph will be mapped into the XSD as an XML logical schema in the process of Forward Engineering, and the data semantics of participation, cardinality, generalization, aggregation, categorization, N-ary and U-ary relationship are preserved in the translated XML schema definition.

Keywords: Relational schema; XML schema definition language; XSD graph; Extended entity relationship model; XML conceptual schema; Semantics transformation

1. Introduction

On the Internet, data retrieval is an important issue relating to the growth of Internet highway. Many companies use legacy systems without Internet-oriented expression such as hierarchical, relational, and object-oriented databases for their daily interoperations on the Internet between users and companies. To overcome this issue, the XML database becomes an important database structure in presenting and storing data for the Internet world. However, XML database has its limitation on presenting its structure using semantics to store data. Biancheri et al.[1] dictated the choice of storing their data using relational database instead of native XML because such a system would enable them to store a larger set of information. In the data storage, the XML technology will be improved for storing the larger set of information. At present, system analyst has no toolset for modeling and analyzing XML system. Our solution applies an XSD Graph as a toolset for XML database to ease analyzing the structure of XML database as shown in Fig. 1. The tree diagram of XSD Graph represents inter-relationship of different elements inside a system. It is important not only for visualizing, specifying, and documenting structural models, but also for constructing executable systems through forward engineering [2] (from design to implementation).

Many papers have focused on Document Type Definition (DTD). We have focused on XML schema definition (XSD). We use XSD Graph to represent the conceptual schema of XML model. The model is introduced in this paper. Many papers have used XML tree to expand not only for the elements, but also for the attributes and data. The benefit is that reader can analyze all components of XML schema on the tree. Ng [3] extended the notion of functional dependency (FD) and compared the values of leaf nodes in a specified context of its corresponding XML tree to form an integrated XML tree. The XML tree consisted of elements, attribute, and data value. Vincent and Jixue Liu [4] modeled an XML

* Corresponding author.
E-mail address: csjfong@cityu.edu.hk (J. Fong).

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.09.010

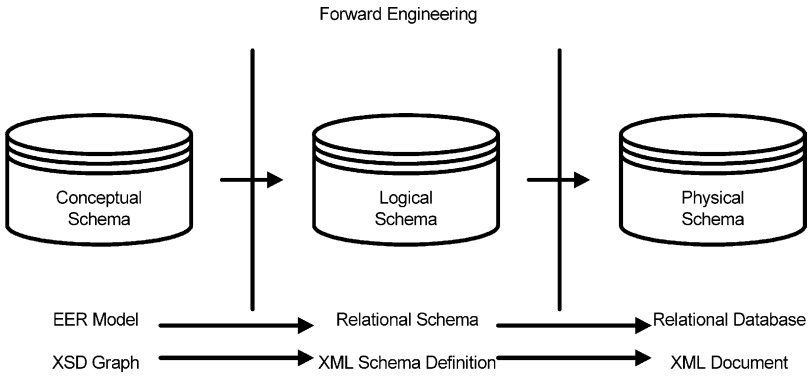


Fig. 1. Architecture of data model.

document as a tree which consisted of element, attribute, and data value. They proposed FD of XML with justification by mapping a relation to a XML document assisting by the XML tree. Yue[5] applied key constraint to create an instance XML tree with element, attribute, and data value.

We propose XSD Graph for representing the conceptual schema of XML model because it shows the data semantics in a more user-friendly approach to readers. In our approach, relational schema is mapped into an EER model in reverse engineering (from implementation to design), and then we map XSD Graph into XML Schema Definition Language [6] as shown in Fig. 2.

The followings are the definition of the data semantics constraints notations:

Functional dependency (FD). A functional dependency is a statement of the form $X \rightarrow Y$, where X and Y are sets of attributes. The FD: $X \rightarrow Y$ holds for relation R if whenever s and t are tuples of R where $s[X] = t[X]$, then $s[Y] = t[Y]$.

Inclusion dependency (ID). An inclusion dependency is a statement of the form $X \subseteq Y$ such that X is a subset of Y. For example, X is a foreign key of a child relation and Y is a referred primary key of its parent relation.

Multi-valued dependency (MVD). Let R be a relation, and let X, Y, and Z be attributes of R. Then Y is multi-dependent on X in MVD: $X \rightarrow\rightarrow Y|Z$ if and only if the set of Y-values matching a given (X-value, Z-value) pair in R depends only on the X-value and is independent of the Z-value.

The structure of paper is organized as follows. Section 2 presents the related works from other researchers. Section 3 presents our methodology of indirect schema translation. We apply the methodology in a case study in Section 4. Section 5 shows a prototype and Section 6 concludes our paper.

2. Related work

Nicolle [7] created an X-Time that consisted of three layers: specification layer, grammar integration layer, and translation layer. He used the XML model to integrate all other models for creating the universal data management system. Fong [8] used XML-based topology for system integration between relational model and XML model. The topology consisted of four different types: (1) functional dependency; (2) multi-valued dependency; (3) join dependency; and (4) M: N cardinality. Chen [9] used XML Schema hierarchical tree to construct a toolkit of enterprise model-based software development and application framework. VXMLR [10] is a system for translating between path expression queries and SQL. They implemented this system by using two data sets. Ndez [11] used the XML model for data exchange between inter-enterprise applications. They called the resulting XML document a VIEW and they proposed SilkRoute for their methodology. XRel [12] is a novel approach for storage and retrieval of XML documents using relational databases. The four schemas are named as Element, Attribute, Text, and Path. A nested and self-describing structure provides a simple yet flexible means for applications to exchange data in XML [13]. Relational tables are flat, while XML documents are tagged, hierarchical and graph-structured. They proposed a few approaches as 'Late/Early tagging', 'Late/Early structuring', 'Inside the Engine' and 'Outside the Engine'.

XPERANTO [14] is a middleware for translating from XML-QL to SQL for the underlying object-relational database system. Clients use an XML-based RPC facility to communicate with the XPERANTO middleware for creating and querying XML views and retrieving the XML results. X-Database [15] was developed for loading an XML schema into a relational database. One important contribution is its

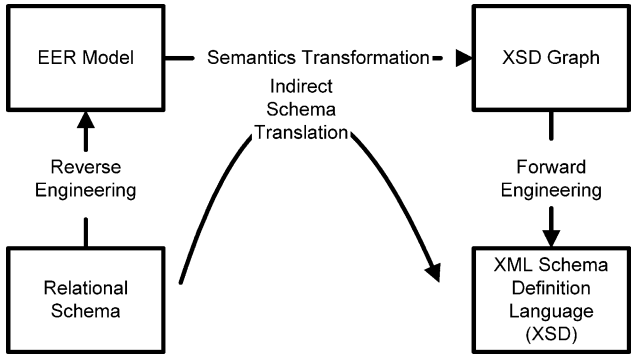


Fig. 2. Processes of recapture data semantics.

own DBCommand for manipulating relational database using XML documents such as DBInsert, DBUpdate, DBDelete, and DBSelect. XML [16] tags describe the data itself. Since XML data is self-describing, it is possible for programs to interpret the data. This means that a program receiving an XML document can interpret it in multiple way, filter the document based upon its content and restructure it to suit the application's needs. Khan [17] proposed a technique for automatic mapping from an XML document to relations within the RDBMS. They constructed a QPathId from XPath and then used an SQL statement to retrieve rows that match this QPathId. Mediator [18] integrates and transforms data from one or several sources using a declarative specification. The key technology consists in query rewriting that means for both data conversion and data integration.

Pare [19] suggested that two foreign keys in table R can be used to construct the ternary relationship among three tables. Pare applied functional and inclusion dependencies for proving his concept. Dahchour and Pirotte [20] presented a reification approach for transforming an N-ary relationship into a new class with n new binary relationships. They proved their theory by using several primitives as cardinality, existence dependency, attribute propagation, exclusion dependency, and inclusion dependency.

Conrad et al. [21] considered designing Object-oriented software (UML) with the XML structure (DTD). They addressed that the DTD lacked clarity and readability. Therefore erroneous design and usage were inevitable. Even though the UML can produce some data semantics such as aggregation, generalization, and association, the UML is in object-oriented view, which is not suitable for the hierarchical structure of the XML model.

Kleiner and Lipeck [22] introduced an automatic generator for translating from conceptual schema (ER model) to XML DTD. They focused on cardinality (binary and N-ary), composite attribute, multi-valued attributes, weak entity, and strong entity. Some data semantics cannot be presented on their paper.

FIXT [23] used the Structural Graph to extract index and sub index for efficient XML transformation for producing a graphical structure as a conceptual view. The graph consisted of root, element, attribute, and data value. XTABLES [24] proposed DTD Graph that mirrored the structure of DTD. The graph consisted of elements, attributes, and operator. The function of DTD Graph is for constructing the desired relational schema. However, they did not mention about the data semantics on their graph.

We focused on XSD and XSD Graph with preserving data semantics as aggregation, generalization/isa, participation, cardinality, categorization, N-ary, and Unary. Our objective is to preserve as much structural information from the relational schema as possible. We have chosen XSD as a logical schema because it is extensible and powerful than Document Type Definition (DTD). We know that XML model is in hierarchical structure. XSD Graph provides a hierarchical view, which closely relates to XML model. In our approach, we map a relational schema into an XSD. In the process, the XSD Graph acts as a diagrammatic representation of XSD. It helps user comprehend the constraints within an XSD.

3. Methodology of indirect schema translation

The first process extracts all features from Relational Schema into EER Model. The second process maps the conceptual schemas from EER model to XSD Graph. The third process translates XSD Graph into XSD. The following processes are outlines of our methodology of indirect schema translation from relational schema to XSD through EER model and XSD Graph.

Process 1: Reverse Engineering—Relational Schema to EER Model

Direct translation is a logical schema translation from relational into XML. In the indirect translation, we recover data semantics of the relational schema into an EER model using reverse engineering, which can be processed in two methods. The first method is mapping a relational schema into an EER model by classification table [25]. The idea is to identify the primary key, composite key and foreign key of each relation in a classification table and classify their relationships in terms of data semantics constraints. The second method is mapping relational database into an EER model by Data Exhaustive Search Algorithm [26]. The idea is to search the data volume of all the attributes in the relations, and derive the data semantics among attributes according to the data volume of their relationships.

Process 2: Semantics Transformation—EER Model to XSD Graph

The transformation between EER model and XSD Graph is a semantic-based methodology. It consists of two different rules composed by 11 steps. The first is General Rule for outlining the basic framework between EER model and XSD Graph. The second is Semantic Rule for capturing relationships and constraints among entities. Besides mapping an EER model to an XSD Graph, we preserve the data semantics of the source relational schema in the target XSD in a hierarchical [27] tree model.

Process 3: Forward Engineering—XSD Graph to XSD

We choose XSD because it is more extensible and powerful than Document Type Definition (DTD). The XSD describes relationship among elements and data type of each element. It supports new data type, inheritance, and namespace, and allows similar attributes to assign as a group. The XSD defines structure and content as well as semantics that can be described in an XSD Graph.

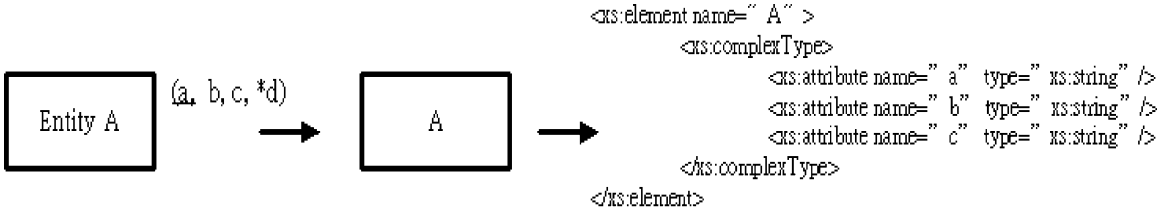
3.1. Indirect schema translation

3.1.1. General rule

Step 1 'Root' relation → Root element

In general, table formatted relational schema and tree formatted XSD are in different data structures. The XSD

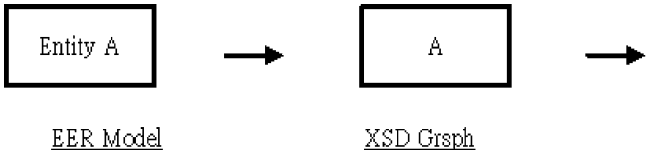
Graph in XML hierarchical tree structure is based on its root elements. Other sub-elements under the root elements must be relevant to the root element. We can only translate a 'tree' view of a relational schema into a hierarchical tree structured XSD. The users need to select a relation as a 'root' relation to be mapped into a root element of XSD. This selected 'root' relation and its relevant relations will be mapped into the elements of an XSD. The relevance of the relations depends on the navigability of the hierarchical tree of the elements. Relations that are in one to many as Primary keys, Foreign keys and Composite keys. In XML model, a unique attribute can be represented as a 'key'. Since 'key' is used in Xpath to specify a node of the element, the 'key' tag can be used in any attributes. Thus, the primary key of EER model is presented by <key> tag in the XML model. Foreign key d is eliminated in the translated XSD because the foreign key between a parent relation and child relations in relational schema is transformed into the hierarchical structure between an element and its sub-elements in an XSD.



cardinality and in superclass to subclass (isa) relationship in a relational schema are considered to be navigable (transferable from top to bottom in a tree structure) and can be mapped into an XSD after schema translation.

Step 2 Entity → Element

'Entities' of EER model are represented as Elements of XML model and 'Attributes' of EER model are represented as 'Attributes' of XML model. We use sub-element for applying the cardinality primitive in XML model. If we find the multi-valued attributes, we place them as sub-elements with 'maxOccurs = unbounded' in the XSD as shown below:



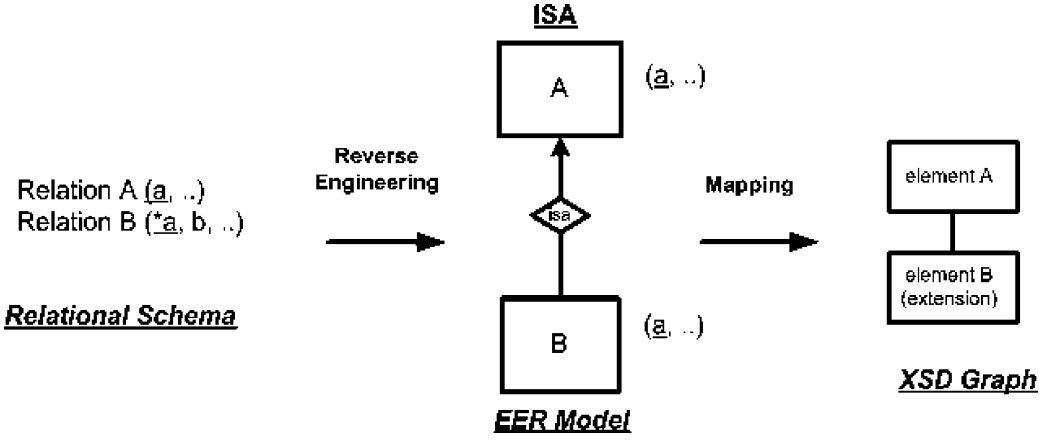
Step 3 Foreign Key → Element/sub-element relationship

The relationship of each entity relies on several keys such

3.1.2. Semantic Rule

Step 4 Isa → Element extension

The relationship between the sub-type and super-type is an 'isa' relationship. When we transform an EER model to an XSD Graph, we consider XSD, which consists of substitution group. We can use the 'extension' tag for the 'isa' relationship. The 'complexType' feature can be applied for this primitive in XML model. The child 'complexType' inherits properties of parent 'complexType' by applying the 'extension' tag on the child definition. Its attributes can be added on to complete the 'complexType' definition.



Definition 1.

EER :	XML :
FD : B.a → A.a ⇒	FD : Instance B → Instance A

The functional dependency is applied in this case. The 'element A' is an element in the XSD Graph, whereas the 'element B' is a sub-element.

Translated XSD

Isa constraint using 'extension' keyword

```
<xs:element name="b" type="b_type"/>
<xs:complexType name="b_type">
  <xs:complexContent>
    <xs:extension base="a"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="a">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

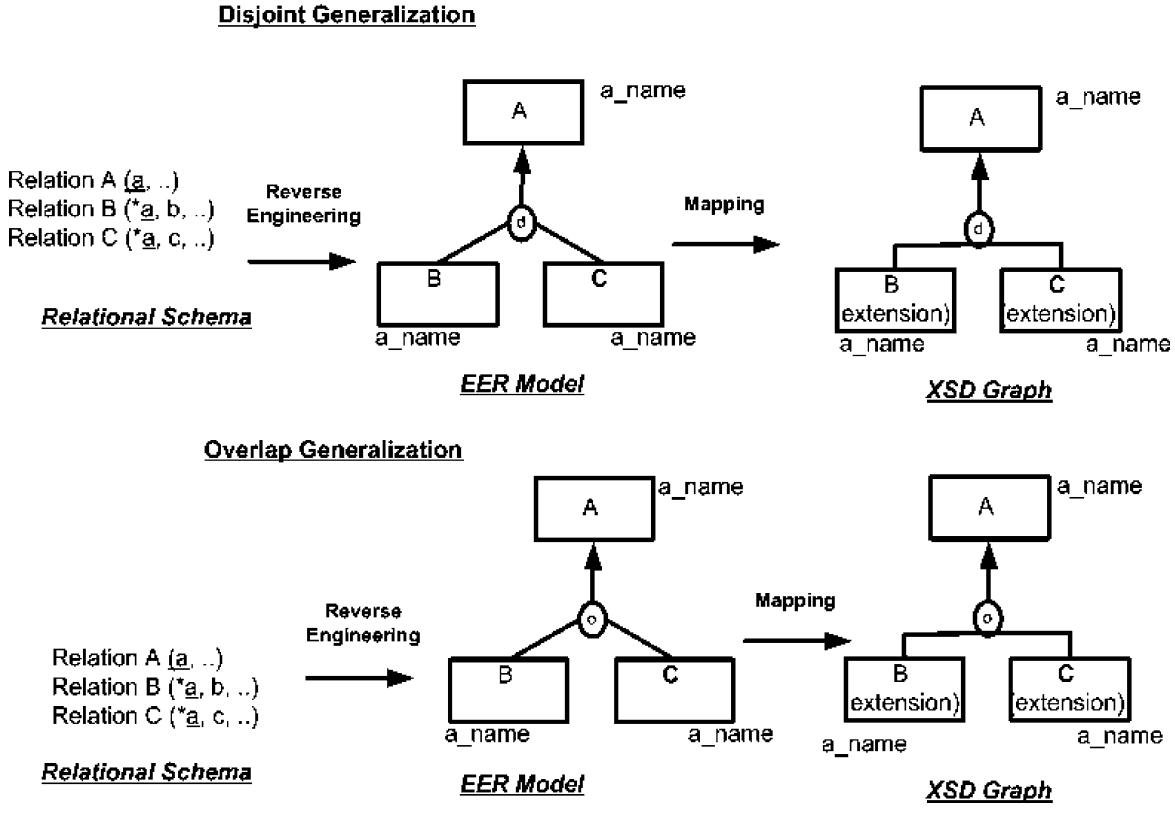
Alternative for Isa with condition using 'restriction' keyword

If we want to inherit selected properties from the top-level element, we can use the following declaration.

```
<xs:element name="b" type="b_type"/>
<xs:complexType name="b_type">
  <xs:complexContent>
    <xs:restriction base="a">
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="a">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="address" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Step 5 Generalization → Element Extension

Generalization is the concept that some entities are subtypes of other entities. The disjoint generalization is mapped into a complex element such that its component elements are mutually exclusive by use of 'choice' keyword. The overlap generalization is mapped into a complex element such that its component elements can be overlapped.



XSD in disjoint generalization	*XSD in overlap generalization*
<xs:element name="A"> <xs:complexType> <xs:sequence> <xs: choice> <xs:element ref="b"/> <xs:element ref="c"/> </xs:choice> <xs:element name="a_name" type = "xs:string"/> </xs:sequence> </xs:complexType> </xs:element> <xs:element name="b" type="b_type"/> xs:complexType name="b_type"> <xs:complexContent> <xs:extension base="a"/> </xs:complexContent> </xs:complexType> <xs:element name="c" type="c_type"/> <xs:complexType name="c_type"> <xs:complexContent> <xs:extension base="a"/> </xs:complexContent> </xs:complexType> <xs:complexType name="a"> <xs:sequence> <xs:element name="a_name" type="xs:string"/> </xs:sequence> </xs:complexType>	<xs:element name="A"> <xs:complexType> <xs:sequence> <xs:element ref="b"/> <xs:element ref="c"/> <xs:element name="a_name" type="xs:string"/> </xs:sequence> </xs:complexType> </xs:element> <xs:element name="b" type="b_type"/> <xs:complexType name="b_type"> <xs:complexContent> <xs:extension base="a"/> </xs:complexContent> </xs:complexType> <xs:element name="c" type="c_type"/> <xs:complexType name="c_type"> <xs:complexContent> <xs:extension base="a"/> </xs:complexContent> </xs:complexType> <xs:complexType name="a"> <xs:sequence> <xs:element name="a_name" type="xs:string"/> </xs:sequence> </xs:complexType>

Definition 2.

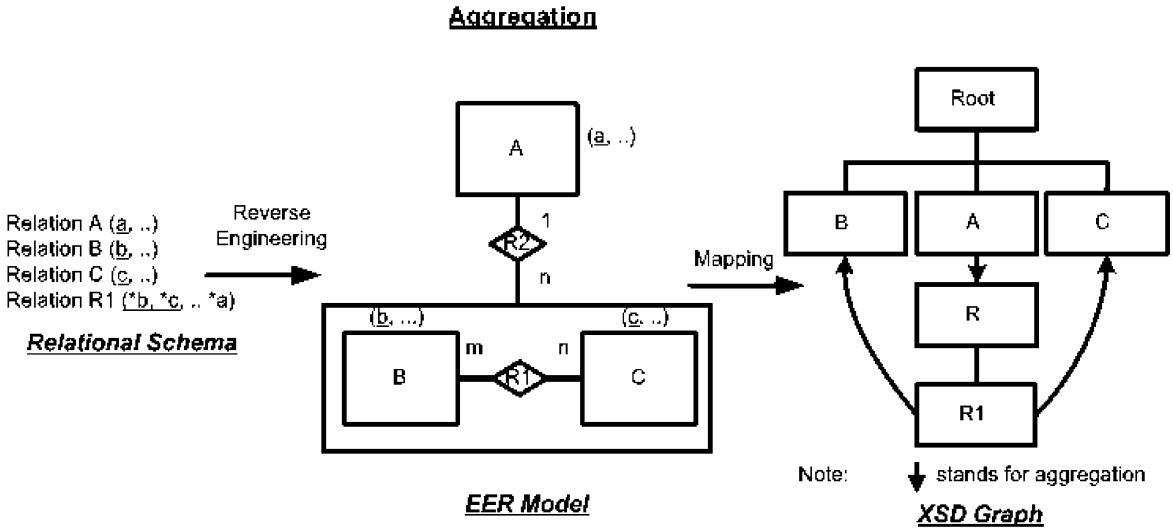
EER : XML :

FD : B.a → A.a. ⇒ FD : Instance B → Instance A

FD : C.a → A.a. ⇒ FD : Instance C → Instance A

Step 6 Aggregation

Aggregation is an abstraction through which relationships are treated as higher-level entities. In the XML schema, the transformation of aggregation is to group sub-elements under an element. Some components of sub-elements are the attributes of an element. In the whole-class element definition, the part-class element is included in the attribute list of the whole-class by using the 'ref' keyword for the type parameter. Other attributes of the whole-class element are included to make the whole-class definition complete. The whole-class elements set up a pointer pointing to the part-class element if the part-class element is pointed by many whole-class elements. The part-class element will be defined as a global element, which can be referred by other elements. If the part-class element is pointed by one whole-class element, it will be coded as a sub-element instead of a global element.



Definition 3.

EER :	XML :
FD : R1.a → A.a ⇒	FD : Instance R1 → Instance A

The functional dependency is applied for the aggregation primitive. The 'sub-element R1' points to 'element B' and 'element C' for creating the m:n cardinality as shown below:

Translated XSD

```
<xs:element name="b">
    <xs:complexType>
        <xs:attribute name="b_name" type=”xs:string”/>>
    </xs:complexType>
</xs:element>
<xs:element name="c">
    <xs:complexType>
        <xs:attribute name="c_name" type=”xs:string”/>>
    </xs:complexType>
</xs:element>
<xs:element name="r">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="r1" minOccurs=”1” maxOccurs=”unbounded”>
                <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="b"/>
                            <xs:element ref="c"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="a">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="r" minOccurs=”1” maxOccurs=”unbounded”/>>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

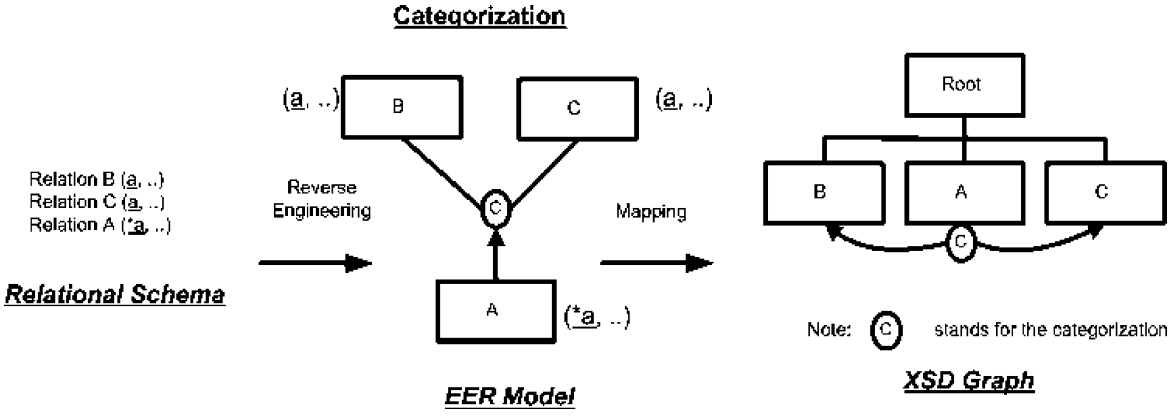
Step 7 Categorization

Categorization is a relationship in connection with two elements and one sub-element. The key in sub-element instance must appear in either one of the two elements. In XML schema, categorization feature can be represented by using element/sub-element relationship. Instances of sub-element must be located in either element. By using 'choice' keyword for making a constraint, this primitive can be functioned in XML model. Either element B or element C appears in the sub-element A. We use the 'group' feature for defining the properties on the element side. It can be referred by sub-element and elements.

```
<xs:choice>
  <xs:group ref="b"/>
  <xs:group ref="c"/>
</xs:choice>
</xs:complexType>
</xs:element>
```

Step 8 Participation

Partial and total participations can be used for distinguishing two types of relationship between parent and child entities. In 1:1, 1:*n*, and *m*:*n* cardinalities, entity relationships cannot be eliminated and need to be converted for maintaining the integrity. In XML schema, there is a more flexible way to maintain the referential integrity such as attribute group, element group or global element with 'minOccurs' and 'maxOccurs'. To control the cardinality for total participation, we use 'mixOccurs=1' keyword. To control the cardinality for partial participation, we use 'key/keyref' keywords as shown below:



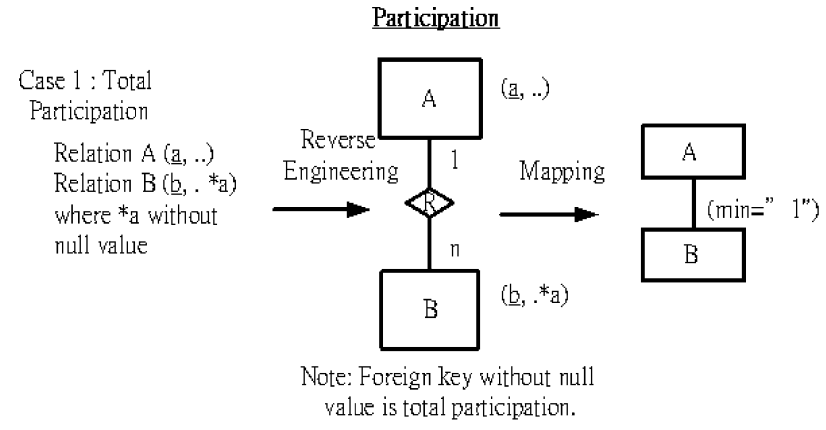
Definition 4.

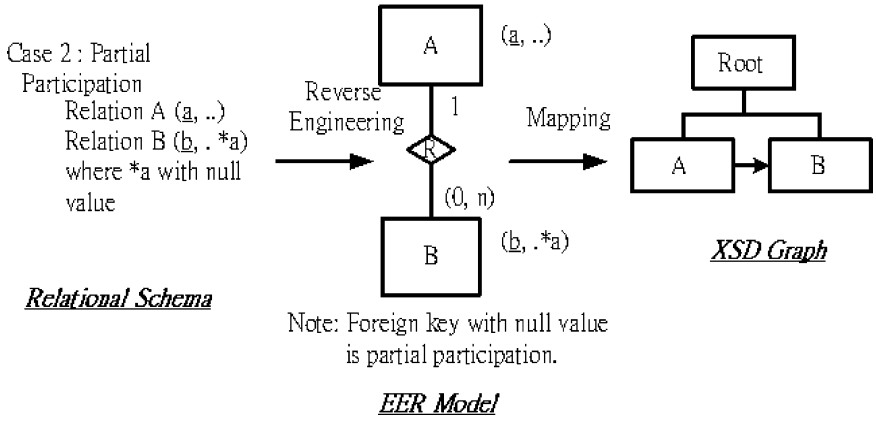
EER :	XML :
ID : A.a ⊆ (B.a ∪ C.a) ⇒	ID : Instance A⊆ (Instance B ∪ Instance C)

Instance A is a subset of Instance B ∪ Instance C linked by 'choice' and 'ref'. In this case, the inclusion dependency is applied and the following schema is referred.

Translated XSD

```
<xs:element name="a">
  <xs:complexType>
```





Definition 5.

EER :	XML :
FD : B.a → A.a ⇒	FD : Instance B → Instance A

The functional dependency is applied for this case. The occurrence of element 'B' is not allowed for null value (i.e. 'minOccurs = 1') that is a total participation. Otherwise, if we set the 'key/keyref' keywords for connecting the element 'a' and element 'b', this primitive is a partial participation.

Translated XSD

Case 1: Total participation

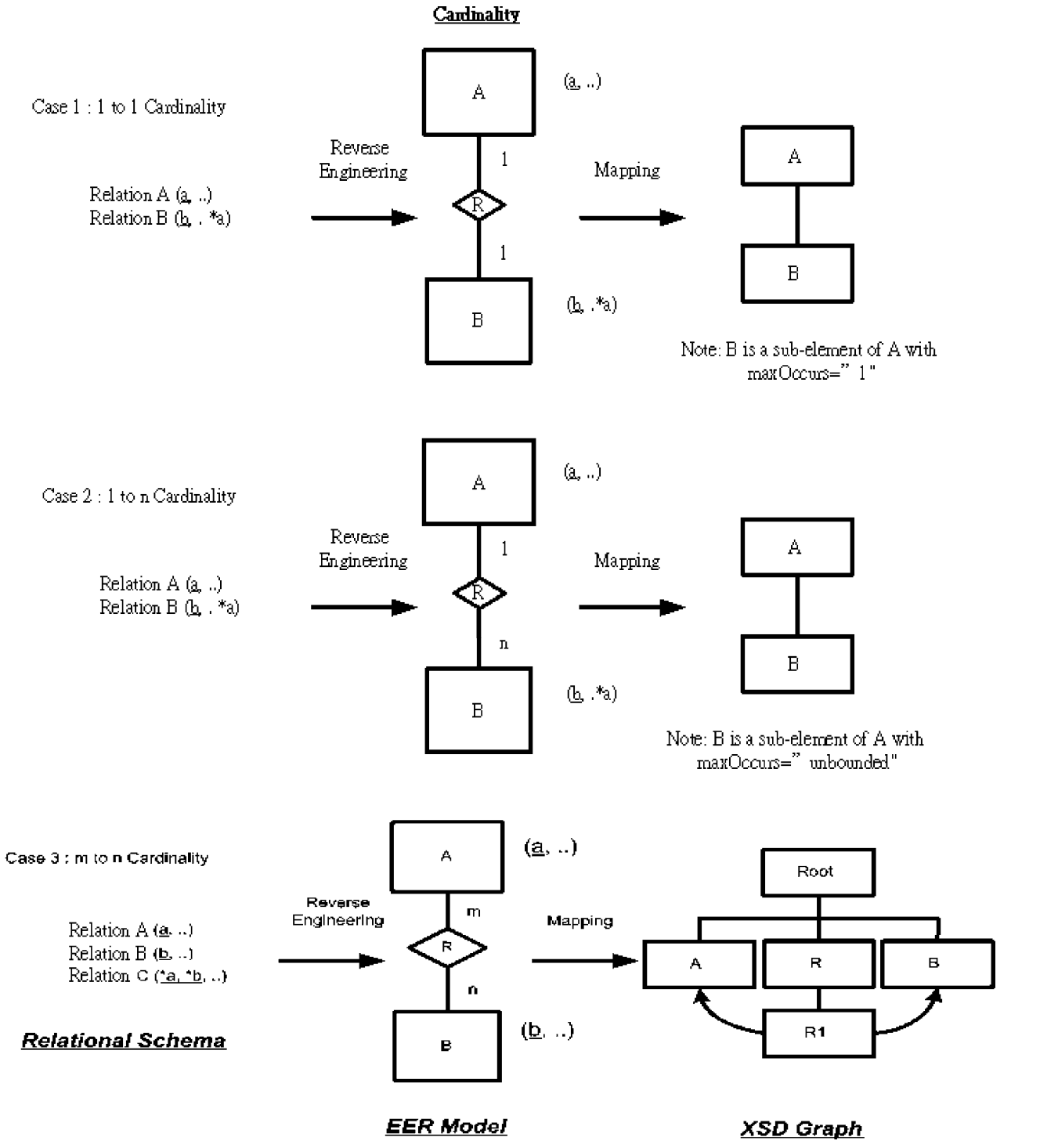
```
<xs:element name="a">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="b" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
                  <xs:attribute name="a_type" type="xs:string"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="a_type" type="xs:string"/>
   </xs:complexType>
</xs:element>
```

Case 2: Partial participation

```
<xs:element name="root">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="b">
                        <xs:complexType>
                          <xs:attribute name="locationID"/
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="a" >
                        <xs:complexType>
                          <xs:attribute name="locationID"/
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      <xs:keyref name="atobRef" refer="atobKey">
            <xs:selector xpath="b"/>
            <xs:field xpath="@locationID"/>
      </xs:keyref>
      <xs:key name="atobKey">
            <xs:selector xpath="a"/>
            <xs:field xpath="@locationID"/>
      </xs:key>
</xs:element>
```

Step 9 Cardinality

We capture 1:1, 1:*n*, and *m*:*n* cardinalities in this step. The cardinality in the XSD Graph is represented as a sub-element or a global element[9]. Name of associating element is the association name in m:n relationship. The associating element refers to the corresponding associated elements. The associating element could be treated as a pointer referring to the associated elements. The association is mapped according to the graphical representation and is assisted by the keyword of 'minOccurs' defined on the element declaration. If an element is referred by two or many elements, it is treated as a global element in the 1:1 and 1:*n* cardinalities.



Definition 6.

EER : XML :

(Case 1) B.a → A.a ⇒ FD : Instance B → Instance A

(Case 2) B.a → A.a ⇒ FD : Instance B → Instance A

(Case 3) B.b → →A.a ⇒ MVD : Instance B → →Instance A

A.a → →B.b ⇒ MVD : Instance A → →Instance B

The functional dependency and multi-valued dependency are applied in the above cases.

Translated XSD

Case 1: 1 to 1 Cardinality

```
<xs:element name="a">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="b" minOccurs=”1” maxOccurs=”1”>
            <xs:complexType>
                  <xs:attribute name=”b_name” type="xs:string"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Case 2: 1 to n Cardinality using sub-element

```
<xs:element name="a">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="b" minOccurs=”1” maxOccurs=”unbounded”>
            <xs:complexType>
                  <xs:attribute name=”b_name” type="xs:string"/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Alternative for 1:*n* cardinality using ‘key/keyref’ keywords

In the following example, the elements ‘b’ and ‘a’ contain a constraint in 1:n cardinality. We have used the ‘key/keyref’ keywords to construct the constraint. This is an alternative comparing with our proposed method. We think the following is more complicated than the previous method.

```
<xs:element name=”root”>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="a">
               <xs:complexType>
                  <xs:attribute name=”locationID”/>
               </xs:complexType>
            </xs:element>
            <xs:element name="b" >
               <xs:complexType>
                  <xs:attribute name=”locationID”/>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
      <xs:keyref name=”atobRef” refer=”atobKey”>
            <xs:selector xpath=”a”/>
            <xs:field xpath=”@locationID”/>
      </xs:keyref>
       <xs:key name=”atobKey”>
            <xs:selector xpath=”b”/>
            <xs:field xpath=”@locationID”/>
      </xs:key>
</xs:element>
```

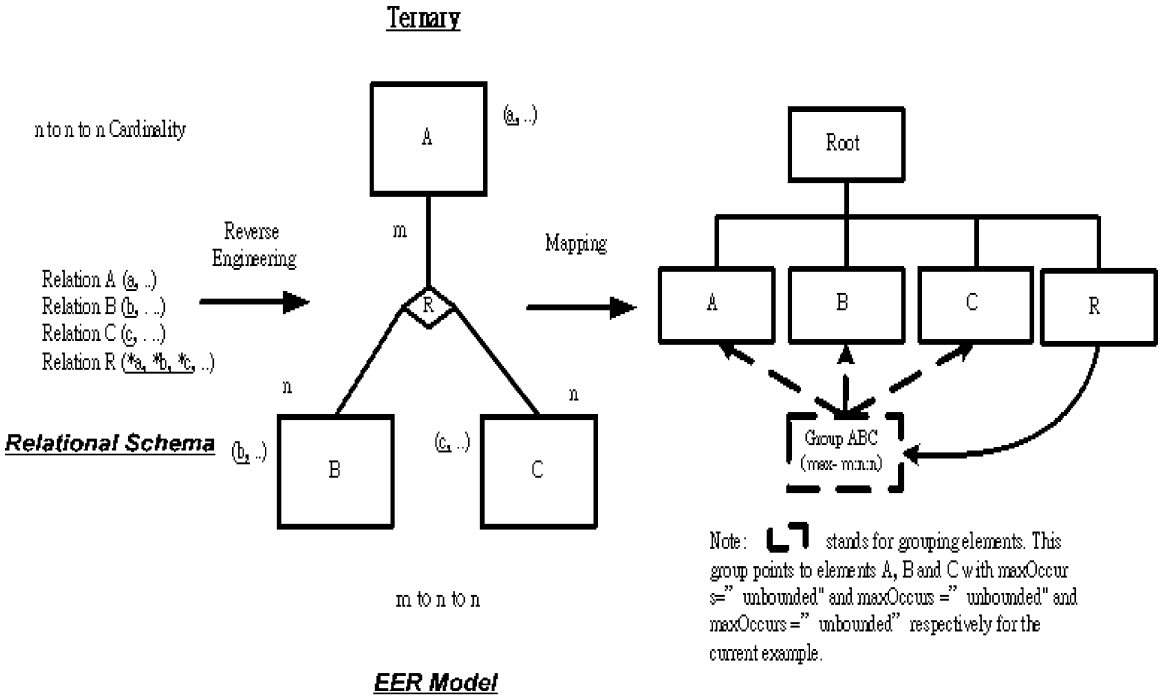
Case 3: *m* to *n* Cardinality

```
<xs:element name="a">
   <xs:complexType>
      <xs:attribute name="a_name" type=”xs:string”/>>
   </xs:complexType>
</xs:element>
<xs:element name="b">
   <xs:complexType>
      <xs:attribute name="b_name" type=”xs:string”/>>
   </xs:complexType>
</xs:element>
<xs:element name="r">
   <xs:complexType>
      <xs:sequence>
```

```
            <xs:element name="r1" minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="a"/>
                            <xs:element ref="b"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Step 10 N-ary relationship

We apply the concept of ternary relationship in XSD Graph. The associating relation is placed at the centre of three associated relations in the EER diagram. In XSD Graph, the associating elements are transformed as an element associating with three elements using 'group element' function. The occurrences of the associating elements depend on the distributions of cardinality. Therefore, in XML schema, the coding is accorded to XSD Graph and assisted by the keywords of 'minOccurs' and 'max-maxOccurs' for controlling the occurrences.



Definition 7.

EER :	XML :
FD : R.a → A.a⇒	FD : Instance Ra → Instance A
FD : R.b → B.b⇒	FD : Instance Rb → Instance B
FD : R.c → C.c⇒	FD : Instance Rc → Instance C

The functional dependency is applied for the N-ary.

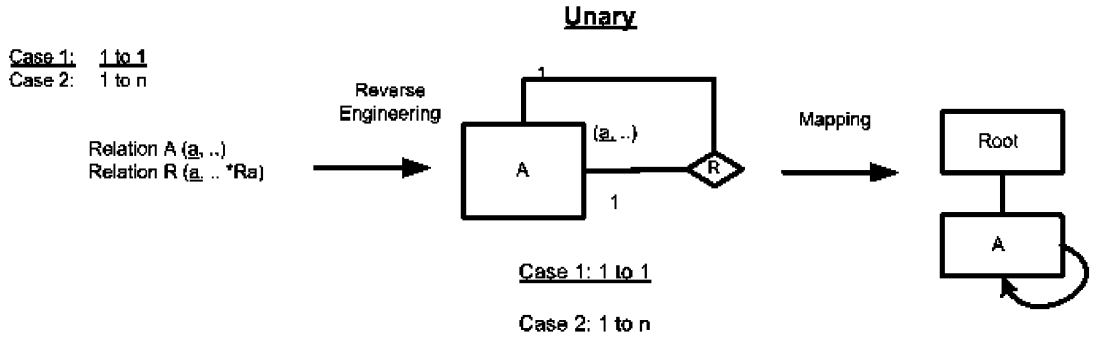
Translated XSD

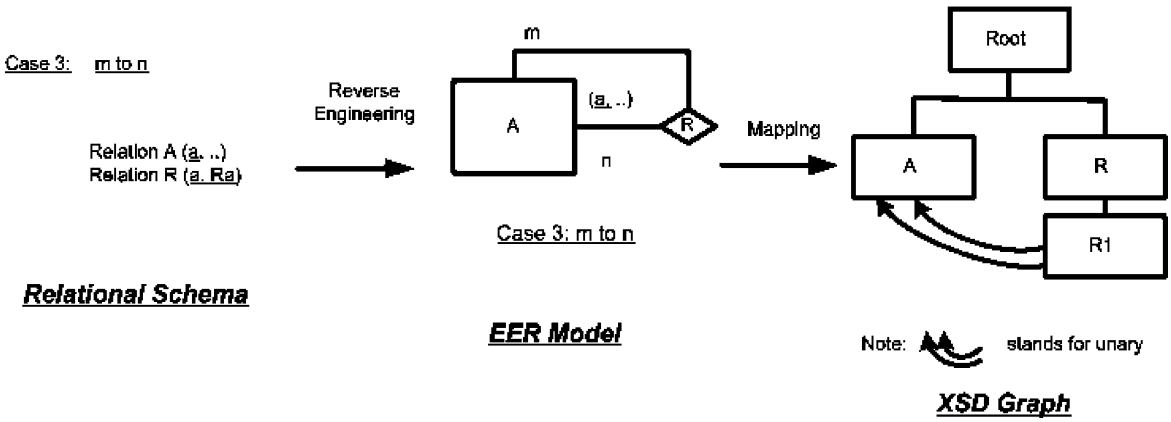
```
<xs:element name="a">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="a_name" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="b">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="b_name" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="c">
    <xs:complexType>
```

```
        <xs:sequence>
            <xs:element name="c_name" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="r">
    <xs:complexType>
        <xs:sequence>
            <xs:group ref="groupABC"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:group name="groupABC">
    <xs:sequence>
        <!-- the elements r1, r2 and r3 in this example are applied for the m to n to n cardinality -->
        <xs:element name="group_a" maxOccurs="unbounded">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="a"/>
                    <xs:element name="group_b" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="b"/>
                                <xs:element name="group_c" maxOccurs="unbounded">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element ref="c"/>
                                        </xs:sequence>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:group>
```

Step 11 Unary relationship

The unary relationship can be described as a relationship for self-reference. It deals with one relation. Three combinations of the unary relationship can be found as *m*:*n*, 1:*n*, and 1:1 cardinalities. We use a keyword 'complexType' for elements which refer to itself in the 1:n and 1:1 cardinalities. A keyword 'ref' is used in the shadow element for the *m*:*n* cardinality. One referring element with two sub-elements is created for the *m*:*n* cardinality of unary relationship in XML schema. With regard to the 1:1 and 1:n cardinalities, we use the element and 'complexType' method with the keywords of 'minOc-minOccurs' and 'maxOccurs' for transforming them into XML schema.





Definition 8.

	EER :	XML :
(Case 1)	FD : R.Ra → A.a	FD : Instance A → Instance A
(Case 2)	FD : R.Ra → A.a	FD : Instance A → Instance A
(Case 3)	MVD : R.Ra → →A.a & R.Ra → →A.a	MVD : Instance R1 → Instance A

(R) is an associating entity for *m*:*n* cardinality in connection with (A) in the EER model.

The functional dependency and the multi-valued dependency are applied in these cases.

Translated XSD

(Case 1: 1 to 1 Unary)

```
<xs:element name="a">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a_name" type="xs:string"/>
      <xs:element name="ref_a" type="a" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="a">
  <xs:sequence>
    <xs:element name="a_name" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

(Case 2: 1 to n Unary)

```
<xs:element name="a">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="a_name" type="xs:string"/>
      <xs:element name="ref_a" type="a" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="a">
  <xs:sequence>
    <xs:element name="a_name" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

(Case 3: m to n Unary)

```
<xs:element name="a">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="a_name" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="r">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="r1" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="a"/>
                        <xs:element ref="a"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

3.2. XSD Graph and its structure

Each entity is represented as an element and the attributes of the entity are treated as the element's attributes. In the relational model, we connect all the tables linking up with relationships. We convert them as global elements in XML model. The 'minOccurs' and 'maxOccurs' are the keywords for declaring the cardinality on XML schema. According to the declaration, XML document is built from primitive data semantics. We have created some symbols for XSD Graph. These symbols are new and used for illustrating the structure between tree and elements.

In XSD Graph, user inputs a name for the root element for the whole XML document. The function of this root element wraps up the rest of elements and it may not have any attributes. For instance, if we do a project in connection with a factory, we could simply provide a root name called 'factory' for the project. The next is to select global elements. The root element is the top level of tree. According to each semantic, we get small document trees representing their semantic group that may not be linked up. The function of root is to construct the whole document tree for the project. The second level of tree will be contained global elements, which are the root of each semantic group. We will search all the nodes and find out the cardinality factor. If an entity has 1:1 relationship with another entity, it will be translated as an element with '1' indicator in the secondary level. If an entity has 1:1 and m:n relationships with two or more entities, it will be translated as a sub-element with 'N' indicator in the third level or forth level. The indicator will depend upon the cardinality factor of its associated entity/entities.

4. Case study for indirect schema translation

In a case study, XSD Graph and XML schema definition focuses on products of a factory. The products can be categorized in different levels. Each product is composed by many small parts and consists of many features. They have their own value. In relational database, a relational schema is created and sample data are loaded into the physical repository. We employ XSD Graph for recapturing data semantics from the relational database. The relational schema is mapped into an XSD Graph. According to the mapped XSD shown in Appendix A, sample data for an XML DB repository will be loaded as an XML document.

4.1. Relational schema

CREATE TABLE category (*item_code*, name, descript)
CREATE TABLE feature (*name*, descript, multivalue, *cate_id,)
CREATE TABLE featurevalue (*value_id*, value, *name)
CREATE TABLE catalogitem (*item_no*, cata_name, descript)
CREATE TABLE part (*item_no*, parttype)
CREATE TABLE supplier (*item_no*, name, address)
CREATE TABLE catalog (*item_code*, name, descript, startdate, enddate)
CREATE TABLE productbundle (*log_item*, name, descript)
CREATE TABLE product (*name*, descript, url, *item_no)
CREATE TABLE service (*name*, descript, *unitoftime)
CREATE TABLE unitoftime (*time_code*, name, hour, day, week, month, year)
CREATE TABLE resource (*item*, loc, name, *log_item)
CREATE TABLE cat (**item_code*,**name*)

4.2. EER diagram

According to the Reverse Engineering of Process 1, the relational schema is transformed to EER model by examining the structural constraints of the relational schema. The following is the EER diagram after transforming from the relational schema (Fig. 3).

4.3. Translation processes

Step 1 'Root' relation → Root element

Since users select relation Category as 'root' relation and all other relations are navigable, we include that all relations are relevant and map them into an XSD.

Step 2 – Entity → Elements

Fig. 4

Step 3 – Foreign Key → Elements/Sub-element structure

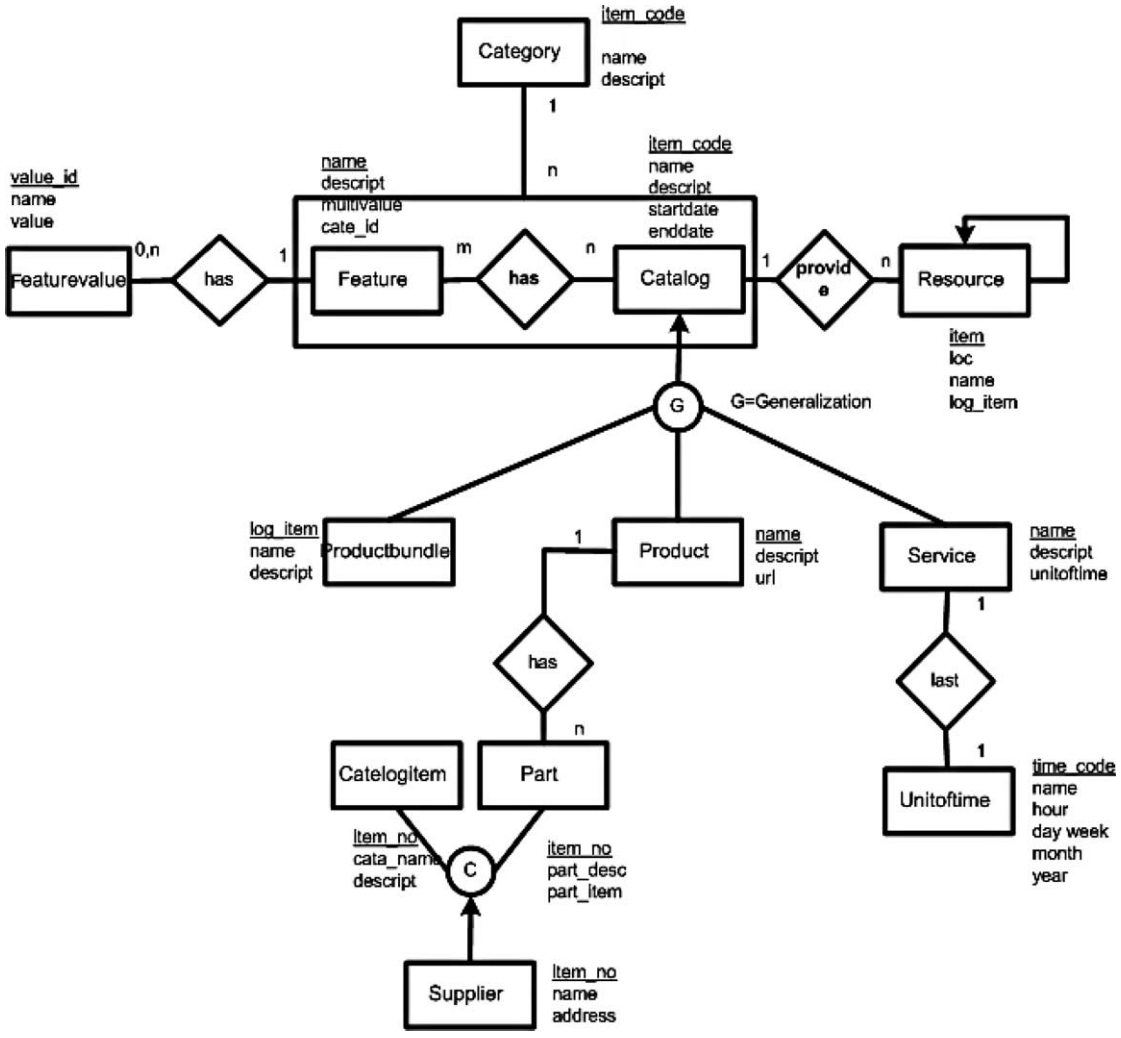


Fig. 3. EER diagram transformed from relational schema.

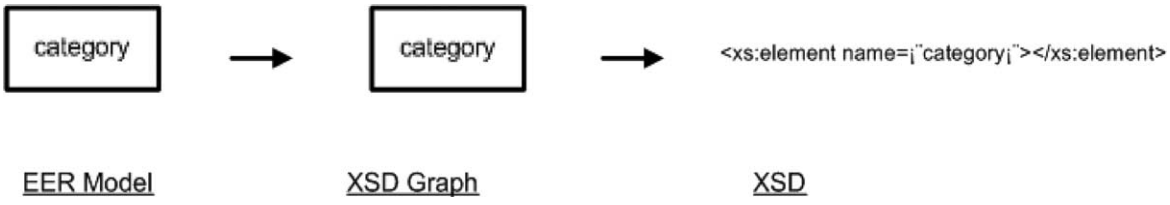


Fig. 4. Mapping entity into elements.

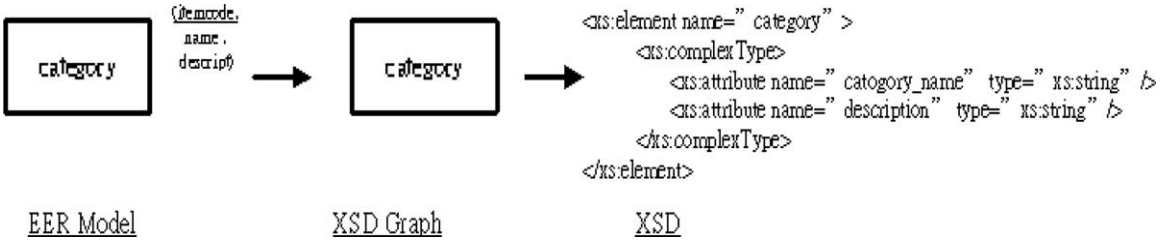


Fig. 5. Mapping foreign key into element and sub-element.

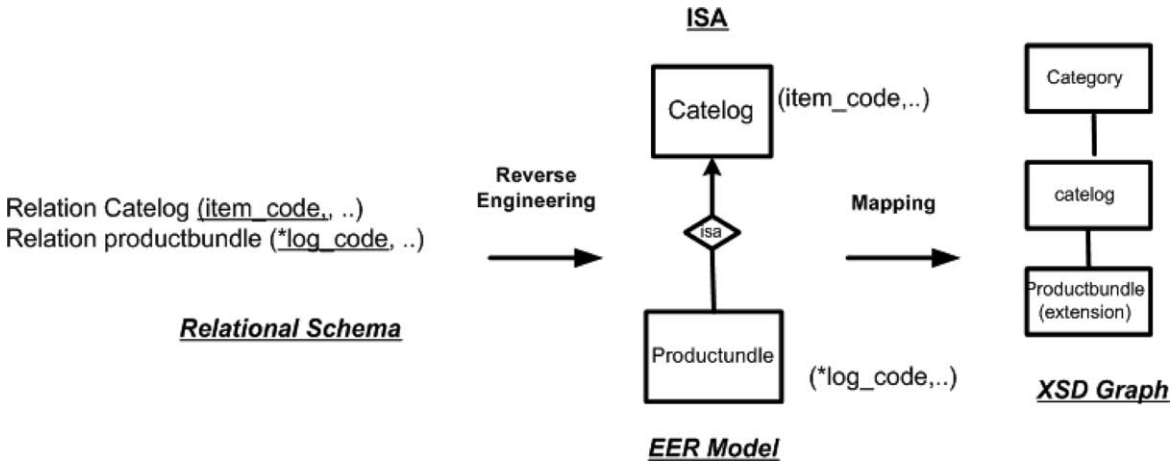


Fig. 6. Mapping generalization into element extension.

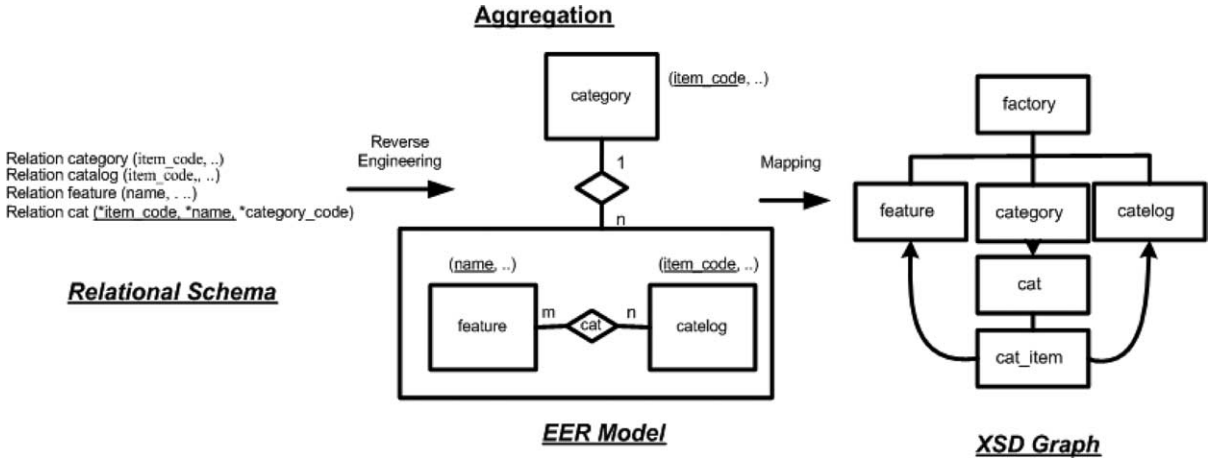


Fig. 7. Mapping diagram for aggregation.

Fig. 5
Step 4 – isa → Elements Extension
Fig. 6
Step 5 Generalization

Not applied
Step 6 Aggregation
Fig. 7
Step 7 Categorization

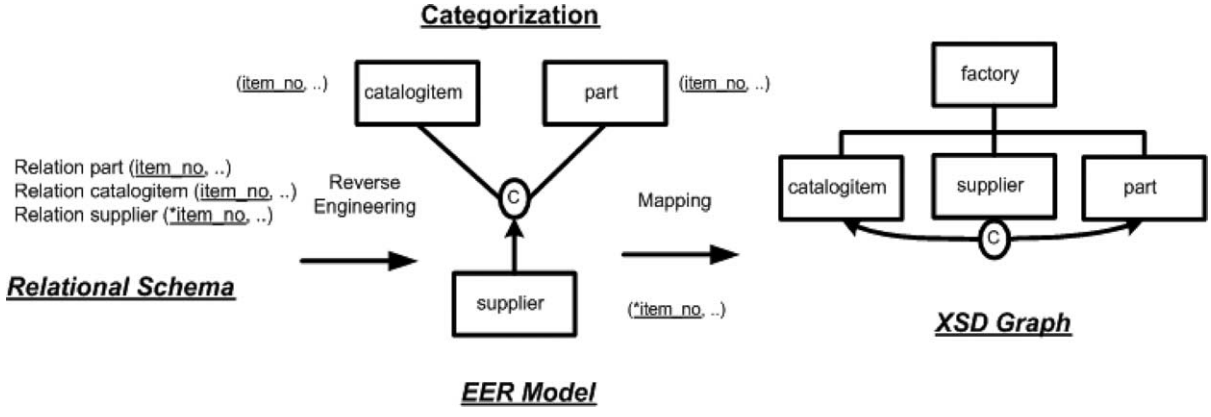


Fig. 8. Mapping diagram for categorization.

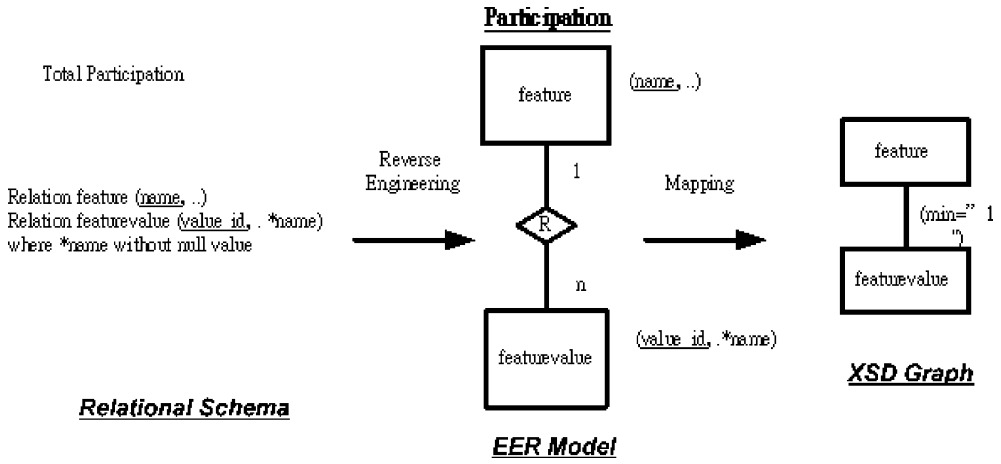


Fig. 9. Mapping diagram for participation.

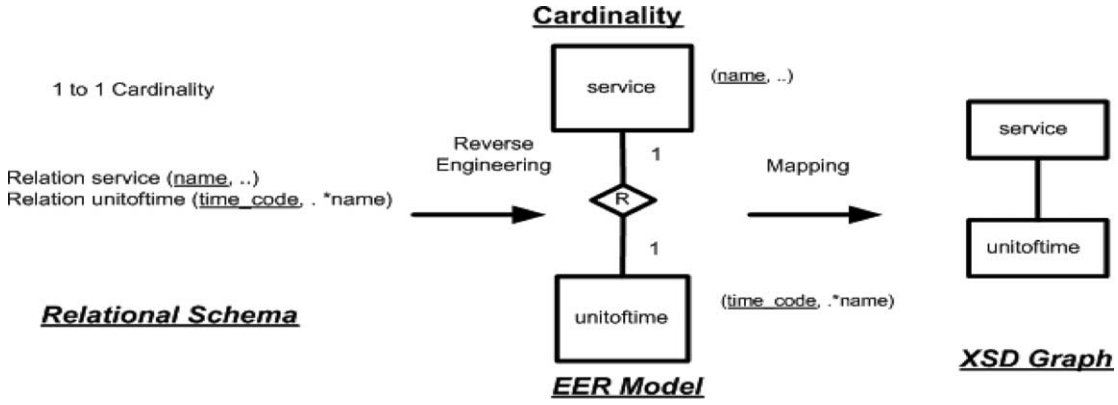


Fig. 10. Mapping diagram for cardinality.

Fig. 8
Step 8 Participation
Fig. 9
Step 9 Cardinality
Fig. 10
Step 10 N-ary relationship
Not applied
Step 11 Unary relationship
Fig. 11

4.4. XSD Graph

Appendix I shows the translated XSD for 'factory.xsd' (Fig. 12).

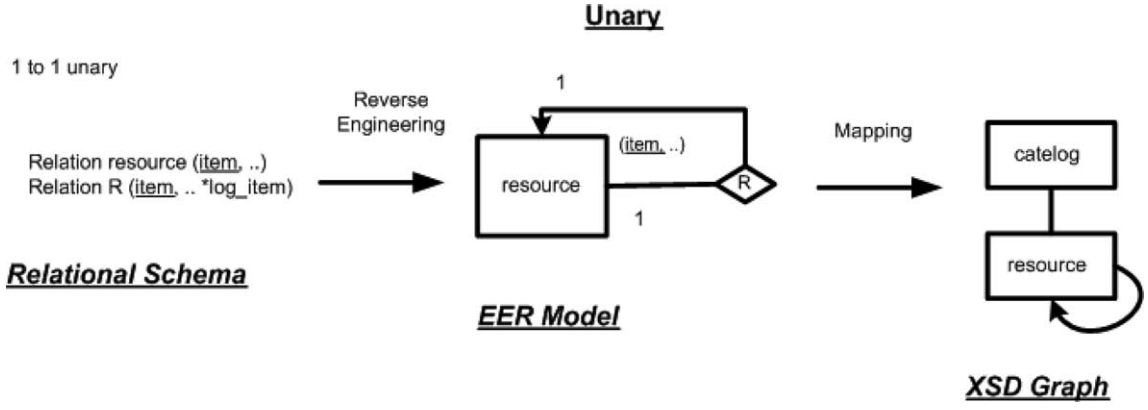


Fig. 11. Mapping diagram for Unary.

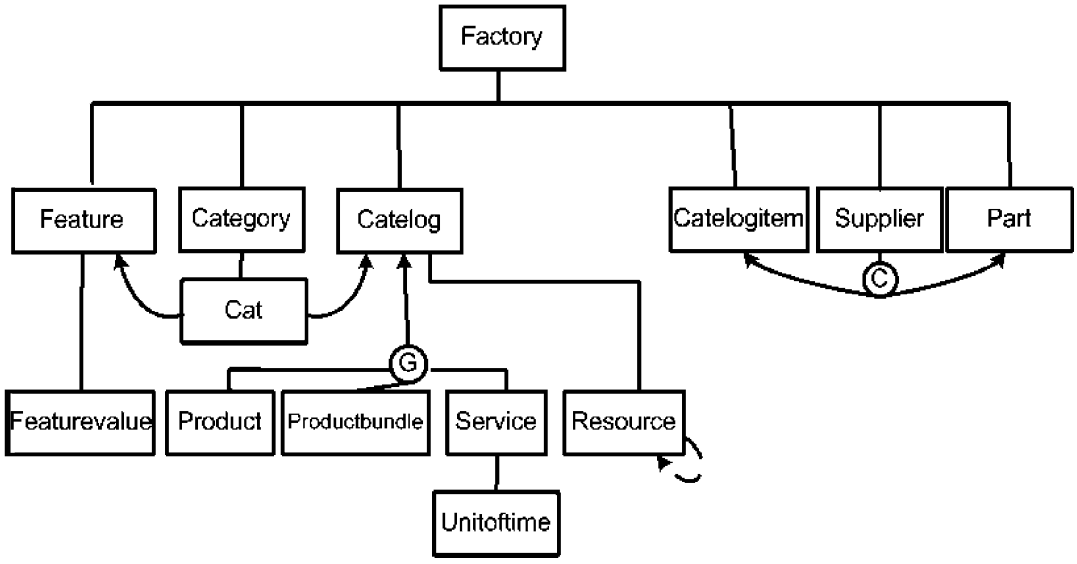


Fig. 12. XSD graph transformed from EER model.

5. Prototype

5.1. Relational schema→XSD

The General and Semantic Rules are applied in this prototype. The former is to outline the scope of application and to identify necessary information to be completed during the transformation. The latter is to recapture data semantics.

5.2. Relational database

The following information extracted from the relational schema is stored for transforming the basic EER entities into XML elements.

Supplier

Item_no	Name	Address
1	ABC Company	Flat H, 3 Main Road, HK
2	Whitehouse Company	Flat R, 34 Main Road East, HK

Catalogitem

Item_no	Cata_name	Descript
1	Door handle	For sport car

Category

Item_code	Name	Descript
C1	Sport car 1	Low price sport car product

Catelog

Item_code	Name	Descript	Startdate	Enddate
Log1	Wiper	For windscreen	13/8/2003	13/8/2003

Feature

Name	Descript	Multivalue	Cate_id
Speed	Describe the speed	1	C1

Featurevalue

Value_id	Value	Name
V1	10	Speed

Part

Item_no	Part_desc	Part_item
1	For car wiper	Wiper

Product

Name	Descript	Url
Wiper	For windscreen	/sport_car/wiper.htm

Productbundle

Log_item	Name	Descript
Log1	Wiper	For windscreen

Resource

Item	Loc	Name	Log_item
1	Floor 6	Car accessories	Log1

Service

Name	Descript	Unitoftime
Wiper	For windscreen	1

Cat

Item_code	Name	Category_code
Log1	Speed	C1

Unitoftime

Time_code	Name	Hour	Day	Week	Month	Year
Time1	Wiper	1	13/8/2003	0	0	0

5.3. XSD graph and its structure

We use the following indicator for locating elements in different levels of XSD Graph. Indicator '1' represents the elements for the first level under a root and indicator 'N' represents the elements for the second and third levels (i.e. sub-elements of the first level).

Indicator {feature = 1, category = 1, featurevalue = N, catalogitem = 1, part = 1, supplier = 1, catalog = 1, productbundle = N, product = N, service = N, resource = N, cat = N, unitoftime = N}

In this step, we construct an XSD Graph according to the information collected by the previous procedures. The root element named 'factory' contains six elements in the first level. Two Groups have been defined and named as 'Catelogitem' and 'Part'. Five ComplexTypes can be located and named as 'productbundle', 'product_type', 'service_type', 'resource', and 'catalog'. The XML document for the 'factory' case is shown in Appendix B.

5.4. Validation

We evaluate our methodology by observing the difference between the relational schema and the XML schema according to the results provided by the relational software and the XML software. We load the schema and sample data into two systems. In the relational model, we employ the 'FoxPro' software for the validation. As for the XML model, we employ the 'XMLSPY' software for the validation. The latter is an editor for creating the XML schema and the XML document. It provides a function to evaluate the schema and document through the XPath. We use its SQL engine for checking the results whether they are the same as the XML model or not. The followings are the results for different views extracted from the XML documents and relational tables.

Retrieving for properties using mapping condition of other element, it searches from the catelog element and finds all properties from 'product' sub-element with catelog name = 'wiper'. The results of query for the XML model and the relational model are shown as follows.

XPath statement [1] = //catelog/catelog_item[service/catelog_item[@catelog_name = "wiper"]]/product/*

Result from XML document:

Catelog_name	Description	photourl	Startdate	Enddate
Wiper	For wind screen	/sport_car/wiper.htm	2003-08-13	2003-08-14

SQL statement [1] = SELECT catelog.name, catelog.descript, product.url, catelog.startdate, catelog.enddate from product, service, catelog WHERE service.name = "wiper" AND service.name = product.name AND product.name = catelog.name

Result from Relational table:

Name	Descript	Url	Startdate	Enddate
Wiper	For wind screen	/sport_car/wiper.htm	08/13/03	08/14/03

Finding for the supplier information using a simple retrieval, it searches from the supplier element and look for the supplier name that is called 'ABC Company'. The results of query for XML model and the relational model are shown as follows.

XPath statement [2] = //supplier[@supplier_name = "ABC Company"]//*

Result from XML document:

Catalogitem_name	Description
Door handle	For sport car

SQL statement [2] = SELECT * from catalogitem WHERE supplier.name = "ABC Company" AND (supplier.item_no = catalogitem.item_no)

Result from Relational table:

Cata_name	Descript
Door handle	For sport car

Finding for the feature information using a simple retrieval, it searches from the category element and look for the category name that is called 'sport car 1'. The results of query for XML model and the relational model are shown as follows

XPath statement [3] = //category[@category_name = "sport car 1"]//feature

Result from XML document:

Feature_name	Feature_desc	Multivalued
Speed	Describe the speed	1

SQL statement [3] = SELECT * from feature WHERE category.name = "sport car 1" AND (feature.cate_id = category.item_code)

Result from Relational table:

Name	Descript	Multivalue
Speed	Describe the speed	1

During the validation, we cannot cover the integrity constraints in XML model because of the difference of data structure between them. However, we can apply the 'unique' keyword in our paper. The following is an alternative for adding up an identity constraint.

Alternative for identity constraint using 'unique' keyword

The 'unique' function in XML model can be interpreted as a primary key in the relational model. Values of attribute defined by the 'unique' keyword cannot be duplicated. If we want to create a relationship with other elements, we must add the 'key/keyref' keywords.

XSD

```
<xs:element name=”a”>
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="a_name" type=”a_nameType”/>
            </xs:sequence>
     </xs:complexType>
     <xs:unique name=”unique_a_name”>
            <xs:selector xpath=”a_name”/>
            <xs:field xpath=”@a_nameID”/>
    </xs:unique>
</xs:element>
<xs:complexType name=”a_nameTYpe”>
    <xs:sequence>
            <xs:attribute name=”a_nameID” type=”xs:string”/>
    </xs:sequence>
</xs:complexType>
```

6. Conclusion

After validating the results between the source relational schema and the target XSD by implementing in the 'xmlspy' and 'foxpro', respectively, our methodology is feasible because both can come up with the same result. We conclude that seven semantics can be converted from Relational model to XML model including aggregation, generalization/isa, participation, cardinality, categorization, N-ary, and Unary. Although there is no key concept in XML model, we can use 'complexType' and 'element and sub-element' for linking up the relationship among elements. We can use the 'choice' keyword as a constraint to restrict the instances for categorization primitive. XSD Graph is employed as a media for the semantic-based transformation between Relational and XML schemas. The target schema for XML model is XML schema definition (XSD) instead of DTD because we expect that XML schema definition will be the standard of XML model. The entire XML DB can be formed according to the relational schema definition in XML DB repository. The future research of this paper is data conversion from relational database into XML database after schema translation.

Appendix A

XSD for "factory.xsd"

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="factory">
        <xs:annotation>
            <xs:documentation>factory is the root element.</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:element name="category">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="cat" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="category_name" type="xs:string"/>
            <xs:attribute name="description" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="supplier">
        <xs:complexType>
            <xs:sequence>
```

```
            <xs:element name="option" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:choice>
                        <xs:group ref="part"/>
                        <xs:group ref="catalogitem"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="supplier_name" type="xs:string"/>
        <xs:attribute name="address" type="xs:string"/>
    </xs:complexType>
</xs:element>
<xs:element name="catalogitem">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="catalogitem_name" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="feature">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="featurevalue">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="value" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="feature_name" type="xs:string"/>
                <xs:attribute name="feature_desc" type="xs:string"/>
                <xs:attribute name="multivalued" type="xs:boolean"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="catelog">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="catelog_item" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element ref="resource"/>
                                <xs:element ref="product"/>
                                <xs:element ref="service"/>
                                <xs:element ref="productbundle"/>
                            </xs:sequence>
                            <xs:attribute name="catelog_name" type="xs:string"/>
                            <xs:attribute name="description" type="xs:string"/>
                            <xs:attribute name="startdate" type="xs:date"/>
                            <xs:attribute name="enddate" type="xs:date"/>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="resource">
            <xs:complexType>
```

```
        <xs:sequence>
            <xs:element name="resource_item" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="res_loc" type="xs:string"/>
                    <xs:attribute name="resource_name" type="xs:string"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="resource_ref" type="resource"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="cat">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="cat_item" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="feature"/>
                        <xs:element ref="catelog"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="unitoftime">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="unit_time" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="hour" type="xs:int"/>
                    <xs:attribute name="day" type="xs:date"/>
                      <xs:attribute name="week" type="xs:int"/>
                      <xs:attribute name="month" type="xs:int"/>
                      <xs:attribute name="year" type="xs:int"/>
                  </xs:complexType>
              </xs:element>
          </xs:sequence>
      </xs:complexType>
  </xs:element>
  <xs:element name="productbundle" type="productbundle_type"/>
  <xs:element name="product" type="product_type"/>
  <xs:element name="service" type="service_type"/>
  <xs:complexType name="productbundle_type">
      <xs:complexContent>
          <xs:extension base="catelog"/>
      </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="product_type">
      <xs:complexContent>
          <xs:extension base="catelog">
              <xs:sequence>
                  <xs:element name="photourl" type="xs:string"/>
              </xs:sequence>
          </xs:extension>
      </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="service_type">
      <xs:complexContent>
```

```
        <xs:extension base="catelog">
            <xs:sequence>
                <xs:element ref="unitoftime"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="catelog">
    <xs:sequence>
        <xs:element name="catelog_item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="catelog_name" type="xs:string"/>
                <xs:attribute name="description" type="xs:string"/>
                <xs:attribute name="startdate" type="xs:date"/>
                <xs:attribute name="enddate" type="xs:date"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="resource">
    <xs:sequence>
        <xs:element name="resource_item" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="res_loc" type="xs:string"/>
                <xs:attribute name="resource_name" type="xs:string"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:group name="part">
    <xs:sequence>
            <xs:element name="parttype" type="xs:string"/>
        </xs:sequence>
    </xs:group>
    <xs:group name="catalogitem">
        <xs:sequence>
            <xs:element name="catalogitem_name" type="xs:string"/>
            <xs:element name="description" type="xs:string"/>
        </xs:sequence>
    </xs:group>
</xs:schema>
```

Appendix B

XML document for "factory.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<factory xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\joe\PhD-2\factory.xsd">
    <supplier supplier_name="ABC Company" address="Flat H, 3 Main Road, HK">
        <option>
            <catalogitem_name>door handle</catalogitem_name>
            <description>for sport car</description>
        </option>
    </supplier>
    <supplier supplier_name="Whitehouse Company" address="Flat R, 34 Main Road East, HK">
        <option>
            <parttype>wiper</parttype>
```

```
        </option>
    </supplier>
    <category category_name="sport car 1" description="low price sport car product">
        <cat>
            <cat_item>
                <feature feature_name="speed" feature_desc="describe the speed" multivalued="1">
                    <featurevalue>
                        <value>10</value>
                    </featurevalue>
                </feature>
                <catelog>
                    <catelog_item catelog_name="wiper" description="for windscreen" startdate="2002-08-13"
                    enddate="2002-08-14">
                        <resource>
                            <resource_item res_loc="floor 6" resource_name="car accessories"/>
                            <resource_ref>
                                <resource_item res_loc="floor 7" resource_name="repair"/>
                            </resource_ref>
                        </resource>
                        <product>
                            <catelog_item catelog_name="wiper" description="for windscreen"
                            startdate="2003-08-13" enddate="2003-08-13"/>
                            <photourl>/sport_car/wiper.htm</photourl>
                        </product>
                        <service>
                            <catelog_item catelog_name="wiper" description="for windscreen"
                            startdate="2003-08-13" enddate="2003-08-13"/>
                            <unitoftime>
                                <unit_time hour="1" day="2003-08-13" week="0" month="0" year="0"/>
                            </unitoftime>
                        </service>
                        <productbundle>
                            <catelog_item catelog_name="wiper" description="for windscreen"
                            startdate="2003-08-13" enddate="2003-08-13"/>
                        </productbundle>
                    </catelog_item>
                </catelog>
            </cat_item>
        </cat>
    </category>
</factory>
```

References

[1] C. Biancheri, J. Pazzaglia, G. Paddeu, EIHA?!?: Deploying Web and Wap Services Using XML Technology, SIGMOD Record 30 (1) (2001) 5–12.

[2] G. Booch, et al., The Unified Modeling Language User Guide, Addison-Wesley, Reading, MA, 1999. pp. 239–240.

[3] Ng Wilfred, Maintaining Consistency of Integrated XML Trees, Third WAIM Conference, Springer, Berlin, 2002. pp. 145–157.

[4] M.W. Vincent, Jixue Liu, Functional Dependencies for XML, Fifth APWeb Conference, Springer, Berlin, 2003. pp. 23–34.

[5] K. Yue, et al., Constraint Preserving XML Updating, Fifth APWeb Conference, Springer, Berlin, 2003. pp. 47–58.

[6] R. Wyke, A. Watt, XML Schema Essentials, Wiley Computer Publishing, 2002. pp. 3–70.

[7] Nicolle, et al., XML Integration and toolkit for B2B applications, Journal of Database Management 14 (4) (2003) 33–58.

[8] J. Fong, et al., Converting relational database into XML documents with DOM, Information and Software Technology 45 (2003) 335–355.

[9] E.V.D. Vlist, Using W3C XML Schema, http://www.xml.com/pub/a/2000/11/29/sche, October 2001.

[10] A. Zhou, H. Lu, S. Zheng, Y. Liang, L. Zhang, W. Ji, Z. Tian, VXMLR: A Visual XML-Relational Database System, Proceedings of the 27th VLDB Conference, Roma, Italy, 2001 pp. 646–648.

[11] M. Fern Ndez, W.-C. Tan, D. Suciu, SilkRoute: trading between relations and XML, Computer Networks 33 (1–6) (2000) 723–745.

[12] M. Yoshikawa, T. Amagasa, XRel: a path-based approach to storage and retrieval of XML documents using relational databases, ACM Transactions on Internet Technology 1 (1) (2001) 110–141.

[13] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, BruceLindsay, H. Pirahesh, B. Reinwald, Efficiently Publishing Relational Data as XML Documents, Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000 pp. 65–76.

[14] M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, S. Subramanian, XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents, Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000 pp. 646–648.

[15] I. Varlamis, M. Vazirgiannis, Bridging XML-schema and relational databases. A system for generating and manipulating relational databases using valid XML documents, Proceedings of the ACM Symposium on Document Engineering, Atlanta, USA, November, 2001 pp. 105–114.

[16] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999 pp. 302–314.

[17] L. Khan, Y. Rao, A performance evaluation of storing XML data in relational database management systems, Proceeding of the third international workshop on Web information and data management, Atlanta, USA, November, 2001 pp. 31–38.

[18] S. Abiteboul, P. Buneman, D. Suciu, Data on the Web From Relational to Semistructured Data and XML, Morgan-Kaufmann, 2000. pp. 197–207.

[19] R.C. Pare, From Ternary Relationship to Relational Table: A Case Against Common Beliefs, ACM/SIGMOD Record 31, 2002 2 June 2002.

[20] M. Dahchour, A. Pirotte, The Semantics of Reifying N-ary Relationships as Classes, ICEIS, 2002 pp. 580–586.

[21] R. Conrad, D. Scheffner, J.C. Freytag, XML Conceptual Modeling Using UML, Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000), Salt Lake City, UT, USA, October, 2000.

[22] C. Kleiner, U. Lipeck, Automatic Generation of XML DTDs from Conceptual Database Schemas, Workshop Web Databases of Conference of German and Austrian Computer Societies, Session III, Vienna, 2001.

[23] J. Xiao, et al., FIXT: A Flexible Index for XML Transformation, Fifth APWeb Conference, Springer, Berlin, 2003. pp. 144–149.

[24] J.E. Funderburk, et al., XTABLES: Bridging Relational Technology and XML, IBM Systems Journal 41 (4) (2002) 616–641.

[25] S. Navathe, A. Awong, Abstracting Relational and Hierarchical Data with a Semantic Data Model, Entity-relationship Approach to Software Engineering, 1983 pp. 223–248.

[26] J. Fong, R. Pang, A. Fong, F. Pang, K. Poon, Concurrent data materialization for object-relational database with semantic metadata, International Journal of Software Engineering and Knowledge Engineering 13 (3) (2003) 257–291.

[27] J. Gennicr, Make XML Native and Relative, Jan/Feb 2003 Oracle Magazine, 2003 pp. 85–88.