# Developing a Unified Design Methodology Based on Extended Entity-Relationship Model for XML

Mun-Young Choi [1], JongSeon Lim[2], and Kyung-Soo Joo [3]

Dept.of Computer Science and Engineering, College of Engineering SoonChunHyang Uni.,
P.O. Box 97, Asan, ChungNam, Korea, 336-745
[1] griffin@hyejeon.ac.kr, [2] ronmer@chol.com, [3] gsoojoo@sch.ac.kr

**Abstract.** Nowadays an information exchange on XML such as B2B electronic commerce is spreading. Therefore the systematic and stable management mechanism for storing the exchanged in formation is needed. For this goal there are many research activities for connection between XML application and relational database. A unified modeling methodology need to store the XML data on the variety database. In this paper an unified design methodology based on EER data model for XML applications is proposed. For this goal, first a XML modeling guideline for XML DTD based on ER data model is introduced. Second a database modeling methodology for storing on object relational database is proposed.

## 1 Introduction

XML is a text form planned to utilize structured document in web. XML is embossed highly as the standard language about data exchange. XML has many advantages for compatibility and extensity. One of the important tasks of XML is the easy exchange of different data and information in B2B electronic commerce. The important exchanged data or information are stored on existent relational database system, and it is important that the data or information is automatically processed to make XML file.

In this paper a XML modeling guideline for XML DTD based on ER data model is introduced. A database modeling methodology for storing on relational database is proposed. XML document structure is described by using DTD, which is defined by standard, and data should be preserving states as much as possible in conversion by XML document[1], [2].

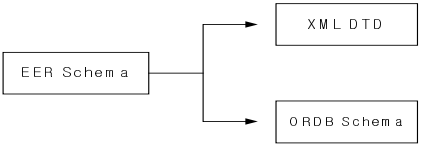


**Fig. 1.** XML modeling and data modeling that use EER schema

The references described about XML modeling and data modeling way in EER schema are[3], [4] But, there is no way that integrate both of two. Therefore, XML modeling and data modeling way in EER schema with Fig. 1 are described in this paper.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 920–929, 2003.

## 2 XML DTD Design Method of ER Base

In this section we will unite the proposed translations from conceptual schema to DTD into an guideline to be applied to any given conceptual schema. The guideline may be validated by applying it to the sample ER schema in section 3 to generate a DTD. Moreover we will comment on the quality of the ER to DTD transformation by analyzing how much information is lost if a retransformation to ER is performed.

The complete guideline in pseudo-code is depicted in Fig. 2. As root element we use an element corresponding to the whole database represented by the schema name. Elements for entities at higher levels are added as direct subelements of the root with the objective to preserve as many constraints from the schema as possible. These elements include elements for entities that are not functionally dependent on other entities as well as relationships of higher cardinality and arity.

In step 3 we start with the detailed contents definition of the elements from the previous step. Whenever we need new elements in the process they are appended to the list of elements in need of detailed definition. The process for each element includes a straight-forward mapping of entity attributes as described in (a), (b) and (e). Subelements are included for relationships to weak entity types (step (c)) and other functionally dependent entities (step (d)) such that a validating XML parser will be able to verify these constraints on a particular document.

In step (f) 1:1-relationships with total participation of the current entity and partial participation of the opposite side are realized by adding all the relationship attributes to the current element as well as an IDREF attribute for the opposite entity type.

Finally in step 4 elements for n:m- or k-ary relation-ships ($k > 2$) are defined as direct subelements of the root element of the document. The references to the participating entities are realized by IDREFs which can only enforce the existence of matching XML elements but not correctness of the desired element type.

A major restriction is that participation of entities in relationships modeled by IDREF attributes cannot be reconstructed. This is a major restriction since several relationships could not be set up properly in a reconstruction process, the relationship Works_On can be reconstructed as binary n:m-relationship but in general it cannot be determined between which entities it holds. To overcome this for a particular XML document one may either obtain the entity information from the key attribute of the XML element pointed to by the reference or use special annotations maintaining the name of the entity participating in a relationship.

Also relationship attributes of 1:1-relationships that were included in the entity element of the total participation side cannot be distinguished from regular attributes of that entity without additional annotations. Weak entities cannot be reconstructed as weak entities since they were modeled in the same way. Role names from the ER schema were not included in the DTD and can therefore not be obtained from DTD or document.

## 3 Example Create XML DTD from ER Schema

In this section we start with a motivating example which illustrates the translation of a well-known ER schema to a DTD. In the second part translation rules from this example are derived and ideas for translating additional modeling constructs are developed..

1. define <SchemaName> to be the root element
2. subelements with cardinality * of root element <SchemaName> are:
   (a) elements for all entities not occuring on the n-side of any 1:n-relationship
   (b) elements for all binary n:m-relationships and all k-ary relationships with k > 2
   (c) elements for all entities occuring only on n sides and only partially in relationships
3. while not all entity elements used are defined in the DTD define the next element by (favor entities having identifying relationships to weak entities):
   (a) introducing a subelement for each composite attribute
   (b) introducing a subelement with cardinality * or + for each multivalued attribute
   (c) defining a subelement for each relationship with weak entity types (cardinality as in the ER schema)
   (d) introducing a subelement for each 1:n-relationship where the current element is on the 1-side with cardinality * or +
   (e) defining XML attributes for all simple-type attributes of the current element (key attributes as ID, others as CDATA)
   (f) defining XML attributes for all attributes of 1:1-relationships where the current element participates totally and the other side is partial; define an IDREF attribute for the opposite side of such a relationship; following the same process, if the opposite side entity is a previously seen entity, even if participation is total
   (g) merging the opposite side entity and all relationship attributes into this element for all 1:1-relationships with total participation on both sides, if the opposite side entity is a new entity
   (h) for each relationship element obtained from (c) or (d):
   h1. defining a subelement for the opposite side of the relationship, if it is a new entity

   h2. defining an IDREF attribute for the opposite side of the relationship, if it is a previously seen entity
   h3. defining attributes for all relationship attributes as above with entity attributes
4. for all relationship elements from n:m- or k-ary relationships with k > 2:
   (a) introducing an IDREF attribute for all entities participating in this relationship
   (b) defining attributes for all relationship attributes as above with entity attributes

**Fig. 2.** Guideline to generate a DTD from an ER schema

In this section the translation is illustrated by applying it to a well-known ER schema of a company database which is shown in Fig. 3. The DTD generated by our guideline can be seen in Fig. 4.

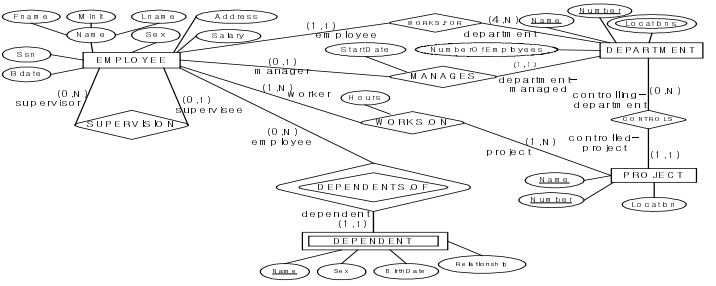


**Fig. 3.** Example ER schema of a company database, schema name Company

At first, there is a root element Company that represents the entire company database. Since entity department is the only entity that is not functionally dependent on another entity it is defined as subelement Department of the root element. Also the only n:m-relationship in the schema appears as direct subelement Works_On of the root element of the document. The structure of Department is given by subelement Location for the multivalued attribute locations and by subelements for the two functional relationships. Their XML cardinality (either * or +) can be derived from the participation of the opposite side in that relationship.

The 1:1-relationship manages is modeled by an IDREF attribute Manager for the opposite side and an attribute for the relationship attribute Start-Date. Subelements of the relationship elements Controls and Works_For introduce the elements for the dependent entities project and employee.

```
<!ELEMENT Company (Department*, WorksOn*)>
<!ELEMENT Department (Location*, Controls*, WorksFor, WorksFor, WorksFor, WorksFor+) >
  <!ATTLIST Department NameNumber ID #REQUIRED>
  <!ATTLIST Department ManagesStartDate CDATA #IMPLIED>
  <!ATTLIST Department Manager IDREF #REQUIRED>
<!ELEMENT Controls (Project) >
<!ELEMENT WorksFor (Employee) >
<!ELEMENT Location EMPTY>
  <!ATTLIST Location Name CDATA #IMPLIED>
<!ELEMENT Project EMPTY>
  <!ATTLIST Project NameNumber ID #REQUIRED>
  <!ATTLIST Project Location CDATA #IMPLIED>
<!ELEMENT Employee (Name, DependentsOf*, Supervision*) >
  <!ATTLIST Employee Ssn ID #REQUIRED>
  <!ATTLIST Employee Bdate CDATA #IMPLIED>
  <!ATTLIST Employee Adress CDATA #IMPLIED>
  <!ATTLIST Employee Sex CDATA #IMPLIED>
  <!ATTLIST Employee Salary CDATA #IMPLIED>
<!ELEMENT DependentsOf (Dependent) >
<!ELEMENT Supervision EMPTY>
  <!ATTLIST Supervision Supervisee IDREF #REQUIRED>
<!ELEMENT Dependent EMPTY>
  <!ATTLIST Dependent Name CDATA #REQUIRED>
  <!ATTLIST Dependent Sex CDATA #IMPLIED>
  <!ATTLIST Dependent BirthDate CDATA #IMPLIED>
  <!ATTLIST Dependent Relationship CDATA #IMPLIED>
<!ELEMENT WorksOn EMPTY>
  <!ATTLIST WorksOn Project IDREF #REQUIRED>
  <!ATTLIST WorksOn Worker IDREF #REQUIRED>
  <!ATTLIST WorksOn Hours CDATA #IMPLIED>
```

**Fig. 4.** DTD for the company schema

Features illustrated by the definition of Employee are the subelement Name for a composite attribute, the modeling of weak entity types similar to regular 1:n-relationships and modeling relationships to the entity itself. According to the type of supervision we use a subelement as before. The difference is in modeling the structure of subelement Supervision: we do not use any further subelements in order to avoid redundant information in the XML file since entity employee is already taken care of. Thus we use an IDREF attribute to point to the particular supervisees. Since we have subelements of Supervision for each supervisee we need only one such attribute.

Each strong entity type E of the ER schema is translated into an XML element of the same name E in the DTD. Each of the simple attributes Ai of E, which are usually assumed to be of a simple data type in ER schemas, is modeled as an XML attribute Ai belonging to element E. Key attributes in the schema should be translated to ID type attributes with the REQUIRED specification. Where as other attributes should be declared as CDATA with either IMPLIED or REQUIRED specification depending on whether they may be null or not. The problem that keys are locally unique whereas IDs are globally unique can be overcome by using the entity name as prefix to the key value in the document instance. Similarly we can use composite strings for composite keys. Of course this is still not an equivalent translation since key datatypes are not preserved and a proper retransformation from DTD to conceptual schema will have to rely on such additional conventions but it is probably the best we can do in the presence of DTDs. Each composite attribute A in the ER schema will be translated into a subelement A of E. The subelement A itself can be defined in much the same way as we developed E.

Each weak entity type W of the ER schema with owner entity type E is modeled by a subelement W of the identifying relationship between E and W. This relationship is itself modeled as subelement of E with cardinality. The identifying relationship itself is modeled as regular 1:n-relationships which are described below. All attributes of

A 1:1-relationship between entity types S and T where both participations are total should be translated by merging the corresponding XML elements S and T into one element. Attributes of the relationship are added to the XML element type.

Each regular binary 1:n-relationship R with entity S on the 1 side and T on the n side of R can be represented by a subelement R of S with cardinality * or + depending on the participation of T in R. The case where S and T are the same entity type has to be treated differently. Since all instances of S will already be included we should not use a subelement of R for the n side but rather an attribute of type IDREF.

Binary n:m-relationships R are mapped to top-level elements R in the DTD. The element R itself consists of two attributes which are defined as IDREF. These references will point to a pair of elements of R in the XML document later. By using IDREF XML documents with minimal redundancy are obtained since each tuple occurs only once and is only referenced at other places.

## 4 Extending the Generation to EER Schemas

Since the constructs of the traditional ER model were not sufficient for advanced application modeling numerous extensions have been proposed. The enhanced ER model (or EER model for short) as used in this article has been described in detail in [5] and uses extensions to the ER model similar to the ones presented in [6] and [7]. The additional constructs available are specialization and generalization, both with partial or total participation and disjoint or overlapping, as well as categories or union types. For our purposes we need not differentiate between specialization and generalization since we work on the finished schema where both have the same meaning.

The first and probably most common form of specialization is a disjoint specialization with total participation. We use XML elements for both super- and subclass entities where the elements of the subclasses contain a subelement representing the superclass. This

mapping is possible since every object belongs to exactly one subclass and the superclass is abstract. Only subclass elements are directly included in the DTD.

For a disjoint, partial specialization we need to include superclass elements in the DTD directly as well since it may be instantiated. Thus we need elements for all entities and include the superclass element directly in the DTD with an optional subelement to be chosen from all possible subclasses. That means, if we have a superclass G with possible subclasses S1,. . . ,Sn in the current specialization we include the term <!ELEMENT G (S1|. . . |SN)?> as description of G.

## 5 Relational Database Design of EER Data Model Base

Relational tables could use to actuality database system from EER model can be planed. But, EER model can see that model is model situated on midstream of relation design with object intention design, therefore partial difficult items are in conversion in relational table [8].

Relation type can be made in individual table with EER model's each Entity type. If the second relation type of Mapping is 1:1 or 1:m's case, relation type using (foreign key) to come form abroad in Entity type instead of is existed in individual table the relation mark can. And individual attribute exists in relation type or it is general that relation type of n:m case has individual table. In the case of generalization, according to restriction condition, various kinds selection is available.

ER schema model's EMPLOYEE has belonged to DEPARTMENT in Fig. 3 and EMPLOYEE large number of relation between each other has belonged by 1:n relational to one DEPARTMENT. Create schema and 1 side composes schema by own attribute including basis height of 1 side in n when change this to relational schema. 1:1 relational can lift relation that chief wealth of one person manages to a DEPARTMENT by relation that only other one individual corresponds to one individual. If change this by relational schema, include basis height in one side and make schema. In the case of n:m relational compose each by relational schema when several PROJECTs that EMPLOYEE plans belong to several PROJECT list and relation between each other composes by schema too. Fig. 5 makes out relational database about the EER schema example.

## 6 Object Relation Database Design of EER Data Model Base

In this chapters Explain about changing EER schema to object relation database. Objective-comparison of relation data type mechanism is same as following.

① Built-in types: They have maturity and high performance because they are compiled into the ORDBMS. They are good as building blocks for other types. They are inflexible, and because they are byte-level in nature, built-in types contribute little semantic value to the data model.

② DISTINCT TYPE: This is the easiest to use of the UDT mechanisms. Each DISTINCT TYPE has the same performance as its underlying type. It is the least flexible of UDT mechanisms, but it is useful if the new type is similar to an existing one (which is surprisingly often).

③ ROW TYPE: ROW TYPE objects are reasonably easy to use and mandatory for some object-relational features (inheritance). It supports compound objects (multi-element types well). In some situations ROW TYPE limits the range of OR-SQL features that can be used in queries involving them. They are not as fast as an OPAQUE TYPE equivalent.
④ OPAQUE TYPE: It is the fastest UDT mechanism, and the most flexible, and it provides good runtime performance on the broadest range of data types.

### 6.1 Strong Entities

For each strong entity in the EER model, create a table in which the table's name is the same as the entity name, each entity attribute has a corresponding column in the table, and the kind of data corresponding to each attribute is represented with the corresponding data type.

Each of the entity's attributes has a corresponding table attribute of the appropriate data type. Mandatory elements are established with NOT NULL column constraints. Strong entities should have at least one candidate key: either a subset of its attributes or a system-generated value to identify the row uniquely. Of course, any other candidate keys should be constrained with a UNIQUE constraint.

Earlier in this chapter we investigated how certain entities may be analyzed in more detail using object-oriented methods. If the entity is found to possess read methods that can be used in a query, the entity is best implemented as follows:
① Use a ROW TYPE to define the table/entity structure.
② Create a table using the ROW TYPE.
③ Add whatever constraints have been identified.

Once these steps are complete, the entity read methods can be implemented using UDFs. If these read methods are sufficiently common, such extensions should be implemented in C so that a functional index can be created later.

Again, we emphasize that using a ROW TYPE to define tables complicates the task of altering a table's structure. Consequently, use this approach only when necessary: either when the table is part of an inheritance hierarchy, or when some value can be computed from the table's rows with a user-defined function.

### 6.2 Weak Entities

You rarely encounter a strong entity on its own. More typically, the conceptual model consists of a network of interrelated weak and strong entities. Weak entities are also represented using table; the table's name is the same as the entity type's name, and it has a set of columns that correspond to the entity type's attributes in much the same way that a strong entity's table does.

When a weak entity is transformed into a table, the columns defining the primary key of the related strong entity are added to the columns defining the weak entity's attributes. Sometimes, these extra columns are called join columns.

We use the name of the table, Strong_Entity, as the name for the new column. Using table names to represent relationships is a convenience for developers who write queries; the new column could have any name that makes sense. Sometimes the role label from the

relationship is used instead. The constraints added after the tables are created enforce the referental integrity.

### 6.3 Modeling Complex Relationships

All examples we have seen involve converting entities into tables, but certain kinds of relationships are also represented this way. For instance, entity relationships that are N:M in their cardinality, or n-ary relationships involving more than one or two entities, are stored as rows in a table. The general idea is to create a table that has columns that are the same as the primary-key columns of the tables involved in the relationship and rules to ensure the correctness of its data.

You might want to enforce rules over the entire relationship in the same way you enforce rules over entity tables. For example, in circumstances in which an Employee may work for multiple Departments, and a single Department will consequently have many Employees, it may be desirable to ensure that each relationship between an Employee and a Department is recorded only once. In other words, the ORDBMS can ensure that a row of values is not repeated in the relationship table. Tables recording relationships are often called resolution tables. One advantage of this approach is that it is relatively straightforward to represent more complex kinds of relationships, such as n-ary relationships and relationships with associated attributes. Handling n-ary relationships also calls for creating a table containing key columns of all related tables, and possibly extending it with additional columns to contain any attributes of the relationship.

### 6.4 General Rules for Table Definitions

There are several "rules of thumb" concerning how attributes are handled. These rules are common to all tables created from EER diagrams.

① NULL values in columns complicate query processing and should be avoided. Some of the most notorious and long-running "wrong answer" problems reported to technical support organizations can be traced to an OR-SQL programmer misunderstanding one of the (many) subtle rules governing three-value logic. They should be avoided wherever possible.

② Columns that are include in a table's keys cannot contain NULL values. Columns where there is an obvious default - a type-specific token that means "Unknown" or "Did not respond," a system-generated value such as TODAY, or a generally acceptable value - should insert that value automatically. The only place NULL values are absolutely required is in columns for a FOREIGN KEY that is not mandatory.

③ Use CHECK constraints to enforce rules on entity attributes. Some rules on the data in a column will be enforced as part of the data type's implementation, but not all.

④ Name all constraints (with the possible exception of the NOT NULL constraints, which should be the de facto standard in the schema). User-defined types provide some control over the values columns can contain. However, when you have identified additional constraints as part of your analysis, it is useful to enforce them. This can lead to a large number of constraint rules per table. Naming constraints simplifies the task of managing them and determines which of them was violated by a particular operation. System-generated constraint names are totally impenetrable.

```
CREATE ROW TYPE DEPARTMENT_Structure (
  Name string NOT NULL, Number string NOT NULL, Locations string NOT NULL );
CREATE FUNCTION Read_Interface_Method
    (Argument DEPARTMENT_Structure)
RETURNS Result_Type
    RETURN Argument.name::CHAR || ' ' ||
    Argument.Number;
END FUNCTION;

CREATE TABLE DEPARTMENT
OF TYPE DEPARTMENT_Structure;

ALTER TABLE DEPARTMENT
ADD CONSTRAINT
  PRIMARY KEY ( Name )
  CONSTRAINT DEPARTMENT_PK;

CREATE TABLE EMPLOYEE (
  Name string NOT NULL, Sex string NOT NULL, Address string NOT NULL, Ssn string NOT
NULL, Bdate string NOT NULL, Salary string NOT NULL );
ALTER TABLE EMPLOYEE
ADD CONSTRAINT
  PRIMARY KEY ( Ssn )
  CONSTRAINT EMPLOYEE_PK;

CREATE TABLE DEPENDENT (
  Name string NOT NULL, Sex string NOT NULL, BirthDate string NOT NULL, Relationship
string NOT NULL   );

ALTER TABLE DEPENDENT
ADD CONSTRAINT
  PRIMARY KEY ( Name )
  CONSTRAINT DEPENDENT_PK;
CREATE TABLE PROJECT (
  NameNumber string NOT NULL, Locations string NOT NULL   );
ALTER TABLE PROJECT
ADD CONSTRAINT
  PRIMARY KEY ( Name )
  CONSTRAINT PROJECT_PK;
```

**Fig. 5.** Object relation database of EER schema example

## 7 Conclusion

The current information on the web is realized into the important property, the information is moving fast in webs, and the importance of information is increasing day after day. ER model is an information modeling tool that use most and show the relation between Entity and objects that display basic object that is required to system. Conversion methods are proposed by object relational database schema with the method changing from ER model are by an information modeling tool to XML DTD structure.

Integration design methodology that proposed in this research can be systematic, and plan efficiently XML application. The embodiment for DTD conversion is gone automatically in relational database schema using design guide given in this paper. The

more studies are necessary about the use of database techniques by the other example such as object intention or native XML finally.

## References

1. Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E.,Subramanian, S.,: XPERANTO: Publishing Object Relational Data as XML. In Suciu, D., Vossen(eds.), G., Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000, Dallas, Texas, USA, May 18–19, 2000, 105-110.
2. Kim Chea-mi, Choi Hak-yeal, Kim Sim-seok, XML Camp with Professional , Mighty Pr Inc, 2001.
3. Carsten Kleiner, Udo W. Lipeck, Automatic Generation of XML DTDs from Conceptual Database Schemas. GI Jahrestagung(1) 2001, 396–405
4. Paul Geoffrey Brown, Object-Relational Database Development: A Plumber's Guide, Prentice Hall PTR, December 22, 2000
5. R. Elmasri, S. B. Navathe: Fundamentals of Database Systems (Third Edition), 3rd edition. World Student Series, Addison-Wesley, Reading, MA, 2000.
6. T. J. Teorey, D. Yang, J. P. Fry: A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. ACM Computing Surveys 18:2 (1986), 197–22.
7. M. Gogolla, U. Hohenstein: Towards a Semantic View of an Extended Entity-Relationship Model. ACM Transactions on Database Systems 16:3 (1991), 369–416.
8. Robert J. Muller, Database Design for Smarties: Using UML for Data Modeling, Morgan Kaufmann Publishers, Inc, 1999.

**Choi Mun-Young** received his BS degree from Dept. of Computer Science, Chungwoon Uni. in 1998 and obtained MS degree from Dept. of Computer Science, Chungwoon Uni. in 2000.

Dept. of Computer Science, Graduate School Soonchunhyang Uni.
E-mail : griffin@hyejeon.ac.kr

**Jong-Seon Lim** received his B.S. degree from Dept. of Computer Science, Korea Uni. in 1997 and obtained M.S. degree from Dept. of Computer Science, Soonchunhyang Uni. in Korea.

Dept. of Computer Science, Graduate school Soonchunhyang Uni.
E-mail : ronmer@chol.com

**Joo Kyung-Soo** received his BS degree from Dept. of Mathematics, Korea Uni. In 1980 and obtained MS degree from Dept. of Computer Science, Korea Uni. In 1985 and obtained Ph.D degree from Dept. of Computer Science, Korea Uni. In 1993

Dept. of Computer Science, College of Engineering Soochunhyang Uni. Prof.
E-mail : gsoojoo@sch.ac.kr